A Graph Matching Attack on Privacy-Preserving Record Linkage[*]

Anushka Vidanage, Peter Christen
Thilina Ranbaduge
{anushka.vidanage,peter.christen,thilina.ranbaduge}@anu.edu.au
The Australian National University
Canberra, ACT, Australia

Rainer Schnell
rainer.schnell@uni-due.de
University Duisburg-Essen
Duisburg, Germany

ABSTRACT

To facilitate advanced analytics, data science projects increasingly require records about individuals to be linked across databases. Generally no unique entity identifiers are available in the databases to be linked, and therefore quasi-identifiers such as names, addresses, and dates of birth are used to link records. The process of linking records without revealing any sensitive or confidential information about the entities represented by these records is known as privacy-preserving record linkage (PPRL). Various encoding and encryption based PPRL methods have been developed in the past two decades. Most existing PPRL methods calculate approximate similarities between records because errors and variations can occur in quasi-identifying attribute values. Even though being used in real-world linkage applications, certain PPRL methods, such as popular Bloom filter encoding, have shown to be vulnerable to cryptanalysis attacks. In this paper we present a novel attack on PPRL methods that exploits the approximate similarities calculated between encoded records. Our attack matches nodes in a similarity graph generated from an encoded database with a corresponding similarity graph generated from a plain-text database to re-identify sensitive values. Our attack is not limited to any specific PPRL method, and in an experimental evaluation we apply it on three PPRL encoding methods using three different databases. This evaluation shows that our attack can successfully re-identify sensitive values from these encodings with high accuracy where no previous attack on PPRL would have been successful.

CCS CONCEPTS

• **Information systems → Entity resolution**; • **Security and privacy → Cryptanalysis and other attacks**; *Privacy-preserving protocols*; *Management and querying of encrypted data.*

KEYWORDS

Graph matching, graph alignment, feature generation, Bloom filter, tabulation hashing, two-step hashing, min hash

[*]Partially funded by the Australian Research Council under grant DP160101934.

ACM Reference Format:
Anushka Vidanage, Peter Christen, Thilina Ranbaduge, and Rainer Schnell. 2020. A Graph Matching Attack on Privacy-Preserving Record Linkage. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411931

1 INTRODUCTION

Linking records that belong to the same individual across different databases has gained increasing interest in the past few decades. Domains ranging from national security and healthcare to population informatics have a high demand for linking records for various reasons including data enrichment, advanced knowledge extraction, and missing value imputation [7]. Since unique entity identifiers across different data sources are generally not available, record linkage often needs to be conducted using sensitive personal information (known as quasi-identifiers [35]) of people, such as their names, addresses, and dates of birth [7]. However, such use of sensitive personal information can raise concerns about individuals' privacy and data security [6, 35]. Accordingly, rules and laws are in place in many countries, that regulate how and when personal information can be used. One example is the EU's General Data Protection Regulation (GDPR, see: https://gdpr-info.eu/).

The need for techniques that can perform record linkage using personal information without compromising the privacy of individuals led to the development of privacy-preserving record linkage (PPRL) methods [35]. Over the last few decades various PPRL methods have been proposed to enable the linkage of sensitive data while protecting the privacy of the individuals whose records are being linked. Generally, PPRL methods are based on encoding or encrypting quasi-identifiers that are used to perform the linkage in such ways that no sensitive information can be extracted during the linkage process. PPRL methods can be divided into two categories, *perturbation* and *secure multi-party computation* (SMC) based techniques [35]. SMC based techniques, while provably secure and accurate [22], generally have high computation and communication costs. Perturbation based techniques, on the other hand, have a trade-off between linkage quality, scalability to linking large databases, and privacy protection [35]. Perturbation based techniques are generally more practical for real-world linkage situations compared to SMC based techniques [6].

One popular perturbation based technique widely used for PPRL is Bloom filter (BF) encoding [4, 30], which we describe in detail in Sect. 2.1. Because of its ability to efficiently calculate approximate similarities between records, and the ease of implementation, BF

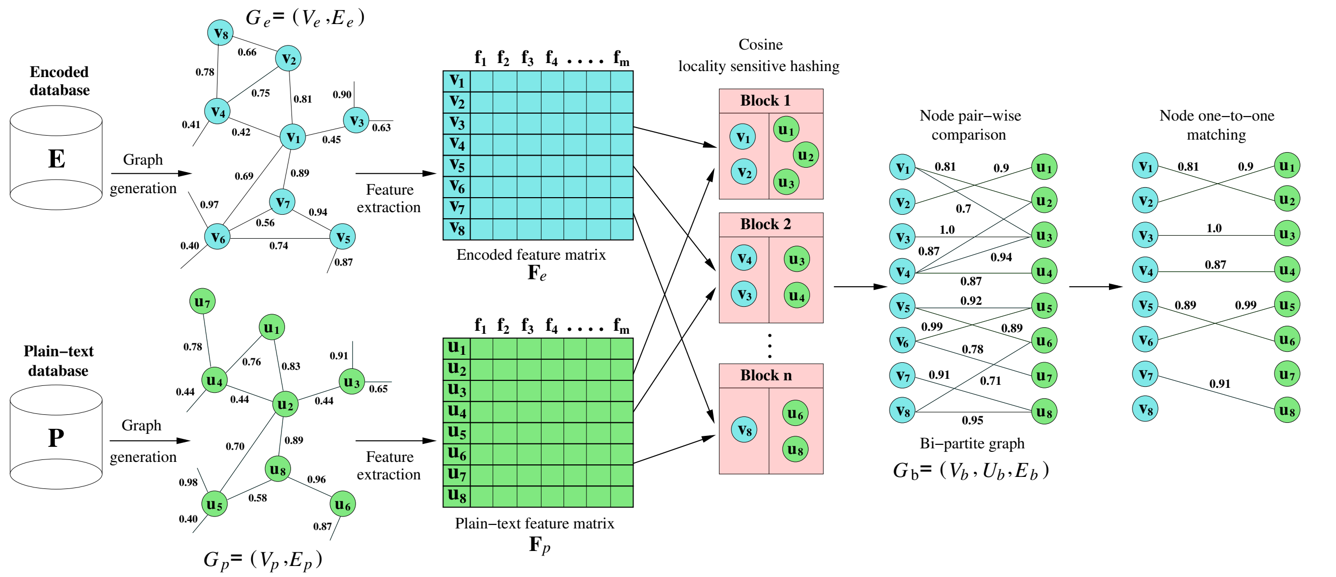


Figure 1: An overview of our proposed graph based attack. As we describe in Sect. 3, in the first step we generate two similarity graphs, G_e and G_p, from the encoded and plain-text databases, E and P, respectively. Next we extract a set of features, f, for each node in both graphs using the node's neighbourhood, resulting in the two feature matrices, F_e and F_p. We then use locality sensitive hashing (LSH) to identify groups of similar feature vectors in F_e and F_p, and build a bi-partite graph connecting nodes $\mathrm{v}_i \in \mathrm{F}_e$ with nodes $\mathrm{u}_j \in \mathrm{F}_p$ based on their similar feature vectors. Finally, we apply one-to-one matching to identify which node v_i from E corresponds to which node u_j from P, allowing us to re-identify the encoded value in v_i from u_j.

encoding is already being used in several real-world PPRL applications [3, 6, 26]. However, research has shown that the bit patterns in BFs, especially frequently occurring patterns, are susceptible to privacy attacks [8, 18, 20, 36], as we discuss in Sect. 2.2.

To overcome the privacy weaknesses of BF encoding, alternative perturbation based encoding techniques for PPRL have been developed. One of these uses tabulation Min-hashing [34] while another recently proposed method uses two hash encoding steps [27]. We discuss these two techniques in detail in Sect. 2.1. Given these encoding techniques for PPRL are to be used in real-world linkage applications where sensitive databases that contain personal information about people are being linked, it is important to understand the limitations and weaknesses of these encoding techniques in terms of the privacy protection they provide.

While the majority of attacks on PPRL are limited to BF encoding, we propose a novel attack technique that exploits the calculated similarities between encoded records using a graph based matching approach. As illustrated in Fig. 1, we generate one similarity graph from the encoded values and a second similarity graph from a set of plain-text values, where graph nodes correspond to the encoded or plain-text values, respectively, and graph edges correspond to the approximate similarities calculated between those values. We then apply graph matching techniques [1, 11, 15] to align nodes across these two graphs that are similar with regard to their local network structure, which allows us to match encoded values to plain-text values. This proposed attack can be applied on any PPRL encoding method that calculates similarities between encoded values.

We investigate different scenarios of the attack, such as different assumptions about the resources an adversary might have access to, in order to understand the strength and limitations of our attack on different PPRL encoding methods. Our experimental results show that in certain situations these encoding methods can be vulnerable to such a graph-based attack where encoded values can be re-identified with high accuracy.

Contributions: We specifically contribute (1) a novel attack using graph matching that works on any PPRL encoding method that calculates similarities between encoded values; (2) a method to adjust similarities between the encoded and plain-text graphs to improve the re-identification accuracy of our attack; and (3) an extensive experimental evaluation using three databases and three perturbation based PPRL techniques (BF encoding and two recent techniques that aim to overcome the vulnerabilities of BFs).

2 BACKGROUND

To provide the reader with an understanding of encoding methods used for PPRL, we now briefly describe the three methods we will use in the experimental evaluation of our attack: popular and practically used Bloom filter (BF) encoding [3, 6, 26, 28, 30], and the more recently proposed tabulation min-hashing (TMH) [34] and two-step hashing (2SH) [27] methods. We then provide an overview of existing attack methods on PPRL in Sect. 2.2.

For notation, throughout this paper we use bold letters for sets and lists (with upper-case bold letters for sets and lists of sets and lists), and normal type letters for integers and strings. Lists are shown with square and sets with curly brackets, where lists have an order while sets do not. For graphs, however, we use standard graph notation [5] where a graph is denoted by $G = (V, E)$ such that V is the set of nodes (vertices) and E is the set of edges.

2.1 Encoding Methods

We now describe how the three encoding methods BF, TMH, and 2SH can be used to encode sensitive values in the context of PPRL.

2.1.1 *Bloom Filter Encoding.*

Bloom filter (BF) [4] based encoding was first proposed in the context of PPRL by Schnell et al. [30] due to the ability of BFs to efficiently calculate set similarities while providing sufficient privacy. A BF $\mathbf{b}$ is a bit vector of length $l = |\mathbf{b}|$, where initially all bits are set to 0. The elements in a set $\mathbf{s}$ are hashed into $\mathbf{b}$ using $k \geq 1$ independent hash functions $h_1, \ldots, h_k$. Each hash function h_i, outputs an index which is used to set the bit at that index (position) in $\mathbf{b}$ to 1, $\mathbf{b}[h_i(s)] = 1$, with $1 \leq i \leq k$, $1 \leq h_i(s) \leq l$, and is performed $\forall s \in \mathbf{s}$.

In PPRL, a set $\mathbf{s}$ is generally the q-grams generated from one or more attribute values selected from each record in a database that needs to be encoded. A q-gram is a sub-string of length q characters extracted from a string using a sliding window approach [7, 30]. Each q-gram in $\mathbf{s}$ is hashed into a BF using k hash functions. Set based similarity functions, such as the Dice coefficient [7], can be used to calculate the similarity between two BFs. Collisions are possible where two different q-grams map to the same bit position. These can lead to increased Dice similarities and potentially to wrongly matched records [30]. Collisions can however also improve the privacy of BF encoding by distorting the frequency distributions of bit patterns [8, 20, 36], as we discuss in Sect. 2.2.

Different methods have been proposed to encode the sensitive attribute values of a record into BFs. Attribute-level BF (ABF) encoding [30] generates one BF per attribute, allowing multiple similarities can be calculated if several attributes are used to compare records. It has, however, been shown that ABF encoding is susceptible to frequency based privacy attacks [8]. Cryptographic long term key (CLK) [31] encoding was proposed to provide increased privacy guarantees. With CLK encoding multiple attribute values from a record are encoded into a single BF so that frequency based attacks would be more difficult [8]. However, CLK encoding can still be vulnerable to pattern mining based attacks [36]. Record level Bloom filter (RBF) encoding [10] was proposed as an alternative to CLK encoding and to overcome the privacy issues of ABF encoding. Using a weighted bit sampling process, RBF generates record level BFs which contain minimum frequency information [10].

To further improve the privacy of BF encoding, various hardening techniques have been proposed, including balancing, salting, XOR-folding, Markov chaining, and Bloom-and-FLIP (BLIP) [31–33]. However, there is generally a trade-off between improved privacy and the linkage quality obtained when using these techniques, because hardened BFs likely result in distorted similarities compared to plain-text similarities.

2.1.2 *Tabulation Min-hash Encoding (TMH).*

A tabulation hashing based method was proposed by Smith [34] as an alternative to BF encoding to encode sensitive attribute values. This method uses more complex hashing which makes it more secure than BF encoding. The basic idea behind this method is to use tabulation based hashing to perform min-hash based locality sensitive hashing.

Similar to BF encoding, in TMH encoding one bit array (of 0s and 1s) of length l is created for each record in the sensitive database to be encoded. First l sets of look-up tables are created, where each set consists of c look-up tables. Each look-up table contains keys of length k with a key space of size 2^k, where $1 < k$, and each key points to a random bit string. Each element in a set $\mathbf{s}$ is then hashed using a one-way hash function (such as MD5 or SHA1 [29]), and a fixed length binary value is extracted from such hash values. This binary value is divided into c sub-keys of length k, and each sub-key is used as the index key into one of the previously generated look-up table to obtain a random bit string. c such random bit strings are retrieved using c sub-keys, and these bit strings are then XORed to generate a single bit string for each of the l positions.

For each $s \in \mathbf{s}$ such a bit string is generated using this table look-up mechanism, and the minimum of all those bit strings is then selected as a min-hash signature. Since there are l sets of look-up tables, for a given set $\mathbf{s}$, l min-hash signatures are generated. Smith argued that generating tabulation min-hash signatures alone will not be able to provide enough security against privacy attacks [34]. Therefore, a 1-bit hashing mechanism is used to prevent against attacks where only the least significant bit of each of the previously generated l min-hash signatures is kept. These l bits are then concatenated into the final bit array that is used as the encoding of a sensitive value that is represented by the set $\mathbf{s}$ of q-grams.

This tabulation hashing based method provides improved security against privacy attacks but at the cost of much higher computational requirements. Generating l sets of look-up tables will increase both the time and memory consumption of the overall encoding process compared to BF encoding.

2.1.3 *Two-Step Hash Encoding (2SH).*

This recently proposed novel PPRL encoding method [27] aims to address both privacy issues of BF encoding as well as the high computational costs of TMH encoding. This method employs two efficient hashing steps based on bit vectors and an integer mapping to provide accurate Jaccard similarity calculations with improved privacy protection.

Similar to BF encoding, the 2SH encoding process first converts the quasi-identifying attribute values selected from each record into a set of q-grams $\mathbf{s}$. Next k independent hash functions, where $k > 1$, are employed to hash each q-gram $s \in \mathbf{s}$ into k bit vectors of length l. Unlike BF encoding, the 2SH encoding process generates one bit vector per hash function, resulting in k bit vectors each of length l, which corresponds to a bit matrix with k rows and l columns at the end of the first hashing step. In the second step of 2SH, each column of this matrix that contains at least one 1-bit is hashed by mapping the column bit vector into an integer value in a defined range. The column number along with a secret key is used in the hashing process such that two columns with the same bit pattern will not result in the same integer value. For hashing each column bit vector into an integer value, either a hash function such as SHA2 [29] or a pseudo-random number generator [29] can be employed. Columns that only contain 0-bits will not be hashed because they do not encode any q-grams, and 0-bit columns that are common in two bit matrices would result in additional common integer values in their corresponding encodings that would lead to incorrect similarity calculations.

At the end of this two-step hashing process each record from a sensitive database will be represented by a set of integers, and these sets can be used to calculate the Jaccard similarity [7] between pairs of encoded records.

2.2 Privacy Attacks on PPRL

Given its popularity and wide use in real-world linkage applications [3, 6, 26], understanding the limitations of BF encoding in terms of its vulnerabilities with regard to privacy attacks is highly important. We now briefly discuss previous attack methods proposed on (mainly) BF encoding for PPRL. No attacks have so far been developed specifically for TMH and 2SH encoding.

The first attack method for BF encoding in PPRL was proposed by Kuzu et al. [20]. The attack assumes that an adversary has access to a plain-text database and knows the number of hash functions k used in the BF encoding. Based on a constraint satisfaction problem solver, the attack aligns frequent q-grams with potentially matching bit positions such that certain constraints are satisfied.

The second attack on BF encoding was proposed by Niedermeyer et al. [24]. It exploits a weakness of the double hashing method used in the original BF encoding method by Schnell et al. [30]. Using potential bit patterns that can encode a single q-gram (called atoms) the attack can re-identify sensitive attribute values encoded in BFs. Using 7,580 unique German last names the attack was able to correctly re-identify the 934 most frequent last names. This attack was then extended into a fully automated attack of several encoded attributes by Kroll and Steinmetzer [18]. This extended attack was able to re-identify 44% of encoded values from 100,000 BFs.

A more recently proposed attack by Christen et al. [8] exploits the weaknesses of the BF construction principle. By aligning frequent BFs in an encoded database with frequent values in a plain-text database, the attack identifies bit positions that can and cannot encode certain q-grams. The attack was able to correctly re-identify the most frequent first and last names in a large database.

A novel type of attack based on pattern mining exploits the way frequent q-grams are hashed into BFs multiple times [36]. The attack does neither require knowledge of the encoding settings nor any frequent BFs in the encoded database, making this attack more practical for real-world scenarios. Using maximal frequent pattern-mining and a language model, the attack first identifies q-grams hashed into BFs. Plain-text values are then re-identified using those q-grams. The attack was able to exactly or partially re-identify around 35,000 encoded values from 220,000 BFs [36].

A graph based dictionary attack on BF encoding was proposed by Mitchell et al. [21]. The attack assumes that the adversary has complete knowledge of all parameters used in the BF encoding process, including the shared secret keys used in the hashing process. Using a brute force method and a directed graph based technique the attack re-identifies encoded plain-text values. The attack was able to correctly re-identify 77% of the encoded values from nearly 470,000 BF encoded records in a US voter database.

Culnane et al. [9] have applied a graph based attack on a PPRL method proposed by the UK Office for National Statistics which is based on a keyed-hash message authentication code (HMAC) [29] and similarity tables [25]. A directed similarity graph can be generated from these tables where nodes in the graph represent the encoded records in the sensitive database, and edges represent the similarities between these records. The problem of re-identifying encoded values in the similarity graph is then solved as a sub-graph isomorphism task. Using a plain-text graph, this attack was able to re-identify up to 93% from over 390,000 encoded records.

3 GRAPH BASED PRIVACY ATTACK

We now describe the four main steps of our graph based privacy attack, as illustrated in Fig. 1. We assume a linkage unit (LU) [35] is employed to conduct the linkage of the encoded databases it receives from the database owners. The LU can therefore build a similarity graph from the record pairs it is comparing. We assume the LU follows the honest-but-curious adversarial model [35], where it is curious to learn about the sensitive information in these encoded databases (the LU can be the adversary in our attack). We also assume the LU does not collude with any of the database owners, because this would allow a much easier attack following the ideas of Mitchell et al. [21] we described above. Furthermore, we assume the LU can guess the domain of the sensitive databases depending on its knowledge about the database owners.

Existing attacks on PPRL also make similar assumptions about the knowledge of an adversary [8, 18, 20, 24, 36]. Therefore, we assume that the adversary has access to an encoded database $\mathbf{E}$ of $n_e = |\mathbf{E}|$ records, which contains sensitive data of people encoded using a PPRL method such as BF, TMH, or 2SH discussed above. We also assume that the adversary has access to a plain-text database $\mathbf{P}$ of $n_p = |\mathbf{P}|$ records, that is from the same domain as $\mathbf{E}$. As most other PPRL attacks do, we assume the adversary does know or can guess the quasi-identifying attributes that were encoded in $\mathbf{E}$ [32].

3.1 Generating Similarity Graphs

In the first step of the attack we generate two similarity graphs using the two databases **E** and **P**. As can be seen from Fig. 1, the main idea behind our attack is that given the two databases are from the same domain then the two graphs will contain similar neighbourhoods for nodes that represent the same value (plain-text or encoded) across the two databases.

Consider a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges connecting those nodes. In a PPRL context, V corresponds to the set of unique values in a database and E to the similarities between those values. For a given pair of nodes $v_1 \in V$ and $v_2 \in V$, its edge attribute is $e_{1,2} = sim(v_1, v_2)$, where $sim()$ is the function used to calculate the similarity between two nodes. This can, for example, be the Dice or Jaccard similarity depending on the encoding method used [27, 30, 34].

We denote the two similarity graphs generated from the two databases **E** and **P** as $G_e = (V_e, E_e)$ and $G_p = (V_p, E_p)$. The encoded similarity graph will contain $|V_e| \leq n_e$ nodes and $|E_e| \leq n_e(n_e - 1)/2$ edges, whereas the plain-text similarity graph will contain $|V_p| \leq n_p$ nodes and $|E_p| \leq n_p(n_p - 1)/2$ edges.

In the plain-text database **P**, a node value is the unique set of q-grams used from a record, while in the encoded database **E** a node value is the unique set of random integers (2SH) or the unique bit vector (BF or TMH) representing an encoded sensitive value. There can be multiple records in **P** that have the same q-grams set, and multiple records in **E** that have the same encoding. In such scenarios we use a single node to represent all records with that q-gram set or encoding, and we set the node frequency as the number of records with that q-gram set or encoding. For example, if there are 10 records with the same first and last name in **P** then they will all have the same q-gram set, and a single node will represent all these records where we set the node frequency to 10.

Table 1: The node, edge, and neighbourhood features we generate for each node in graphs G_e and G_p.

NodeFreq	The number of records in **P** or **E**, respectively, that contain the value represented by a node $v \in V$
NodeLen	Length of q-gram set (plain-text graph), integer encoding set for 2SH, or number of 1-bits (Hamming weight) for BF and TMH
NodeDegr$_{s_m,c_m}$	Number of edges connected to a node $v \in V$ for given minimum similarity threshold s_m and minimum connected component size c_m
EdgeMax	Maximum similarity of the edges connected to a node $v \in V$
EdgeMin$_{s_m,c_m}$	Minimum similarity of the edges connected to a node $v \in V$, for given s_m and c_m
EdgeAvr$_{s_m,c_m}$	Arithmetic average of similarities of the edges connected to a node $v \in V$, for given s_m and c_m
EdgeStdDev$_{s_m,c_m}$	Standard deviation of similarities of the edges connected to a node $v \in V$, for given s_m and c_m
EgonetDegr$_{s_m,c_m}$	Degree of the induced subgraph for a node $v \in V$ which includes v and all of its direct neighbours, for given s_m and c_m
EgonetDens$_{s_m,c_m}$	Density of the induced subgraph for a node $v \in V$ which includes v and all of its direct neighbours, for given s_m and c_m
BetwCentr$_{s_m,c_m}$	Betweenness centrality, the ratio of the number of shortest paths that traverse through a node $v \in V$ to all shortest paths, for given s_m and c_m
DegrCentr$_{s_m,c_m}$	Degree centrality, the ratio of the nodes connected to a node $v \in V$ with regard to all nodes in the connected component, for given s_m and c_m
OneHopHisto$_{s_m,c_m}$	Degree distribution of the one-hop neighbour nodes of a node $v \in V$, for given s_m and c_m
TwoHopHisto$_{s_m,c_m}$	Degree distribution of the two-hop neighbour nodes of a node $v \in V$, for given s_m and c_m

To limit the number of edges between nodes in the graphs we generate, we use a minimum similarity threshold s_m, with $0 \leq s_m \leq 1$ (assuming $0 \leq sim(v_1, v_2) \leq 1$), and only consider similarities between nodes that are at least s_m. Higher values of s_m mean the generated graphs will contain less edges.

3.2 Node Feature Generation

The second step of the attack generates features for each node in V_e and in V_p. These features, as listed in Table 1, represent the characteristics of a given node based on its value and the edges connecting it to neighbouring nodes. We only consider nodes that are part of a connected component with a minimum size c_m (based on a minimum similarity threshold s_m on the edges in a graph, as discussed above), because otherwise the generated features would not contain enough information to distinguish nodes. For example, all nodes with no or only one edge connecting them to a single neighbouring node will likely have the same feature values making them indistinguishable in the graph matching step.

For each node in the two graphs we generate a feature vector, where our attack is based on identifying highly similar feature vectors to match pairs of plain-text and encoded values. The generated feature vectors will differ for different minimum similarity thresholds s_m and minimum connected component sizes c_m, because the neighbourhoods of nodes will be different with different number of edges connecting them. We discuss what values for s_m and c_m provided the best results in our experiments in Sect. 4. We generate three types of features, as shown in Table 1 and described next.

3.2.1 Node Based Features. Each node in the similarity graphs G_e and G_p has two attributes, its frequency and length. Node frequency is how many records in **E** or **P**, respectively, contain a certain q-gram set or encoded values, as we discussed above. As a node's length, for the plain-text graph G_p we use the number of q-grams in a node as its length, while for a node in G_e we use the Hamming weight (number of 1-bits) of the bit vectors for BF and TMH encoding, and the length of the integer set generated for 2SH.

3.2.2 Edge Based Features. For a graph generated based on given values of s_m and c_m, each node that is part of a connected component of size at least c_m will have one or more neighbouring nodes it is connected to. We calculate edge features for each such node v_i based on the set of its neighbouring nodes as $N(v_i) = \{v_j | (v_i, v_j) \in E\}$, and the set of edges connecting each neighbour $v_j \in N(v_i)$ to v_i as $E(v_i) = \{e_{i,j} | v_j \in N(v_i)\}$. As listed in Table 1, these edge features are the degree of the node v_i, $d(v_i) = |N(v_i)|$, and the minimum, $min(E(v_i))$, maximum, $max(E(v_i))$, average, $avg(E(v_i)) = (\sum_{j=1}^{|E(v_i)|} e_{i,j})/|E(v_i)|$, and standard deviation, $std(E(v_i))$, of neighbouring edges.

3.2.3 Structural Features. Under structural features we consider all nodes that are part of a connected component, not just the direct neighbours of a given node v_i. We calculate egonet degree and egonet density using the egonet of a node v_i. For a given node $v_i \in V$, the egonet of that node is the subgraph G' of G where $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$ which includes the node v_i, all its direct neighbours, and all the edges between them [1]. We also calculate centrality metrics such as betweenness centrality and degree centrality [1]. These centrality metrics measure the importance of a node in the graph based on the connectivity of that node within the graph. We also consider the degree distribution of a node's one-hop and two-hop neighbourhoods. We calculate neighbourhood node degree histograms based on a logarithmic scale for each node [16] and use them as features.

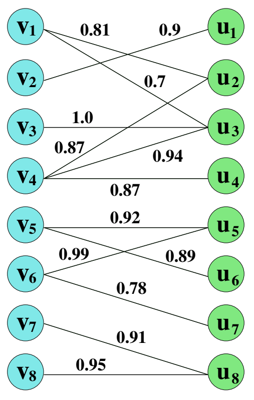
At the end of the feature generation process each node in V_e and in V_p will have a vector **f** of features assigned to it where $m = |\mathbf{f}|$ is the number of features considered. We denote the resulting two feature matrices for databases **E** and **P** as $\mathbf{F}_e$ and $\mathbf{F}_p$, respectively.

3.3 Node Matching

After generating the feature matrices $\mathbf{F}_e$ and $\mathbf{F}_p$, we next match nodes in the two graphs G_e and G_p to identify encoded values in **E** that correspond to plain-text values in **P**. We now describe the steps taken to perform node matching in an effective and efficient manner. Given we are processing feature vectors in both $\mathbf{F}_e$ and $\mathbf{F}_p$ in the same way, to improve readability we refer to both as **F**.

Many graph matching and alignment algorithms have been developed in research domains ranging from bioinformatics to multimedia information retrieval [11]. Such algorithms can be based on graph kernels [17], node and edge embeddings [14], or hashing [15]. In our approach, as discussed below, we use a Cosine locality sensitive hashing (LSH) based approach because it allows us to explicitly set a trade-off between the performance of the attack (as the number of feature vectors to be compared) versus the probability of comparing similar nodes [15].

3.3.1 Reducing Comparison Space using Locality Sensitive Hashing. With n_e nodes in the encoded graph G_e and n_p nodes in the plain-text graph G_p, there can be a maximum of $n_e \cdot n_p$ pairs of feature



Edges	Cosine similarity	Similarity confidence	Degree confidence
(v_1,u_2)	0.81	1.03	0.33
(v_1,u_3)	0.7	0.76	0.25
(v_2,u_1)	0.9	1.5	1.0
(v_3,u_3)	1.0	1.22	0.33
(v_4,u_2)	0.87	0.99	0.25
(v_4,u_3)	0.94	1.09	0.2
(v_4,u_4)	0.87	0.96	0.33
(v_5,u_5)	0.92	0.98	0.33
(v_5,u_6)	0.89	0.97	0.5
(v_6,u_5)	0.99	1.17	0.33
(v_6,u_7)	0.78	0.79	0.5
(v_7,u_8)	0.91	0.96	0.5
(v_8,u_8)	0.95	1.04	0.5

Figure 2: An example bi-partite graph matching, where encoded nodes v_i are connected to plain-text nodes u_i. For each edge the calculated Cosine similarity, similarity confidence, and degree confidence are shown, as discussed in Sect. 3.3.

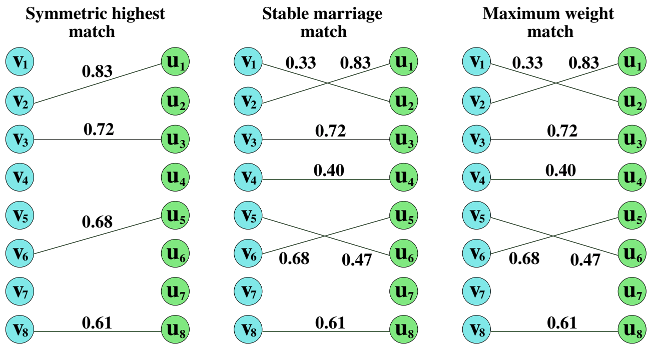


Figure 3: An example solution of the bi-partite graph matching problem from Fig. 2 using the weights $w_{cs} = 0.5$, $w_{sc} = 0.3$, and $w_{dc} = 0.2$ for Cosine similarity, similarity confidence, and degree confidence, respectively.

vectors for which we would need to calculate their similarities. This is not scalable for large databases. We therefore use LSH to reduce the comparison space. Since the node feature vectors, $\mathbf{f}$, are real-valued we use Cosine similarity based LSH [2], where d randomly generated column vectors (hyperplanes) $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_d\} \in \mathbb{R}^m$ are generated that follow a uniform stable distribution such as a Gaussian distribution. We use these column vectors as the LSH family of hash functions $\mathcal{H}$ to map the nodes into blocks using the each node's feature vector $\mathbf{f} \in \mathbf{F}$.

The mapping of node v_i, $h_d(v_i)$, where $h() \in \mathcal{H}$, is based on the sign of the dot product $\mathbf{f}(v_i) \cdot \mathbf{c}_j$:

$$h_d(v_i) = \begin{cases} 0, & \text{if } \mathbf{f}(v_i) \cdot \mathbf{c}_j < 0 \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

where $\mathbf{f}(v_i)$ is the feature vector of the node v_i and $\mathbf{c}_j$ is a random column vector $1 \leq j \leq d$. Given there are d such column vectors, each feature vector $\mathbf{f} \in \mathbf{F}$ of a node is therefore mapped to a binary vector space $\mathcal{H}(v_i) = \{0,1\}^d$ resulting in a bit vector of length d assigned to each node. We next use Hamming LSH on these mapped bit vectors to group the nodes in the two graphs G_e and G_p into similar blocks. Using this method the attack reduces the quadratic comparison space by removing likely non-matching nodes and grouping potential matching nodes into the same blocks.

3.3.2 *Bi-partite graph matching.*

After the LSH based blocking step, the comparison of nodes across the two graphs will become a bi-partite graph matching problem [1]. We compare pairs of encoded and plain-text nodes from similar blocks using a defined similarity metric, and then select pairs with a similarity higher than a user defined threshold b_m. Since node pairs are compared using their respective feature vectors, we use Cosine similarity to calculate their similarity. This results in a bi-partite graph $G_b = (V_b, U_b, E_b)$ where $V_b \subseteq V_e$ and $U_b \subseteq V_p$. It is not necessary to have $|V_b| = |U_b|$. There can be multiple edges connecting one encoded node to multiple plain-text nodes (and vise-versa) in G_b, as shown in Fig. 2. Note that from here onward we refer to a node in V_b as v and to a node in U_b as u to improve readability.

We select a set of edges, $E_s = \{(v,u), v \in V_b, u \in U_b\}$, where $E_s \subseteq E_b$, between encoded and plain-text nodes in such a way that for every pair of edges $e_1, e_2 \in E_s$, e_1 and e_2 do not share common nodes. Formally $\exists(v,u) \in E_s \iff \forall(v',u') \in E_s : (v,u) \neq (v',u'); (v' \neq v) \wedge (u' \neq u)$. Fig. 2 shows an example bi-partite matching problem of two sets of nodes from G_e and G_p. We define three different edge weights: (1) Cosine similarity, (2) similarity confidence, and (3) degree confidence. These can be used when matching nodes in G_b. The Cosine similarity, $cs(v,u)$, of a given edge is calculated as [15] $cs(v,u) = \frac{\mathbf{f}(v) \cdot \mathbf{f}(u)}{||\mathbf{f}(v)|| \cdot ||\mathbf{f}(u)||}$, where $\mathbf{f}(v)$ and $\mathbf{f}(u)$ are the normalised feature vectors calculated for the two nodes v and u. The similarity confidence, $cs(v,u)$, measures the reliability of an edge value with respect to all the other values of the edges connected to the nodes v and u. The similarity confidence, $sc(v,u)$, of a given edge can be calculated as $sc(v,u) = \frac{cs(v,u) \cdot (p+q-2)}{\sum_{i=1}^{p-1} cs(v,u_i) + \sum_{j=1}^{q-1} cs(v_j,u)}$, where $cs(v,u)$ is the Cosine similarity of the node pair (v,u), $p-1$ is the number of other edges connected to the encoded node v, and $q-1$ is the number of other edges connected to the plain-text node u. Finally, the degree confidence, $dc(v,u)$ measures the reliability of an edge (v,u) with respect to the number of other edges connected to the nodes v and u. The degree confidence of a given edge can be calculated as $dc(v,u) = \frac{1}{p+q-1}$, where p is the degree of the encoded node v and q is the degree of the plain-text node u. In Fig. 2 we show the calculation of these edge weights in the given bi-partite graph.

Using the above three weights defined for each edge in G_B we then aim to match encoded nodes with plain-text nodes. The three types of weights can be combined in a weighted way, using w_{cs}, w_{sc}, and w_{dc}, where $w_{cs} + w_{sc} + w_{dc} = 1.0$, based on their importance in being able to identify the correct plain-text node for an encoded node. Note that since the numerical values calculated by these three weight types are not in the same range, we normalise them prior to matching. We discuss the suitability of these three weight types to correctly match nodes across the two graphs in Sect. 5.

The actual matching of nodes can be performed using different methods. Franke et al. [12] proposed three methods of performing one-to-one matching of nodes between a bi-partite graph.

The symmetric highest match (SHM) method finds, for each node in both encoded and plain-text graphs, the edge that has the highest weight. Here the edge weight can be a value calculated using one of the above three weight types, or a weighted average of the three weight types. For instance in Fig. 2 the encoded node v_6 and the plain-text node u_5 have three edges (v_6, u_5), (v_6, u_7), and (v_5, u_5). In Fig. 3, when we calculate the weighted edge values we get $(v_6, u_5) = 0.68$, $(v_6, u_7) = 0.22$, and $(v_5, u_5) = 0.49$. As can be seen, since the highest match for both v_6 and u_5 is the edge connecting these two nodes, it is selected while all the other edges connected to those two nodes are discarded.

The stable marriage match (SMM) method solves the problem of matching nodes by finding stable matches in the bi-partite graph G_b. A matched edge is defined as *stable* if there are no other edges connecting both nodes that have higher weight than the matched edge. We use the Gale–Shapley algorithm [13] to solve this problem. However, compared to the original algorithm [13], our specific case of matching nodes from the encoded graph to the plain-text graph requires two modifications. The original algorithm assumes that each node from one side of the graph G_b has a complete preference list including every node from the other side. As can be seen from Fig. 2 this is not the case for our problem. The original algorithm also assumes equal sized sets of nodes in both side of the graph G_b. This might not be always true in our case because we use a minimum connected component size threshold c_m and LSH based blocking to group similar nodes. We have modified the original algorithm to accept these changes and identify the stable matches across the bi-partite graph G_b.

Finally, the maximum weight match (MWM) method corresponds to the assignment problem in a bi-partite graph where the set of edges E_s is selected in such a way that the sum of the weights of those edges are maximised. This assignment problem can be solved using the Kuhn-Munkres algorithm (also known as the Hungarian algorithm) [19] in polynomial time.

In Fig. 3 we show example graph matching results obtained using the above discussed three matching techniques with weight values $w_{cs} = 0.5$, $w_{sc} = 0.3$, and $w_{dc} = 0.2$ for each edge weighting technique. At the end of this matching step, no node in the resulting set of edges, E_s, will occur in more than one edge.

3.4 Plain-text Value Re-identification

Once nodes are matched across the encoded and plain-text graphs in the bi-partite graph G_b, the final step of our attack is to perform plain-text value re-identification. Since the selected set of edges E_s from the previous step only consists of one-to-one matches, we can align the plain-text value associated with a node from G_p to its corresponding encoded node in G_e using the edges E_s in G_b.

First the attack sorts all the edges in $(v, u) \in E_s$ according to their edge weights in descending order. Then we loop over each edge and retrieve the corresponding encoded and plain-text values from the databases **E** and **P**, respectively. We continue this process only for the top t pairs of nodes with the highest edge weights, where t is a user defined value. With larger t we will potentially match and re-identify incorrect values as their similarities become lower. In this assignment process there can be one-to-many or many-to-many matches of records as a node from either graph, G_e or G_p, can represent more than one record (as discussed in Sect. 3.1), where however all these records have the same value.

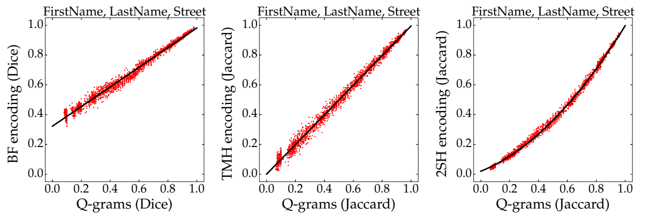


Figure 4: The relationship between encoded and plain-text similarities using the encoding techniques BF (left), TMH (center), and 2SH (right). The regression models (black lines) are build using the EURO database (as discussed in Sect. 4).

3.5 Adjusting Similarities

For our attack method to work accurately, it is important to have similar neighbourhood structures for nodes representing the same value across the two graphs. Assuming that the adversary can estimate the parameter values used in the encoding of values (even if not all parameter values are set correctly), she can perform a regression analysis using the plain-text database to determine the relationship between the encoded and plain-text similarity distributions. Some encoding parameters, such as BF length l, can be learned by analysing the encoded values themselves. Other parameters, such as the number of hash functions used, k, can be approximated by comparing the Hamming weight distribution of BFs to the distribution of the number of q-grams in plain-text values. By referring to previous publications and/or real-world applications where encoding techniques are used for PPRL the likely used parameter settings can also be obtained. It is important to note that by obtaining these parameters an adversary cannot simply reconstruct the exact encodings (BF, THM, or 2SH) because the hash functions and secret key used will not be known to the adversary.

In Fig. 4 we show three example regression models based on the three encoding techniques. As can be seen, the relationships between the encoded and plain-text similarities are approximately linear with BF and TMH encoding, and slightly non-linear with 2SH encoding. However, BF similarities are significantly higher than the corresponding plain-text q-gram set similarities due to collisions. Using such regression models, the adversary can adjust plain-text similarities such that they align with encoded similarities when generating the plain-text graph G_p. Using this additional step during the similarity graph generation of our attack, the accuracy of the attack can be significantly increased because when encoded and plain-text similarities are aligned the overall structure of the two generated graphs, G_e and G_p, will be more similar. This leads to the corresponding nodes in the encoded and plain-text graphs to have highly similar neighbourhoods, and therefore highly similar feature vectors. In Sect. 5 we discuss the accuracy improvements we gained using this technique.

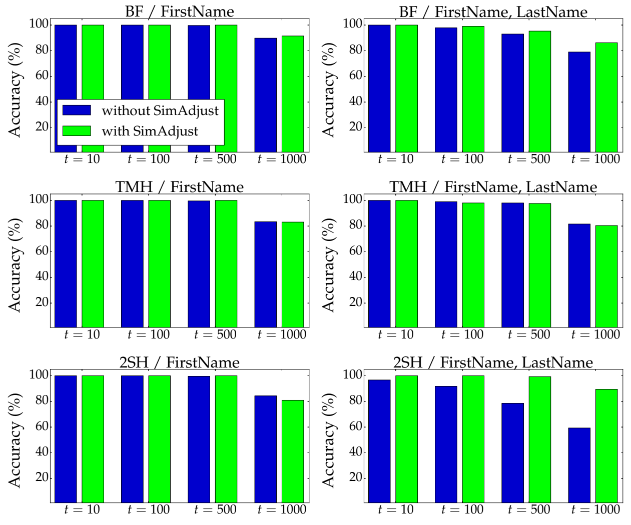


Figure 5: Accuracy results for re-identified plain-text values on the TITANIC database, as discussed in Sect. 3.5.

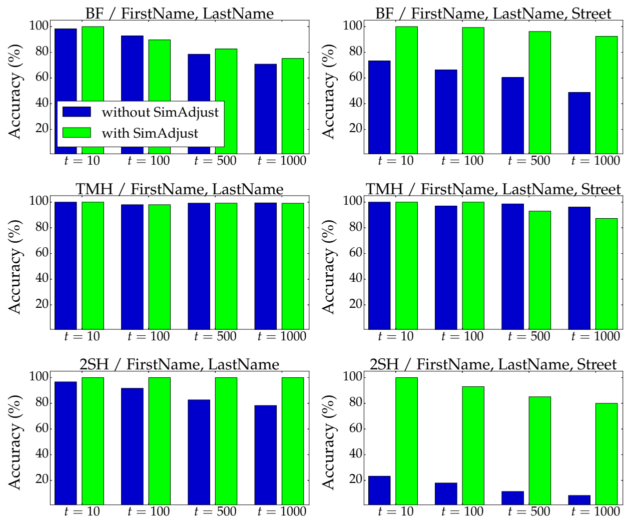


Figure 6: Accuracy results for re-identified plain-text values on the EURO database, as discussed in Sect. 3.5.

4 EXPERIMENTAL EVALUATION

We conducted an experimental evaluation of our proposed graph matching attack on PPRL encoding methods using two real-world databases and one synthetic database. We measured the accuracy of the attack under different parameter settings to investigate the feasibility of our attack in real-world linkage situations.

Databases: The first database we used is the Titanic (TITANIC) database[1] [34] which contains the names of 1,317 passengers. We extracted the *FirstNames* and *LastNames* of the passengers separately into two attributes to be encoded. The second database is

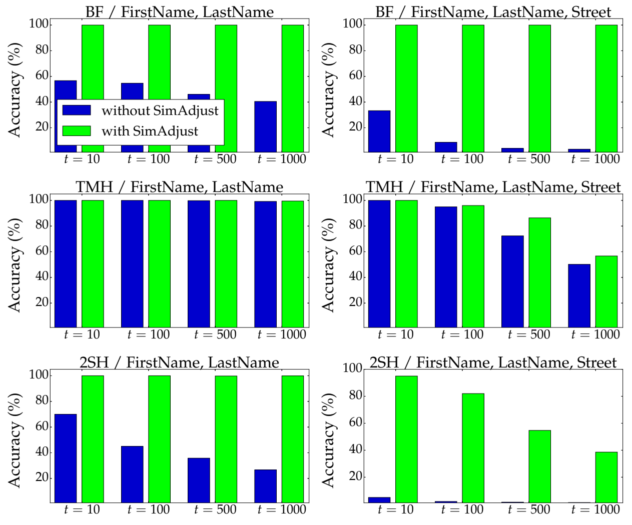


Figure 7: Accuracy results for re-identified plain-text values on the NCVR database, as discussed in Sect. 3.5.

a synthetic European census (EURO) database[2] generated to represent real observations of the decennial census. This database contains names and addresses of 25,343 people. The third database was the North Carolina Voter Registration (NCVR) database[3] where we used two snapshots from February and April 2019, each containing 100,000 voter records with 100% overlap of voter identifiers. The NCVR database also contains name and address details of voters.

For the TITANIC and EURO databases we used the same database as both the encoded, **E**, and plain-text, **P**, databases, while for the NCVR database we used the snapshot from April 2019 as the encoded database **E** and the snapshot from February 2019 as the plain-text database **P**. We used different combinations of the attributes *FirstName*, *LastName*, and *StreetAddress* to generate different instances of encoded databases using the three encoding methods BF, TMH, and 2SH. These attributes are commonly used in PPRL to link records across different databases [35].

Evaluation Criteria and Setup: To evaluate the quality of re-identification of plain-text values from encoded values we analysed the accuracy of the top one-to-one t re-identifications (when sorted according to the edge weight between encoded and plain-text nodes, as discussed in Sect. 3.4) where $t = [10, 100, 500, 1000]$. We then calculated the percentage of correct re-identifications from these top t identified values as the accuracy we report. We also discuss the usability of the edge weighting and bi-partite graph matching approaches to accurately match plain-text nodes from **P** to encoded nodes from **E**, as described in Sect. 3.3.

Following earlier work [30, 35], for BF encoding we used the parameter settings: $q = 2$, $l = 1000$, and $k = [10, 15, 20]$. BFs were encoded using both the CLK and RBF approaches [10, 31]. Following [34], for TMH encoding we used $c = 8$ look-up tables per min-hash signature, where in each look-up table bit keys of length $k = 8$ were used to refer to 64 bit random bit strings. Using the

[1] Available at: https://en.wikipedia.org/wiki/Passengers_of_the_RMS_Titanic

[2] Available at: https://ec.europa.eu/eurostat/cros/content/job-training_en

[3] Available at: https://dl.ncsbe.gov/?prefix=data

Table 2: Percentages of accurate re-identifications using different bi-partite graph matching and edge weighting methods, as discussed in Sec. 3.3.2, for the EURO database with respect to all 25,343 potential matches. Percentages are shown without and with similarity adjustments applied. In the *all-three* edge weighting method we used the weights $w_{cs} = 0.6$, $w_{sc} = 0.3$, and $w_{dc} = 0.1$ for Cosine similarity, similarity confidence, and degree confidence, respectively.

Attribute Combination	Encoding Method	Graph matching method			Graph edge weighting method			
		SHM	SMM	MWM	$cs(v,u)$	$sc(v,u)$	$dc(v,u)$	*all-three*
$FirstName, LastName$	BF	16.9 / 32.0	21.1 / 34.0	14.8 / 32.4	19.0 / 39.4	21.5 / 39.8	8.5 / 12.2	21.5 / 39.8
	TMH	40.7 / 41.4	47.1 / 47.8	35.6 / 36.7	51.9 / 53.3	52.7 / 54.0	5.9 / 5.5	54.0 / 55.1
	2SH	20.6 / 43.4	25.5 / 49.7	16.8 / 40.6	24.7 / 56.7	27.3 / 57.8	4.6 / 5.0	27.2 / 58.8
$FirstName, LastName, StreetAddress$	BF	7.6 / 57.0	12.5 / 62.5	5.1 / 60.1	8.2 / 66.3	10.0 / 66.9	5.5 / 38.8	10.0 / 67.3
	TMH	23.2 / 24.0	31.3 / 31.7	19.4 / 17.1	30.9 / 31.1	32.0 / 30.3	2.7 / 2.2	32.9 / 33.4
	2SH	0.4 / 16.1	0.8 / 22.0	0.2 / 11.6	0.3 / 19.6	0.4 / 21.6	0.6 / 2.5	0.4 / 22.5

obtained min-hash signatures, bit arrays of length $l = 1000$ were generated for each encoded value [34]. For 2SH encoding, we used the parameter settings: $q = 2$, $l = 1000$, and $k = [10, 15, 20]$ [27]. The accuracy results discussed below for each database were averaged over all these parameter settings. We set $s_m = [0.2, 0.3, 0.4]$ for the minimum similarity threshold and $c_m = [5, 10, 50, 100]$ for the minimum connected component size threshold. Furthermore, we set $b_m = [0.8, 0.9]$ as the minimum similarity threshold when generating the bi-partite graph as discussed in Sect. 3.3.2. For the alignment of similarities between encoded and plain-text databases we used polynomial regression [23] and built regression models using a sample size of 5,000 edge (similarity) pairs from the plain-text graph. These parameter values provided accurate results in a series of set-up experiments.

We implemented the proposed graph matching attack method using Python 2.7 and all experiments were run on a server with 64-bit Xeon 2.1 GHz 16-Core CPU, 512 GBytes of memory and running Ubuntu 18.04. To facilitate repeatability the prototype programs are available from: https://dmm.anu.edu.au/pprlattack/.

5 RESULTS AND DISCUSSION

In Fig. 5 we show accuracy results for the TITANIC database with two different attribute combinations and both without and with similarity adjustments (as discussed in Sect. 3.5). As can be seen, the attack was able to re-identify encoded sensitive values with more than 80% accuracy with all three encoding techniques. With 2SH encoding the accuracy of re-identifications without similarity adjustment drops because of the similarity differences as well as the slightly non-linear relationship between encoded and plain-text similarities (as shown in Fig. 4).

Figures 6 and 7 illustrate re-identification accuracy results for the EURO and NCVR databases, respectively. As can be seen, with two attribute combinations our attack achieves high re-identification accuracy for all three encoding techniques. However, the results without similarity adjustment drop when more attributes are encoded because with more attributes the neighbourhoods of dissimilar nodes (which corresponds to non-matching values) across the two graphs G_e and G_p become less distinct as they get connected by more edges. This can be seen in the results for both BF and 2SH encoding. Out of the three encoding techniques, TMH gave the best results, both with and without similarity adjustments. This is because the similarities between TMH encodings are highly similar to the corresponding plain-text q-gram set similarities (as can be seen in the center plot of Fig. 4). This results in encoded and plain-text graphs, G_e and G_p, with highly similar node neighbourhoods. BF encoding provided slightly better re-identification accuracy both with and without similarity adjustments because of the linear relationship between encoded and plain-text similarities (as shown in the left plot in Fig. 4). The worst re-identification results are obtained for 2SH encoding with three attribute combination. Overall, our attack achieved better re-identification accuracy with similarity adjustment compared to without similarity adjustment.

Table 3: Runtime results (in minutes) for different steps of our proposed graph attack (averaged over all other parameter settings).

Database	Encoding Method	Graph Gen.	Feat. Gen.	Node Match.	Plain-text Re-ident.
EURO	BF	7.11	449.36	30.0	< 1
	TMH	5.75	364.46	148.34	< 1
	2SH	23.78	514.75	201.2	< 1
NCVR	BF	12.25	774.21	85.26	< 1
	TMH	11.21	419.54	116.43	< 1
	2SH	34.67	647.88	86.69	< 1

In Table 2 we show the percentages of accurate one-to-one re-identifications (averaged over all other parameter settings) for the three different bi-partite graph matching techniques, SHM, SMM, and MWM, and the three different bi-partite graph edge weighting approaches, $cs()$, $sc()$, and $dc()$. Overall, the SMM matching method performed relatively better compared to other two matching methods, where SHM performed slightly better than MWM. Furthermore, the experiments with MWM matching consistently took considerably longer time compared to the SHM and SMM. From the three edge weighting approaches, $cs()$ and $sc()$ performed better than $dc()$ in almost all experiments, where the $sc()$ approach performed slightly better than $cs()$. However, in general the weighted use of all three approaches (*all-three*) provided better results compared to using a single approach. Similar results were obtained for the other two databases (TITANIC and NCVR). These results are excluded from the paper due to space constraints.

There are two main limitations to our proposed graph matching attack: First, if the differences between the encoded and plain-text similarities are high and if the relationship between those similarities is not linear, then it will be difficult to obtain accurate results. Second, the more attributes are encoded the lower the accuracy of re-identification. This is because of the differences in similarity values across the two graphs as well as having similar graph neighbourhoods for dissimilar nodes.

In Table 3 we show the runtimes of our attack. As can be seen, feature generation is the most time consuming step while the plain-text re-identification step took only seconds to complete.

These experimental results of our proposed graph matching attack illustrate the limitations of encoding techniques such as BF, TMH, and 2SH, when used with certain parameter settings. Even though the TMH and 2SH encoding methods are proposed to overcome the privacy vulnerabilities of BF encoding, they are still vulnerable to privacy attacks, especially when only one or two attributes are encoded. With proper similarity adjustments, attribute values encoded using these methods can still be re-identified even when more than two attributes were encoded. Furthermore, our attack will work on any other encoding technique which calculates approximate similarities between record pairs.

Given the popularity of perturbation based encoding techniques to be used in real-world PPRL applications [3, 6, 26, 28, 30], it is important to carefully consider and evaluate the privacy implications of such techniques before using them in practical applications. Database owners can use our attack to evaluate the privacy guarantees of encoding techniques and the parameter settings they plan to use. Moreover, our attack has emphasised the need of further research into developing more advanced privacy techniques to be used in PPRL, where such techniques still need to be scalable to large databases and provide high linkage quality.

6 CONCLUSION AND FUTURE WORK

We have presented a novel graph matching attack on privacy-preserving record linkage (PPRL) that, unlike earlier attack methods for PPRL, can be applied on any PPRL method that calculates similarities between encoded values. Our attack re-identifies sensitive values in an encoded database from a known plain-text database by using graph matching. We successfully applied our attack on three encoding methods, popular Bloom filters [6, 10, 26, 28, 30], tabulation min-hashing [34] and two-step hashing [27]. Given the increasing demand for such perturbation based encoding methods in real-world PPRL applications it is essential to investigate the privacy limitations of such methods.

As future work, we plan to improve the attack in three ways: (1) investigate supervised and unsupervised machine learning methods to conduct the alignment between encoded and plain-text similarities without having knowledge of the encoding parameters used; (2) explore how our attack works with privacy improvements such as hardening techniques proposed on perturbation based encoding methods [31–33], and (3) compare the accuracy and efficiency of our attack with other privacy attacks proposed for PPRL.

REFERENCES

[1] Charu Aggarwal and Haixun Wang. 2010. *Managing and Mining Graph Data.* Advances in Database Systems, Vol. 40. Springer.
[2] Alexandr Andoni and Piotr Indyk. 2008. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM* 51 (2008).
[3] Manfred Antoni and Rainer Schnell. 2017. The past, present and future of the German Record Linkage Center. *Journal of Economics and Statistics* 239, 2 (2017).
[4] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.
[5] Adrian Bondy and Uppaluri Murty. 2011. *Graph Theory.* Springer, London.
[6] James H Boyd, Sean M Randall, and Anna M Ferrante. 2015. Application of Privacy-Preserving Techniques in Operational Record Linkage Centres. In *Medical Data Privacy Handbook.* Springer, 267–287.
[7] Peter Christen. 2012. *Data matching – Concepts and techniques for record linkage, entity resolution, and duplicate detection.* Springer.
[8] Peter Christen, Thilina Ranbaduge, Dinusha Vatsalan, and Rainer Schnell. 2019. Precise and Fast Cryptanalysis for Bloom Filter Based Privacy-Preserving Record Linkage. *IEEE TKDE* 31, 11 (2019), 2164–2177.
[9] Chris Culnane, Benjamin Rubinstein, and Vanessa Teague. 2017. Vulnerabilities in the use of similarity tables in combination with pseudonymisation to preserve data privacy in the UK Office for National Statistics' Privacy-Preserving Record Linkage. *arXiv Preprint* (2017).
[10] Elizabeth Durham, Murat Kantarcioglu, Yuan Xue, Csaba Toth, Mehmet Kuzu, and Bradley Malin. 2014. Composite Bloom filters for secure record linkage. *IEEE TKDE* 26, 12 (2014), 2956–2968.
[11] Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. 2016. Fifty years of graph matching, network alignment and network comparison. *Information Sciences* 346 (2016), 180–197.
[12] Martin Franke, Ziad Sehili, Marcel Gladbach, and Erhard Rahm. 2018. Post-processing Methods for High Quality Privacy-Preserving Record Linkage. In *DPM/CBT@ESORICS.* Springer, Cham, 263–278.
[13] David Gale and Lloyd Shapley. 1962. College Admissions and the Stability of Marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15.
[14] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *KDD.* ACM, New York, NY, USA, 855–864.
[15] Mark Heimann, Wei Lee, Shengjie Pan, Kuan-Yu Chen, and Danai Koutra. 2018. HashAlign: Hash-Based Alignment of Multiple Graphs. In *PAKDD.* Melbourne.
[16] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. 2018. REGAL: Representation Learning-Based Graph Alignment. In *CIKM.* ACM, Torino, 117–126.
[17] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. 2019. A Survey on Graph Kernels. *Applied Network Science* 5 (2019), 6.
[18] Martin Kroll and Simone Steinmetzer. 2015. Who Is 1011011111...1110110010? Automated cryptanalysis of Bloom filter encryptions of databases with several personal identifiers. In *BIOSTEC.* Springer, Lisbon, 341–356.
[19] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2, 1-2 (1955), 83–97.
[20] Mehmet Kuzu, Murat Kantarcioglu, Elizabeth Durham, and Bradley Malin. 2011. A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In *PET.* Springer, Waterloo, 226–245.
[21] William Mitchell, Rinku Dewri, Ramakrishna Thurimella, and Max Roschke. 2017. A graph traversal attack on Bloom filter-based medical data aggregation. *IJBDI* 4, 4 (2017), 217–226.
[22] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge University Press.
[23] Douglas Montgomery, Elizabeth Peck, and Geoffrey Vining. 2012. *Introduction to Linear Regression Analysis* (5 ed.). Wiley.
[24] Frank Niedermeyer, Simone Steinmetzer, Martin Kroll, and Rainer Schnell. 2014. Cryptanalysis of basic Bloom filters used for privacy preserving record linkage. *JPC* 6, 2 (2014), 59–79.
[25] Office for National Statistics UK. 2013. Beyond 2011 Matching Anonymous Data. Methods and Policies Report M9.
[26] Robespierre Pita, Clícia Pinto, Samila Sena, Rosemeire Fiaccone, Leila Amorim, Sandra Reis, Mauricio Barreto, Spiros Denaxas, and Marcos Ennes Barreto. 2018. On the accuracy and scalability of probabilistic data linkage over the Brazilian 114 million cohort. *IEEE JBHI* 22, 2 (2018), 346–353.
[27] Thilina Ranbaduge, Peter Christen, and Rainer Schnell. 2020. Secure and Accurate Two-sep Hash Encoding for Privacy-Preserving Record Linkage. In *PAKDD.* Springer, Singapore, 139–151.
[28] Sean Randall, Anna Ferrante, James Boyd, Jacqueline Bauer, and James Semmens. 2014. Privacy-preserving record linkage on large real world datasets. *JBI* 50 (2014), 205–212.
[29] Bruce Schneier. 1996. *Applied cryptography: Protocols, algorithms, and source code in C* (2 ed.). John Wiley and Sons, Inc., New York.
[30] Rainer Schnell, Tobias Bachteler, and Jorg Reiher. 2009. Privacy-preserving record linkage using Bloom filters. *BMC Med Inform Decis Mak* 9, 41 (2009).
[31] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. 2011. A Novel Error-Tolerant Anonymous Linking Code. *SSRN Electronic Journal* (01 2011).
[32] Rainer Schnell and Christian Borgs. 2016. Randomized response and balanced Bloom filters for privacy preserving record linkage. In *ICDMW DINA.* Barcelona.
[33] Rainer Schnell and Christian Borgs. 2016. XOR-Folding for Bloom Filter-based Encryptions for Privacy-preserving Record Linkage. *SSRN* (2016).
[34] Duncan L Smith. 2017. Secure pseudonymisation for privacy-preserving probabilistic record linkage. *J. Inf. Secur. Appl.* 34 (2017), 271–279.
[35] Dinusha Vatsalan, Peter Christen, and Vassilios Verykios. 2013. A taxonomy of privacy-preserving record linkage techniques. *Elsevier IS* 38, 6 (2013), 946–969.
[36] Anushka Vidanage, Thilina Ranbaduge, Peter Christen, and Rainer Schnell. 2019. Efficient Pattern Mining Based Cryptanalysis for Privacy-Preserving Record Linkage. In *IEEE ICDE.* IEEE, Macau, 1698–1701.