Efficient Pattern Mining based Cryptanalysis for Privacy-Preserving Record Linkage

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

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

***Abstract*—Privacy-preserving record linkage (PPRL) is the process of identifying records that correspond to the same entities across several databases without revealing any sensitive information about these entities. One popular PPRL technique is Bloom filter (BF) encoding, with first applications of BF based PPRL now being employed in real-world linkage applications. Here we present a new type of cryptanalysis attack method that can re-identify sensitive attributes values that are encoded in BFs.**

Our method applies maximal frequent itemset mining on a BF database to first identify sets of frequently co-occurring bit positions that correspond to encoded frequent q-grams (character substrings extracted from plain-text values). Using a probabilistic language model, we then identify additional q-grams by applying pattern mining on subsets of BFs that encode a previously identified frequent q-gram. Experiments on real-world databases show that our attack can successfully re-identify sensitive values even when each BF in an encoded database is unique.

***Index Terms*—Bloom filter, re-identification, maximal frequent itemset mining, data linkage, probabilistic language model.**

I. Introduction

Data analytics projects in domains ranging from national security and healthcare to social science research often require records about individuals to be linked across different databases [1]. Because unique entity identifiers are rarely available in all databases to be linked, the linking of records generally relies upon identifying personal details (such as names, addresses and dates of birth) [1]. Known as quasi-identifiers (QIDs) [2], values in such attributes are in general sufficiently well correlated with entities to allow accurate linkage. Using such personal information however often leads to privacy and confidentiality concerns [2], [3].

Research into *privacy-preserving record linkage* (PPRL) aims to link records in different databases that refer to the same entities across while ensuring the privacy of these entities [2]. The idea behind PPRL is to conduct the linkage based on encode QID values. At the end of the PPRL process the participating parties only learn which of their records are matched (are highly similar), but they cannot learn any other information about the other database(s) [2].

One widely used privacy technique for PPRL is Bloom filter (BF) encoding [4] because it allows accurate and efficient PPRL of large databases, as we describe in Section **??**. However, recent research has shown that basic BFs are vulnerable to attacks that can successfully re-identify some values encoded in a BF database [5]–[10]. These attacks exploit the frequency counts and bit patterns in a set of BFs, and how these correlate to frequent plain-text values to identify BF bit positions that can encode certain q-grams. This then enables the re-identification of values from a plain-text database.

In this paper we present a novel attack method on BF encoding for PPRL that employs a maximal frequent itemsets mining approach. Our attack does not require any knowledge of the parameters used in the BF encoding process. We illustrate the basic idea of our attack in Figure 1. This attack is a substantial improved version of our previous attack method in [5]. We extend our attack by using a probabilistic language model to expand the set of identified q-grams by applying pattern mining on subsets of BFs that are known to encode a certain previously identified frequent q-gram. Based on the identified q-grams we then re-identify plain-text values encoded in individual BFs. An evaluation on two real data sets shows that our attack method can identify bit positions of q-grams with high precision, and it can re-identify plain-text values even when each BF in an encoded database is unique. In such a scenario no previous attack would have been successful.

II. Related Work

The first attack method on BFs for PPRL, proposed by Kuzu et al. [8], employed a constraint satisfaction problem (CSP) solver which assigns values to variables such that a set of constraints is satisfied. The attack was evaluated on a real patient database where it was successful in re-identifying four out of 20 frequent names correctly [9]. Niedermeyer et al. [10] more recently proposed an attack based on the counts of q-grams extracted from frequent German surnames which they were able to manually re-identify the most frequent ones. Kroll and Steinmetzer [7] extended this attack into a cryptanalysis of several attributes, which was able to correctly re-identify 44% of plain-text encoded in BFs. However, both these attack methods cannot be used with all hashing approaches on BFs.

Christen et al. [6] recently proposed a novel attack method that aligns frequent BFs with frequent plain-text values. The attack was able to correctly re-identify the most frequent plain-text values in a large database efficiently. This attack however requires both frequent BFs and frequent plain-text values.

This work was funded by the Australian Research Council under DP130101801 and DP160101934. P. Christen likes to acknowledge the support of ScaDS Dresden/Leipzig (BMBF grant 01IS14014B) and the University of Duisburg-Essen, where parts of this work were conducted.

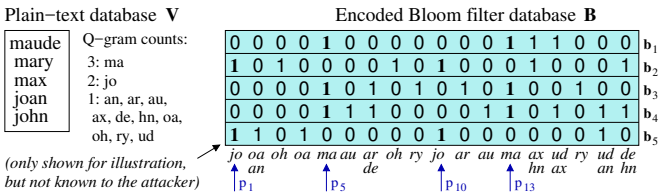

Fig. 1. An example illustrating the basic idea of our proposed attack method. We first identify that bit positions p_5 and p_{13} have co-occurring 1-bits in the *same three BFs* ($\mathbf{b}_1$, $\mathbf{b}_3$ and $\mathbf{b}_4$) and therefore must encode 'ma' which is the only q-gram that occurs in *three plain-text values*. Next, we find that p_1 and p_{10} must encode 'jo' because they have co-occurring 1-bits in the *same two BFs* ($\mathbf{b}_2$ and $\mathbf{b}_5$) and 'jo' is the only q-gram that occurs in *two plain-text values*. We now know that BFs $\mathbf{b}_2$ and $\mathbf{b}_4$ can only encode 'john' and 'joan', while $\mathbf{b}_1$, $\mathbf{b}_3$ and $\mathbf{b}_4$ can encode either 'maude', 'mary' or 'max'.

Notation of inputs and outputs as used by Algorithms 1 to 4	
$\mathbf{B}$:	List of BFs from the sensitive encoded database, one BF per record
$\mathbf{V}$:	List of plain-text values from a public database, one string per record
l_q:	Length of substrings to extract from plain-text values
d:	Min. percentage difference between two most frequent q-grams
m:	Min. partition size (a subset of BFs in $\mathbf{B}$) as a number of BFs
$\mathbf{F}$:	List of identified frequent q-grams with their BF bit positions
$\mathbf{A}^+$:	Lists of must have q-gram sets assigned to BFs in $\mathbf{B}$
$\mathbf{A}^-$:	Lists of cannot have q-gram sets assigned to BFs in $\mathbf{B}$
$\mathbf{Q}$:	List of q-gram sets, one per plain-text value in $\mathbf{V}$
$\mathbf{B}_c$:	Column-wise storage of BF database $\mathbf{B}$
k_{est}:	Estimated number of hash functions used to encode q-grams
$\mathbf{L}$:	Language model as one graph $\mathbf{G}_f$ per frequent q-gram $q_f \in \mathbf{F}$
$\mathbf{F}_E$:	Expanded list of identified q-grams and their BF bit positions
$\mathbf{A}_E^+$:	Expanded lists of must have q-gram sets assigned to BFs in $\mathbf{B}$
$\mathbf{A}_E^-$:	Expanded lists of cannot have q-gram sets assigned to BFs in $\mathbf{B}$
n_m:	Minimum number of required identified q-grams in a BF from $\mathbf{B}$
$\mathbf{R}$:	List of re-identified plain-text values from $\mathbf{V}$ for BFs from $\mathbf{B}$

III. Background

We now briefly describe BF encoding and the frequent pattern mining algorithm we used in our approach.

Bloom Filter Encoding: A BF $\mathbf{b}$ is a bit array of length $l_b = |\mathbf{b}|$ with all bits initialized to 0 [11]. k independent hash functions, $\mathbf{H} = \{h_1, \ldots, h_k\}$, are used to map each element s in a set $\mathbf{s}$ into $\mathbf{b}$ by setting the bit positions $\mathbf{b}[h_j(s)] = 1$, with $1 \leq j \leq k$, $1 \leq h_j(s) \leq l_b$, and $\forall s \in \mathbf{s}$. In PPRL, where commonly name and address values are used, the set $\mathbf{s}$ can be generated from character q-grams [1] extracted from strings. A set based similarity function, such as the Dice coefficient [1], can be used to calculate the similarity between two BFs [4]. Hashing of q-grams often leads to collisions [12], which can improve privacy but results in false positives [4].

Pattern Mining Techniques: The way BFs are generated leads to some specific aspects of how pattern mining can be applied in our attack: (1) We assume each BF in an encoded BF database is unique and therefore each itemset is unique, and (2) we are only interested in the maximal (longest) frequent itemsets which have an expected length of k, the number of hash functions used to encode q-grams into BFs.

Various algorithms that have been proposed over the years to efficiently mine frequent patterns from large databases however not all pattern mining techniques are suitable for our attack. We investigated several algorithms with regard to the requirements of our attack, and based on our set-up experiments we found Max-Miner [13] was one of the best performing algorithms in terms of efficiency and we therefore use it in our attack.

IV. A Pattern Mining based Attack

We first summarize our attack method, as illustrated in Figure 1 and detailed in Algorithms 1 to 4. In Table **??** we summarise the notation of inputs and outputs as used and/or generated by these algorithms. We use bold letters for BFs, sets, and lists (with upper-case bold letters for sets and lists of BFs, 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.

As with existing attacks on BFs for PPRL, we assume an adversary has access to an encoded BF database, $\mathbf{B}$, where it is unknown which plain-text value(s) are encoded in a BF; and a plain-text database, $\mathbf{V}$, that contains values from one or several attributes. An attacker can guess which attributes are encoded in $\mathbf{B}$ and what value was used for q based on the distribution of 1-bits of BFs in $\mathbf{B}$ [14].

In the first step of our attack we identify sets of frequently co-occurring bit positions in $\mathbf{B}$ that can encode q-grams that are frequent in $\mathbf{V}$, as well as the number of hash functions (k) that was used to encode values into BFs. In the second step we then identify additional q-grams that have a high conditional probability of co-occurring with one of the frequent q-grams identified in the first step. In the third step we use all identified q-grams to find plain-text values $v \in \mathbf{V}$ that can be encoded in a BF $\mathbf{b} \in \mathbf{B}$ based on the bit positions of where the q-grams identified in the first two steps must or cannot occur. Our attack exploits the way BFs are constructed as follows:

Assuming a q-gram q occurs in $n_q < n$ records in a plain-text database $\mathbf{V}$ that contains $n = |\mathbf{V}|$ records, and $k \geq 1$ independent hash functions are used to encode q-grams from $\mathbf{V}$ into the encoded database $\mathbf{B}$ of n BFs, i.e. $|\mathbf{V}| = |\mathbf{B}|$. Then, (1) each BF bit position that can encode q must contain a 1-bit in at least n_q BFs in $\mathbf{B}$, and (2) if $k > 1$ then up to k bit positions must contain a 1-bit in the same subset of BFs $\mathbf{B}_q \subseteq \mathbf{B}$, with $n_q = |\mathbf{B}_q|$, that encode q.

Our pattern mining based attack method relies on a set of bit positions that have frequently co-occurring 1-bit patterns. However, because there can be possible collisions of the k hash functions when a q-gram is hashed into BFs [11], [12], potentially less than k bit positions will encode a certain q-gram. Therefore less than k bit positions might be co-occurring frequently for a certain frequent q-gram.

Our attack should therefore be able to clearly identify with high precision up to k frequently co-occurring bit positions that encode the same q-gram, as we experimentally validate in Section V.

Identifying Co-occurring Bit Positions: The first step of our attack (proposed in [5]), as detailed in Algorithm 1, identifies the sets of frequently co-occurring bit positions in

Algo. 1: *Identify frequent q-grams and co-occurring BF bit positions*

1: $\mathbf{B}_c = ConvColWise(\mathbf{B})$
2: $l_b = |\mathbf{B}_c|$
3: $\mathbf{Q} = GenQGramSets(\mathbf{V}, l_q)$
4: $\mathbf{F} = [\,]$, $\mathbf{A}^+ = [\,]$, $\mathbf{A}^- = [\,]$, $\mathbf{m} = \{\}$, $\mathbf{n} = \{\}$, $\mathbf{k} = [\,]$
5: $\mathbf{r} = \{b : 1 \leq b \leq |\mathbf{B}|\}$, $\mathbf{c} = \{p : 1 \leq p \leq l_b\}$
6: $Q = [(|\mathbf{B}|, \mathbf{c}, \mathbf{r}, \mathbf{m}, \mathbf{n})]$
7: **while** $|Q| > 0$ **do**:
8: $(ps_i, \mathbf{c}_i, \mathbf{r}_i, \mathbf{m}_i, \mathbf{n}_i) = Q.pop()$
9: $q_1, f_1, q_2, f_2 = GetTwoMostFreqQGrams(\mathbf{Q}, \mathbf{m}_i, \mathbf{n}_i)$
10: $d_i = 2(f_1 - f_2)/(f_1 + f_2) \cdot 100$
11: **if** $d_i \geq d$ **then**:
12: $s_i = |\mathbf{B}| \cdot (f_1 + f_2)/(2|\mathbf{V}|)$
13: $\mathbf{f}_i = GetMaxFreqCoOccurBitPos(\mathbf{B}_c, s_i, \mathbf{c}_i, \mathbf{r}_i)$
14: **if** $|\mathbf{f}_i| > 0$ **then**:
15: $\mathbf{F}.append((q_1, \mathbf{f}_i))$
16: $\mathbf{k}.append(|\mathbf{f}_i|)$
17: $\mathbf{c}_i = \mathbf{c}_i \setminus \mathbf{f}_i$
18: $\mathbf{r}_i^+, \mathbf{r}_i^- = UpdateRowFilter(\mathbf{f}_i, \mathbf{r}_i)$
19: $\forall b \in \mathbf{r}^+ : \mathbf{A}^+[b] = \mathbf{A}^+[b] \cup \{q_1\}$
20: $\forall b \in \mathbf{r}^- : \mathbf{A}^-[b] = \mathbf{A}^-[b] \cup \{q_1\}$
21: **if** $|\mathbf{r}_i^+| \geq m$ **then**:
22: $Q.append([(\mathbf{r}_i^+|, \mathbf{c}_i, \mathbf{r}_i^+, \mathbf{m}_i \cup \{q_1\}, \mathbf{n}_i))$
23: **if** $|\mathbf{r}_i^-| \geq m$ **then**:
24: $Q.append((|\mathbf{r}_i^-|, \mathbf{c}_i, \mathbf{r}_i^-, \mathbf{m}_i, \mathbf{n}_i \cup \{q_1\}))$
25: $Q.sort\_largest\_first()$
26: $k_{est} = mode(\mathbf{k})$
27: **return** $\mathbf{F}$, $\mathbf{A}^+$, $\mathbf{A}^-$, $\mathbf{Q}$, $\mathbf{B}_c$, k_{est}

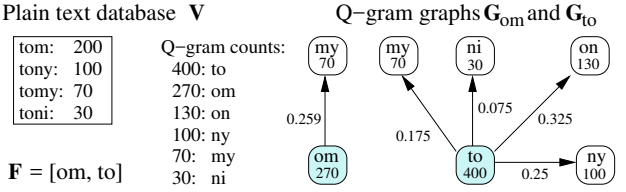

Fig. 2. An example q-gram graph generation from four plain-text values containing six q-grams, where 'om' and 'to' are frequent q-grams identified in step one of our attack. Edges between q-grams show conditional probabilities of the destination q-gram assuming the source q-gram occurs in a value in $\mathbf{V}$. For each q-gram in $\mathbf{F}$, in step two of our attack we select the not-frequent q-grams with the highest conditional probabilities, such as 'my' for 'om' and 'on' for 'to'. As described in Section **??**, we then apply pattern mining on subsets of BFs in $\mathbf{B}$ where we know they contain a frequent q-gram to identify the bit positions that can encode a selected not-frequent q-gram.

the BF database $\mathbf{B}$ that correspond to the frequent q-grams that occur in the plain-text database $\mathbf{V}$.

Language Model Based Q-gram Set Expansion:

In this second step of our attack we expand the list $\mathbf{F}$ by identifying not frequent q-grams and their bit positions that have a high conditional probability of co-occurring with one of the frequent q-grams in $\mathbf{F}$ in values in $\mathbf{V}$. As illustrated in Figure 2, conceptually we build a star graph $\mathbf{G}_f$ for each frequent q-gram $q_f \in \mathbf{F}$, where the center node is the frequent q-gram q_f as identified in the first step of our attack, and all leaf nodes are the other q-grams q_i that co-occur with q_f in any value $v_j \in \mathbf{V}$. The node attributes are the number of values in $\mathbf{V}$ that contain a certain q-gram q_i, calculated as $f_i = |\{q_i \in \mathbf{q}_j : \mathbf{q}_j \in \mathbf{Q}\}|$, where $\mathbf{q}_j$ is the q-gram set for value v_j as generated in line 3 in Algorithm 1.

A directed edge, $q_f \rightarrow q_i$ in $\mathbf{G}_f$ represents the conditional probability $p_{f,i}$ of q_i occurring in a value $v_j \in \mathbf{V}$ given the frequent q-gram q_f occurs in v_j, calculated as $p_{f,i} = f_i/f_f$.

Algo. 2: *Generate language model for q-gram set expansion*

1: $\mathbf{L} = [\,]$
2: **for** $q_f \in \mathbf{F}$ **do**:
3: $\mathbf{G}_f = (V = \{q_f\}, E = \emptyset)$
4: **for** $\mathbf{q}_j \in \mathbf{Q}$ **do:**
5: **if** $q_f \in \mathbf{q}_j$ **then:**
6: $\mathbf{G}_f[q_f] = \mathbf{G}_f[q_f] + 1$
7: **for** $q_i \in \mathbf{q}_j \setminus \{q_f\}$ **do**:
8: **if** $q_i \notin \mathbf{G}.V$ **then**:
9: $\mathbf{G}_f.V = \mathbf{G}_f.V \cup \{q_i\}$; $\mathbf{G}_f[q_i] = 0$
10: $\mathbf{G}_f[q_i] = \mathbf{G}_f[q_i] + 1$
11: **for** $q_i \in \mathbf{G}_f.V \setminus \{q_f\}$ **do**:
12: $\mathbf{G}_f.E = \mathbf{G}_f.E \cup \{(q_f, q_i)\}$
13: $\mathbf{G}_f[q_f, q_i] = \mathbf{G}_f[q_i]$ / $\mathbf{G}_f[q_f]$
14: $\mathbf{L}.append([q_f, \mathbf{G}_f])$
15: **return** $\mathbf{L}$

For example, in $\mathbf{G}_{to}$ in Figure 2, the conditional probability that 'on' occurs given 'to' occurs is $130/400 = 0.325$.

Algorithm 2 details the steps for generating the language model $\mathbf{L}$.

Once the language model $\mathbf{L}$ is generated, we can use it to expand the set of identified q-grams. The idea of our expansion step is that for each frequent q-gram, q_f, and its corresponding graph, $\mathbf{G}_f$, we select the other q-grams $q_i \in \mathbf{G}_f.V$ that have the highest conditional probabilities of co-occurring with q_f. We then try to identify the bit positions where a q-gram q_i could have been hashed to based on the subset of BFs in $\mathbf{B}$ where we know (from Algorithm 1) that q_f was hashed into. This subset of BFs that can encode q_f, named $\mathbf{B}_f$, are those BFs that have a 1-bit in all bit positions identified for q_f, as available in the list $\mathbf{F}$ generated in Algorithm 1. We then apply maximal frequent pattern mining on this subset $\mathbf{B}_f$ of BFs with the aim to identify the bit positions of where q_i can be hashed into. The expected frequency (support count) of 1-bits of co-occurring bit positions for q_i needs to correspond to the conditional probability $p_{f,i}$ of q_i occurring in a value $v_j \in \mathbf{V}$ where we know q_f occurs in, calculated as $s_i = p_{f,i} \cdot |\mathbf{B}_f|$. We describe this approach in detail in Algorithm 3.

In Algorithm 3 we now describe the detailed steps involved in the q-gram set expansion step based on the language model $\mathbf{L}$ as generated in Algorithm 2.

Plain-text Value Re-identification:

As detailed in Algorithm 4, the final step of our attack aims to re-identify values from $\mathbf{V}$ that could have been encoded into BFs in $\mathbf{B}$ using the lists of identified q-grams and their bit positions, $\mathbf{F}_E$, and the lists $\mathbf{A}_E^+$ and $\mathbf{A}_E^-$ of must have and cannot have q-gram sets for BFs, as generated in Algorithm 3. We only consider plain-text values and BFs that contain at least n_m must have q-grams, because considering a single or only a few identified q-grams would result in too many plain-text values that could match a BF (for example, nearly 4,000 surnames in the experimental NCVR data set of 224,061 records contain the q-gram 'sm').

V. Experiments and Results

We evaluated our attack using the real North Carolina Voter Registration (NCVR) database (http://dl.ncsbe.gov/data/), where we used one snapshot from April 2014 as $\mathbf{B}$ and

Algo. 3: ***Identify further q-grams and co-occurring BF bit positions***

1: $\mathbf{F}_E = \mathbf{F}$, $\mathbf{A}_E^+ = \mathbf{A}^+$, $\mathbf{A}_E^- = \mathbf{A}^-$, $Q = [\,]$
2: $\mathbf{c} = \{p : 1 \leq p \leq |\mathbf{B}_c|\} \setminus \{\cup p : \forall p \in \mathbf{f}_i : \forall \mathbf{f}_i \in \mathbf{F}\}$
3: **for** $(q_f, \mathbf{G}_f) \in \mathbf{L}$ **do**:
4: $\quad \mathbf{r}_f = \{b : 1 \leq b \leq |\mathbf{B}| : \forall p \in \mathbf{f}_f : \mathbf{B}[b][p] = 1 : \mathbf{f}_f \in \mathbf{F}\}$
5: $\quad f_f = \mathbf{G}_f[q_f]$
6: $\quad Q.append((f_f, q_f, \mathbf{G}_f, \mathbf{r}_f))$
7: $Q.sort\_most\_freq\_first()$
8: **while** $|Q| > 0$ **do**:
9: $\quad (f_f, q_f, \mathbf{G}_f, \mathbf{r}_f) = Q.pop()$
10: $\quad \mathbf{G}_f.V.sort\_most\_freq\_first()$
11: $\quad$ **for** $[(q_i, p_{f,i}), (q_{i+1}, p_{f,i+1})] \in \mathbf{G}_f.V \setminus \{q_f\}$ **do**:
12: $\quad\quad d_i = 2(p_{f,i} - p_{f,i+1})/(p_{f,i} + p_{f,i+1}) \cdot 100$
13: $\quad\quad$ **if** $d_i \geq d$ **then**:
14: $\quad\quad\quad s_i = |\mathbf{r}_f| \cdot (p_{f,i} + p_{f,i+1})/2$
15: $\quad\quad\quad \mathbf{f}_i = GetMaxFreqCoOccurBitPos(\mathbf{B}_c, s_i, \mathbf{c}, \mathbf{r}_f)$
16: $\quad\quad\quad$ **if** $0 < |\mathbf{f}_i| \leq k_{est}$ **then**:
17: $\quad\quad\quad\quad \mathbf{F}_E.append((q_i, \mathbf{f}_i))$
18: $\quad\quad\quad\quad \mathbf{c} = \mathbf{c} \setminus \mathbf{f}_i$
19: $\quad\quad\quad\quad \mathbf{r}_i^+, \mathbf{r}_i^- = UpdateRowFilter(\mathbf{f}_i, \mathbf{r}_f)$
20: $\quad\quad\quad\quad \forall b \in \mathbf{r}_i^+ : \mathbf{A}_E^+[b] = \mathbf{A}_E^+[b] \cup \{q_i\}$
21: $\quad\quad\quad\quad \forall b \in \mathbf{r}_i^- : \mathbf{A}_E^-[b] = \mathbf{A}_E^-[b] \cup \{q_i\}$
22: $\quad\quad$ **else**: // $d_j < d$
23: $\quad\quad\quad$ **exit for loop**
24: **return** $\mathbf{F}_E$, $\mathbf{A}_E^+$, $\mathbf{A}_E^-$

Algo. 4: ***Re-identify plain-text values in Bloom filters***

1: $\mathbf{I}_B = [\,]$, $\mathbf{R} = [\,]$
2: $\mathbf{I}_Q = GenValQGramInvIndex(\mathbf{V}, \mathbf{Q})$
3: **for** $b \in \mathbf{A}_E^+$ **do**:
4: $\quad \mathbf{q}^+ = \mathbf{A}_E^+[b]$, $\mathbf{q}^- = \mathbf{A}_E^-[b]$
5: $\quad$ **for** $(q_i, \mathbf{f}_i) \in \mathbf{F}_E$ **do**:
6: $\quad\quad$ **if** $\exists p \in \mathbf{f}_i : \mathbf{B}[b][p] = 0$ **then**:
7: $\quad\quad\quad \mathbf{q}^- = \mathbf{q}^- \cup \{q_i\}$
8: $\quad \mathbf{I}_B[(\mathbf{q}^+, \mathbf{q}^-)].add(b)$
9: **for** $(\mathbf{q}^+, \mathbf{q}^-) \in \mathbf{I}_B$ **do**:
10: $\quad$ **if** $|\mathbf{q}^+| \geq n_m \vee |\mathbf{I}_B[(\mathbf{q}^+, \mathbf{q}^-)]| = 1$ **then**:
11: $\quad\quad \mathbf{v} = \{\forall q_i \in \mathbf{q}^+ : \cap \mathbf{I}_Q[q_i]\} \setminus \{\forall q_i \in \mathbf{q}^- : \cup \mathbf{I}_Q[q_i]\}$
12: $\quad\quad$ **for** $b \in \mathbf{I}_B[(\mathbf{q}^+, \mathbf{q}^-)]$ **do**:
13: $\quad\quad\quad$ **for** $v \in \mathbf{v}$ **do**:
14: $\quad\quad\quad\quad \mathbf{R}[b] = \mathbf{R}[b] \cup \{v\}$
15: **return** $\mathbf{R}$

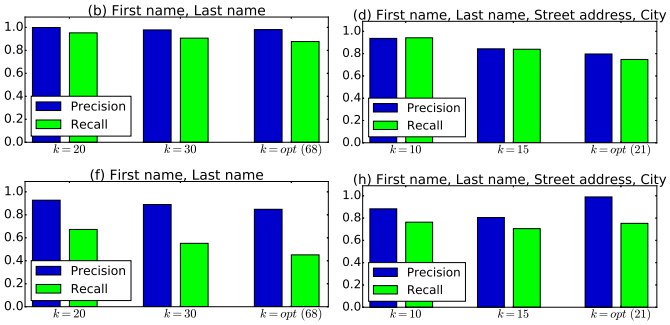

Fig. 3. Precision and recall results for the NCVR database for the identified frequent q-grams in $\mathbf{F}$ from Algorithm 1 (top row, plots (a) to (d)) and in $\mathbf{F}_E$ from Algorithm 3 (bottom row, plots (e) to (h)) with different numbers of hash functions k, as described in Section V.

a second from June 2014 as $\mathbf{V}$. We extracted pairs of records that correspond to the same voter but had name and/or address changes over time, resulting in two files of 222,251 and 224,06 records, respectively. We encoded different combinations of the attributes *first name*, *last name*, *street address* and *city* into BFs. For combinations of three and four attributes, all values in $\mathbf{V}$ and all bit patterns in $\mathbf{B}$ were unique.

We used the cryptographic long term key BF encoding [14] with double and random hashing [14], with *q=2*, l_b*=1,000* and different values for k as shown in Figure 3. We calculated the optimal number, *opt*, of hash functions such that the average number of 1-bits in a BF is 50% to minimize the false positive rate in BFs [15]. We set the minimum percentage difference as *d=[1.0, 5.0]*, the minimum partition size *m=[2,000, 10,000]*, and the minimum q-gram tuple size n_m*=3* as these provided good results in set-up experiments.

We present the quality of the identified frequent q-grams in $\mathbf{F}$ and $\mathbf{F}_E$ from Algorithms 1 and 3 as the precision and recall of how many bit positions were correctly identified for a q-gram in $\mathbf{F}$ or $\mathbf{F}_E$ averaged over all q-grams. For Algorithm 4, we evaluated the quality of re-identified values in $\mathbf{R}$ as the percentages of (1) *exact* matches of a plain-text value with the true value encoded in a BF, (2) *partial* matches where not all words in a value matched (for example, *first* and *last name* were the same but *city* was different), and (3) *wrong* matches where no word matched. We only considered BFs that had 10 or less plain-text values assigned to them in $\mathbf{R}$, and we present averaged results for *1-to-1*, with $|\mathbf{R}[b]| = 1$, and *1-to-m* (many), with $1 < |\mathbf{R}[b]| \leq 10$, re-identifications.

We compared our attack method with the attack by Christen et al. [6] (the only other attack that does not require knowledge of the BF encoding parameters) which aligns frequent BFs and plain-text values to re-identify the most frequent plain-text values, as well as the initial version of our current attack method [5]. We implemented all methods using Python 2.7 and ran experiments on a server with 64-bit 2.4 GHz CPUs, 128 GBytes of memory and running Ubuntu 16.04. The programs and data sets are available from: https://dmm.anu.edu.au/pprlattack.

Discussion: In Figure 3 we show precision and recall results for the q-grams identified in $\mathbf{F}$ and $\mathbf{F}_E$. As can be seen, both precision and recall of the identified frequent q-grams in $\mathbf{F}$ are high, above 0.88, for different values of k and different encoded attribute combinations. This validates that our attack can successfully identify bit positions of frequent q-grams with high accuracy even when each BF in an encoded database is unique. The precision of q-grams in $\mathbf{F}_E$ stays above 0.8 for different values of k and different attribute combinations. However, as we discussed in Section IV, the recall of bit positions of the additional q-grams identified in the second step of our attack drops because we mark a bit position as assigned once a q-gram hashed to it has been identified.

The lower recall in Figure 3 (e) and (f) for one and two encoded attributes (compared to three and four) is because we are able to identify more frequent q-grams already in the first step of our attack, and therefore less bit positions are left in the second step to be assigned to additional q-grams (in the column filter set, $\mathbf{c}$, in line 2 of Algorithm 3).

Table I shows the re-identification results from the third step of our attack (Algorithm 4). Compared to the frequency based attack by Christen et al. [6], our attack can exactly or partially re-identify a considerable percentage of encoded plain-text

TABLE I

RE-IDENTIFICATION PERCENTAGES ON NCVR OF EXACT (E), PARTIAL (P) AND WRONG (W) MATCHES OF PLAIN-TEXT VALUES FROM ALGORITHM 4, AVERAGED OVER PARAMETERS DESCRIBED IN SECTION V, AND COMPARED TO OUR EARLIER ATTACK METHODS.

		Two attributes E / P / W	Three attributes E / P / W	Four attributes E / P / W
Christen et al. (2018) [6]	1-to-1	0.0 / 0.0 / 100.0	0.0 / 0.0 / 100.0	0.0 / 0.0 / 100.0
	1-to-m	13.5 / 0.0 / 86.5.0	0.0 / 0.0 / 100.0	0.0 / 0.0 / 100.0
Christen et al. (2018) [5]	1-to-1	20.7 / 30.9 / 48.4	0.2 / 61.0 / 38.8	0.5 / 73.2 / 26.3
	1-to-m	27.5 / 46.5 / 26.0	0.4 / 83.5 / 16.1	0.5 / 87.9 / 11.6
Our new method	1-to-1	52.6 / 16.3 / 31.0	0.6 / 54.3 / 45.1	0.3 / 48.2 / 51.5
	1-to-m	55.8 / 22.3 / 21.9	0.6 / 76.0 / 23.4	0.3 / 76.4 / 23.3

TABLE II

COMPARISON OF THE NUMBER OF EXACT, PARTIAL, AND WRONG RE-IDENTIFICATIONS BETWEEN THE INITIAL VERSION OF OUR ATTACK [5] AND THE NEW METHOD CONDUCTED ON THE NCVR DATABASE.

Num. attr.	1-to-1 matches Exact	1-to-1 matches Partial	1-to-1 matches Wrong	1-to-m matches Exact	1-to-m matches Partial	1-to-m matches Wrong
Two	2,010 / 10,956	577 / 2,546	823 / 3,918	5,368 / 18,037	2,424 / 7,825	1,831 / 9,250
Three	2 / 111	187 / 6,825	128 / 3,915	15 / 163	2,052 / 17,644	420 / 7,898
Four	75 / 88	8,837 / 8,995	2,976 / 6,680	147 / 189	24,911 / 36,796	3,112 / 13,980

values. While some re-identification percentages are lower compared to the initial attack version [5] (for three and four attributes), as the numbers of re-identified plain-text values in Table II show, our new attack method is able to re-identify a substantially higher number of plain-text values. We expected the numbers of correct exact re-identifications for three and four attributes to be very low because less than 1,800 of over 222,000 combined values between the two NCVR snapshots are exact matches.

These results show that basic BFs, even when each BF in an encoded database is unique, can successfully be attacked using our pattern mining method. Our work highlights the need to improve BF encoding, and to develop new encoding methods for PPRL that do not exhibit the weaknesses of basic BFs.

VI. CONCLUSIONS AND FUTURE WORK

We have presented a pattern mining based attack method on BF encoding as used for PPRL which can successfully re-identify encoded q-grams and plain-text values even when all BFs in a database are unique. Given that BF based PPRL is now employed in real-world applications [3], it is important to study the limits of BF encoding. Our attack helps to make BF based PPRL more secure because it allows data custodians to ensure their encoded databases are not vulnerable to such attacks. As future work we plan to improve the second step of our attack by analyzing the differences in bit patterns between BFs to identify additional q-grams with high accuracy to allow an improved re-identification of plain-text values in the third step. We will also explore how BF *hardening* techniques [14], such as balancing or XOR-folding, will influence the feasibility of our attack method.

REFERENCES

[1] P. Christen, *Data matching*. Springer, 2012.

[2] D. Vatsalan, Z. Sehili, P. Christen, and E. Rahm, "Privacy-preserving record linkage for Big Data: Current approaches and research challenges," in *Handbook of Big Data Technologies*. Springer, 2017.

[3] J. Boyd, S. Randall, and A. Ferrante, "Application of privacy-preserving techniques in operational record linkage centres," in *Med Data Privacy Handbook*, 2015.

[4] R. Schnell, T. Bachteler, and J. Reiher, "Privacy-preserving record linkage using Bloom filters," *BMC Med Inform Decis Mak*, 2009.

[5] P. Christen, A. Vidanage, T. Ranbaduge, and R. Schnell, "Pattern-mining based cryptanalysis of Bloom filters for privacy-preserving record linkage," in *PAKDD*, Melbourne, 2018.

[6] P. Christen, T. Ranbaduge, D. Vatsalan, and R. Schnell, "Precise and fast cryptanalysis for Bloom filter based privacy-preserving record linkage," *IEEE TKDE (accepted 29 Sep)*, 2018.

[7] M. Kroll and S. Steinmetzer, "Automated cryptanalysis of Bloom filter encryptions of databases with several personal identifiers," in *BIOSTEC*, Lisbon, 2015.

[8] M. Kuzu, M. Kantarcioglu, E. Durham, and B. Malin, "A constraint satisfaction cryptanalysis of Bloom filters in private record linkage," in *PET*, Waterloo, 2011.

[9] M. Kuzu, M. Kantarcioglu, E. Durham *et al.*, "A practical approach to achieve private medical record linkage in light of public resources," *JAMIA*, vol. 20, no. 2, pp. 285–292, 2013.

[10] F. Niedermeyer, S. Steinmetzer *et al.*, "Cryptanalysis of basic Bloom filters used for privacy preserving record linkage," *JPC*, 2014.

[11] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge Uni. Press, 2005.

[13] R. Bayardo, "Efficiently mining long patterns from databases," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 85–93, 1998.

[14] R. Schnell and C. Borgs, "Randomized response and balanced Bloom filters for privacy preserving record linkage," in *DINA*, Barcelona, 2016.

[15] D. Vatsalan and P. Christen, "Privacy-preserving matching of similar patients," *JBI*, vol. 59, pp. 285–298, 2016.