Inference and Record-Injection Attacks on Searchable Encrypted Relational Databases

Mohamed Ahmed Abdelraheem, Tobias Andersson and Christian Gehrmann

{mohamed.abdelraheem, tobias.andersson, chrisg}@sics.se

SICS Swedish ICT AB

Abstract. We point out the risks of providing security to relational databases via searchable encryption schemes by mounting a novel inference attack exploiting the structure of relational databases together with the leakage of searchable encryption schemes. We discuss some techniques to reduce the effectiveness of inference attacks against searchable encryption schemes. Moreover, we show that record-injection attacks mounted on relational databases have worse consequences than their file-injection counterparts on unstructured databases which have been recently proposed at USENIX 2016.

1 Introduction

One of the practical solutions for searching on encrypted data is provided by searchable symmetric encryption (SSE) schemes. The very first such scheme was proposed by Song et al. in [25]. Later, Curtmoula et al.'s [14] introduced two security notions for SSE schemes, namely, the non-adaptive semantic security definition and the adaptive semantic security definition. Subsequent SSE schemes [10,9,8] are all based on Curtmoula et al.'s security model. The price of the efficiency offered by searchable symmetric encryption schemes comes at the cost of leaking the frequency of each keyword after it has been queried. This makes them vulnerable to frequency attacks.

Beside SSE schemes, there are several practical solutions proposed to execute SQL queries on an encrypted database. Recently, Popa et al. proposed CryptDB as a solution to protect confidentiality for applications using SQL databases [24]. CryptDB uses column-level encryption to encrypt the database tables. CryptDB uses a trusted proxy server to communicate between the client application and the cloud server. The proxy server translates the SQL queries in plaintext format to an encrypted format to enable the cloud server in executing the SQL on the encrypted tables. To enable equality searches, CryptDB uses deterministic encryption [4]. Order preserving encryption [2,5] is used to enable range and comparison queries on encrypted data. This is the weakest encryption scheme used in CryptDB whose design concept is based on the trade-off between functionality and confidentiality. Recently, Naveed et al. [23] mounted frequency attacks that recovered the plaintext from CryptDB's columns protected by deterministic encryption and order preserving encryption schemes.

Another line of research in preserving database privacy is achieved by distributing and fragmenting the database table across two or more servers [1,11,13]. The fragmentation technique employed in all the previous schemes is vertical fragmentation where the tables' columns are partitioned across the servers. Privacy in [1] is provided under the assumption that the two cloud servers are unable to communicate directly with each other while the privacy in [11,13] is achieved without such assumption. The idea here is to use encryption as little as possible. In general, fragmentation is preferred over encryption to break the associations among the attributes. In [12], privacy is preserved by using fragmentation only and no encryption is employed. This comes at the cost of saving sensitive data in the clear at the data owner, i.e. the client. All these privacy constraints schemes are efficient but at the cost of having plaintext fields and only encrypting sensitive data which makes them vulnerable to be attacked by an adversary with background information about the database. Also adding or modifying a record reveals the relation among the fragments to passive adversary monitoring the fragments.

Comparing the above methods for searching on encrypted data in terms of security, one can see that SSE schemes offer better security than deterministic or order preserving encryption schemes since they do not leak the frequency of a keyword before querying it. They also provide better security than the data fragmentation method via privacy constraints [11] since they encrypt all the plaintext data and they can also securely manage a dynamic database [8].

However, still SSE schemes suffer from leaking the access pattern of a queried keyword which make them vulnerable to inference attacks as demonstrated by Islam et al. [17] and Cash et al. [7]. In this paper, we propose a novel inference attack targeting relational databases via SSE schemes and study the effect of these inference attacks as well as the recent file-injection attacks [27] on relational databases. We also propose a suitable inference control to safeguard relational databases secured via SSE schemes from being completely recovered by strong adversaries with background knowledge about the relational database.

Our Contribution. We show the severe consequences of inference attacks [17,7] as well as injection attacks [27] on a relational database secured via an SSE scheme. We propose a novel inference attack on SSE schemes that exploits the structure of relational databases beside the access pattern's leakage inherent in SSE schemes. Moreover, we show that record-injection attacks are a serious threat for SSE schemes that are not forward secure. Furthermore, we propose the use of privacy constraints [1,11,13] to distribute the encrypted index of an SSE scheme into several fragments or servers to reduce the effectiveness of frequency attacks exploiting the access pattern leakage [17,7] which is inherent in SSE schemes.

Related Work. Query recovery attacks exploiting the access pattern leakage of SSE schemes was proposed by Islam et al. [17] and later improved by Cash et al. [7]. By exploiting the structure of relational databases in order to attack them, our new attack requires lesser background information about the database under attack compared to full database knowledge required by Cash et al. and Islam et al. Moreover, we empirically emphasize the obvious observation that the query recovery attacks in [17,7] might lead to complete record-recovery attacks when the database under attack is a relational database. Furthermore, the padding countermeasure against query recovery attacks proposed in [17,7] does not fully prevent the generalized Count attack proposed in [7] (cf. Section 5). However, in this work, we propose the use of privacy constraints as an additional countermeasure that should be used together with padding to reduce the effectiveness of frequency attacks. Note that the privacy constraints as defined in [1,11,13] were mainly used to depart completely from the use of encryption or to use encryption as less as possible. However, in this paper we propose using them to strengthen the security of SSE schemes against frequency attacks.

Naveed el al. [23] proposed a number of attacks targeting relational database columns encrypted using deterministic encryption [4] or order preserving encryption algorithms [2,5] in CryptDB [24] where encrypted values belonging to the same column whose name is encrypted are collected together as one set. While seems similar, their column finder procedure is different than our attribute recovery attack in SSE schemes. Their column finder procedure takes as input the set of encrypted values belonging to the same unknown column whose name is encrypted. It recovers the encrypted name by matching the number of distinct encrypted values with each column's cardinality defined in the set of plaintext columns (i.e. a column name and it possible values are known) belonging to the auxiliary or background data. Their procedure relies mainly on the attributes' (i.e. columns) cardinalities. In contrast, our attack described in Algorithm 1 takes as input all the observed encrypted queries in an SSE scheme. Then using the access pattern leakage inherent in SSE schemes in addition to basic background data about the number of records and attributes' cardinalities, it divides the observed encrypted queries into different classes where each class contains a set of encrypted queries belonging to the same attribute. Thus, each class or set of encrypted queries found by our attack is actually the input used by Naveed et al's column finder procedure.

The recently proposed file-injection attack by Zhang et al. at USENIX 2016 [27] recovers only a set of keyword in encrypted document and could prevented by limiting the content of each injected file. However, in this paper we show that record-injection attacks while being similar to file-injection attacks, they have more severe consequences represented in an almost full record

recovery or reconstruction attack on a relational database. Moreover, record-injection attacks cannot be simply prevented by limiting the content of an injected record as done to prevent file-injection attacks [27] since that would hinder the addition of new complete records to the relational database. Therefore, our work shows that the batch update process suggested by some SSE schemes is the right way to prevent file-injection attacks.

The recent generic attacks [19] proposed at CCS 2016 target any secure database systems supporting range queries but leaking either access pattern such as order-preserving encryption schemes without any prior knowledge about the dataset under attack. Moreover, their generic attacks target any secure encrypted search method supporting range queries leaking communication volume such as fully homomorphic encryption or ORAM schemes. However, their attacks only target systems supporting range queries and they require the attacker to gather at least N^4 queries (N is the domain size) to mount the attack successfully.

Organization of the paper. Section 2 gives a brief overview about SSE schemes and the frequency attacks on them. In Section 3, we point out the security risk of using SSE schemes in relational databases by proposing a new frequency attack exploiting the properties of relational databases. In Section 4, we revisit the file-injection attacks in the context of relational databases. In Section 5, we propose the use of privacy constraints as an inference control and countermeasure to reduce the risk of frequency attacks.

2 Background

In this Section, we define SSE schemes and explain their leakage. We also explain the frequency attacks exploiting the leakage of SSE schemes.

2.1 SSE Schemes

Let DB denotes a database of n documents and m unique keywords. Let $\mathsf{ID} = (\mathsf{id}_i)_{i=1}^n$ denotes the set of all document identifiers in DB and let $\mathsf{W} = (w_j)_{j=1}^m$ denotes the set of all unique keywords in DB. Let $|\mathsf{W}|$ denote the number of keywords in the database. The database can be represented as an inverted index where each keyword is associated with a list of identifiers, i.e. $\mathsf{DB} = (w_i, \mathsf{DB}(w_i))_{i=1}^m$ where $\mathsf{DB}(w_i)$ denotes the set of documents' identifiers containing the keyword w_i. Let $|\mathsf{DB}(w_i)|$ denote size of $\mathsf{DB}(w_i)$.

Definition. An SSE scheme takes as inputs the plaintext index $\mathsf{DB} = (w_i, \mathsf{DB}(w_i))_{i=1}^m$ together with the client's secret keys and outputs an encrypted and frequency-hiding database index EINDEX where a keyword w is transformed into a token t using a deterministic encryption algorithm and its corresponding document identifiers are encrypted using a randomized algorithm. The SSE scheme also encrypts the original documents using a randomized encryption algorithm and stores in an encrypted database EDB indexed by the document identifiers. Both the encrypted index EINDEX and the encrypted database EDB are sent to the cloud server. To search for a keyword w, the client generates its token t and sends it to the server which retrieves the corresponding encrypted document identifiers from EINDEX and decrypts them and consequently sends the corresponding encrypted documents from EDB to the client.

Leakage Profile. An SSE scheme leaks the *access pattern*: the result of the query or the document IDs corresponding to the queried keyword w_i, $\{\mathsf{DB}(\mathsf{w_i}) : \mathsf{w_i} \in \mathsf{W}\}$, and also leaks the *search pattern*: the fact that whether two searches are the same or not.

Attack Model. All recent SSE schemes follow the adaptive security definition proposed by Curtmoula et al. [14] where security is achieved against an honest-but-curious server is a passive adversary following the protocol but curious to use the leakage profile to learn about the encrypted queries and the encrypted documents.

Attacks on SSE Schemes. Two passive attacks against SSE schemes exploiting the access pattern leakage have been proposed recently by Islam et al. [17] and later developed by Cash et al. [7]. These passive attacks are frequency attacks mounted by an honest-but-curious server who knows all or a significant number of the client's plaintext documents. Another class of attacks outlined by Cash et al. [7] are the chosen-document attack and the chosen query attacks. Both attacks are mounted by an active adversary who is somehow able to deceive the client into including her own chosen-document into the documents set and into choosing her favorite queries respectively. Recently Zhang et al. [27] presented a concrete description of a chosen-document attack (file-injection) where the attacker is able to recover all the queries without any prior knowledge about the client's dataset under attack.

2.2 Frequency Attacks

Frequency attacks are mounted on SSE schemes to recover the plaintext of encrypted keywords involved on previous queries issued by the client and observed by the attacker. This kind of attack is called *query recovery* and was proposed by Islam, Kuzu and Kantarcioglu (IKK) in [17]. Their attack, known in the literature as the IKK attack, targets the strongest kinds of SSE schemes which are those proved to be secure under the adaptive security definition. The IKK attack models the problem of recovering the unknown keywords as an optimization problem solved using a simulated annealing algorithm. Recently, Cash et al. improved the IKK by proposing another frequency attack, called the *Count attack*, that is simpler and more efficient than the IKK attack [7]. Both attacks require a complete knowledge about the dataset under attack. The attacker knows at least the frequency of each keyword also knows the co-occurrence between any two keyword. We denote the frequency knowledge by $\mathcal{K}_{\mathcal{F}}$ and the co-occurrence knowledge by K_O. Next, we briefly describe the IKK attack and the Count attack.

The IKK Attack. The IKK attack assumes that the attacker knows the entire document set in plaintext and the goal is to recover the queries issued on the encrypted document set which is of course unknown to the attacker. The IKK attack can be briefly described as follows: assume that the number of unique keywords indexed by an SSE scheme is m. The attacker constructs an $m \times m$ matrix M where its entry $M[i, j]$ holds the probability of having the ith unique keyword and jth unique keyword together in any random document in the database D indexed by the SSE scheme under consideration. The matrix M represents the background knowledge of the attacker about the encrypted database. The percentage of success of the attack depends on the accuracy of this background matrix M. From an l observed queries (encrypted keywords), the attacker calculates another $l \times l$ matrix Q where the entry $Q[i, j] = \frac{|R_i \cap R_j|}{n}$ where R_i is the result set of the ith query Q_i (or encrypted keyword) and R_j is the result set of the jth query Q_j (or encrypted keyword) and n is the number of documents in the database D. Now if the Euclidean squared distance $(Q[i, j] - M[s, t])^2$ is close to zero, then the attacker might know with high probability that the ith and the jth queries (encrypted keywords) correspond to the plaintexts keywords corresponding to the sth and tth entry of the background knowledge matrix M. Therefore, the attack model is to find the best best mapping: $(i, j) \rightarrow (s, t)$, that minimizes the following summation carried over all the l observed queries: $\sum_{Q_i, Q_j \in Q}(Q[i, j] - M[s, t])^2$. The minimization is done under constraints which are set from the subset of known queries $S \subset Q$ and the attackers goal is to assign the unknown queries $q \in (Q - S)$ to their corresponding keywords in the background knowledge represented by the matrix M. A simulated annealing algorithm is used to solve the above optimization problem.

The Count Attack. Similar to the IKK attack, Cash et al. proposed the count attack which does not only use a background knowledge matrix M representing the co-occurrence probabilities of the keywords in plaintext but also a background knowledge about the frequency of each individual keyword w in the database D, say $\mathbf{count}(w)$. This simple additional knowledge improves significantly the effectiveness of query recovery attacks and it does not use any numerical optimization technique. Obviously, assuming that the attacker background knowledge consists of all plaintext

document set, then any keyword w with unique frequency will be easily retrieved when queried as its result set $\mathbf{count}(q)$ will match the attacker's background knowledge $\mathbf{count}(w)$.

The unique counts approach will retrieve all the keywords with unique frequency which can be significant in some databases. Keywords with non-unique frequency are recovered by comparing the frequency of the elements of the query co-occurrence matrix with the frequency of the elements of the co-occurrence background matrix. Assume that (q', t) is a known query where t is the keyword and q' is the queried keyword (encrypted keyword). Assume that we want to recover the query q which has a non-unique frequency. Now, If $Q[q, q']$ and $M[s, t]$ are equal, then the keyword 's' is probably the plaintext of the query 'q'. Experiments in [7] show that the co-occurrence testing phase can successfully recover almost all the queries. At the end of Section 3.1, we show that our attribute attack could resolve some queries unresolved by the Count attack.

3 On the use of SSE to secure relational databases

The frequency attacks on adaptively secure SSE schemes are only query recovery attacks that only recover the plaintexts of the keywords associated with a document or a record but do not translate to partial document recovery which is the case for some searchable schemes with higher leakage as shown in [7] (such schemes are not secure under adaptive security definition [14]).

Traditionally, SSE schemes are designed to protect a set of unstructured documents. However, they could also be used to provide privacy for relational databases where each record is treated as a separate document and each ($attribute_i$, value) is treated as a keyword as done in [9,8]. Now, one can see that frequency attacks will probably lead to a partial or full record recovery since query recovery directly translates to partial record recovery in this case. This is because recovering a keyword means recovering its attribute and its value. To recover a record with identity id_i, one has to look for all (or part of) the keywords corresponding to the record with identity id_i. Thus if all the encrypted queries are recovered using for example the above Count attack. Then, one can fully reconstruct some target records by creating a forward index from the access pattern leakage of the observed encrypted queries and replacing each encrypted query with its corresponding plaintext obtained before through the Count attack. We see this as a real obvious threat which makes relational databases secured via SSE schemes at risk (c.f. Section 3.2).

Moreover, the structure of relational databases would enable an attack on them without resorting to the co-occurrence knowledge K_O. In the following, we describe a new attack called "Attribute Recovery" targeting relational databases protected by SSE schemes. The attack recovers the attribute names of queries, hence the name "Attribute Recovery" under the assumption that the attacker knows only the number of records and attribute names $\mathcal{A}$ together with their cardinalities. We define this as the attacker's basic background knowledge $\mathcal{K}_{\mathcal{B}}$.

Moreover, our attack could also recover the attribute value of a given attribute name 'a' under the assumption that the attacker has frequency knowledge $\mathcal{K}_{\mathcal{F}}$. Similar to the Cash et al.'s Count attack, the knowledge of frequency values $|\mathsf{DB}(w)|$ enables us to recover the keywords w whose frequency value $|\mathsf{DB}(w)|$ is unique. Similar to the Count attack, we also use the observed co-occurrence knowledge between any two observed queries which can be computed from the access pattern leakage of the two queries. However, the Count attack resolves the queries with non-unique frequency values using co-occurrence knowledge $\mathcal{K}_{\mathcal{O}}$ about the dataset whereas our attack resolves such queries by exploiting the properties of relational datasets. This is a clear advantage over Cash et al.'s Count attack when it comes to attacking relational datasets since it does not require any co-occurrence knowledge between plaintext keywords as done by Cash et al. [7] and Islam at al. [17].

3.1 Attribute Recovery Attack

We make use of the following simple observation about the co-occurrence of observed queries on searchable encrypted relational datasets (i.e. SSE is employed to protect the database and at the same time make it searchable). We assume that the attacker has basic background knowledge $\mathcal{K}_{\mathcal{B}}$

about the relational database under attack (i.e. the number of records n and the cardinality of each attribute and its possible values). Some notable SSE schemes such as [10,26,22,18] already leak the number of documents or records n.

Observation 1. The co-occurrence between any two keywords or observed queries with similar attributes is zero.

Our attribute recovery attack is based on the proposition below which follows immediately from Observation 1 and the fact that the total frequency of all possible values of an attribute a in an encrypted relational database EDB equals the number of records in the encrypted relational database EDB under the assumption that no padding countermeasure has been done to prevent frequency attacks.

Proposition 1. *Let $\boldsymbol{t} = \{t_1, \cdots, t_l\}$ be the tokens set of observed queries. Let $|t_i|$ denote the result size of query token t_i. Let C_O be the observed co-occurrence matrix. Let n be number of records in an encrypted index EDB. Let a be an attribute in the EDB whose cardinality $|a|$ is unique. Then there exists a subset $\boldsymbol{s} \subseteq \boldsymbol{t}$ where $\sum_{t_i \in \boldsymbol{s}} |t_i| = n$ and $|\boldsymbol{s}| = |\boldsymbol{t}|$ and $\forall t_i, t_j \in \boldsymbol{s}\,, C_O[t_i, t_j] = 0$.*

One can see that the above proposition can be used to develop an algorithm that distinguishes between observed queries. Such distinguisher can be mounted by a weak attacker who observes only the access pattern leakage of queries and has no prior knowledge other than the number of records, attribute names and their cardinalties. Assuming that an attacker observes all the possible $|\mathsf{W}|$ queries and that all attributes have unique cardinalities, a distinguisher algorithm separating and dividing all the queries according to their attribute name can succeed with probability one. However, an attacker does not need to collect all the possible $|\mathsf{W}|$ queries to mount his attack. If the attacker's observed queries contain all the possible values of an attribute a with unique cardinality, then observed queries whose attribute name is "a" will be recovered with probability one. Algorithm 1 takes a set of observed query tokens $\mathbf{t} = \{t_1, ..., t_l\}$ and the attacker's basic background knowledge $\mathcal{K}_{\mathcal{B}}$. It divides these tokens according to their attribute name into different sets where each set G_a represents the tokens whose attribute name is 'a'.

Discussion about Algorithm 1. One way to look at Algorithm 1, is to consider the co-occurrence matrix as the adjacency matrix of a weighted graph G_{Co} whose nodes are the queries and any two nodes are connected by an edge whose weight is the co-occurrence value between the two connected nodes (i.e. zero co-occurrence means no edge or edge with weight zero). This allows us to consider the elements of the list $L[\mathsf{ctr}]$ created in Algorithm 1 as nodes in another smaller weighed graph $G_{L[\mathsf{ctr}]}$ whose adjacent matrix is a submatrix of the co-occurrence matrix (i.e. adjacent matrix of the bigger graph G_{Co} containing all queries). Now rather than looking at the subsets of $L[\mathsf{ctr}]$ (Note that the first element $L[\mathsf{ctr}, 1]$ should be included in all subsets) whose size is equivalent to a given cardinality $|a|$, one can look at the independent sets of the graph $G_{L[\mathsf{ctr}]}$ corresponding to the list $L[\mathsf{ctr}]$ whose size is equivalent to $|a|$ with the additional condition that the total sum of the frequency of each node (i.e. query) equals the total number of records n. This is the known independent set NP-Complete problem with an additional filtering condition.

Another way to look at Algorithm 1 is to consider that Step 7 and Step 8 form the known k-SUM problem (i.e. Given $A = \{a_1, \cdots, a_s\}$ and a target sum t. Is there any subset of indices $\{i_1, \cdots, i_k\}$ such that $\sum_{j=1}^{k} a_{i_j} = t$?) with an additional condition that the co-occurrence between any two elements in the subset is zero. The k-Sum problem is a parameterized version of subset sum problem which is a known NP-complete problem. The brute force algorithm for the k-SUM problem takes $O(s^k)$ where s is the size of the given set. There are simple algorithms solving this problem in $O(s^{\frac{k}{2}} \log s)$ when k is even and $O(s^{\frac{k+1}{2}})$ when k is odd [3,15,16]. However, employing the co-occurrence condition might reduce the above complexity times but this needs further investigation. Obviously, Algorithm 1 performs well when k is small (i.e. the attribute cardinality is small). When the target sum t is not very large (i.e. number of records n is not very large), one can use the known dynamic programming technique to solve the subset sum problem in pseudo-polynomial time $O(st)$ [20].

Algorithm 1 Attribute Recovery Attack

Require: $\mathcal{K}_{\mathcal{B}}\vee$ Observed tokens $\mathbf{t} = \{t_1, ..., t_l\}$ and their results. $|a| \equiv$ cardinality of $a \in \mathcal{A}$ and $|t_i| \equiv$ result size of query token t_i.

Ensure: Recover the attribute name of observed queries.

1: Set $R = \{\}$. Compute the co-occurrence matrix C_O for observed queries tokens $\mathbf{t} = \{t_1, ..., t_l\}$.
2: For each t_i, create a list Q_i holding t_i ($Q_i[1] = t_i$) and all other tokens t_j's where $C_O[t_i, t_j] = 0$.
3: Sort all the lists Q_i according to their size in ascending order.
4: Add all the sorted lists Q_i's to a lists' container L. ▷ $L[i, j]$ is the jth entry in the ith list $L[i]$.
5: Choose an attribute $a \in \mathcal{A}$ where $|a|$ is the minimum cardinality.
6: Set $G_a \leftarrow \{\}$ and $\mathsf{ctr} = 1$.
7: **for all** $S \subseteq L[\mathsf{ctr}]$ where $|S| = |a|$ & $L[\mathsf{ctr}, 1] \in S$ **do**
8: **if** $\sum_{t_u \in S} |t_u| = n$ & $C_O[t_u, t_v] = 0 \quad \forall t_u, t_v \in S$ **then**
9: $G_a \leftarrow S$
10: $R \leftarrow R \cup G_a$
11: $L[\mathsf{ctr}] \leftarrow \emptyset$
12: **for all** i **do**
13: **if** $L[i, 1] \in G_a$ **then**
14: $L[i] \leftarrow \emptyset$
15: **else**
16: $L[i] \leftarrow L[i] \backslash G_a$
17: $\mathcal{A} \leftarrow \mathcal{A} \backslash a$
18: **if** $\mathcal{A} = \emptyset$ **then return** R
19: **else**
20: **goto** Step 5
21: $\mathsf{ctr} = \mathsf{ctr} + 1$.
22: **if** $\mathsf{ctr} \leq l$ **then**
23: **goto** Step 7.
24: **else**
25: **print**("No valid subset found corresponding to the chosen attribute ",a) and **goto** Step 5.

Now one can ask, which approach (i.e. independent set algorithms or k-SUM algorithms or dynamic programming of subset sum) is better to implement Step 7 and Step 8 in Algorithm 1. The answer depends on the dataset under attack and the available queries. We have implemented the dynamic programming algorithm for the subset problem where all the solutions are traced back and the co-occurrence condition is evaluated after finding each solution. This is practical if $O(st)$ is pseudo-polynomial which means that the number of records t is not large and if for each cardinality there exists a constructed list $L[\mathsf{ctr}]$ whose size s is not large. As future work, we leave investigating the other approaches, namely, the independent set problem with the additional filtering condition (total sum of frequencies) and the k-SUM approach with the additional zero co-occurrence filtering condition.

Recovering Attribute Values. Algorithm 1 might enable an attacker to recover the attribute names of some queries without any knowledge except $\mathcal{K}_{\mathcal{B}}$. Now with the basic knowledge $\mathcal{K}_{\mathcal{B}}$, the attacker knows the possible values of a given attribute name. Moreover, with the access pattern leakage, the attacker knows the result set size (i.e. frequency) of each observed query. However, in order to recover, without any prior knowledge such as those used in the Count attack $\mathcal{K}_{\mathcal{F}}$ and $\mathcal{K}_{\mathcal{O}}$, the attribute values of observed queries whose attribute names are recovered with Algorithm 1, an attacker needs to know at least the rank-size or rank-frequency distribution [1] of the attribute values. Assume that L_a is a sorted list containing the attribute values of an attribute named a according to their rank-size in descending order.

[1] The knowledge of the rank-size or rank-frequency distribution of a given attribute value does not necessarily imply the knowledge of the frequency of each attribute value $\mathcal{K}_{\mathcal{F}}$.

Now, one can see that by sorting the observed queries in descending order according to the size of their results and adding the sorted observed queries to a list L_q, then $L_a[L_q[t_i]]$ will be the attribute value of an observed query token t_i whose attribute name is a. If all the result sizes of queries in L_q are unique, then the above attack succeeds with probability one. Otherwise, there would be an error whenever we have a tie between two or more queries in L_q. This is a standard frequency attack similar to the one described by Naveed et al. [23] for attacking columns of a relational database encrypted using deterministic encryption. However, our attack targets relational databases encrypted using SSE schemes and firstly recovers the attribute name of a given query through Algorithm 1 and then secondly recovers its value through a standard frequency attack without any prior knowledge other than the rank-size of the attribute values of the attribute names belonging to the dataset under attack.

A strong version of the above attack can be mounted if the attacker knows only the frequency of each attribute value $\mathcal{K}_{\mathcal{F}}$. Here the attacker first recovers the observed queries whose results are unique using the standard frequency attack and second employs a variant of Algorithm 1 which we describe in what follows. Note that our attacker's goal is to recover the queries Q' with non-unique result size without resorting to any co-occurrence knowledge K_O which is a clear advantage over the Count attack.

The attack procedure starts by recovering the observed encrypted queries Q with unique result size (i.e. $|\mathsf{DB}(w)|$ is unique) using the attacker's background knowledge. Then, we divide the recovered queries Q into subsets according to their attributes. We then use our knowledge about the attributes of the dataset under attack. Let G_a be the subset of Q containing the recovered queries of a *discrete* attribute 'a'. We focus here on discrete attributes, non-discrete and variable attributes can be recovered using a guess-and-determine attack where at each attempt a different cardinality is guessed until the right one is determined. Now, we find the number of missing values of the attribute a in G_a, say $m_a = |a| - |G_a|$. If $m_a = 0$, then we have already recovered all queries whose attribute is 'a'. Otherwise ($m_a > 0$), we try to find the missing values from the set of unresolved queries Q'. In order to do so, we create a list Q'_a and add to it each element in Q' that has a zero co-occurrence with all the elements in G_a. Note that due to the zero co-occurrence property between similar attributes, described in Observation 1, $|Q'_a|$ would probably be less than or equal to m_a but it could also be larger than m_a. We treat the three cases as follows.

1. If $|Q'_a| = m_a$ and the co-occurrence between any two queries in Q'_a is zero, then all the queries in Q'_a probably have the same attribute which is probably 'a' similar to the elements of G_a unless there are queries with attribute 'a' that have not yet been queried. This means that they do not exist in the original unresolved queries set Q' and at the same time there is another attribute b where $m_b = m_a$ and all the unresolved queries of the attribute 'b' have zero co-occurrence with all the resolved and unresolved queries with attribute 'a'. However, if the sum of the queries' result sizes $\mathsf{totalFreq}(Q'_a)$ (i.e. $\sum_{t_i \in Q'_a} |t_i|$ where $|t_i|$ is the result size of query token t_i) is equal to n minus the sum of the resolved queries with attribute a, $\mathsf{totalFreq}(G_a)$, i.e. $\mathsf{totalFreq}(Q'_a) = n - \mathsf{totalFreq}(G_a)$, then with high probability we are confident that the attribute of the queries in Q'_a is 'a' unless $\mathsf{totalFreq}(G_b) = \mathsf{totalFreq}(G_a)$. Thus, for a database whose attributes' cardinalities are unique, an attacker with background frequency knowledge $\mathcal{K}_{\mathcal{F}}$ will be able to match between the frequency of the keywords in $\mathcal{K}_{\mathcal{F}}$ whose attribute is 'a' and the result size of the queries in Q'_a. In other words, the value of a query token $t_i \in Q'_a$ will be equivalent to a keyword $w \in \mathcal{K}_{\mathcal{F}}$ whose frequency is the result size of the query token t_i.
2. If $|Q'_a| > m_a$, then there are $\binom{|Q'_a|}{m_a}$ subsets of size m_a. We add a subset S' to the solution set $\mathsf{Sol}(G_a)$ iff the co-occurrence between any two elements in S' is zero and at the same time the total sum of frequencies of its elements $\mathsf{totalFreq}(S')$ equals to $n - \mathsf{totalFreq}(G_a)$. Now, if $|\mathsf{Sol}(G_a)| = 1$, then we could easily recover the missing values of the attribute a in case all the queries tokens in $S' \in \mathsf{Sol}(G_a)$ have different result sizes as shown in the first case using the background frequency knowledge $\mathcal{K}_{\mathcal{F}}$. If there some tokens in S' with similar result sizes, then we will only recover the queries in S' with different result sizes and recover only the attribute name (i.e. 'a') for queries in S' with similar result sizes. If the solution set has more than one solution, $|\mathsf{Sol}(G_a)| > 1$ which is a rare scenario, then performing matching between the

frequency of keywords in $\mathcal{K}_{\mathcal{F}}$ and the result size of queries tokens in Q'_a will probably yield one solution unless there are two different solutions in $\mathsf{Sol}(G_a)$ with similar result sizes.

3. If $|Q'_a| < m_a$, then there are $m_a - |Q'_a|$ queries with attribute 'a' that have not yet been queried. Now, the attacker can use his frequency knowledge $\mathcal{K}_{\mathcal{F}}$ to see whether all the values of the attribute 'a' have different frequency values. In case they have different frequencies, then the attacker can easily match between the frequency background knowledge and the result set size of the queries of Q'_a and thus recover the attribute values of the queries in Q'_a. Otherwise, the attacker will only manage to recover the attribute name but not its value for queries in Q'_a with similar frequency values.

Combined Count Attack and Attribute Attack. One can also combine the Count attack with our attribute attack in order to break the ties unresolved by the Count attack through the co-occurrence knowledge K_O but this will only break ties between any two queries with different attribute names and similar frequency values. So the gain here depends on the dataset.

3.2 Experimental Results

We performed two experiments on a small relational dataset. The first experiment (Exp. I) shows the practical viability of Algorithm 1 where the attacker has no knowledge beyond the observed co-occurrence matrix computed from the SSE's leakage and the basic knowledge $\mathcal{K}_{\mathcal{B}}$. The second experiment (Exp. II) shows that our attribute recovery attack can be used to recover also the values if the attacker knows (or can guess) the frequency of the keywords in the dataset under attack but not the joint frequency knowledge assumed by the Count attack. We also made a comparison between our attack and the Count attack. Moreover, we confirm the obvious observation that query recovery attacks via the Count attack on small or medium-sized relational dataset could lead to complete records recoveries.

Data Set. We used the Adult dataset [21] to conduct our experiments. It consists of $32561 \approx 2^{15}$ rows (i.e. records) and 14 columns (i.e. attributes). Out of these 14 attributes eight are discrete and six are non-discrete (variable or continuous). The Adult dataset has 498 distinct keywords. However, most of these keywords belong to two non-discrete attributes, namely, capital-gain and capital-loss. So for the sake of explaining the effectiveness of our attribute recovery attack on discrete attributes and emphasizing our point that query recovery attacks on small datasets are equivalent to record recovery attacks, we remove the capital-gain and capital-loss attributes from the Adult dataset[2]. The resulting dataset contains only 287 distinct keywords from 12 different attributes (8 discrete and 4 non-discrete). The eight attribute names are "sex" with its two values, "race" with 5 possible values, "relationship" has 6 values, "marital-status" with 7 values, "work-class" with 8 possible values, "occupation" with 14 possible values, "education" with 16 possible values, and "native-country" with 41 possible values. In fact, only 2 non-discrete attributes namely "age" and "hours-per-week" since "education-num" (this is also equivalent to the discrete attribute "education") and salary "class" have fixed values 16 and 2 respectively.

Query Generation. Our dataset is encrypted using a standard implementation of a basic SSE scheme supporting single-keyword queries. The queries are chosen randomly and the query result set and the access pattern leakage are recorded by our honest-but-curious attacker. Using this observed-queries knowledge, our attacker can compute the co-occurrence value between any two queries and thus the whole observed co-occurrence matrix. In Exp. I, we generated all the possible queries whereas different numbers of queries are generated in Exp.II.

[2] To look for a specific integer or floating-point variable attribute such as capital-gain or capital-loss, range queries are usually used. In such scenarios, applying the attack proposed in [19] is preferable.

Attribute Recovery Attack (Exp. I). Here we assume that our attacker knows only the basic knowledge $\mathcal{K}_{\mathcal{B}}$ about the relational dataset under attack beside to the observed co-occurrence matrix C_o computed from the leakage of the used SSE scheme (i.e. access pattern and in some schemes the number of records n can also be leaked). Algorithm 1 is implemented using the known dynamic programming procedure to solve the subset sum problem in each set $L[\mathsf{ctr}]$. The time complexity is $O(2^{15} \cdot |L[\mathsf{ctr}]|)$ where $|L[\mathsf{ctr}]|$ (averaged over all lists yields a value much less than the total number of queries) beside the time taken to trace back all the possible solutions. The size of the first five elements in the sorted list's container L was below 20. All the discrete attributes whose cardinality is unique have been recovered successfully. However, attributes with the same cardianlity such as the "sex" and salary "class", where each has 2 values, have been distinguished from the other attributes but we can not tell which of the 2 values point to the "sex" attribute and which point to the "class" attribute. Similarly, each of the "education" and "education-num" attributes has 16 values[3]. However, all their values were in one list ranked at position 29 in the sorted lists' container and at the same time each value in each attribute has zero co-occurrence with all the other values except one value. This make it impossible to separate the values of each attribute as we did for the "sex" and "class" attributes. In fact, our dynamic programming implementation of Algorithm 1 generated exactly $32678 = 2^{15}$ solutions. The reason as that the first element of each list is included in each solution but each of all the other 15 values has two possibilities which gives us in total 2^{15} solutions. In such scenarios Algorithm 1 fails completely to recover the attribute name of a class of keywords.

Attribute and Value Recovery Attack (Exp. II). In addition to the basic knowledge $\mathcal{K}_{\mathcal{B}}$ and the observed co-occurrence computed from the SSE's leakage, our attacker knows the frequency of each keyword $\mathcal{K}_{\mathcal{F}}$ but does not know the co-occurrence between any two keywords. So the attacker first recovers queries with unique result size using his frequency knowledge $\mathcal{K}_{\mathcal{F}}$ and then applies the attribute value recovery procedure (i.e. the three cases) outlined above to break the ties between queries with similar result size. Figure 1 shows that the attacker managed to recover more keywords with our attack compared to the standard unique-frequency attack. However, the Count attack is clearly more effective than our attack since it assumes that the attacker has co-occurrence knowledge $\mathcal{K}_{\mathcal{O}}$ about the dataset which we consider as a strong assumption that might rarely hold in practice.

Record Recovery via Count Attack (Exp. II). The queries recovered during our second experiment using the Count attack lead to complete record recovery as pointed above in Section 3. Figure 1 shows that the Count attack recovered approximately 90-99% of the available queries (regardless of the amount of available queries). Figure 2 shows that the average recovery per record is dependent on the number of available queries, ranging between 30% and 100%. One might argue that our keyword space is too small but this is often the case in a single relational database table especially if it only holds discrete attributes or non-discrete attributes with small domain. Therefore, we believe that this result shows that protecting small or moderate-size relational datasets via SSE schemes could be a risk.

4 Record-Injection Attacks

Zhang et al.'s file-injection binary attack [27] cannot be applied exactly here since we are dealing with a structured text governed by a relational database rather than unstructured text. Depending on the relational database under attack, there might be a large number of records needed to be injected in order to recover all the possible encrypted queries if the database contains attributes

[3] There is one-to-one correspondence between education and education-num, if we know the value of education we can determine the value of education-num with probability 1. While it might be awkward to include them in one database table, this serves as a good example to explain when Algorithm 1 can fail.

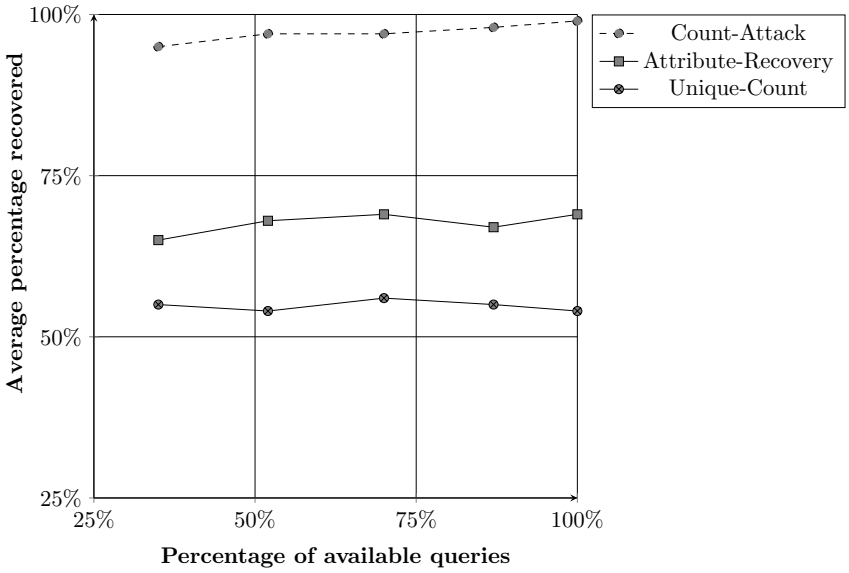

Fig. 1: Displays the query recovery for available queries with three different attacks, i.e. Unique-Count, Attribute-Recovery and Count-Attack.

whose values are variables (not discrete) with a big range. However, if the attacker is concerned about a small subset of keywords. Then she can inject a number of records by focusing on some attributes whose values are discrete with a small range. This will reduce the number of injected records and will lead to query recovery and consequently partial record recovery without any prior knowledge. In the following, we discuss how to estimate the minimum number of records needed to be injected in order to completely recover an encrypted relational database. Similar to Zhang et al. [27], we assume that the attacker can identify the record ID of each injected record. Let D be a relational database with n records and m attributes or columns where each attribute is denoted by a_i and its cardinality is denoted by $|a_i|$, $1 \leq i \leq m$. Assume that the number of records need to be injected in D in order to cover the whole keyword space K or a target subset of keywords S is l. Suppose that $R = r_1 r_2 \cdots r_l$ is the search result on the injected records regarding an observed query q, where $r_i = 1$ iff the ith injected record is part of the result set of the query q, otherwise $r_i = 0$. Clearly $l \geq |a_i|$ for all i, otherwise the injected records will not recover all the values of the attribute a_i.

Assume that there are t attributes $(a_{i1}, \cdots a_{it})$ with the same cardinality d, then in order to cover all the $d \cdot t$ values one can construct a bijective mapping from the attributes values to the search result string by simply injecting $d \cdot t$ records in a certain way, as follows. Let the first d records contain all the values of the 1st attribute and the other attributes belonging to S are empty. Also let the second d records contain all the values of the 2nd attribute and the other attributes belonging to S are empty and so on until the last and tth d records containing all the values of the tth attribute and the other attributes belonging to S are empty. Now one can see that the search result on the injected files regarding any keyword in S will yield a binary with Hamming weight one where the location of the active bit i indicates that the keyword is located at position $i \mod d$ (or last position if $i \mod d = 0$) in the $\lceil i/d \rceil$th attribute. However, one can do better by injecting much less number of records by gradually incrementing the number of records l starting from $l = d$ and checking whether it is possible to construct a bijective mapping between attribute values and search result on injected records.

The following proposition enables us to estimate and come close to the number of injected records. One can brute-force search for the smallest number l satisfying the inequality below $l \leq d \cdot t \leq \binom{l}{l/d}$ in order to find the minimum number of injected records needed to uniquely identify and recover all the queries.

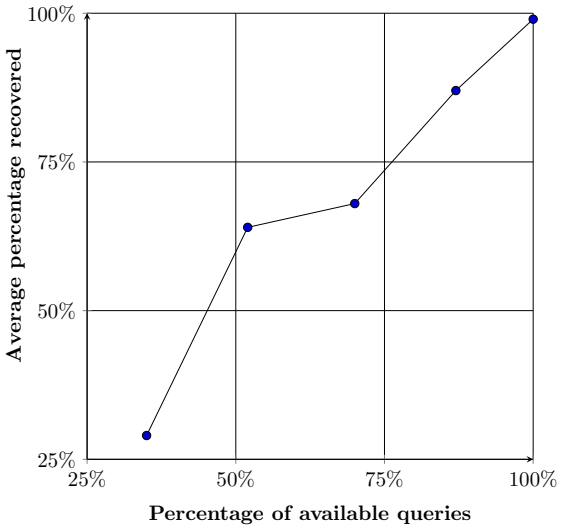

Fig. 2: Displays the average percentage of records recovered of the entire dataset given different numbers of available queries.

Proposition 2. *Let d be the cardinality of a subset of attributes S of size t in a database D. Then the number of injected records l needed to uniquely identify all the values of S satisfies the inequality $l \leq d \cdot t \leq \binom{l}{l/d}$.*

Proof. Assume that l/d is an integer. Clearly injecting $d \cdot t$ records would allow unique encodings for the search results on the injected records. So $l \leq d \cdot t$. However, it is also possible to inject a much smaller number of records l that is strictly smaller than $d \cdot t$ and still uniquely identify all the keywords. In order for such number of records to yield a unique search result on the injected records for all the $d \cdot t$ keywords, we need to make sure that each keyword can be uniquely represented by an l-bit string representing the search result on the injected records and has Hamming weight equal to l/d. This can be done by constructing a bijective mapping between the keyword values and the l-bit search result on the injected records. Replacing each occurrence of a keyword value in a column in the injected records by '1' and each non-occurrence by '0' yields a binary string with Hamming weight l/d equivalent to the search result of the keyword value on the injected records. Consequently, we get a bijective mapping. Now, to enable the unique encoding of all possible $d \cdot t$ keyword values, the number of l-bit strings $\binom{l}{l/d}$ should be $\geq d \cdot t$ since otherwise all the possible keyword values will not be covered. □

The corollary below follows from the above proposition and it gives a sufficient number of records that need to be injected in order to obtain a unique search result on the injected records for all possible keywords.

Corollary 1. *Let S be a set of m attributes. Let $a_{\max}$ be the maximum cardinality in the set S. Then injecting $2 \cdot a_{\max}$ records is enough to get a unique search result on the injected records as long as $m \leq 2 \cdot a_{\max} - 1$.*

Proof. If the number of injected records can uniquely hold the maximum number of possible keywords $a_{\max} \cdot m$, then it will be able to uniquely hold any number of keywords less than $a_{\max} \cdot m$. Therefore we assume that all the m attributes have cardinality $a_{\max}$. Then using the above inequality where $d = a_{\max}$, $t = m$ and $l = 2a_{\max}$, we find

$$a_{max} \cdot m \leq \binom{2a_{\max}}{2} = 2a_{\max}^2 - a_{\max}$$

Thus, $m \leq 2a_{\max} - 1$. □

The following corollary gives the precise minimum number of injected records needed to cover all the keyword space for some datasets whose maximum cardinality is more than or equal to the double of the cardinality of other attributes.

Corollary 2. *Let S be a set of m attributes. Let there be a single attribute with the maximum cardinality in S, $a_{\max}$. If there is no any attribute b such that $a_{\max} < 2|b|$. Then injecting $a_{\max}$ records where each attribute c contains $a_{\max}/|c|$ instances of each possible column value is the minimum number needed to uniquely identify all the possible keyword values of S as long as $m \leq 2|b|$.*

Proof. Similar to the proof of Corollary 1 except here we set $t = m - 1$ (excluding the attribute column with maximum cardinality), $l = 2 \cdot |b|$ and $d = |b|$ to get $m \leq 2 \cdot |b|$. Note that $l = 2 \cdot |b|$ will cover all attributes except the ones with maximum cardinality $a_{\max}$. So we need to inject not less than $a_{\max}$ records to cover the keywords about the values of the attribute with the maximum cardinality. Since $a_{\max} > l$ then injecting $a_{\max}$ is the exact number needed to cover all possible keywords. □

Following the above corollaries, the number of records will be in the interval $[a_{\max}, c \cdot a_{\max}]$ where $c \geq 1$ is a small constant. If we have an attribute whose values are not discrete, then $a_{\max}$ might be large. However, one can reduce the number of injected records by predicting a small subset that is dominant in the dataset.

Discussion. It is clear that once an attacker is able to inject his own records then record-injection will lead query recovery and eventually could lead to full record recovery as pointed above in our second experiment where we showed that the Count attack could lead to complete record recovery. The above record injection attack works under the assumption that the attacker can identify the record identifiers of the injected records. This assumption is also adopted by Zhang et al. [27]. If the client updates one record at a time, then the attacker will always be able to identify the identifiers of the injected records. However, if the client does only batch updates, then the attacker should inject records in a way such that each keyword has a unique number of appearances in all the injected records.

The file-injection countermeasure proposed by Zhang et al. [27] which restricts the number of keywords per document to a certain threshold T (e.g. $T \ll |W|/2$ where $|W|$ is the number of keywords in the dataset) cannot be applied here since a relational database record has a certain number of keywords equivalent to its number of attributes and any restriction would hinder the work of any application using the SSE-encrypted database. So one needs to use a forward secure SSE scheme such as the one proposed in [6] in order to protect relational datasets against record-injection attacks. Note that a forward secure SSE scheme does not provide protection against injection attacks if the tokens corresponding to the injected keywords have not been queried before the injection attack. So additional system-based or application-based countermeasures protecting the update process in the application using the encrypted-dataset from being attacked by an active adversary performing record-injection attacks need to be employed.

5 Countermeasures Against Attacks on SSE Schemes

Countermeasures against frequency attacks must be used in order to reduce the effectiveness of the frequency attacks demonstrated above. A well known technique is *padding* which is proposed in [17,7] as a potential countermeasure to reduce the effectiveness of frequency attacks. Basically, during the setup of the encrypted database, the client adds dummy document or record IDs to each keyword in the index order to hide the actual frequency of the keyword. Also, the client adds an encrypted dummy document corresponding to each dummy ID added in the index. Later, during search, the client filters out the dummy documents. Experiments in [7], show that a padding level that increases the index size by 15% for a real world sample dataset and 30% for another real

world sample dataset, does not affect the success rate of the generalized Count attack [7] which is a slight improvement of the above described Count attack that basically does not depend on the keywords with unique frequency which will not exist in a padded SSE scheme but it initially guesses these keywords. The detection of a wrong guess is done during the co-occurrence testing phase which does equality matches in a window or a range of a fixed size to nullify the noise coming from the dummy documents causing false co-occurrences. Thus, the generalized Count attack presented in [7] suggests that padding alone does not reduce the effectiveness of frequency attacks as matches in a range can be done through the observed co-occurrence matrix.

Countermeasure Against the Attribute Recovery Attack. The above padding countermeasure which hides the actual size of each query prevents our attribute recovery attack. However, depending on the padding level, a variant of the attribute recovery attack that does not look for the exact number of records could still work. So another countermeasure is needed such as requiring a unified cardinality for some or all attributes to prevent distinguishing attributes with unique cardinality. This can be done by adding dummy values for each attribute together with their corresponding dummy record identifiers in order to have at least two attributes with similar cardinalities. As shown in Exp. I, attributes with similar cardinality are difficult to distinguish from each other using only the basic knowledge $\mathcal{K}_{\mathcal{B}}$. Also requiring the minimum cardinality to be large would increase the time complexity of Algorithm 1 and thus make our attack impractical. Moreover, an SSE scheme leaking the number of records n should not be used unless dummy records are added before encrypting the index in order to hide the actual number of records n. All these countermeasures prevent our attack at the expense of storage and time efficiency of SSE schemes.

Countermeasure Against the Count Attack. Beside reducing the effectiveness of the keyword count leakage by padding, one might think of reducing the effectiveness of the keyword co-occurrence leakage by having the observed co-occurrence of some keywords to be zero. One can see that if q_i and q_j are distributed in different fragments according to a defined privacy constraints, then $Q[q_i, q_j]$ will be zero when each query is executed in only one fragment and fragments are not allowed to interact together to evaluate queries. Now an equality match or a window equality match with the background matrix $M[s_i, t_j]$ as done in [7] will never happen which will significantly reduce the effectiveness of the Count attack. This can be done by applying vertical fragmentation to a relational database table according to a pre-defined set of *privacy constraints* on its columns. The aim behind the privacy constraints is hiding the association among the attributes which means that there should be no joint appearance of the attributes in the privacy constraints [1,11,13]. For example, consider the above defined Adult dataset. The privacy constraint $c_1 = \{$sex,occupation$\}$ prevents the "sex" column and "occupation" column from being together in one fragment. Another example, consider the relation of the following attributes about patients in a hospital: Name, Date of Birth (DOB), Disease, Medical Doctor (MD), and ZIP. Now one can define the following privacy constraints $c_1 = \{$Name, DOB$\}$, $c_2 = \{$Name, Disease$\}$, $c_3 = \{$Name, ZIP$\}$, $c_4 = \{$Name, MD$\}$, $c_5 = \{$DOB, ZIP, Disease$\}$ and $c_6 = \{$DOB, ZIP, MD$\}$. Now a privacy constraint prevents some columns from being together, so the privacy constraint $c_1 = \{\mathsf{Name}, \mathsf{DoB}\}$ prevents the Name column or attribute from being together in one fragment with the DoB column since they might reveal together more information about a specific person if frequency attacks are applied on an SSE scheme protecting the fragment.

We note that the privacy constraints should be used to produce the minimal amount of fragments possible using the heuristic algorithm proposed in [11]. For instance, a valid minimal fragmentation for the relation and privacy constraints defined above is the following $\mathcal{F} = \{\mathcal{F}_1 = \{$Name$\}$, $\mathcal{F}_2 = \{$DoB,ZIP$\}$, $\mathcal{F}_3 = \{$Disease,MD$\}\}$. After applying the fragmentation, we need to ensure that each query is executed in only one fragment in order to prevent an attacker monitoring all the fragments (or collaborative honest-but-curious fragment servers) from gaining any information about the correlation of the records between any two fragments which will obviously break the pre-defined the privacy constraints set by the data owner. Moreover, we need to have different

record IDs for the same record at each fragment in order to achieve security against an attacker monitoring all fragments (or collaborative honest-but-curious fragment servers) and also apply random shuffling for the fragment's records. After that we can apply the same SSE scheme in each fragment using a different key. This ensures that applying frequency attacks on each fragment is not effective since the encrypted attributes within each fragment does not provide sufficient information if they are recovered. Note that the generalized Count attack is effective in each fragment and it could probably recover entire records in each fragment. However, the fragments are defined according to the privacy constraints which means that the recovered records are unlinkable and thus will not reveal useful information. If all fragments are recovered, the attacker will not be able to link or combine them and find the original record before fragmentation since each fragment is shuffled differently and each fragment's record has a different record ID for the same original record.

Security Gain From Privacy Constraints. Vertical fragmentation using privacy constraints prevents full record recovery but the fragmented SSE scheme will still leak the access pattern inside each fragment as well as leakage the attribute of the keywords in each fragment. The access pattern leakage can be reduced by padding and the attribute leakage can also be reduced if one can pad dummy values in order to have the same cardinality for the fragment's attributes. We performed an experiment to show the amount of security gained when we employ vertical fragmentation via privacy constraints. In this experiment, the dataset is split into three different fragments. All fragments are protected using the same SSE scheme but each fragment has its own secret keys and has its own encrypted index. No interaction between the fragments are allowed during an execution of a query. We applied the Count attack on each fragment. Figure 3 shows the results of this experiment. Each of the first and third fragments contain few attributes, so when all possible queries in each fragment are queried, we are able to recover all the queries using the Count attack and thus consequently recover all the records in each fragment. However, an attacker cannot recover the same record distributed between the two fragments since each fragment is shuffled differently and keywords belonging to the same record are fragmented into two records with different IDs. One can see that in the second fragment which contains 7 attributes, we have only recovered approximately less than 75% of each record in the second fragment and thus around 43.75% of each record in the original dataset before fragmentation. This is a clear advantage compared to the record recovery attacks shown in Figure 2 which recovers almost 100% of each record when all the possible queries are available.

Performance of SSE Schemes under Fragmentation. The drawback with the fragmentation approach is the additional computational work in the use case of a multi-keyword query whose keywords exist in different fragments. But this can be combated by the use of a fragmentation algorithm which takes usage data into account [13]. This usage data is compiled in the form of an affinity matrix. For each pair of attributes there exists an affinity value. This value defines the frequency for which this specific pair of attributes exist in a multi-keyword query. This will allow us to find a suitable minimal fragmentation providing efficient execution for multi-keyword queries and the same time meeting the security demands set by the privacy constraints.

6 Conclusion

In this paper, we proposed a novel attack exploiting the structure of relational datasets. Our attack targets small relational datasets and requires only the access pattern leakage inherent in all SSE schemes beside basic information about the database under attack. We also showed that our attack can be easily prevented by requiring some or all attributes to have the same cardinality to hide the cardinality of each attribute. However, this will increase the size of stored encrypted index and thus decrease the efficiency.

Moreover, we showed that record-injection attacks pose a real threat for SSE schemes that does not provide forward security [6,27]. Furthermore, we proposed the use of privacy constraints

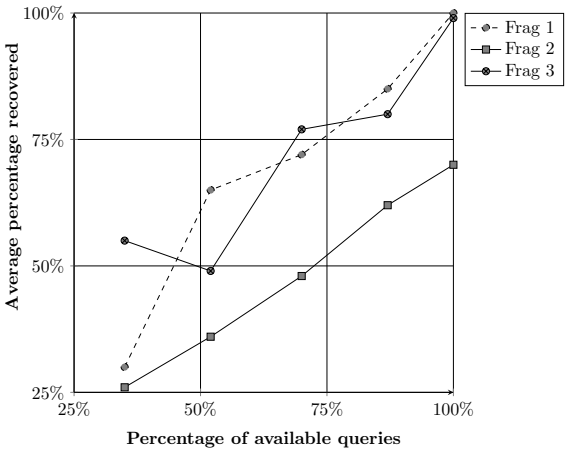

Fig. 3: Displays the result of the average percentage recovered per record for the fragmented dataset. Frag. 1 includes "sex, marital-status and relationship", which totals at approximately 5% of the keyword space. Frag. 2 includes "age, work-class, education, education-num, occupation, hours-per-week and salary class", which totals at approximately 78% of the keyword space. Frag. 3 includes "native-country and race", which totals at approximately 17% of the keyword space. All queries are single keyword searches where for each experiment the queries are split between the three fragmentations but not uniformly split as this depends on the distribution of attributes' values in each fragment.

together with padding on top of any SSE scheme in order to reduce the effectiveness of the frequency attacks proposed in [17,7]. Clearly the privacy constraints would affect the performance of searchable symmetric encryption schemes. However, several optimizations similar to the ones proposed in [13] can be performed to improve the performance.

Acknowledgments

We would like to thank Erik Zenner for helpful comments and suggestions on an earlier draft of this paper. This work was supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 644814, the PaaSword project within the ICT Programme ICT- 07-2014: Advanced Cloud Infrastructures and Services.

References

1. Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnaram Kenthapadi, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, and Ying Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR 2005*, 2005.
2. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2004.
3. Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *Journal of the ACM (JACM)*, 52(2):157–171, 2005.
4. Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference*, pages 535–552. Springer, 2007.
5. Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.
6. Raphael Bost. $\sigma o \varphi o \varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154. ACM, 2016.

7. David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 668–679. ACM, 2015.
8. David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. *IACR Cryptology ePrint Archive*, 2014:853, 2014.
9. David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.
10. Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010*, pages 577–594. Springer, 2010.
11. Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Computer Security–ESORICS 2007*, pages 171–186. Springer, 2007.
12. Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *Computer Security–ESORICS 2009*, pages 440–455. Springer, 2009.
13. Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):22, 2010.
14. Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88. ACM, 2006.
15. Jeff Erickson. Lower bounds for linear satisfiability problems. In *SODA*, pages 388–395, 1995.
16. Omer Gold and Micha Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. *CoRR*, abs/1512.05279, 2015.
17. Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, volume 20, page 12, 2012.
18. Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.
19. Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340. ACM, 2016.
20. Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.
21. Ronny Kohavi and Barry Becker. Adult data set. https://archive.ics.uci.edu/ml/machine-learning-databases/adult/, 1996. [Last Accessed 2016-10-18].
22. Kaoru Kurosawa and Yasuhiro Ohtaki. Uc-secure searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pages 285–298. Springer, 2012.
23. Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.
24. Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.
25. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.
26. Peter van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. *Secure Data Management: 7th VLDB Workshop, SDM 2010, Singapore, September 17, 2010. Proceedings*, chapter Computationally Efficient Searchable Symmetric Encryption, pages 87–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
27. Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. Cryptology ePrint Archive, Report 2016/172, 2016. `http://eprint.iacr.org/2016/172`.