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

RISE SICS 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 [28]. Later, Curtmola 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 Curtmola 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. Besides 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 [26]. 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. [24] mounted frequency analysis 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 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 a passive adversary monitoring the fragments.

Traditionally, SSE schemes are designed to protect a set of unstructured documents (e.g. email archive or a backup or any set of sensitive text files). However, recently under the Intelligence Advanced Research Project Activity (IAPRA) SPAR program [27], researchers from IBM along with other researchers proposed an efficient and elegant SSE scheme at Crypto 2013 [9] which in their own words, quoted from page 4 in [9], "*We develop the first non-generic sublinear SSE schemes supporting conjunctive keyword search (and more general Boolean queries, see below) with a non-trivial combination of security and efficiency. The schemes performance scales to very large datasets and arbitrarily-structured data, including free-text search.*", targets relational databases as well as unstructured data. Besides providing confidentiality to data and queries in relational databases and unstructured data, their solution also provides more functionality compared to previous SSE schemes by supporting a large subset of Boolean queries as well as better efficiency as it achieves a query speed that is comparable to the unprotected MySQL (release 5.5) query but with a storage cost of up to seven times the unencrypted data [9,17]. Later, under another IARPA project, a dynamic SSE scheme is proposed to support the protection of dynamic databases [8]. To the best of our knowledge, the SSE schemes developed under the IARPA projects are the first SSE schemes aiming towards protecting relational databases using SSE schemes by improving the functionality in [9] and the efficiency and secure support for updates in [8].

Comparing the above methods for searching on encrypted relational databases in terms of security, one can argue that SSE schemes offer better security than deterministic or order preserving encryption schemes since they do not leak the frequency of an attribute-value pair 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, to the best of our belief, the security gain from protecting relational databases via SSE schemes has not been studied and analyzed before as the security of SSE schemes have only been studied against inference attacks [18,7] in their traditional use case scenario where documents datasets such as email archive are protected. These inference attacks represent a major threat against SSE schemes since they can be launched by passive adversaries with background information about the target data. Therefore, in this paper, we study the implications of inference attacks on searchable encrypted relational databases (i.e. relational databases that are protected via SSE schemes). Our work basically shows that the structure of relational databases adds more leakage that enables an attacker to infer more information about the attribute names of protected queries which represent very sensitive information since revealing them will obviously enable an attacker to collect all queries sharing the same attribute name in a single column and consequently apply frequency analysis attacks similar to the ones proposed by Naveed et al [24]. We also study the effect of the recent file-injection attacks [30] on relational databases.

In order for our attacks to work, the attacker needs a background information about the target relational database table under attack such as the structural metadata of the database table in addition to the number of records. It is important to note that our attacks do not target the SSE schemes [9,8] developed under the IARPA projects as they specifically do not leak the number of records needed to launch our attack. However, it might be reasonable to assume that the attacker knows the number of records especially in relational databases that are static (e.g. archived databases).

Our Contribution. We show the severe consequences of inference attacks [18,7] 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 besides 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 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. Specifically, our inference control is 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 inference attacks exploiting the access pattern leakage [18,7] which is inherent in SSE schemes.

Related Work. Query recovery attacks exploiting the access pattern leakage of SSE schemes were proposed by Islam et al. [18] and later improved by Cash et al. [7]. By exploiting the structure of relational databases, our new attack requires lesser background information about the relational 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 [18,7] might lead to complete record-recovery attacks when the dataset under attack is a relational database. Furthermore, the padding countermeasure against query recovery attacks proposed in [18,7] does not fully prevent the generalized Count attack proposed in [7] (cf. Section 5). Therefore, 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 et al. [24] proposed a number of attacks targeting relational database columns encrypted using deterministic encryption [4] or order preserving encryption algorithms [2,5] in CryptDB [26] where encrypted values belonging to the same column whose name is encrypted are collected together as one set or more precisely one column vector. While seems similar, their column finder procedure is different than our attribute recovery attack in SSE schemes. It takes as input the set of encrypted values (i.e. column vector 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 its possible values are known) belonging to the attacker's 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 recent generic attacks [20] proposed at CCS 2016 target any secure database systems supporting range queries but leaking the access pattern without any prior knowledge about the dataset under attack. Most notably, one of their generic attacks target even secure encrypted search methods supporting range queries and only leaking the communication volume (i.e. query result size) such as fully homomorphic encryption or ORAM schemes. Their attacks only target range queries and they require the attacker to gather at least N^4 queries (N is the domain size) to mount the attack successfully. However, our work targets relational databases protected by SSE schemes dealing with equality queries. The only thing in common between our attribute recovery attack and their attack that targets strong encryption schemes (e.g. ORAM and FHE) is the requirement of knowing the number of records in addition to the query result size (or communication volume as defined in [20]) as an important information needed to launch the attacks.

The recently proposed file-injection attack by Zhang et al. at USENIX 2016 [30] recovers only the set of keywords in an encrypted document (and not the actual plaintext document) and could be 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 [30] since that would hinder the addition of new complete records to the relational database.

Organization of the paper. Section 2 gives a brief overview of SSE schemes and the frequency attacks on them. In Section 3, we point out the security risks 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 inference attacks.

2 Background about SSE Schemes

In this Section, we define SSE schemes and explain their leakage. We also explain the inference attacks exploiting the leakage of SSE schemes. Let DB denote a database of n documents and m unique keywords. Let $\mathsf{ID} = (\mathsf{id}_i)_{i=1}^{n}$ denote the set of all records' (or documents') identifiers in DB and let $\mathsf{W} = (w_j)_{j=1}^{m}$ denote the set of all unique attribute-value pairs (or documents' keywords) in DB. Let $|\mathsf{W}|$ denote the number of unique attribute-value pairs (or unique keywords in a set of unstructured documents) in the database. The database can be represented as an inverted index where each unique attribute-value (or keyword) is associated with a list of records (or documents) identifiers, i.e. $\mathsf{DB} = (w_i, \mathsf{DB}(w_i))_{i=1}^{m}$ where $\mathsf{DB}(w_i)$ denotes the set of records (or documents') identifiers containing the attribute-value pair $w_i = (attribute_i, value)$ or (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 $(w_i, \mathsf{DB}(w_i))_{i=1}^{m}$ 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 record (or document) identifiers are encrypted using a randomized encryption algorithm. The SSE scheme also encrypts the original database records (or documents) using a randomized encryption algorithm and stores it in an encrypted database EDB indexed by the record (or 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 record (or document) identifiers from EINDEX and decrypts them and consequently sends the corresponding encrypted records (or documents) from EDB to the client.

Leakage Profile. An SSE scheme leaks the *access pattern*: the result of the query or the record (or document) IDs corresponding to the queried keyword w_i, $\mathsf{DB(w_i)}$, 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 Curtmola et al. [14] where security is achieved against an honest-but-curious server. That means a passive adversary following the protocol but curious to use the leakage profile to learn about the queries (tokens) and the encrypted records (or documents).

Inference Attacks on SSE Schemes. Classical ciphers were broken by frequency analysis which is a standard example of an inference attack where an attacker can recover a plaintext character by inferring some information about its corresponding ciphertext character using language statistics. Similarly, using publicly-available auxiliary data, an inference attack can be mounted on adaptive SSE schemes to recover the plaintext of queries (tokens) involved on previous queries issued by the client and observed by the attacker (e.g. honest-but-curious server or passive external attacker). This kind of attack performs *query recovery* and was proposed by Islam, Kuzu, and Kantarcioglu (IKK) in [18]. 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 assumes knowledge about the *joint frequency (or co-occurrence count)* of any two plaintext keywords $w_i, w_j \in \mathsf{W}$ and also assumes knowledge about a subset of queries (tokens) issued by the client in plaintext. A similar inference attack has been proposed recently by Cash et al. is called the *Count attack* [7]. The Count attack also assumes knowledge about the frequency of each keyword (attribute-value pair) over all the documents (records) which means $|\mathsf{DB}(w_i)|$ where $w_i \in \mathsf{W}$ and DB is the original plaintext dataset. We denote this knowledge by $\mathcal{K}_{\mathcal{F}}$. Similar to the IKK attack, it also requires knowledge about the joint frequency (or co-occurrence count) of any two keywords (attribute-value pairs) $w_i, w_j \in \mathsf{W}$. Both, the IKK and the Count attacks, represent the joint frequency knowledge in a matrix called the *co-occurrence knowledge-matrix*, C_w. Therefore, both attacks could require a complete knowledge about the dataset under attack in order to form the co-occurrence knowledge-matrix. Both attacks exploit the access pattern leakage inherent in SSE schemes to compute the result size of any observed query (token) and also compute the joint frequency of any two observed queries (tokens) which is equal to the size of the set resulting from the intersection between the result sets of the two queries. A joint *co-occurrence token-matrix*, C_t, is then formed and compared to the *co-occurrence knowledge-matrix*, C_w in both attacks.

Active Attacks on SSE Schemes. The above inference attacks are passive attacks mounted by an honest-but-curious server who knows all or a significant number of the client's plaintext dataset. Another class of attacks outlined by Cash et al. [7] are the chosen-document attacks and the chosen-query attacks. Both attacks are mounted by an active adversary who is somehow capable of deceiving the client into including her own chosen-document into the documents set as well as into choosing her favorite queries respectively. Recently, Zhang et al. [30] 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. The equivalent of file-injection in the context of searchable encrypted relational databases is record-injection and it does have worse consequences that go beyond query-recovery such as full database record recovery or partial record recovery (cf. Section 4).

3 On the use of SSE to secure relational databases

The previously discussed inference attacks on adaptively secure SSE schemes are query recovery attacks that only recover the keyword of a query token. It allows us to the keywords associated with a document or but do not translate to full or partial document recovery which is the case for some less-secure searchable schemes with higher leakage as shown in [7] (such schemes are not secure under adaptive security definition [14]). However, when an adaptive SSE scheme protects a relational database, the implications of inference attacks will not be similar to the case where SSE schemes protect unstructured databases. Obviously, inference attacks on any adaptive SSE scheme protecting a relational database 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 query in the context of searchable encrypted relational databases means recovering one cell of a database record (i.e. an attribute-value pair). To recover a record with identity id_i, one has to look in the access pattern leakage of an SSE scheme for all (or part of) the unique queries (tokens) corresponding to the record with identity id_i. Thus if all the queries (tokens) 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 queries (tokens) and replacing each query (token) with its corresponding plaintext keyword obtained before through the Count attack. We see this as a real obvious threat which puts relational databases secured via SSE schemes at risk (cf. Section 3.2).

Moreover, the structure of relational databases does enable an attack on them without resorting to the co-occurrence knowledge-matrix C_w about the dataset under attack. 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 (tokens) which are also sensitive information beside the attribute values, hence the name "Attribute Recovery". Under the assumption that enough queries have been observed and that the attacker knows only the number of records, the unique attribute or column names $\mathcal{A}$ and cardinalities of the dataset under attack. We define this as the attacker's basic background knowledge $\mathcal{K}_{\mathcal{B}}$. This is definitely much less than the prior knowledge required by previously discussed inference attacks and can be easily guessed or acquired. For example, if an honest-but-curious server located at a hospital holds an encrypted database, then an attacker can easily acquire the columns' or attributes' names of the protected dataset by referring to publicly-available information such as the meta data about database tables used in standard medical software applications such as OpenEMR [25] which is well-known open source medical software supporting Electronic Medical Records (EMR). The number of records, which can be dynamic depending on the protected dataset under attack, could be gained either through a guess-and-determine process especially for small size datasets or through a leakage of the SSE scheme under attack as some notable SSE schemes such as [10,29,23,19] already leak the number of documents or records.

Moreover, our attack could also recover the attribute value of a given attribute name 'a' under the assumption that the attacker knows $\mathcal{K}_{\mathcal{F}}$. Similar to Cash et al.'s Count attack, the knowledge of $\mathcal{K}_{\mathcal{F}}$ enables the attacker to recover the queries (tokens) t_i whose result-size $|t_i|$ is unique. Also similar to the Count attack, we use the observed co-occurrence token-matrix C_t. However, the Count attack resolves the queries (tokens) with non-unique frequency values using the co-occurrence knowledge-matrix C_w about the dataset whereas our attack resolves such queries by exploiting the properties of relational databases (cf. Observation 1). This is a clear advantage over Cash et al.'s Count attack when it comes to attacking relational databases since it does not require any co-occurrence knowledge-matrix C_w as done by Cash et al. [7] and Islam et al. [18] but only $\mathcal{K}_{\mathcal{B}}$ if the attacker's goal is to recover the attribute names of queries and $\mathcal{K}_{\mathcal{F}}$ if the attacker's goal is to recover the value of each query whose attribute name is recovered.

3.1 Attribute Recovery Attack

We make use of the following simple observation about the joint frequency (or the co-occurrence count) of observed queries (tokens) sharing the same attribute name on searchable encrypted relational databases.

Observation 1. *The joint frequency (or the co-occurrence count) is zero for any two different queries (tokens) with the same attribute name that are observed from the access pattern leakage of an SSE scheme protecting a relational database.*

The observation should be clear from the fact that each relational database record has only one value for each column or attribute name. For example, let t_1 be the token corresponding to "$\mathsf{Sex : Male}$", t_2 be the token corresponding to "$\mathsf{Sex : Female}$", and t_3 be the token corresponding to "$\mathsf{Age : 18}$". Now, the joint frequency of t_1 and t_2 must be zero as there cannot be a relational database record whose "Sex" value is both "Male" and "Female". Also there is no guarantee that the joint frequency of t_1 and t_3 (or t_2 and t_3) is zero since their corresponding attribute names are different. More generally, Observation 1 might allow an attacker to answer the following question "Do the queries tokens t_i and t_j have the same attribute name ?". Here the attacker does not need any knowledge other than the observed access pattern leakage C_t. If the value $C_t[t_i, t_j]$ does not equal zero, then t_i and t_j definitely have different attribute names. Otherwise, the attacker cannot answer. Our attribute recovery attack is based on Proposition 1 which follows immediately from Observation 1 and the fact that the total frequency of all the domain values of an attribute a in a searchable encrypted relational database EDB equals the number of records in EDB. Going back to our previous example and assuming that the number of records in EDB is n, one can see that since t_1 and t_2 have the same attribute name and $|\mathsf{Sex}| = 2$ (i.e. the cardinality of "Sex") then the following equation will be satisfied: $|t_1| + |t_2| = n$. This gives an example that explains the following proposition.

Proposition 1. *Let $\boldsymbol{t} = \{t_1, \cdots, t_l\}$ be the query tokens set of observed queries where $l \le |\mathsf{W}|$. Let $|t_i|$ denote the result size of query token t_i. Let C_t be the observed co-occurrence token-matrix. Let n be number of records in a searchable encrypted relational database EDB. Let 'a' be an attribute name in the EDB whose cardinality $|a|$ is unique. Then there exists a subset $\boldsymbol{s} \subseteq \boldsymbol{t}$ where $\sum_{t_i \in \boldsymbol{s}} |t_i| = n$, $|\boldsymbol{s}| = |a|$, and $\forall t_i, t_j \in \boldsymbol{s}\,, C_t[t_i, t_j] = 0$ with success probability 1 if $l = |\mathsf{W}|$ and $\binom{|\mathsf{W}|-|a|}{l-|a|}/\binom{|\mathsf{W}|}{l}$ otherwise.*

Proof. The existence of a subset s with probability 1 when all possible queries are observed follows from Observation 1. We prove the success probability when $l < |\mathsf{W}|$. Assume that the queries were generated randomly without repetition of any query. Let X be a discrete random variable representing the number of appearances of a specific attribute 'a' after l queries. One can see that the probability of randomly choosing $X = k$ queries belonging to the attribute name 'a' from a set of l queries is equivalent to $\binom{|a|}{k}\binom{|\mathsf{W}|-|a|}{l-|a|}/\binom{|\mathsf{W}|}{l}$ since the discrete random variable X follows a hypergeometric distribution [1]. Now setting $k = |a|$ yields the success probability $\square$.

The above proposition can be used to develop an algorithm that distinguishes between observed queries (tokens). Such distinguisher can be mounted by a weak attacker (hence a strong attack) who observes only the access pattern leakage of queries and has no prior knowledge other than $\mathcal{K}_{\mathcal{B}}$ (i.e. number of records, attribute names and their cardinalties). 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 and classifies these tokens according to their attribute name into different sets where each set G_a represents the tokens whose attribute name is 'a'.

Algorithm 1 Attribute Recovery Attack

```
Require: $\mathcal{K}_{\mathcal{B}}$, 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 token-matrix $C_t$ 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 $C_t[t_i, t_j] = 0$.
 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 $j$th entry in the $i$th 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 $\sum_{t_u \in S} |t_u| = n$ & $C_t[t_u, t_v] = 0 \quad \forall t_u, t_v \in S$ 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.
```

Discussion about Algorithm 1. One way to look at Algorithm 1, is to consider the co-occurrence token-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 joint frequency (or co-occurrence count) value between the two connected nodes (i.e. A zero value for the joint frequency (or co-occurrence count) 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 token-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

[1] The hypergeometric probability distribution models the distribution of the number of green marbles that will be obtained in a sample without replacement of N marbles from a box containing K green marbles and $N - K$ non-green marbles. The probability of selecting k green marbles after l selections is $\binom{K}{k}\binom{N-K}{l-k}/\binom{N}{l}$.

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 joint frequency (or co-occurrence count) value between any two elements in the subset is zero. The k-Sum problem is a parameterized version of the 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 joint frequency (or co-occurrence count) 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 $O(st)$ [21]. 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 joint frequency (or co-occurrence count) condition is evaluated after finding each solution. This is practical when $O(st)$ is pseudo-polynomial which means that the number of records t is not large and 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 joint frequency (or co-occurrence count) filtering condition.

To appreciate what Algorithm 1 achieves, we note that Cash et al [7] showed, in one of their experiments about query recovery rate on the Enron dataset protected by an SSE scheme, that neither the IKK attack nor the Count attack succeed in recovering any query without assuming that the attacker has more than 50% of the complete knowledge about the dataset which is needed to compute the co-occurrence matrix-knowledge C_w and acquire the $\mathcal{K}_{\mathcal{F}}$ knowledge used in the Count attack. Now, assume that an attacker have observed enough queries on a searchable encrypted relational database, then Algorithm 1 can give us an answer to: "Have all possible values about attribute 'x' been queried ?". To answer such a question, an attacker needs to know the number of records n which could be possible if the relational database is static and not dynamic such as Archive databases. The attacker also needs to know $|x|$ which could also be possible as many relational database tables are standard such as the sensitive OpenEMR [25] relational databases. Employing Algorithm 1, under enough available queries, will return all the queries with the same attribute that have result sizes whose sum is equivalent to n. If there is only one attribute whose cardinality equals '$|x|$', then Algorithm 1 will yield one solution if all values of 'x' have been queried. The ability to answer the above question does break the query privacy meant to be provided by using SSE and confirms that the leakage resulting from protecting relational databases with SSE schemes is more than the leakage resulting from protecting unstructured data with SSE schemes.

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 domain or space values of a given attribute name. Moreover, with the access pattern leakage, the attacker knows the result set size (i.e. number of records holding the corresponding attribute-value pair for the query token) of each observed query. However, in order to recover the attribute values of observed queries (tokens) whose attribute names are recovered with Algorithm 1, without any prior knowledge such as those used in the Count attack $\mathcal{K}_{\mathcal{F}}$ and C_w, an attacker needs to know at least the rank-size or rank-frequency distribution [2] of the attribute values. Using rank-size distribution knowledge only instead of $\mathcal{K}_\mathcal{F}$ knowledge, an attacker can create a list L_a containing the attribute values of an attribute named a that is sorted according to their rank-size in descending order. Let L_q be a list containing the result-size of of each query such that $L_q[i]$ contains the result size of query t_i. Let $\mathsf{Sort}(L_q)$ be the list obtained after sorting L_q in descending order. Let $\mathsf{Find}(\mathsf{Sort}(L_q), L_q[i])$ be a function that gives the location corresponding to t_i in the sorted list. Then the value of the token t_i will be $L_a[\mathsf{Find}(\mathsf{Sort}(L_q), L_q[i])]$. If all the result sizes of queries in L_q are unique, then the above attack succeeds with probability one. Otherwise, there might be an error whenever we have a tie in the result sizes between two or more queries in L_q.

Using $\mathcal{K}_\mathcal{F}$ knowledge which is a very strong assumption compared to the rank-size distribution knowledge, an attacker can populate a list of lists data structure, say L_a, where $L_a[j]$ gives the value (or a list of values) of the attribute 'a' whose frequency value (values) equals j. Assuming, for example, that Algorithm 1 has recovered the attribute name of a token t_i to be 'a', then one can see that by adding the result-size of observed queries (tokens) to a dictionary D_q (i.e. $D_q[t_i]$ gives the result-size of query token t_i), then $L_a[D_q[t_i]]$ will be the attribute value (or the possible attribute values) of an observed query token t_i whose attribute name is a. If each list in L_a (i.e. $L_a[j]$) has size 1, then this process yields one value for the token t_i whose frequency matches the result-size of t_i, $D_q[t_i]$. Otherwise, it will return the list of all the possible values of the token t_i which appear $D_q[t_i]$ times over all the database records.

The two above procedures are standard frequency analysis attacks similar to the one described by Naveed et al. [24] for attacking columns of a relational database encrypted using deterministic encryption. However, they target searchable encrypted relational databases by firstly recovering the attribute name of a given query through Algorithm 1 and then secondly recovering its value.

Our second attack procedure employs Algorithm 1 to gather all the queries belonging to a specific attribute name and then uses $\mathcal{K}_\mathcal{F}$ knowledge to recover the value. This can be considerably improved by firstly using $\mathcal{K}_\mathcal{F}$ knowledge to recover the observed queries (tokens) whose result sizes are unique and secondly employing a variant of Algorithm 1 to recover the observed queries (tokens) whose result sizes are non-unique without resorting to any co-occurrence knowledge-matrix. One can see that the $\mathcal{K}_\mathcal{F}$ knowledge will speed up the attack as the queries with non-unique result size can be resolved with a subset sum problem that is easier to solve. The new improved attack procedure can be described as follows. We start by recovering the observed queries Q with unique result size using the attacker's $\mathcal{K}_\mathcal{F}$ knowledge. Then, we divide the recovered queries Q into subsets according to their attribute names. 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 $m_a = |a| - |G_a|$ (note that $|a|$ is known from $\mathcal{K}_\mathcal{B}$). 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 joint frequency with all the elements in G_a. Now we have three cases according to $|Q'_a|$. We treat them as follows.

1. If $|Q'_a| = m_a$ and the joint frequency between any two queries in Q'_a is zero, then all the queries in Q'_a probably have the same attribute name which is probably 'a' similar to the elements of G_a. However, there might be queries with attribute 'a' that have not yet been queried and thus 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 joint frequency with all the resolved and unresolved queries with attribute 'a'. In such case, if the sum of the queries' result sizes $\mathsf{totalFreq}(Q'_a)$ (i.e. $\sum_{t_i \in Q'_a} |t_i|$) is equal to n minus the sum of

[2] 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 $\mathcal{K}_\mathcal{F}$. For example, an attacker might know that in a certain country, from Census data, that the number of females exceeds the number of males without knowing the exact numbers.

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 name 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 $\mathcal{K}_\mathcal{F}$ knowledge will be able to match between the frequency of the attribute-value pairs in $\mathcal{K}_\mathcal{F}$ whose attribute is 'a' and the result sizes 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 joint frequency 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 $\mathcal{K}_\mathcal{F}$ knowledge. If there are 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 attribute-value pairs 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 $\mathcal{K}_\mathcal{F}$ knowledge 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 attribute-value pairs in $\mathcal{K}_\mathcal{F}$ knowledge and the result set size of the queries of Q'_a and thus recovers 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 result sizes.

Combined Count Attack and Attribute Attack. One can also combine the Count attack with our attribute recovery attack in order to break the ties unresolved by the Count attack through the co-occurrence knowledge-matrix C_w. However, this will only break ties between any two queries with different attribute names but similar result sizes. So the gain here really depends on the relational database under attack.

3.2 Experimental Results

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

Target Relational Database Table. We used the Adult dataset [22] to extract a relational database table in order 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 have discrete values and six are non-discrete values (variable or continuous). The Adult dataset has 498 distinct attribute-value pairs. However, most of these attribute-value pairs 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 relational databases are equivalent to record recovery attacks, we extract a relational database table from the Adult dataset by removing the capital-gain and capital-loss attributes [3]. The extracted relational database table contains only 287 distinct attribute-value pairs 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. Thus we have only 2 non-discrete attributes namely "age" and "hours-per-week" since "education-num" (which is equivalent to the discrete attribute "education") and salary "class" have fixed values 16 and 2 respectively.

Query Generation. Our extracted relational database table is transformed to a searchable encrypted relational database table using a standard single-keyword SSE scheme where a Bitmap encrypted index leaking the number of records n is used similar to the single-keyword Bitmap index scheme described in [29,18]. The queries were chosen randomly from the set of all available queries by a client and each query result set and its access pattern leakage were recorded by an honest-but-curious server. Using this observed-queries knowledge, our attacker (i.e. the honest-but-curious server) can compute the joint frequency (or co-occurrence count) value between any two queries and thus the whole co-occurrence token-matrix, C_t. In Exp. I, we generated all the possible queries (i.e. 287 queries) whereas different numbers of queries are generated in Exp. II, namely 100, 150, 200, 250, and 287 queries.

Attribute Recovery Attack (Exp. I). Here we assume that our attacker knows only the basic knowledge $\mathcal{K}_{\mathcal{B}}$ about the target data (i.e. Adult dataset) beside to the observed co-occurrence token-matrix C_t computed from the leakage of the used SSE scheme. Algorithm 1 is implemented using the known dynamic programming procedure to solve the subset sum problem in each set $L[\mathsf{ctr}]$. Since the number of records is $\approx 2^{15}$, the time complexity taken by Algorithm 1 will be $O(2^{15} \cdot |L[\mathsf{ctr}]|)$ 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 with unique cardinalities 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[4]. 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 joint frequency (or co-occurrence count) value with all the other values except one value. This makes 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 is 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 queries.

Attribute and Value Recovery Attack (Exp. II). In addition to the basic knowledge $\mathcal{K}_{\mathcal{B}}$ and the observed co-occurrence token-matrix computed from the SSE's leakage, our attacker have $\mathcal{K}_{\mathcal{F}}$ knowledge (i.e. the number of occurrences of each attribute-value pair over all the original records in the relational database) but does not know the co-occurrence knowledge-matrix C_w required by both the IKK attack and the Count attack. So the attacker first recovers queries with unique result size using his $\mathcal{K}_{\mathcal{F}}$ knowledge and then applies a variant of Algorithm 1 (i.e. the three cases) outlined in Appendix A to resolve queries with non-unique result sizes. Figure 1 shows that the attacker managed to recover more queries with our attack compared to a standard unique-frequency attack that recovers only queries with unique result sizes. However, the Count

[3] 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 [20] is preferable.

[4] 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.

attack is clearly more effective than our attack since it assumes that the attacker has co-occurrence knowledge-matrix C_w about the relational datasbase.

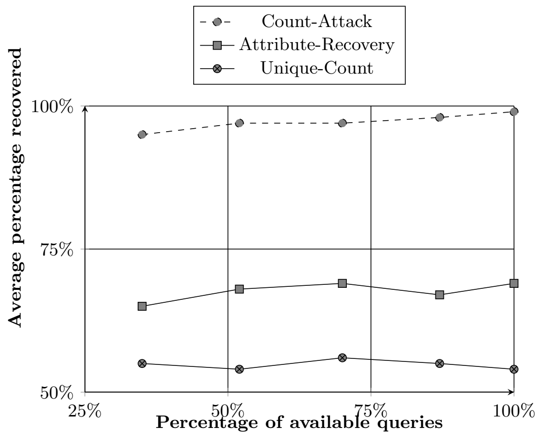

Fig. 1: Shows the query recovery of available queries with three different attacks, i.e. Unique-Count, Attribute-Recovery and Count-Attack.

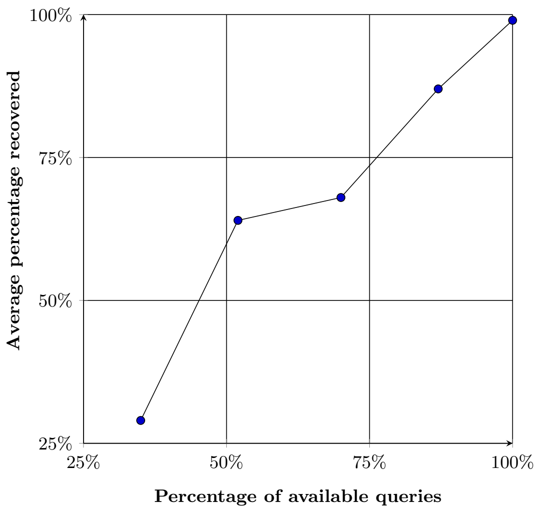

Fig. 2: Shows the average percentage of records recovered of the entire dataset given different numbers of available queries.

Record Recovery via Count Attack (Exp. II). The query recovery during our second experiment using the Count attack leads 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 databases via SSE schemes could be a risk.

4 Record-Injection Attacks

Zhang et al.'s non-adaptive file-injection binary attack [30] 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 whose values are variables (not discrete) with a big range. However, if the attacker is concerned about a small subset of attribute-value pairs. Then he 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 a searchable encrypted relational database. Our focus here is on non-adaptive record injections as adaptive injection attacks need background knowledge about the target encrypted database and they can be prevented by using a forward secure SSE scheme. Similar to Zhang et al. [30], 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 attribute-value pair space $|\mathsf{W}|$ or a target subset of attribute-value pairs 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 attribute-value pairs 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 records regarding any attribute-value pair in S will yield a binary string with Hamming weight one where the location of the active bit i indicates that the attribute value 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 results 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.

Proposition 2. *Let S be a subset of attributes of size t. Let all the attributes of S have the same cardinality 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 attribute-value pairs. In order for such number of records to yield a unique search result on the injected records for all the $d \cdot t$ attribute value pairs, we need to make sure that each attribute-value pair 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 attribute-value pairs and the l-bit search result on the injected records. Replacing each occurrence of an attribute-value pair 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 attribute-value pair on the injected records. Consequently, we get a bijective mapping. Now, to enable the unique encoding of all possible $d \cdot t$ attribute-value pairs, the number of l-bit strings $\binom{l}{l/d}$ should be $\geq d \cdot t$ since otherwise all the possible attribute-value pairs 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 attribute-value pairs.

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 a whose values are not discrete, then $a_{\max}$ might be large which makes it difficult to recover all the values of the the non-discrete attribute a. However, one can reduce the number of injected records by predicting a small and dominant subset of all the possible values of the non-discrete attribute a.

Discussion. It is clear that once an attacker is able to inject his own records then record-injection will lead to 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. [30]. 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 attribute-value pair has a unique number of appearances in all the injected records.

The file-injection countermeasure proposed by Zhang et al. [30] which restricts the number of keywords per document to a certain threshold T (e.g. $T \ll |\mathsf{W}|/2$) 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 databases against adaptive injection attacks (Note that the above described attacks are non-adaptive attacks but an adaptive injection attacks similar to the one proposed in [30] can easily be realized). Note that a forward secure SSE scheme does not provide protection against non-adaptive injection attacks. So additional system-based or application-based countermeasures protecting the update process of the the encrypted database from being attacked by an active adversary performing record-injection attacks need to be employed.

5 Countermeasures Against Attacks on SSE Schemes

Countermeasures against inference attacks must be used in order to reduce the effectiveness of the inference attacks demonstrated above. A well known technique is *padding* which is proposed in [18,7] as a potential countermeasure to reduce the effectiveness of inference attacks. Basically, during the setup of the encrypted database, the client adds dummy record (or document) IDs to each attribute-value pair (or keyword) in the index order to hide the actual frequency of the keyword. Also, the client adds an encrypted dummy record (or document) corresponding to each dummy ID added in the index. Later, during search, the client filters out the dummy records (or 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 Count attack. It basically does not depend on resolving queries with unique frequency which will not exist in a padded SSE scheme but it initially guesses these queries. 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 records (or documents) causing false co-occurrence count values. 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 Attribute Recovery. The above padding countermeasure which hides the actual result 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 cardinalities. 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 equivalent cardinalities. As shown in Exp. I, attributes with equivalent cardinalities 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 makes 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 more storage and time efficiency of SSE schemes.

Countermeasure Against the Count Attack. Beside reducing the effectiveness of the actual query result size by padding, one might think of reducing the effectiveness of the observed query co-occurrence token-matrix C_t by forcing the observed joint frequency between some queries 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 $C_t[q_i, q_j]$ will be zero when each query is executed in only one fragment and the fragments are not allowed to interact with each other to evaluate any query. Now an equality match or a window equality match with the background knowledge-matrix $C_w[s_i, t_j]$ 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 Adult dataset. The privacy constraint $c_1 = \{\text{sex,occupation}\}$ 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 $c_1 = \{\text{Name, DOB}\}$, $c_2 = \{\text{Name, Disease}\}$, $c_3 = \{\text{Name, ZIP}\}$, $c_4 = \{\text{Name, MD}\}$, $c_5 = \{\text{DOB, ZIP, Disease}\}$ and $c_6 = \{\text{DOB, ZIP, MD}\}$. 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 $\mathcal{F} = \{\mathcal{F}_1 = \{\text{Name}\}, \mathcal{F}_2 = \{\text{DoB,ZIP}\}, \mathcal{F}_3 = \{\text{Disease,MD}\}\}$. 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 inference 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 to recover the original record before fragmentation. This is because each fragment is shuffled differently and each fragment's record has a different record ID for the same original record.

Security Gain. 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 leaking the attribute of the queries in each fragment. We performed an experiment to show the amount of security gained when we employ vertical fragmentation via privacy constraints. In this experiment, the relational database table extracted from the Adult dataset is split into three different fragments. All fragments are protected using the same SSE scheme (without any padding countermeasures) 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 three fragments since each fragment is shuffled differently and attribute-value pairs belonging to the same record are fragmented into three 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 Gain. The drawback with the fragmentation approach is the additional computational work in the use case of a multi-keyword query whose keywords or attribute-value pairs exist in different fragments. But this can be improved using 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 databases. Our attack targets small relational databases and requires the access pattern leakage inherent in all SSE schemes besides basic information about the target relational database. Our attack can be easily prevented by requiring some or all attributes to have the same cardinality which can be by adding dummy values to each attribute. Moreover, we showed that record-injection attacks also pose a real threat for searchable encrypted relational databases. Furthermore, we proposed the use of privacy constraints together with padding on top of any SSE scheme in order to reduce the effectiveness of the frequency attacks proposed in [18,7]. Clearly, the privacy constraints would

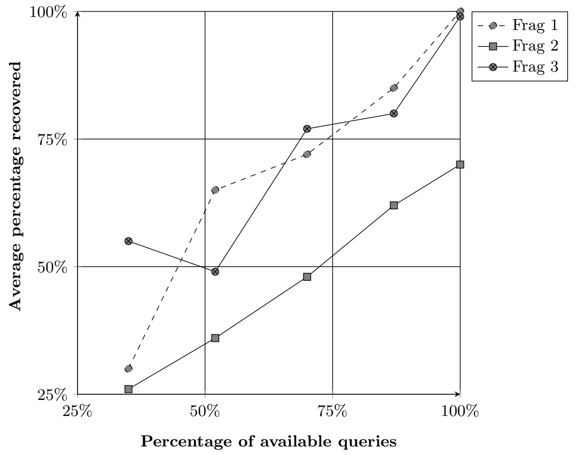

Fig. 3: Shows the result of the average percentage recovered per record for the fragmented dataset. Frag. 1 includes "sex, marital-status and relationship", which totals at approximately 5% of the attribute-value pair space. Frag. 2 includes "age, work-class, education, education-num, occupation, hours-per-week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

affect the performance of SSE 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. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR 2005.*
2. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.*
3. N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. *Journal of the ACM (JACM)*, 2005.
4. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference*, 2007.
5. A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Crypto 2011.*
6. R. Bost. $\sigma o\varphi o\varsigma$: Forward secure searchable encryption. In *CCS 2016.*
7. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS 2015.*
8. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. *IACR Cryptology ePrint Archive*, 2014.
9. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013.*

10. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010*.
11. V. Ciriani, S. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *ESORICS 2007*.
12. V. Ciriani, S. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *ESORICS 2009*.
13. V. Ciriani, S. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Transactions on Information and System Security (TISSEC)*, 2010.
14. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, 2006.
15. J. Erickson. Lower bounds for linear satisfiability problems. In *SODA 1995*.
16. O. Gold and M. Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. *CoRR*, abs/1512.05279, 2015.
17. IARPA. Poster about protecting privacy and civil liberties. `https://www.iarpa.gov/images/files/programs/spar/09-SPAR_final_v21.pdf`.
18. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*.
19. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, 2013.
20. G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *CCS*, 2016.
21. J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.
22. R. Kohavi and B. Becker. Adult data set. https://archive.ics.uci.edu/ml/machine-learning-databases/adult/, 1996. [Last Accessed 2016-10-18].
23. K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, 2012.
24. M. Naveed, S. Kamara, and C. Wright. Inference attacks on property-preserving encrypted databases. In *CCS 2015*.
25. OpenEMR. `http://www.open-emr.org/`. Accessed: March 2017.
26. R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *ACM Symposium on Operating Systems Principles 2011*.
27. IARPA. Security and Privacy Assurance Research (SPAR) Program BAA, 2011. `https://www.iarpa.gov/index.php/research-programs/spar`.
28. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Security and Privacy. S&P 2000.*
29. P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Workshop on Secure Data Management*, 2010.
30. Y. Zhang, J. Katz, and C. 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`.