Practical Attacks on Relational Databases Protected via Searchable Encryption

Mohamed Ahmed Abdelraheem[1] *, Tobias Andersson[2], Christian Gehrmann[3] and Cornelius Glackin[1]

[1] Intelligent Voice Ltd., London, UK
[2] RISE SICS, Lund, Sweden
[3] Lund University, Sweden

Abstract. Searchable symmetric encryption (SSE) schemes are commonly proposed to enable search in a protected unstructured documents such as email archives or any set of sensitive text files. However, some SSE schemes have been recently proposed in order to protect relational databases. Most of the previous attacks on SSE schemes have only targeted its common use case, protecting unstructured data. In this work, we propose a new inference attack on relational databases protected via SSE schemes. Our inference attack enables a passive adversary with only basic knowledge about the meta-data information of the target relational database to recover the attribute names of some observed queries. This violates query privacy since the attribute name of a query is secret.

1 Introduction

Searchable symmetric encryption (SSE) schemes provide one of the practical solutions for searching on encrypted data. It was firstly proposed by Song et al. in [26] and later improved by Curtmola et al.'s [10]. Based on Curtmola et al.'s security model many SSE schemes were proposed such as [9,8,7]. The efficiency of SSE schemes comes at the cost of some leakage that might make them vulnerable to inference attacks [15,6].

Traditionally, SSE schemes are designed to protect a set of unstructured documents (e.g. email archives or a backup or any set of sensitive text files). However, recently Cash et al. proposed an elegant scheme [8] that targets relational databases by increasing the functionality through supporting a large subset of Boolean queries. Their scheme also has good performance as it achieves a query speed that is comparable to the query speed under the unprotected MySQL (release 5.5) but with a storage cost of up to seven times the unencrypted data [8,14].

The security of SSE schemes has been well studied against inference attacks [15,6] in their traditional use case scenario where document datasets such as email archives are protected. More recently, the work in [1] presented the first security analysis of searchable encrypted relational databases (i.e. relational

* Most of this work was done while the author was a postdoc at RISE SICS in Sweden.

databases protected via SSE) by proposing an inference attack, called *Relational-Count*, exploiting the structural properties of relational databases and using only knowledge about the frequency distribution of the attribute-value pairs which is much less than the prior knowledge (i.e. joint frequency of attribute-value pairs which is equivalent to knowledge of the whole plaintext database) required by the *Count* attack [6].

In this work, we further improve the analysis of searchable encrypted relational databases and show that there is an extra leakage that could reveal the attribute name (also known as column name) of all the queries (i.e. encrypted attribute-value pairs) belonging to a specific database column.

Henceforth, we will use the words "query" and "query token" interchangeably to mean an encrypted attribute-value pair that has been queried (or an encrypted keyword in the context of unstructured databases). Thus, query recovery means finding the actual plaintext of the query which is an attribute-value pair (or a keyword). Also, relational database tables protected via SSE schemes will either be called SSE-protected databases or searchable encrypted relational databases. Finally, the words "attribute" and "column" will also be used interchangeably throughout the paper.

Our Contribution. The contribution of this paper is twofold. *First*, we propose a new attack on searchable encrypted relational databases. Our attack recovers the attribute names or column names of queries and thus we call it the *Attribute-Name recovery* attack. Our attack breaks the query privacy by recovering the secret attribute names of some of the observed queries. We propose two simple algorithms (one is deterministic and the other is heuristic), that take as input the set of issued queries and divide it into different subsets where each subset represents all the values belonging to a specific column. Table 2 shows the number of columns recovered by our attack on three different real world databases protected by an SSE scheme. It is apparent from Table 2 that for the Adult [19] and the Bank [25] database tables which have 14/32561 and 17/4521 columns/records respectively, our attack recovers almost all the columns and thus completely breaks the privacy of queries. However, regarding larger database tables such as the Census which has 40/299285 columns/records, we are able to recover slightly more than half of the columns. Most notably, our attack works using only the meta-data information (i.e. column names and their cardinalities where the cardinality of a column is defined as the number of unique elements in the column) and the number of records about the protected relational database table under attack. This lesser amount of required knowledge makes our Attribute-Name recovery attack more applicable in practice than all previous attacks on SSE schemes [15,6,1] as they require at least prior knowledge about the frequency distribution of the keywords or attribute-value pairs in the target database.

Second, we combine our Attribute-Name recovery attack with the *Relational-Count* attack [1]. This gives us a second attack that recovers the values of queries but requires knowledge about the frequency distribution of the attribute-value pairs in the target database. The results about the number of recovered queries in different SSE-protected relational databases are shown in Table 3.

Related Work. Query recovery attacks exploiting the access pattern leakage of SSE schemes were proposed by Islam et al. [15] and later improved by Cash et al. [6]. The recent inference attack on searchable encrypted relational databases by Abdelraheem et al. [1] exploits the properties of relational databases. This reduces the attacker's knowledge from full database knowledge required by previous inference attacks [15,6] to only partial knowledge represented by the frequency distribution of the attribute-value pairs in the target database. Unlike Abdelraheem et al.'s *Relational-Count* attack [1] which still requires the frequency distribution of the attribute-value pairs in the target database, our first proposed attack recovers the attribute names of queries by requiring only the attacker's knowledge about the meta-data information of the target protected database. Moreover, as shown in Table 3, our second proposed attack recovers more queries than those recovered by Abdelraheem et al.'s *Relational-Count* attack [1].

Naveed et al. [22] 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 or more precisely one column vector. While the approach seems similar, their column finder procedure is significantly different than our attribute recovery attack. 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. Their approach 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 [17] proposed at CCS 2016 target any secure database systems supporting range queries but leaking the access pattern without any prior knowledge about the database 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. There are only two things in common between our attack and theirs,

and that is the requirement of knowing the number of records and the query result size (or communication volume as defined in [17]).

Organization of the paper. Section 2 gives a brief overview of SSE schemes and inference attacks. 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 present detailed experimental results demonstrating our attacks and then discuss some countermeasures. Section 5 concludes the paper.

2 Background about SSE Schemes

Definition. An SSE scheme takes as inputs a plaintext database index together with the client's secret keys and outputs an encrypted index where each keyword w in the plaintext index is transformed into a token t using a deterministic encryption algorithm and its corresponding documents' identifiers are encrypted using a randomized encryption algorithm. An SSE scheme encrypts the original documents by employing a randomized encryption algorithm using the client's secret keys and stores the encrypted documents in an encrypted database indexed by the document identifiers. Both the encrypted index and the encrypted documents database are sent to the cloud server. To search for for a keyword w, the client issues a query by generating its token t using its secret keys and sends it to the server. The server responds by sending the corresponding encrypted documents from the encrypted documents database to the client.

Our focus will be on analyzing the security of relational databases protected by SSE schemes that are adaptively secure as defined by Curtmola et al. [10]. The use of SSE schemes to protect SSE schemes was firstly proposed by Cash el al. [8,7] where the database records are considered as documents and the attribute-value pairs are considered as keywords (e.g. each keyword is represented as $w_i = (attribute_i, v_i)$ where $attribute_i$ is the attribute name and v_i is its value).

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, $\mathsf{DB}(\mathsf{w_i})$, 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. [10] 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 and the encrypted records.

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 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 [15]. 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 attribute-value pairs (or two keywords) $w_i, w_j \in \mathsf{W}$. It also assumes knowledge about the plaintext of a subset of queries issued by the client.

A similar inference attack has been proposed recently by Cash et al. is called the *Count* attack [6]. The *Count* attack assumes the attacker's knowledge about the number of occurrences of each keyword/attribute-value over all the documents/records (i.e.$|\mathsf{DB}(w_i)|$ where $w_i \in \mathsf{W}$ and DB is the original plaintext dataset). This is also called keyword frequency knowledge and we denote this knowledge by $\mathcal{K}_{\mathcal{F}}$. Similar to the IKK attack, it also assumes knowledge about the joint frequency (or co-occurrence count) of any two plaintext attribute-value pairs (or two keywords) $w_i, w_j \in \mathsf{W}$. 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 require a complete knowledge about the plaintext dataset under attack in order to form the co-occurrence knowledge-matrix. Both attacks exploit the access pattern leakage inherent in SSE schemes, in this case enabling the computation of the result size of any observed query and also the observed joint frequency of any two observed queries. In the relational database setting, this is equal to the size of the set resulting from the intersection between the result sets of the two queries. A joint *co-occurrence query-matrix*, C_t, is then formed and compared to the *co-occurrence knowledge-matrix*, C_w in both attacks.

The recent *Relational-Count* [1] inference attack also performs query recovery but on searchable encrypted relational databases. Its query recovery rate is less than the *Count* attack but it is more practical than the *Count* attack since it requires only the knowledge of the frequency distribution of the plaintext attribute-value pairs in the target database rather than the joint frequency knowledge required by the *Count* attack. The difference lies in the fact that the *Relational-Count* attack uses Observation 1 (See Section 3) to discard the wrong candidates in the list of possible candidates for a query with non-unique result size in contrast to the *Count* attack which uses prior joint frequency knowledge to discard wrong candidates from the same list.

3 Attacking Searchable Encrypted Relational Databases

As demonstrated in the recent *Relational-Count* inference attack [1], the structure of relational databases does enable an attack on searchable encrypted relational databases without resorting to the co-occurrence knowledge-matrix C_w about the relational database under attack. In the following, we describe our Attribute-Name recovery attack. Our attack works under the assumption that enough queries have been observed and that the attacker knows the meta-data

information about the target database (e.g. column names $\mathcal{A}$ and the cardinality of each column which is equivalent to the number of unique values in the column).

Note that the meta-data information represents the least possible prior knowledge that could be acquired by an attacker. 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 could be guessed or acquired from public data. For example, if an honest-but-curious server holds an encrypted medical data, then an attacker can easily acquire the columns' or attributes' names of the protected data by referring to publicly-available information such as the meta-data about database tables used in standard medical software applications such as OpenEMR [23][4].

In addition to this basic knowledge, our attack assumes also the attacker's knowledge of the number of records in the target database which can be dynamic depending on the protected database under attack. The number of records could be gained either through a guess-and-determine process especially for small or medium sized databases or through a leakage of the SSE scheme under attack which is the case in some notable SSE schemes such as [9,27,20,16]. Moreover, our attack could also recover the attribute value of a given attribute name 'a' under the assumption that the attacker knows $\mathcal{K}_{\mathcal{F}}$.

3.1 Attribute-Name Recovery Attacks (First Attack)

We make use of the following simple observation, proposed in [1], about the joint frequency (or the co-occurrence count) of observed queries sharing the same attribute name on searchable encrypted relational databases.

Observation 1. *The observed joint frequency (or observed co-occurrence count) between any two different queries is non-zero only when their corresponding attribute names are different.*

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 query token corresponding to the plaintext attribute-value pair "Sex : Male", t_2 be the query token corresponding to the plaintext attribute-value pair "Sex : Female", and t_3 be the query token corresponding to the plaintext attribute-value pair "Age : 18". Now, the observed 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 observed 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 t_i and t_j have the same attribute name ?". Here the attacker does not need any knowledge other than the observed access pattern

[4] OpenEMR is a well known open source medical software supporting Electronic Medical Records (EMR)

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 first proposed attack, the attribute-name recovery attack, is based on Observation 2 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 observation.

Observation 2. *Let $\boldsymbol{t} = \{t_1, \cdots, t_l\}$ be the set of observed queries where $l \leq |\mathsf{W}|$. Let $|t_i|$ denote the result size of query token t_i. Let C_t be the observed co-occurrence query-matrix. Let n be number of records in a searchable encrypted relational database* EDB*. Let 'a' be an attribute name in the* EDB *with cardinality $|a|$. Then there exists a subset* [5] *$\boldsymbol{s} \subseteq \boldsymbol{t}$ where $\sum_{t_i \in \boldsymbol{s}} |t_i| = n$, $|\boldsymbol{s}| = |a|$, and $\forall t_i, t_j \in \boldsymbol{s}\,, C_t[t_i, t_j] = 0$ when $l = |W|$.*

Observation 2 can be used to develop an algorithm that distinguishes between observed queries. Such a distinguisher algorithm 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, column names and their cardinalities).

Description of Algorithm 1. Algorithm 1 takes as input 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 names into different sets where each set G_a belongs to one attribute name $a = (a_i)$ or multiple attributes names $a = (a_i, a_j)$ sharing the same cardinality $|a_i|$. Note that the pseudo-code of Algorithm 1 provides only an overview about the steps and does not represent a description of our implementation. In the following, we discuss how to efficiently implement Algorithm 1.

Step 7 and Step 8 can give us an idea about the time complexity of Algorithm 1 since they 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 observed 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 [3,11,13].

However, employing the observed 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

[5] If the cardinality $|a|$ is not unique, then k subsets will exist where k is the number of attributes whose cardinalities are equal to $|a|$.

Algorithm 1 Attribute Recovery Attack

Require: $\mathcal{K}_{\mathcal{B}}$ and observed query tokens $\mathbf{t} = \{t_1, ..., t_l\}$ and their results. $|a| \equiv$ cardinality of $a \in \mathcal{A}$ where a can represent a single attribute name or multiple attribute names sharing the same cardinality $|a|$. $|t_i| \equiv$ result size of query token t_i.

Ensure: Recover the attribute name of observed queries.

1: Set $R = \{\}$. Compute the co-occurrence query-matrix C_t from query tokens $\mathbf{t}$.
2: For each t_i, create a list Q_i holding t_i in its head (i.e. $Q_i[1] = t_i$) and all other query 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. Add all the sorted lists Q_i's to a lists' container L where $L[i, j]$ is the jth query in the ith list $L[i]$.
4: Set $G_a \leftarrow \{\}$ for each attribute a and $\mathsf{ctr} = 1$.
5: **while** $\mathsf{ctr} \leq l$ **AND** $\mathcal{A} \neq \emptyset$ **do**
6: $\quad L[\mathsf{ctr}] \leftarrow L[\mathsf{ctr}] \backslash R$
7: $\quad$ **for all** $S \subseteq L[\mathsf{ctr}]$ **where** $L[\mathsf{ctr}, 1] \in S$ **do**
8: $\quad\quad$ **if** $\sum_{t_u \in S} |t_u| = n$ **and** $C_t[t_u, t_v] = 0 \ , \forall t_u, t_v \in S$ **then**
9: $\quad\quad\quad$ **for all** $a \in \mathcal{A}$ **do**
10: $\quad\quad\quad\quad$ **if** $|S| = |a|$ **then**
11: $\quad\quad\quad\quad\quad G_a \leftarrow G_a \cup S$
12: $\quad\quad\quad\quad\quad R \leftarrow R \cup G_a$
13: $\quad\quad\quad\quad\quad \mathcal{A} \leftarrow \mathcal{A} \backslash a$
14: $\quad\quad\quad\quad\quad$ **print**("Possible solution for cardinality ",$|a|$, " is ",G_a)
15: $\quad\quad\quad\quad\quad$ **goto** Step 16
16: $\quad \mathsf{ctr} = \mathsf{ctr} + 1$
17: **return** R.

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 $O(st)$ [18].

Another way to look at Algorithm 1 is to consider the co-occurrence query-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 observed 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 query-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. We have implemented Step 8 in Algorithm 1 using a greedy solution to the independent set problem where a node with the maximum degree is included in the solution and all its neighbors

are discarded (See Section 4). Appendix A gives an example where the queries are represented in a graph and the problem of finding the attribute name of each query is equivalent to finding solutions to the independent set problem of a given graph.

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 target database table 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.

However, Algorithm 1 will not be practical to apply when the constructed lists are large. Therefore, we propose Algorithm 2, which is an efficient heuristic algorithm that can find valid solutions faster. Algorithm 2 takes as input the constructed lists $L[\mathsf{ctr}]$, the number of records, and the observed query-matrix. The algorithm is based on the observation that "*the intersection between lists whose heads belong to a similar attribute name gives a list containing queries whose result sizes add up to the number of records*". The problem here lies in identifying the lists whose heads belong to the same attribute name. Our heuristic approach allows the intersection between lists whose heads are queries that are disjoint. Its time complexity is $O(|W|^2)$ where $|W|$ is the number of attribute-value pairs which is equivalent also the number of lists $L[i]$.

Unlike Algorithm 1 which needs to find all the solutions of a subset sum problem (see Step 8 in Algorithm 1), the heuristic approach only needs to find the intersection (see Step 8 in Algorithm 2) between two sorted sets at each iteration which takes $O(m_1 \log(m_2))$ where m_1 is the size of the small list and m_2 is the size of the larger. For each list $L[i]$, it outputs a possible solution $\mathsf{R}[i]$. Note that a correct solution $\mathsf{R}[i]$ is more likely to appear and to be confirmed many times but not more than $|\mathsf{R}[i]|$ times. No doubt false positives will appear especially for columns with small cardinalities, even if the number of confirmations for a solution $\mathsf{R}[i]$, $C[\mathsf{R}[i]]$, is exactly $|\mathsf{R}[i]|$. The order of lists $L[i]$ affects the results of the intersections, so the algorithm might not succeed in finding the right solution. So further work is required to study the success rate of our heuristic approach.

However, experiments show that such false positive solutions are discarded by Step 17. As long as there are no intersections between solutions, any number of confirmations more than one will give us confidence that our solution is correct. In our experiments, for small cardinalities ≤ 15, we require that the number of confirmations to be exactly equivalent to the number of confirmations. But for other cardinalities, we require that the number of confirmations ≥ 10.

Experiments show that Algorithm 2 managed to recover many attributes in large databases such as the Census database [21] while Algorithm 1 succeeded

in recovering almost all the columns (12/14) in the Adult database [19] which is a small database with only 498 attribute-value pairs.

Algorithm 1 and Algorithm 2 break query privacy after observing enough queries using a less amount of required knowledge. To realize their effect, assume that an attacker has 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 as noted above. The attacker also needs to know the cardinality of x, $|x|$, which could also be possible as many relational database tables are standard such as the sensitive OpenEMR [23] relational databases. Employing Algorithm 1, after observing enough queries, might 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 the recent results [1] that the leakage resulting from protecting relational databases with SSE schemes is more than the leakage resulting from protecting unstructured data with SSE schemes.

Algorithm 2 Heuristic Approach

```
1: procedure HEURISTIC(L, C_t, n, t = {t_1, ..., t_l}, A)
2:     R = {}
3:     C = {}                          ▷ Hash table to map similar entries in R to one entry
4:     while L is decreasing do
5:         for all L[i] do
6:             R[i] ← L[i]
7:             for all L[j] where L[j,1] ∈ L[i] do
8:                 S ← R[i] ∩ L[j]
9:                 if Σ_{t_u ∈ S} |t_u| ≥ n then
10:                    R[i] ← S
11:            if Σ_{t_u ∈ R[i]} |t_u| = n  &  C_t[t_u, t_v] = 0 , ∀t_u, t_v ∈ R[i]  then
12:                R[i] ← Sort(R[i])                    ▷ Sort according to query no.
13:                if R[i] ∈ C then
14:                    C[R[i]] ← C[R[i]] + 1
15:                else
16:                    C[R[i]] ← 0
17:        for all R[i] where |R[i]| ∈ |A| & (R[i] ∩ R\R[i] = φ) do
18:            if  C[R[i]] ≈ |R[i]| then
19:                print(R[i], " holds queries of an attribute with card. ",|R[i]|)
20:                for all L[j] do
21:                    if L[j,1] ∈ R[i] then
22:                        L ← L\L[j]
23:                    else
24:                        L[j] ← L[j]\R[i]
```

3.2 Recovering Attribute Values (Second Attack)

Algorithm 1 and Algorithm 2 might enable an attacker to recover the attribute names of some queries without any knowledge except the basic knowledge, $\mathcal{K}_{\mathcal{B}}$. Now with $\mathcal{K}_{\mathcal{B}}$, 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 whose attribute names are recovered with Algorithm 1, without any prior joint frequency 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 of the attribute values. Note that 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.

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 token 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 query 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 query token t_i to be 'a', then one can see that by adding the result-size of observed queries 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 query 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 query 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. [22] for attacking columns of a relational database encrypted using deterministic encryption.

However, our attacks target searchable encrypted relational databases by firstly recovering the attribute name of a given query using only the meta-data information about the databases and then secondly recovering its value when

background knowledge about the databases (i.e. the distribution of the attribute-value pairs) under attack is known. Moreover, rather than firstly recovering all the attribute names of the issued queries and then secondly recovering all the values using the attacker's frequency knowledge, we can use the frequency knowledge and use our attribute-name recovery attack and at the same time apply the *Relational-Count* attack proposed by Abdelraheem et al. [1]. Combining the *Attribute-Name Recovery* attack and the *Relational-Count attack* allows us to recover more queries. This combination represents our second proposed attack. See the graph on the right in Fig. 1 at Appendix A to see the effect of our second attack when combined with our first attack. In Section 4, we give some experimental results on the Bank database [25] where this combination significantly increases the number of recovered queries and closes the security gap between deterministic encryption and searchable encryption schemes.

4 Experimental Results and Countermeasures

In this section, we show the practical viability and threat posed by our first proposed attack (attribute-name recovery) as well as our second proposed attack which recovers the actual plaintext values of the queries.

Our first attack is represented by Algorithm 1 (subset sum approach or Independent set approach) and Algorithm 2. We also combine both Algorithm 2 and Algorithm 1 (subset sum approach) by using Algorithm 2 first and then using the reduced and unresolved lists produced by Algorithm 2 as an input for Algorithm 1 (subset sum approach) in order to tackle the unresolved lists with small sizes. Our attack takes as input the access pattern of the observed queries (i.e. access pattern = record IDs and result size), which enables the attacker to compute the observed co-occurrence matrix C_t. It also takes as input the basic background information represented in *only* the meta-data information about the relational database under attack (i.e. $\mathcal{K}_{\mathcal{B}}$). Our goal is to distinguish between queries belonging to different attribute names and gather all queries belonging to the same attribute in a single unit column. This will reduce that the security offered by SSE and make it close to the security offered by deterministic encryption when enough queries are issued which will enable an attacker with background knowledge about the actual database or its distribution to recover all queries and thus all attribute values of the target database as demonstrated by Naveed et al. [22] and discussed above in Section 3.2.

We tested our attacks against the three relational database tables used in [1], namely, the Adult [19], the Bank [25], and the Census [21]. The implementation of our attacks can be accessed from the link https://goo.gl/WxmSU4. Before describing our experimental results, we give a short description about these three databases.

Relational Databases. The three relational database tables are publicly available online at the UCI Machine Learning Repository [12]: (1) Adult [19]: consists of 32561 rows (records), 14 columns (attributes), and has 498 distinct attribute-value pairs. (2) Bank [25]: consists of 4521 rows, 17 columns (attributes), and

has 3720 distinct attribute-value pairs. (3) Census [21]: consists of 299285 rows, 40 columns (attributes), and has 4001 distinct attribute-value pairs.

Query Generation. Using a single-keyword Bitmap-based SSE scheme similar to the ones described in [27,15], each target relational database is transformed into a separate searchable encryption relational database where all possible queries within it are issued. The queries were chosen randomly from the set of all available queries (498 queries for the Adult database, 3720 queries for the bank database and 3993 queries for the Census database) 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 query-matrix, C_t.

Sizes of the Sorted Lists. Both Algorithm 1 and Algorithm 2 depend on the lists $L[\mathsf{ctr}]$ whose sizes determine the time complexity of both algorithms. The sizes of the lists $L[\mathsf{ctr}]$ vary depending on the database under attack and the number of issued queries. For example, Table 1 shows that in the Adult database when all the 498 queries are issued, the smallest list has size 13 and the largest list has size 485 whereas in the Bank database when all the 3720 queries are issued, the smallest list has size 37 and the largest list has size 3704 and in the Census database when all the 4001 queries are issued, the smallest list has size 11 and the largest list has size 3962.

4.1 Experimental Results of our first attack

Experiments with Algorithm 1. Algorithm 1 is implemented using two approaches. The first approach is the known dynamic programming with a backtracking procedure to solve the subset sum problem in each list $L[\mathsf{ctr}]$ and the second approach is a greedy algorithm that solves the independent set problem. Table 2 shows the results of using the two approaches of implementing Algorithm 1 on the above three databases. The results show that the subset sum implementation is more successful than the independent set implementation when the database table has a small number of attribute-value pairs such as the Adult database which has only 498 attribute-value pairs and thus at most 498 queries. So regarding the Adult database, all the columns or attribute names whose cardinalities are unique have been recovered successfully. However, columns with the same cardinality such as the "sex" and salary "class", where each has two values, have been distinguished from the other attributes but one can not tell which of the two 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[6]. However, all their values were in one list ranked at position $\mathsf{ctr} = 13$ 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

[6] There is one-to-one correspondence between education and education-num.

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. However, the set of all queries belonging to "education" and "education-num" will be identified and separated from the other queries but not from each other. In such scenarios Algorithm 1 fails completely to recover the attribute name of a class of queries.

Table 1: *Sizes of the Sorted Lists (increasing order) in each of the three databases (i.e. Adult, Bank and Census) when all the possible queries are issued in each protected database. The 1st list (L[1]) for the Adult database has size 13, the 2nd list (L[2]) has size 28 and so on.* ***last*** *refers to the last list which corresponds to the 498th list in the Adult database, the 3720th list in the Bank database and the 4001th list in the Census database.*

Data Set	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**...**	**last**
Adult	13	28	41	45	52	67	89	89	93	118	118	119	129	135	136	...	485
Bank	37	288	374	662	662	662	770	852	954	1184	1465	1518	1812	1878	1963	...	3704
Census	11	25	25	30	34	55	107	110	111	127	127	152	157	197	198	...	3962

Regarding the Census and the Bank databases, the greedy approach of the independent set implementation of Algorithm 1 is more effective as it recovers more columns. Note that the subset sum approach is deterministic but it takes too long time to resolve a single list $L[\mathsf{ctr}]$ with a large size. For example, we are only able to resolve the first three lists in the Bank database using the subset sum implementation but we cannot resolve the 4th ranked list in the Bank database whose size is 662 and all the other lists corresponding to the Bank database in a reasonable time using the subset sum implementation. However, when we use the greedy approach for the independent set (note that this approach is heuristic so we check the intersections between solutions similar to Step 17 in Algorithm 2 to discard false positives), we are able to recover 10/17 columns compared to only 3/17 recovered columns when using the subset sum implementation.

Experiments with Algorithm 2. The heuristic approach presented in Algorithm 2 was more effective than the deterministic approach presented in Algorithm 1. We tested Algorithm 2 against the three above databases. The results are depicted in Table 2 and they show clearly that the heuristic approach presented in Algorithm 2 recovers more columns. For example, it recovered 20/40 columns in the Census database compared to only 6/40 when Algorithm 1 (independent set approach) is employed. Moreover, when we combine the intersection heuristic approach with the subset sum approach, we recover even more columns in the Census database as shown in the last row of Table 2 where 22/40 queries are recovered.

4.2 Experiments to Recover the Attribute Values

Under the assumption that the attacker has background knowledge represented only in the frequency distribution, we apply both the *Attribute-Name Recovery* attack and the *Relational-Count* attack on the Bank database. Our goal is to recover more queries compared to the case where only the *Relational-Count* attack is applied.

Table 2: *Attribute-Name Recovery Results on Different Relational Databases when all the possible queries are issued. The entry* 22/40 *indicates that we have recovered 22 columns out of 40 columns in the Census database.*

Algorithm	**Census**	**Bank**	**Adult**
1 (Subset sum)	6/40	3/17	12/14
1 (Independent set)	6/40	10/17	2/14
2 (Intersection)	20/40	12/17	10/14
2 (Int.)+1 (Sub.)	22/40	12/17	12/14

Combining Algorithm 2 and the *Relational-Count* attack allows us to recover more queries than those recovered when only the *Relational-Count* is applied [1]. Table 3 summarizes the results of experiments to recover the attribute values using our second attack (i.e. combination of the Attribute-Name and the *Relational-Count* attack). The table shows that when all queries are issued, our second attack recovers more queries than those recovered using the *Relational-Count* attack. So one can conclude that for some relational database tables (e.g. small number of columns, unique cardinality, non-zero co-occurrence between attribute-value pairs belonging to different columns), SSE schemes security level is very close to the security level provided by deterministic encryption schemes. Note that the maximum number of queries that can be recovered in each SSE-protected database is upper bounded by the number of queries that can be recovered when the columns' of the relational database is protected using a deterministic encryption algorithm. This upper bound is equivalent to 272, 150 and 829 in the Adult, Bank and Census databases respectively due to the existence of some values within the same column that have the same frequency in these databases.

Table 3: *The table shows the number of queries recovered from three SSE-protected relational databases (Adult, Bank and Census) using both the Relational-Count attack [1] and our second attack.*

Attack	**Adult**	**Bank**	**Census**
Our Second Attack	240/272	147/150	760/829
Relational-Count Attack [1]	236/272	122/150	757/829

4.3 Countermeasures

The padding countermeasure, which is proposed in [15,6], hides the actual result size of each query and therefore might prevent our *Attribute-Name recovery* attack. However, depending on the padding level, a variant of the *Attribute-Name recovery* attack that does not look for the exact number of records but for a range of possible values for the number of records might still allow the attacker to find queries belonging to the same column.

5 Conclusion

In this paper, we proposed two attacks on relational databases protected via SSE schemes. Our first attack breaks query privacy as it classifies the set of issued queries into different subsets where each subset holds the queries belonging to a specific column. Remarkably, our first attack is more practical than all other proposed attacks on SSE schemes [15,6,1] which require prior knowledge about the frequency distribution of the attribute-value pairs in the target database. This is because our attribute-name recovery attack does not require any prior knowledge about the target database other than the meta-data information and the number of records. An important message of this paper is that SSE schemes leaking the number of records n should not be used.

Moreover, we improved the *Relational-Count* attack [1] by combining it with our first attack. This combination allows us to recover more queries on some databases. Our work is important because it shows that protecting relational databases via SSE schemes breaks query privacy as it allows an attacker to recover the attribute names of some of the observed queries. In particular, the queries belonging to columns with low cardinality will be easily distinguished from other queries without waiting for all queries to be issued.

Our experiments assume that all the possible queries on a searchable encrypted database are issued which is one limitation of our work. Observation 2 will not hold for some attributes when the number of issued queries is less than the number of all the attribute-value pairs in the encrypted database since the existence of a subset of observed queries belonging to a certain attribute is not certain. Estimating the existence probability of a certain subset of observed queries depends on the distribution of the issued queries which varies from one user to another. So further work is required to estimate this probability.

Acknowledgments

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. Mohamed Ahmed Abdelraheem, Tobias Andersson, and Christian Gehrmann. Searchable encrypted relational databases: Risks and countermeasures. In *The 12th Data Privacy and Management Workshop*, 2017.
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. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *CCS 2015*.
7. 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.
8. 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*.
9. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010*.
10. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, 2006.
11. J. Erickson. Lower bounds for linear satisfiability problems. In *SODA 1995*.
12. Center for Machine Learning and Intelligent Systems. University of california, irvine. `https://archive.ics.uci.edu/ml/datasets.html`. [Last Accessed June 2017].
13. O. Gold and M. Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. *CoRR*, abs/1512.05279, 2015.
14. IARPA. Poster about protecting privacy and civil liberties. `https://www.iarpa.gov/images/files/programs/spar/09-SPAR_final_v21.pdf`.
15. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*.
16. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, 2013.
17. G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *CCS*, 2016.
18. J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.
19. R. Kohavi and B. Becker. Adult data set. `https://archive.ics.uci.edu/ml/machine-learning-databases/adult/`, 1996. [Last Accessed June 2017].
20. K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, 2012.
21. Terran Lane and Ronny Kohavi. Census-income (kdd) data set. `https://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld/`, 2000. [Last Accessed June 2017].
22. M. Naveed, S. Kamara, and C. Wright. Inference attacks on property-preserving encrypted databases. In *CCS 2015*.

23. OpenEMR. http://www.open-emr.org/. Accessed: March 2017.
24. 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*.
25. R. Laureano S. Moro and P. Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In P. Novais et al. (Eds.), Proceedings of the European Simulation and Modelling Conference - ESM'2011, pp. 117-121, Guimarães, Portugal, October, 2011. EUROSIS, https://archive.ics.uci.edu/ml/datasets/Bank+Marketing. [Last Accessed June 2017].
26. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Security and Privacy. S&P 2000.*
27. P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Workshop on Secure Data Management*, 2010.

A Example Explaining Our Attribute-Name Recovery Attack

In the following, we give a toy example to demonstrate our attack. Assume that we have a relational database table as shown in Table 4. Using a deterministic encryption algorithm to encrypt the "Sex" and "Education" columns and using an order preserving encryption will transform our relational database table to an encrypted relational database table as shown in Table 5. However, most secure SSE schemes will transform an inverted index such as the one displayed in Table 6 into a length-hiding encrypted index where the server does not know the frequency or result length of each keyword token before being queried.

Table 4: *The table shows a plaintext relational database table.*

ID	**Sex**	**Education**	**Age**
1	M	Bsc	40
2	F	Msc	39
3	F	PhD	30
4	M	PhD	45
5	M	Bsc	25
6	F	Bsc	23
7	M	Msc	30

After observing all the queries issued on the encrypted index shown in Table 6. Our attribute-name recovery attack tries to resolve the attribute name of each observed query by exploiting the access pattern leakage. Fig. 1 shows three graphs whose nodes represent the observed queries. The graph on the left shows the server's knowledge (represented by the frequencies or result lengths of observed queries gained from the access pattern leakage) before launching our

Table 5: *The table shows an encrypted relational database table. DET refers to Deterministic Encryption and OPE refers to Order Preserving Encryption. Column names can either be replaced by random labels or deterministically encrypted using the table unique ID.*

ID	$DET(Education\|\|tableID, K_0)$	$DET(Age\|\|tableID, K_0)$	$DET(Sex\|\|tableID, K_0)$
1	$DET(Bsc, K_2)$	$OPE(40, K_3)$	$DET(M, K_1)$
2	$DET(Msc, K_2)$	$OPE(39, K_3)$	$DET(F, K_1)$
3	$DET(PhD, K_2)$	$OPE(30, K_3)$	$DET(F, K_1)$
4	$DET(PhD, K_2)$	$OPE(45, K_3)$	$DET(M, K_1)$
5	$DET(Bsc, K_2)$	$OPE(25, K_3)$	$DET(M, K_1)$
6	$DET(Bsc, K_2)$	$OPE(23, K_3)$	$DET(F, K_1)$
7	$DET(Msc, K_2)$	$OPE(30, K_3)$	$DET(M, K_1)$

Table 6: *The table shows an inverted index for the relational database table shown in Table 4. Each keyword w is represented as $w = (a : v)$ where a refers to its attribute name and v refers to its value.*

Keyword	**Record IDs**
Sex:F	2, 3, 6
Sex:M	1, 4, 5, 7
Education:Bsc	1, 5, 6
Education:Msc	2, 7
Education:PhD	3, 4
Age:23	6
Age:25	5
Age:30	3, 7
Age:39	2
Age:40	1
Age:45	4

attacks. When we apply the Attribute-Name recovery attack using only as background knowledge the meta-data information about the table and the number of records, the Server will know only the attribute names, "Education", "Sex" and "Age" represented by the graph on the middle in Fig. 1. Note that Naveed et al. [22] attack recovers column names and values of the encrypted database table shown in Table 5 using public background data. However, our attribute-name recovery attack recovers the query issued on the encrypted index shown in Table 7 using only meta-data information about the database table in addition to the number of records which can be leaked by some SSE schemes or guessed by the attacker.

Moreover, when we apply both the Attribute-Name recovery attack and the *Relational-Count* attack using the frequency distribution knowledge, the Server will know both the attribute names and their corresponding actual values. This additional knowledge is represented in Fig. 1 by the graph on the right.

Table 7: *The table shows a length-hiding encrypted index before being randomly permuted for the inverted index shown in Table 6. DET refers to deterministic encryption and Enc refers for a randomized encryption algorithm. Note that this encrypted index is not secure as the rows needs to be securely and randomly shuffled and the number of rows needs to be padded to the maximum number of keywords possible.*

Keyword	**Record IDs**
$DET(Sex : F, K_0)$	$Enc(2\|\|3\|\|6\|\|*, K_1)$
$DET(Sex : M, K_0)$	$Enc(1\|\|4\|\|5\|\|7, K_1)$
$DET(Education : Bsc, K_0)$	$Enc(1\|\|5\|\|6\|\|*, K_1)$
$DET(Education : Msc, K_0)$	$Enc(2\|\|7\|\|*\|\|*, K_1)$
$DET(Education : PhD, K_0)$	$Enc(3\|\|4\|\|*\|\|*, K_1)$
$DET(Age : 23, K_0)$	$Enc(6\|\|*\|\|*\|\|*, K_1)$
$DET(Age : 25, K_0)$	$Enc(5\|\|*\|\|*\|\|*, K_1)$
$DET(Age : 30, K_0)$	$Enc(3\|\|7\|\|*\|\|*, K_1)$
$DET(Age : 39, K_0)$	$Enc(2\|\|*\|\|*\|\|*, K_1)$
$DET(Age : 40, K_0)$	$Enc(1\|\|*\|\|*\|\|*, K_1)$
$DET(Age : 45, K_0)$	$Enc(4\|\|*\|\|*\|\|*, K_1)$

Fig. 1: Nodes on the graphs represent all the possible queries that can be issued in the relational database index table shown in Table 6 after being encrypted by an SSE scheme. An edge between nodes (queries) exists if the intersection between their corresponding result sets is non-zero. The graph on the left shows queries as nodes of a graph labeled by their result lengths before applying our attacks. The graph on the middle shows what the server will learn after applying our attribute-name recovery attack. Note that the green color refers to the "Sex" attribute name and the blue color refers to the "Education" attribute name and the red color refers to the "Age" attribute name. The graph on the right shows what the server could learn after applying our second attack (Attribute-Name recovery attack combined with the *Relational-Count* attack).