Using Bitmaps for Executing Range Queries in Encrypted Databases

Lil María Rodríguez-Henríquez and Debrup Chakraborty
Departamento de Computación, CINVESTAV-IPN
Av. Instituto Politécnico Nacional No. 2508, Col. San Pedro Zacatenco, Mexico D.F. 07300, Mexico

Keywords: Symmetric Encryption, Database Encryption, Deterministic Encryption, Bitmap Index, Provable Security.

Abstract: Privacy of data stored at un-trusted servers is an important problem of today. A solution to this problem can be achieved by encrypting the outsourced data, but simple encryption does not allow efficient query processing. In this paper we propose a novel scheme for encrypting relational databases so that range queries can be efficiently executed on the encrypted data. We formally define the syntax and security of the problem and specify a scheme called ESRQ1. ESRQ1 uses a deterministic encryption scheme along with bitmap indices to encrypt a relational database. We provide details of the functionality of ESRQ1 and prove its security in the specified model.

1 INTRODUCTION

Encryption, if applied to relational databases, should be done in such a way that a large class of queries can be executed on the encrypted tables. For example, if we consider the relation $R1$ shown in Table 1 some representative queries on this relation can be: (1) What are the names of the employees whose age is 18 years? (2) What are the names of the employees whose age is more than 30 years? (3) What is the average age of an employee?

If the relation $R1$ is encrypted normally, i.e., encrypting separately each cell of the relation with a strong (say a IND-CPA secure) encryption, then none of the above queries can be executed on the encrypted table.

Regarding the queries that we posed above, query (1) can be addressed if the relation is encrypted using a *deterministic encryption scheme*, such encryption preserves the equality relation of plaintexts. For query (2), it is required that the ciphertexts maintains the order of the plaintexts, this can be achieved by *order preserving encryption*. Finally, for query (3) it is required that meaningful computations can be performed on the ciphertexts, *homomorphic encryption schemes* can support such computations on encrypted data. Generally to treat a large class of queries different encryption mechanisms on the same database is used. Among others, this paradigm is followed in a recent work (Popa et al., 2011). In this work we will mainly focus on processing a class of range queries (query (2) in the example above).

To enable range queries in an encrypted database the ciphertext values should provide order information of the plaintexts. This can be achieved by an order preserving encryption (OPE) scheme. An OPE scheme $\mathbf{E}$ is such a scheme where $\mathbf{E}(x) \geq \mathbf{E}(y)$, iff $x \geq y$. This interesting primitive has received lot of attention in the current years. The first concepts appeared in the paper (Agrawal et al., 2004), where the main aim was to design a scheme where efficient range queries can be executed on encrypted data. This work does not delve into formal definitional and security perspectives of OPE. The first work which formally deals with OPE is (Boldyreva et al., 2009), where the ideal security notion for an OPE scheme, IND-OCPA, was introduced. The IND-OCPA definition specifies that the main goal for an OPE is to reveal no additional information about the plaintext values besides their order. In (Boldyreva et al., 2009) it was shown that it is infeasible to achieve IND-OCPA security with a stateless encryption scheme. As a result, they settled on a weaker security guarantee that was later shown in (Boldyreva et al., 2011) to leak at least half of the plaintext bits. Later, several other order preserving schemes were proposed (Boldyreva et al., 2011; Lee et al., 2009; Yum et al., 2012), but as it is suggested in (Popa et al., 2013) none of them has achieved the ideal IND-OCPA security.

Recently, in (Popa et al., 2013), the first ideal-security order-preserving encoding scheme was proposed, where the ciphertexts reveal nothing except the order of the plaintext values. The insight that allow them to avoid the infeasibility result in (Boldyreva

María Rodríguez-Henríquez L. and Chakraborty D..
Using Bitmaps for Executing Range Queries in Encrypted Databases.
DOI: 10.5220/0005111604320438
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 432-438
ISBN: 978-989-758-045-1

Table 1: The relation $R1$.

EmpId	Name	Age
TRW	Tom	18
MST	Mary	17
JOH	John	52
MRH	Mary	33
ASY	Anne	18
RZT	Rosy	36

et al., 2009) is that the encryption protocol proposed is interactive, and a small number of ciphertexts of already-encrypted values change, as new plaintext values are encrypted. A property which the authors terms as "mutable".

Our Contributions. We see the problem of executing range queries in an encrypted database from a different direction. The main component of our solution is not to use any special type of encryption scheme, i.e., the encryption scheme we use is not an OPE scheme. We encrypt the tables with a deterministic encryption scheme, additionally we maintain another table which contains just the order information of the various attribute values. Both of these tables are kept with the server, and the server can respond to range queries without the knowledge of the real attribute values in the table. We propose a generic framework for a database encryption scheme supporting range queries (ESRQ), and define its syntax and security notion. We also provide a complete scheme called ESRQ1. A novel component of ESRQ1 is that it uses bitmap indices to encode the order information of the attributes. Bitmap indices are gaining popularity in the current days for accelerated query processing, and many commercial databases now implement them. Thus it is likely that our solution can be adopted in an existing database easily and efficiently. To our knowledge bitmaps have not been used previously to ensure privacy of databases.

2 PRELIMINARIES

Relations. In what follows, by $R(A)$ we would denote a relation over a set of attributes A. If $A = \{a_1, a_2, \cdots a_m\}$, we shall sometimes write $R(a_1, a_2, \cdots, a_m)$. Given an attribute a, $\mathsf{Dom}_R(a)$ would represent the distinct values of the attribute a in the relation R.

By cardinality of an attribute a in R we shall mean the cardinality of $\mathsf{Dom}_R(a)$, i.e. $\mathsf{Card}_R(a) = |\mathsf{Dom}_R(a)|$.

A tuple t in a relation over a set of attributes is a function that associates with each attribute a value in its specific domain. Specifically if $A = \{a_1, a_2, \cdots a_m\}$ and $R(A)$ be a relation, then the j^{th} tuple of $R(A)$ would be denoted by t_j^R. And for $a_i \in A$, by $t_j^R[a_i]$ we shall denote the value of attribute a_i in the j^{th} tuple in R. For $B \subseteq A$, $t_j^R[B]$ will denote the set of values of the attributes in B in the j^{th} tuple. We shall sometimes omit the subscripts and superscripts from t_j^R and denote the tuple by t if the concerned relation is clear from the context and the tuple number is irrelevant.

Binary Strings. The set of all binary strings would be denoted by $\{0,1\}^*$, and the set of n bit strings by $\{0,1\}^n$. For $X_1, X_2 \in \{0,1\}^*$, by $X_1||X_2$ we shall mean the concatenation of X_1 and X_2; $|X_1|$ will denote the length of X_1 in bits; and $\overline{X_1}$ will denote the bitwise complement of X_1. If $|X_1| = |X_2|$, and $\circledcirc$ be a Boolean operator then by $X_1 \circledcirc X_2$ we will mean that the operation $\circledcirc$ is applied bitwise. For $X \in \{0,1\}^n$ and $1 \leq k \leq n$, $\mathsf{bit}_k(X)$ will denote the k^{th} bit of X.

We shall always consider that the domains of all attributes in the relations and attribute names are subsets of $\{0,1\}^*$, this convention would allow us to apply transformations and functions on the values of the tuples in a relation without describing explicit encoding schemes.

Bitmaps. Consider a relation $R(a_1, \ldots, a_m)$ with nT many rows. Consider that for each attribute a_i, $\mathsf{Dom}_R(a_i) = \{v_1^i, v_2^i, \ldots v_{\lambda_i}^i\}$, thus $\mathsf{Card}_R(a_i) = \lambda_i$ for $1 \leq i \leq m$. We define the bitmap of an attribute a_i corresponding to its value v_j^i in the relation R as $\mathsf{BitMap}_R(a_i, v_j^i) = X$, where X is a binary string, such that $|X| = \mathsf{nT}$ and for $1 \leq k \leq \mathsf{nT}$,

$$\mathsf{bit}_k(X) = \begin{cases} 1 & \text{if } t_k^R[a_i] = v_j^i \\ 0 & \text{otherwise.} \end{cases}$$

The encoding for the previous definition is known as equality encoding, we shall call such bitmaps as e-encoded bitmaps. In the literature there are other kinds of bitmap encodings to allow different kinds of queries. We would be interested in two types of range bitmaps $\mathsf{BitMap}^<$ and $\mathsf{BitMap}^>$ which we will further call as l-encoded and g-encoded bitmaps respectively. For the relation R described above, we have $\mathsf{BitMap}_R^<(a_i, v_j^i) = Y$, where

$$\mathsf{bit}_k(Y) = \begin{cases} 1 & \text{if } t_k^R[a_i] < v_j^i \\ 0 & \text{otherwise.} \end{cases}$$

From the l-encoding and e-encoding other bitmap encodings can be easily derived. For example, we define

$$\begin{aligned} \mathsf{BitMap}_R^>(a_i, v_j^i) &= \overline{\mathsf{BitMap}_R^<(a_i, v_j^i)} \oplus \mathsf{BitMap}_R(a_i, v_j^i) \\ \mathsf{BitMap}_R^\leq(a_i, v_j^i) &= \mathsf{BitMap}_R^<(a_i, v_j^i) \vee \mathsf{BitMap}_R(a_i, v_j^i) \\ \mathsf{BitMap}_R^\geq(a_i, v_j^i) &= \overline{\mathsf{BitMap}_R^<(a_i, v_j^i)} \end{aligned}$$

In what follows we will sometimes use two procedures related to bitmaps. Let nT be a positive integer and $S \subseteq \{1,2\ldots,\mathsf{nT}\}$. $\mathsf{MakeBitMap}(\mathsf{nT},S)$ outputs a string X such that $|X| = \mathsf{nT}$ and for $1 \leq k \leq \mathsf{nT}$, $\mathsf{bit}_k(X) = 1$ if $k \in S$ and $\mathsf{bit}_k(X) = 0$, otherwise. If $Y \in \{0,1\}^{\mathsf{nT}}$, then $\mathsf{MakeSet}(Y)$ returns the set $S = \{i : 1 \leq i \leq \mathsf{nT} \text{ and } \mathsf{bit}_i(Y) = 1\}$.

Deterministic Encryption Schemes: A deterministic encryption scheme with associated data (DEAD) is a deterministic function $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{C}$, where $\mathcal{K}$, $\mathcal{T}$, $\mathcal{M}$ and $\mathcal{C}$ are the key space, tweak (associated data) space, message space and cipher space, respectively. Thus, $\mathbf{E}$ receives as input a key $K \in \mathcal{K}$, a tweak $T \in \mathcal{T}$ and a message $M \in \mathcal{M}$ and produces as output a cipher $C \in \mathcal{C}$. We shall often write $\mathbf{E}_K(T,M)$ instead of $\mathbf{E}(K,T,M)$. $\mathbf{E}$ also has an inverse function $\mathbf{D} : \mathcal{K} \times \mathcal{T} \times \mathcal{C} \to \mathcal{M}$, such that for every $K \in \mathcal{K}$, every $T \in \mathcal{T}$ and every $M \in \mathcal{M}$, $\mathbf{D}_K(T,\mathbf{E}_K(T,M)) = M$. This implies that for every $T \in \mathcal{T}$ and every $K \in \mathcal{K}$, $\mathbf{E}_K(T,\cdot) : \mathcal{M} \to \mathcal{C}$ is an injective function.

To define security of $\mathbf{E}$ we consider an adversary $\mathcal{A}$ which is given an oracle O. The oracle O can either be $\mathbf{E}_K(\cdot,\cdot)$, i.e., the encryption scheme which is instantiated with a uniform random key K selected from $\mathcal{K}$, or it can be $\$(\cdot,\cdot)$ which is an oracle which on input $(T,M) \in \mathcal{T} \times \mathcal{M}$ outputs a random string of size $|\mathbf{E}_K(T,M)|$. One important restriction of $\mathcal{A}$ is that it is not allowed to repeat any query. We formally define the advantage of $\mathcal{A}$ as

$$\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(\mathcal{A}) = |\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{E}_K(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1]|$$

We define $\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(q,\sigma_n,t)$ by $\max_{\mathcal{A}}\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(\mathcal{A})$ where maximum is taken over all adversaries which makes at most q queries having at most σ_n many blocks (of n bits) and runs for time at most t. We consider an encryption scheme to be (ε,q,t) det-cpa secure if $\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(q,\sigma_n,t) \leq \varepsilon$.

3 ESRQ: BASIC NOTIONS

An encryption scheme supporting range queries (ESRQ) consists of a tuple of algorithms $(\mathcal{K},\mathsf{Enc},\Phi,\Psi,\mathsf{Dec})$, which work as follows.

The **key Generation Algorithm** $\mathcal{K}$ runs at the client side and outputs a set of keys that are randomly selected from a pre-specified key space. We will sometimes denote the key space also by $\mathcal{K}$.

Game $\text{ESRQ}^{\mathcal{A}}_{\Upsilon}$
1. The challenger selects $K \xleftarrow{\$} \mathcal{K}$
2. A selects two relations R_0, R_1, such that $R_0 \approx R_1$, and gives them to the challenger.
3. The challenger selects a bit $b \xleftarrow{\$} \{0,1\}$
4. The challenger computes $\mathcal{R}' \leftarrow \Upsilon.\mathsf{Enc}_K(R_b)$, and gives it to $\mathcal{A}$.
5. $\mathcal{A}$ outputs a bit b'.
6. **if** $b = b'$ **output** 1
7. **else output** 0

Figure 1: Game used to define security of ESRQ.

The **Privacy Transform** Enc receives as an input a relation called R and the set of keys generated by $\mathcal{K}$. This transform generates a new set of encrypted relations called $\mathcal{R}'$. We denote this operation as $\mathcal{R}' \leftarrow \mathsf{Enc}_K(R)$. This transform is executed in the client side.

The **Query Translator** Φ is a keyed transform that runs at the client. It takes in the key K and a query q meant for the relation R and converts it to a query q' which can be executed in $\mathcal{R}'$. For convenience, we shall sometimes refer to a query meant for R to be a R-query and a query meant for $\mathcal{R}'$ as $\mathcal{R}'$-query.

The **Response Procedure** Ψ runs at the server side. Ψ allows the server to answer a query $\Phi(q)$ on $\mathcal{R}'$. The output of Ψ for a query is the response of the server, called ρ. ρ contains the encrypted answer, the server returns it to the client.

The **Decryption Procedure** is a keyed transform Dec_K which runs in the client. It receives as input the server response ρ and outputs the answer ans for the query q.

3.1 Security Notion

Definition 1. *Given a relation R over a set of attributes A, we classify the attributes in two different classes: the ones where a range query is valid (range attributes) and the ones where such queries are not valid. Define* $\mathsf{ty} : A \to \{0,1\}$, *where for any $a \in A$,* $\mathsf{ty}(a) = 0$, *if range queries on the attribute a is not applicable and* $\mathsf{ty}(a) = 1$, *otherwise.*

Definition 2. *Two relations R and S are said to be equivalent (denoted by $R \approx S$) if the following conditions hold.*

1. *R and S are defined over the same set of attributes A.*
2. *R and S contains the same number of tuples* nT.
3. *For every $i,j \in \{1,2,\ldots,\mathsf{nT}\}$, and every $a \in A$, $t^R_i[a] = t^R_j[a] \iff t^S_i[a] = t^S_j[a]$.*

4. *For every* $i, j \in \{1,2,\ldots,\mathsf{nT}\}$, *and every* $a \in A$, *such that* $\mathsf{ty}(a) = 1$,

$$t_i^R[a] \geq t_j^R[a] \Longleftrightarrow t_i^S[a] \geq t_j^S[a].$$

5. *For every* $i \in \{1,2,\ldots,\mathsf{nT}\}$, *and every* $a \in A$, $|t_i^R[a]| = |t_i^S[a]|$.

The basic goal of ESRQ is to transform R to $\mathcal{R}'$ in such a way that $\mathcal{R}'$ should not contain any information beyond the order relation between the attribute values. We formally define security of an ESRQ Υ as a game between an adversary and a challenger, see Figure 1.

Definition 3. *The advantage of an adversary* $\mathcal{A}$ *in attacking an* ESRQ Υ *is defined as*

$$\mathbf{Adv}_{\Upsilon}^{\text{esrq}}(\mathcal{A}) = \left|\Pr[\text{ESRQ}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|.$$

The relevant resources of an adversary attacking an ESRQ scheme is its running time and the size of the relations that it chooses and submits to its challenger. We measure the size of the relation in two different ways, we denote by q the number of cells. Let s_i be the size of each cell, then we define $\sigma_n = \sum_{i=1}^{q} \lceil s_i/n \rceil$. We call q as the cell complexity of $\mathcal{A}$ and σ_n as the query complexity of $\mathcal{A}$. We define $\mathbf{Adv}_{\Upsilon}^{\text{esrq}}(q,\sigma_n,t) = \max \mathbf{Adv}_{\Upsilon}^{\text{esrq}}(\mathcal{A})$, where the maximum is taken over all adversaries $\mathcal{A}$ which runs for time at most t and has cell complexity and query complexity of at most q and σ_n, respectively. Moreover, we say that an ESRQ scheme Υ is $(\varepsilon, q, \sigma_n, t)$ secure, if $\mathbf{Adv}_{\Upsilon}^{\text{esrq}}(q,\sigma_n,t) \leq \varepsilon$.

3.2 ESRQ1

Here we discuss a specific scheme ESRQ1 for encrypting relations such that simple select and range queries can be executed in the encrypted relations. Consider a relation $R(A)$ where $A = \{a_1, a_2, \ldots a_{|A|}\}$, and the function $\mathsf{ty} : A \to \{0,1\}$ defined on A. We consider that a client wants to outsource this generic relation $R(A)$ to a server. To ensure privacy, the client, encrypts the relation $R(A)$ using ESRQ1 and delegates this encrypted relation instead of the original one. The client will pose queries to the server and expects that the server to execute these queries on his/her behalf without knowing the real contents of the relation $R(A)$.

In what follows, we present a generic description of the scheme, also throughout we discuss a specific example based on the relation shown in Table 1. The only cryptographic object used by ESRQ1 is a deterministic encryption scheme $\mathbf{E}$ which is required to be det-cpa secure. We assume that $\mathbf{E} : \mathcal{K} \times \{0,1\}^{\lambda} \times \{0,1\}^* \to \{0,1\}^*$. Other than the deterministic encryption scheme $\mathbf{E}$, ESRQ1 uses bitmap indices.

The various algorithms involved in ESRQ1 are discussed next in order.

ESRQ1.$\mathcal{K}$: This procedure selects a key K uniformly at random from $\mathcal{K}$. Where $\mathcal{K}$ is the key space of the deterministic encryption scheme $\mathbf{E}$ involved.

ESRQ1.Enc: Given $R(A)$ and the key K as input, ESRQ1.Enc outputs two relations R_α and R_β. We assume that $R(A)$ contains nT many tuples and $A = \{a_1, a_2, \ldots, a_{|A|}\}$. To each attribute $a_i \in A$ we associate an unique identifier $\mathsf{id}_i \in \{0,1\}^{\lambda}$. Among other possibilities, this identifier can be the (appropriately encoded) name of the attribute or a counter.

R_α contains nT tuples and is defined over the attributes $B = \{\mathtt{Row}\} \cup \{b_1, b_2, \ldots, b_{|A|}\}$. Where $b_i = \mathbf{E}_K(\mathsf{id}^*, a_i)$ for some $\mathsf{id}^* \in \{0,1\}^{\lambda}$ such that $\mathsf{id}^* \notin \{\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_{|A|}\}$. Hence, R_α has one attribute more than in R, the extra attribute is RowNo, the other attributes of R_α are the encryption of the attribute names in A. The specific way in which R_α is created from R is shown in Figure 2, which shows that R_α contains the encryption of the values present in R.

The relation R_β contains the attributes $\{\mathtt{Name}, \mathtt{SearchKey}, \mathtt{BitMap}\}$, irrespective of the attributes in relation R. The way the relation R_β is populated is shown in Figure 2. R_β stores information regarding each range attribute in A. For a range attribute a_i all its values occurring in R are encrypted. These encrypted values along with the corresponding attribute name and the l-encoded bitmap are stored in R_β.

For a concrete example, consider that ESRQ1.Enc has as input the relation $R1$ as shown in Table 1. The only attribute in $R1$, where range queries are meaningful is the attribute Age. Then ESRQ1.Enc$(R1)$ would produce as output the relations $R1_\alpha$ and $R1_\beta$ as shown in Table 2. While applying encryption, the unique identifier of each column is used as the associated data. The attribute names of the original relation $R1$ occur in $R1_\alpha$ in the encrypted form.

The relation $R1_\beta$ is created as described in Figure 2. The basic idea is to store the order information of all the values corresponding to the range attributes in a suitable manner. The table stores the encrypted values along with the encrypted attribute names. The order information is stored in the form of l-encoded bitmaps. Note that, for encrypting a specific value the identifier of the attribute is used as the associated data.

ESRQ1.Φ: The transform Φ receives as input a query meant for R and converts it to a query which can be executed in (R_α, R_β). The allowed set of queries are simple select queries and range queries. The generic format of an allowed query is

Q: SELECT * FROM R WHERE $(a_1 \lhd_1 v_1)$ $\circledcirc_1$ $(a_1 \lhd_2 v_2)$ $\circledcirc_2$ ..... $\circledcirc_{\ell-1}$ $(a_\ell \lhd_1 v_\ell)$,

Table 2: Relations $R1_\alpha$ and $R1_\beta$.

Relation $R1_\alpha$

RowNo	$\mathbf{E}_K(\mathsf{id}^*,\mathtt{EmpId})$	$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Name})$	$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Age})$
1	$\mathbf{E}_K(\mathsf{id}_1,\text{TRW})$	$\mathbf{E}_K(\mathsf{id}_2,\text{Tom})$	$\mathbf{E}_K(\mathsf{id}_3,18)$
2	$\mathbf{E}_K(\mathsf{id}_1,\text{MST})$	$\mathbf{E}_K(\mathsf{id}_2,\text{Mary})$	$\mathbf{E}_K(\mathsf{id}_3,17)$
3	$\mathbf{E}_K(\mathsf{id}_1,\text{JOH})$	$\mathbf{E}_K(\mathsf{id}_2,\text{John})$	$\mathbf{E}_K(\mathsf{id}_3,52)$
4	$\mathbf{E}_K(\mathsf{id}_1,\text{MRH})$	$\mathbf{E}_K(\mathsf{id}_2,\text{Mary})$	$\mathbf{E}_K(\mathsf{id}_3,33)$
5	$\mathbf{E}_K(\mathsf{id}_1,\text{ASY})$	$\mathbf{E}_K(\mathsf{id}_2,\text{Anne})$	$\mathbf{E}_K(\mathsf{id}_3,18)$
6	$\mathbf{E}_K(\mathsf{id}_1,\text{RZT})$	$\mathbf{E}_K(\mathsf{id}_2,\text{Rosy})$	$\mathbf{E}_K(\mathsf{id}_3,36)$

Relation $R1_\beta$

Name	SearchKey	BitMap
$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Age})$	$\mathbf{E}_K(\mathsf{id}_3,17)$	000000
$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Age})$	$\mathbf{E}_K(\mathsf{id}_3,18)$	010000
$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Age})$	$\mathbf{E}_K(\mathsf{id}_3,33)$	110010
$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Age})$	$\mathbf{E}_K(\mathsf{id}_3,36)$	110110
$\mathbf{E}_K(\mathsf{id}^*,\mathtt{Age})$	$\mathbf{E}_K(\mathsf{id}_3,52)$	110111

Creating R_α
1. **for** $j = 1$ to $|A|$,
2. $\quad b_j \leftarrow \mathbf{E}_K(\mathsf{id}^*, a_j)$;
3. **end for**
4. **for** $j = 1$ to nT,
5. $\quad t_j^{R\alpha}[\mathtt{RowNo}] \leftarrow j$;
6. $\quad$ **for** $i = 1$ to $|A|$,
7. $\quad\quad t_j^{R\alpha}[b_i] \leftarrow \mathbf{E}_K(\mathsf{id}_i, t_j^R[a_i])$;
8. $\quad$ **end for**
9. **end for**

Creating R_β
1. $j \rightarrow 1$
2. **for** $i = 1$ to $|A|$
3. $\quad$ **if** $\mathsf{ty}(a_i) = 1$,
4. $\quad\quad$ **for all** $v \in \mathsf{Dom}(a_i)$,
5. $\quad\quad\quad t_j^{R\beta}[\mathtt{Name}] \leftarrow \mathbf{E}_K(\mathsf{id}^*, a)$;
6. $\quad\quad\quad t_j^{R\beta}[\mathtt{SearchKey}] \leftarrow \mathbf{E}_K(\mathsf{id}_i, v)$;
7. $\quad\quad\quad t_j^{R\beta}[\mathtt{BitMap}] \leftarrow \mathsf{BitMap}_R^{\leq}(a_i, v)$;
8. $\quad\quad\quad j \leftarrow j+1$
9. $\quad\quad$ **end for**
10. $\quad$ **end if**
11. **end for**

Figure 2: Creating R_α and R_β.

where a_i represent an attribute name and v_i a value of the attribute. $\lhd_i \in \{>, <, \leq, \geq\}$, and $\circledcirc_i$ can be an arbitrary Boolean connective, say $\vee$, $\wedge$ etc.

Given as input a valid query Q, $\Phi(Q)$ is the translation of the original query which just hides the attribute names and the values assigned to the attributes by encryption. As the input query has a specific structure, hence the translated query $\Phi(Q)$ is just

$$Q'\text{: } (c_1 \lhd_1 c_1') \circledcirc_1 (c_2 \lhd_2 c_2') \circledcirc_2 \ldots\ldots \circledcirc_{\ell-1} (c_\ell \lhd_1 c_\ell'),$$

where $c_i = \mathbf{E}_K(\mathsf{id}^*, a_i)$ and $c_i' = \mathbf{E}_K(\mathsf{id}_i, v_i)$.

Going back to the concrete example, consider the following query $Q1$ for the relation $R1$

$Q1$: `SELECT * FROM` R `WHERE Age` ≥ 18 `AND Age` ≤ 36

The transformation $\Phi(Q1)$, will output

$Q1'$: $(\mathbf{E}_k(\mathsf{id}^*, \mathtt{Age}) \geq \mathbf{E}_K(\mathsf{id}_3, 18)) \bigwedge (\mathbf{E}_k(\mathsf{id}^*, \mathtt{Age}) \leq \mathbf{E}_K(\mathsf{id}_3, 36))$

This transformed query is sent to the server.

ESRQ1.Ψ: As discussed, Ψ is the transform that runs in the server to execute the client's instructions. The instruction from the client comes to the server encoded as Q', and it executes the procedure $\Psi(Q')$ as described in Figure 3. Ψ treats the equality/inequality conditions in Q' separately, and constructs a bitmap for each of these conditions. Once the bitmaps for the individual conditions are constructed, it aggregates the bitmaps using the given Boolean connectives. The final bitmap B constructed in line 13 contains the information regarding the tuples in R_α, which satisfies the client's query.

Going back to our example, the query $Q1'$ consists of two inequality conditions. The first one is $(\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age}) \geq \mathbf{E}_K(\mathsf{id}_3, 18))$, the bitmap for this condition is constructed by checking the relation R_β, the bitmap corresponding to $\mathtt{Name} = \mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})$ AND $\mathtt{SearchKey} = \mathbf{E}_K(\mathsf{id}_3, 18)$ is 010000. But recall that the bitmap stored in R_β is the l-encoded bitmap. Hence, the bitmap for the condition $(\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age}) \geq \mathbf{E}_K(\mathsf{id}_3, 18))$ would be the complement of the stored bitmap, i.e., 101111. The bitmap for the second condition $(\mathbf{E}_k(\mathsf{id}^*, \mathtt{Age}) \leq \mathbf{E}_K(\mathsf{id}_3, 36))$ is computed by retrieving the l-encoded bitmap for $\mathtt{Name} = \mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})$ AND $\mathtt{SearchKey} = \mathbf{E}_K(\mathsf{id}_3, 36)$ that is equal to 110110. Then the e-encoded bitmap is computed from R_α, i.e., 000001. Finally these two bitmaps are operated by an OR bitwise, giving as result 110111. As in the query the two conditions are connected by an `AND`, hence the bitmap for the query is $101111 \wedge 110111 = 100111$. This bitmap corresponds to the rows 1, 4, 5 and 6 of the table R_α. Hence the server sends the tuples in R_α corresponding to these row numbers.

ESRQ1.Dec: The decryption procedure receives as input the response S from the server and the keys. This procedure uses the inverse of the encryption $\mathbf{E}_K^{-1}$ to decrypt the server response.

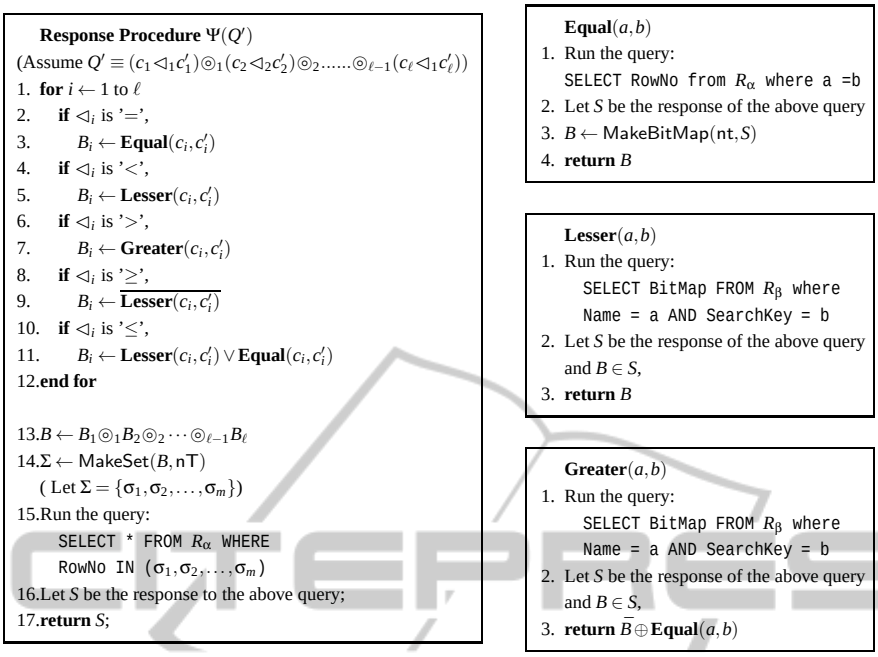

Figure 3: The response procedure Ψ.

Security of ESRQ1**:** Let ESRQ1[**E**] denote a ESRQ scheme where the deterministic encryption used is **E**.

Theorem 1. *Fix natural numbers q, σ_n, t, and a deterministic encryption scheme $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{C}$, such that the smallest ciphertext in $\mathcal{C}$ has size s. Let $\Upsilon =$ ESRQ1[**E**]. Then*

$$\mathbf{Adv}^{\text{esrq}}_{\Upsilon}(q, \sigma_n, t) \leq \mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(q', \sigma'_n, t') + \frac{q^2}{2^{s+1}},$$

where $q' \leq q$, $\sigma'_n \leq \sigma_n$ and $t' = O(t)$.

The above Theorem relates the esrq security of ESRQ1[**E**], with the det-cpa security of **E**, and it implies that the ESRQ1 scheme provides almost the same security as that of **E** with a small degradation.

4 CONCLUSION

In this paper we described a generic framework for database encryptions which enables range queries. We also specified a novel scheme called ESRQ1 which uses deterministic encryption and bitmap indices. This initial proposal is being extended in several ways: management for dynamic databases, a new scheme which would provide just det-cpa security, and an efficient implementation of this scheme on a postgres SQL database. These results would be soon appear in an extended version of this paper.

ACKNOWLEDGEMENTS

The authors acknowledge the support from CONACyT project 166763.

REFERENCES

Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2004). Order-preserving encryption for numeric data. In Weikum, G., König, A. C., and Deßloch, S., editors, *SIGMOD Conference*, pages 563–574. ACM.

Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. (2009). Order-preserving symmetric encryption. In Joux, A., editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 224–241. Springer.

Boldyreva, A., Chenette, N., and O'Neill, A. (2011). Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Rogaway, P., editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 578–595. Springer.

Lee, S., Park, T.-J., Lee, D., Nam, T., and Kim, S. (2009). Chaotic order preserving encryption for efficient and secure queries on databases. *IEICE Transactions*, 92-D(11):2207–2217.

Popa, R. A., Li, F. H., and Zeldovich, N. (2013). An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy*, pages 463–477. IEEE Computer Society.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: protecting confidentiality

with encrypted query processing. In *SOSP*, pages 85–100.

Yum, D., Kim, D., Kim, J., Lee, P., and Hong, S. (2012). Order-preserving encryption for non-uniformly distributed plaintexts. In Jung, S. and Yung, M., editors, *Information Security Applications*, volume 7115 of *Lecture Notes in Computer Science*, pages 84–97. Springer Berlin Heidelberg.