# Randomly Partitioned Encryption for Cloud Databases

Tahmineh Sanamrad[1], Lucas Braun[1], Donald Kossmann[1],
and Ramarathnam Venkatesan[2]

[1] Systems Group, Computer Science Departement, ETH Zurich, Switzerland
{sanamrat,braunl,donaldk}@inf.ethz.ch
[2] Microsoft Research, Redmond CA, USA
venkie@microsoft.com

**Abstract.** With the current advances in Cloud Computing, outsourcing data has never been so tempting. Along with outsourcing a database comes the privacy versus performance discussion. Order-Preserving Encryption (OPE) is one of the most attractive techniques for database encryption since it allows to execute range and rank queries efficiently without decrypting the data. On the other hand, people are reluctant to use OPE-based techniques in practice because of their vulnerability against adversaries with knowledge of the domain, its frequency distribution and query logs. This paper formally defines three real world driven attacks, called *Domain Attack*, *Frequency Attack* and *Query Log Attack*, typically launched by an honest-but-curious database or systems administrator. We also introduce measures to capture the probability distribution of the adversary's advantage under each attacker model. Most importantly, we present a novel technique called *Randomly Partitioned Encryption* (RPE) to minimize the adversary's advantage. Finally, we show that RPE not only withstands real world database adversaries, but also shows good performance that is close to state-of-art OPE schemes for both, read- and write-intensive workloads.

**Keywords:** Database Encryption, Efficient Query Processing, Domain Attack, Frequency Attack, Query Log Attack, Randomly Partitioned Encryption.

## 1 Introduction

Believing the trade press, cloud computing is the next big thing. Cloud computing promises reduced cost, flexibility, improved time to market, higher availability, and more focus on the core business of an organization. Virtually all players of the IT industry are jumping on the cloud computing band wagon. The only issue that seems to be able to stop cloud computing are security concerns [8]. The events that motivated this work were privacy violations in a *private* cloud by *honest-but-curious* adversaries.[1]

Encryption is a possible way to protect data against such attackers. The spectrum of available encryption schemes ranges from strong semantically secure (but typically inefficient) encryption schemes to weak (but efficient) encryption schemes. The problem that *Randomly Partitioned Encryption* tries to solve is to combine the advantages of low-security/high-performance schemes like OPE [1] with high-security/low-performance

[1] Adversaries that do not actively manipulate data, but try to infer information as defined in [19].

V. Atluri and G. Pernul (Eds.): DBSec 2014, LNCS 8566, pp. 307–323, 2014.

schemes like Probabilistic AES (AES in CBC mode [19]), in order to achieve *good security* and *reasonable performance*. Performance refers to the average query response times of the TPC-H benchmark, while security, in this paper, is the ability to resist the following attacks (thoroughly defined in section 3):

- **Domain Attack:** The Adversary has knowledge of the plaintext domain.
- **Frequency Attack:** The Adversary has knowledge of the plaintext domain and its frequency distribution.
- **Query Log Attack:** The Adversary has knowledge of the plaintext domain and has access to the database query logs.

### 1.1 Background and State of the Art

The main idea of *RPE* is to randomly partition the domain and apply an order preserving encryption scheme to each partition. This makes *RPE* a partially order-preserving encryption as each partition is ordered, but the total order is hidden. In the following, we try to summarize the state of the art in the context of existing OPE schemes and argue why they cannot withstand *Domain, Frequency* and *Query Log Attack*. A more detailed overview of related work can be found in section 9.

**Order-Preserving Encryption (OPE).** An order-preserving symmetric encryption (OPE) scheme is a deterministic symmetric encryption scheme whose encryption algorithm produces ciphertexts that preserve numerical ordering of the plaintexts. This property makes OPE very attractive for database applications, since it allows efficient range and rank query processing on encrypted data. However, the order relationship between plaintext and ciphertext remains intact after encryption, making it an easy target for a *Domain Attack*. Moreover, being deterministic, makes OPE particularly vulnerable against *Frequency Attacks*. OPE was first proposed in the database community by Agrawal et al. [1], and treated cryptographically for the first time by Boldyreva et al. in [7], followed by [6], [20], [23], [33] in search for an "ideal object". However, the problem of dealing with *Domain* and *Frequency Attacks* remains.

**Probabilistic Order-Preserving Encryption (Prob-OPE).** A probabilistic order-preserving encryption scheme is a probabilistic symmetric encryption scheme whose algorithm not only produces ciphertexts that preserve numerical ordering of the plaintexts, but also generates different ciphertexts for the same plaintext. This property flattens out the original frequency distribution of the plaintext values, therefore resisting any statistical analysis. Examples are [11], [18], [31], [35]. However, probabilistic schemes still leak total order and are exposed to *Domain Attacks*.

**Partially Order Preserving Encryption (POP).** A partially order-preserving encryption scheme is a symmetric encryption scheme whose algorithm partitions the domain. Within each partition the order is preserved but across the partitions the order is distorted. There are several ways how to partition the domain: [17], [26] partition the domain into several bins. These solutions are secure against *Domain Attacks*, but vulnerable against *Frequency and Query Log Attacks* because the queries leak the bin boundaries. *RPE* on the other hand, partitions the data in a fine-grained manner and proposes a security-tunable method to rewrite queries.

### 1.2 Contributions and Main Results

To the best of our knowledge, this paper is the first that formally defines *Domain Attack*, *Frequency Attack* and *Query Log Attack*. Moreover, we introduce three novel encryption methods and analyze their security under these attacks. We start with a scheme that protects against the *Domain Attack* which we call *Deterministic Randomly Partitioned Encryption* (Det-RPE). We then make this scheme probabilistic (Prob-RPE) in order to address the *Frequency Attack* and finally introduce a new query-rewrite mechanism called *Fixed-Range Query Rewrite* (FR) to additionally protect RPE from *Query Log Attacks*. FR can be applied to both, Det-RPE and Prob-RPE, yielding two additional encryption schemes called Det-RPE-FR, resp. Prob-RPE-FR.

What is more, we assess the security and performance of these three encryption methods and compare them to relevant related work. Table 1 shows a summary of this assessment. The first three columns depict security and have a tick if a method is secure against a certain attack (a tick in brackets means that the security depends on a tuning parameter). The last column shows a very rough performance measure that states whether or not range queries can be answered within 30 minutes in the 10-GB TPC-H dataset. As we can see, the three *RPE* variants are Pareto-optimal as they have unique privacy/performance characteristics that differentiate them from existing solutions.

**Table 1.** Qualitative Security & Performance Analysis of selected Database Encryption Schemes

| Adversary Model | Domain Attack | Frequency Attack | Query Log Attack | Performance |
|---|---|---|---|---|
| OPE [1], [6], [7], [20], [23], [33], [34] | × | × | × | ✓ |
| Modular OPE [6] | ✓ | × | × | ✓ |
| Probabilistic OPE [11], [18], [35] | × | ✓ | × | ✓ |
| Partially OPE [17], [26] | [✓] | × | × | × |
| AES-CBC [19] | ✓ | ✓ | ✓ | × |
| Det-RPE | ✓ | × | × | ✓ |
| Prob-RPE | ✓ | ✓ | × | ✓ |
| Det-RPE-FR | ✓ | × | [✓] | ✓ |
| Prob-RPE-FR | ✓ | ✓ | [✓] | ✓ |

One important feature of our methods is their composability. RPE schemes can be composed with any order-preserving encryption scheme approved by the security community, thereby inheriting their latest breakthroughs. Additionally, by adding a layer of randomness on top of a chosen underlying OPE scheme, RPE schemes amend the weaknesses of OPE schemes.

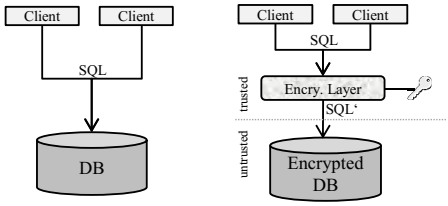
### 1.3 Overview

The remainder of this paper is organized as follows: Section 2 presents our assumed client-server architecture. Section 3 formally defines the newly introduced adversary models. Section 4 starts with describing the basic idea of *Randomly Partitioned Encryption*, then formally defines Det-RPE and analyses the security of Det-RPE and other OPE schemes under *Domain Attack*. Section 5 formally defines *Probabilistic RPE* and analyses its security under *Frequency Attack*. In Section 6, the *Fixed Range Query Rewrite* mechanism is explained and its security is analyzed under *Query Log Attack*. Section 7 and 8 describe implementation details and how they influence performance of the TPCH benchmark. Section 9 discusses related work in detail and section 10 briefly concludes.

## 2 Client-Server Architecture

Figure 1a shows the traditional client-server architecture of running applications on top of a database system. The application or end user issues SQL statements to the database server. The database server executes these SQL statements and returns the results.

Figure 1b shows the extended architecture assumed in this paper. This architecture has also been assumed by all related work on client-side database security; e.g. in [3], [9], [24], [27]. In this architecture, the application remains unchanged and issues the same (unencrypted) SQL statements as in the traditional system of Figure 1a. The confidentiality is implemented as part of an *Encryption Layer*. The *Encryption Layer* has two significant methods: First, *rewriting the queries and updates* that are to be submitted to the encrypted database. Second, *decrypting and post-processing* the query results returned from the encrypted database. Thus, the *Encryption Layer* encapsulates encryption/decryption and makes security issues transparent to the application developer and end user.




(a) Traditional (b) Encryption Layer

**Fig. 1.** Client-Server Database Architecture

The *Encryption Layer* is assumed to be *thin* and *trusted*. Thin means that not much computational power is needed to implement the *Encryption Layer*; the heavy weight-lifting of executing joins, aggregates, etc. is expected to be carried out in the database. It should be possible to deploy the *Encryption Layer* on a smart phone or laptop. The goal is to omit any administration requirements for the *Encryption Layer*.

## 3 Adversary Models

A couple of concrete usecases from the financial industry have given birth to new adversary models for cloud databases. These models have not been cryptographically treated so far. In this section we formally introduce the new adversary models and security metrics.

**Notation.** Let $\mathcal{X}$ be the set of plaintext values in a domain, and $\mathcal{Y}$ be the set of ciphertext values. The size of $\mathcal{X}$ is denoted as $X = |\mathcal{X}|$; the same applies for the size of $\mathcal{Y}$, $Y = |\mathcal{Y}|$. Plaintext elements are denoted as $x$ and ciphertext elements as $y$. Additionally, we define the Key space to be $\mathcal{K}eys$ and $K$ is denoted as an element from the key space. $\mathcal{K}$ is a function that randomly selects an element from $\mathcal{K}eys$, denoted as $K \xleftarrow{\$} \mathcal{K}eys$. The $\$$ sign on top of the $\leftarrow$ shows that the selection was random. Let $\mathcal{E}nc$ be the encryption function having a key, $K$, and a plaintext value, $x$, as its input parameters; thus, we have: $y = \mathcal{E}nc(K, x)$. Symmetrically, $\mathcal{D}ec$ will be the decryption function, taking $y$ and $K$ as input, yielding: $x = \mathcal{D}ec(K, y)$. Let $Rank_x$ be the rank of $x$ in $\mathcal{X}$. Symmetrically, we have $Rank_y$ which denotes the rank of $y$ in $\mathcal{Y}$. Let $rank$ be the function that returns the rank of an element in its corresponding space. The frequency distribution of $\mathcal{X}$ and $\mathcal{Y}$ is denoted as $\mathcal{F}_{\mathcal{X}}$ and $\mathcal{F}_{\mathcal{Y}}$ respectively. Let $freq$ be the function that returns the frequency of an element in its corresponding space.

### 3.1 Domain Attack

*Domain Attack* is launched by an adversary, $\mathcal{A}^D$ that has a-priori knowledge of the plaintext domain. In our motivating usecase from the financial industry, the database administrator had a list of all customers and needed to retrieve other relevant account information for selected customers.

**Rank One-Wayness.** In order to measure the success probability of $\mathcal{A}^D$, in breaking an encryption scheme, $\mathcal{ES}$, we introduce a new notion called Rank One-Wayness (ROW). The ROW advantage is defined to be the probability of Experiment 1 returning *1*.

$$Adv_{\text{ES}}^{\text{ROW}}(\mathcal{A}^D) = Pr[Exp_{\text{ES}}^{\text{ROW}}(\mathcal{A}^D) = 1)] \tag{1}$$

**Experiment 1 . $\text{Exp}_{\text{ES}}^{\text{ROW}}(\mathcal{A}^D)$**

1: $K \xleftarrow{\$} \mathcal{K}eys; x \xleftarrow{\$} \mathcal{X}$
2: $y \leftarrow \mathcal{E}nc(K, x)$
3: $Rank_y \leftarrow rank(y)$
4: $x' \xleftarrow{\$} A^D(\mathcal{X}, Rank_y)$
5: **if** $x = x'$ **then return** 1
6: **else return** 0

To further clarify Experiment 1, let us consider the following example. Let $\mathcal{X} = \{'Beatles','Metallica','U2'\}$ and $\mathcal{ES} = OPE$. Assume the ciphertext space to be $\mathcal{Y} = \{143, 465, 706\}$. We start the experiment by choosing a random element from $\mathcal{X}$, for instance $'Metallica'$ and encrypt it. We provide $\mathcal{A}^D$ with $\mathcal{X}$ and $Rank_{465} = 2$. $\mathcal{A}^D$ has to return an element from $\mathcal{X}$ which he thinks corresponds to the second element of $\mathcal{Y}$. The probability that $\mathcal{A}^D$ guesses $'Metallica'$ correctly is called the *ROW advantage*.

### 3.2 Frequency Attack

A *Frequency Attack* is an attack that is launched by an adversary, $\mathcal{A}^{DF}$ that has a-priori knowledge of the plaintext values as well as their frequency distribution.

**Frequency One-Wayness.** In order to measure the success probability of $\mathcal{A}^{DF}$ in breaking an encryption scheme, $\mathcal{ES}$, we introduce a new notion called Frequency One-Wayness (FOW). The FOW advantage is defined to be the probability of Experiment 2 returning *1*.

$$Adv_{\text{ES}}^{\text{FOW}}(\mathcal{A}^{DF}) = Pr[Exp_{\text{ES}}^{\text{FOW}}(\mathcal{A}^{DF}) = 1] \tag{2}$$

Again we illustrate this with an example. Let $\mathcal{X} = \{'Beatles','Beatles','U2'\}$ and $\mathcal{ES} = OPE$. The ciphertext space will be $\mathcal{Y} = \{143, 143, 706\}$. We start the experiment by choosing a random element from $\mathcal{X}$, for instance $'Beatles'$ and encrypt it. We give $\mathcal{A}^{DF}$, $\mathcal{X}$, $\mathcal{F}_{\mathcal{X}} = \{2, 1\}$ and $freq_{143} = 2$. $\mathcal{A}^{DF}$ has to return an element from $\mathcal{X}$ which he thinks corresponds to the element of $\mathcal{Y}$ that appears twice. The probability that $\mathcal{A}^{DF}$ guesses correctly is called the *FOW advantage*.

**Experiment 2 . $\text{Exp}^{\text{FOW}}_{\text{ES}}(\mathcal{A}^{DF})$**

1: $K \xleftarrow{\$} \mathcal{K}eys; x \xleftarrow{\$} \mathcal{X}$
2: $y \leftarrow \mathcal{E}nc(K, x)$
3: $Freq_y \leftarrow freq(y)$
4: $x' \xleftarrow{\$} A^{DF}(\mathcal{X}, \mathcal{F}_x, Freq_y)$
5: **if** $x = x'$ **then return** 1
6: **else return** 0

### 3.3 Query Log Attack

A *Query Log Attack* is launched by an adversary, $\mathcal{A}^Q$, that in addition to domain knowledge, has access to the database query logs, $Q_{DB}$. In our concrete usecase, since the adversary is the database administrator, access to query logs is granted to him. Depending on the encryption scheme, the query logs may reveal more information about the underlying data than what the encryption scheme was initially intended to leak. Thus, it is crucial to add query log analysis to the list of possible cryptanalysis on database encryption schemes. In general, the query logs leak (1) content, (2) origin (e.g. an IP address of the client submitting the query), (3) frequencies, and (4) time stamp of the query. In this paper we will focus on the *Query Content*. How to perform query analysis and what an adversary can obtain from it, depends on the underlying encryption scheme.

We define the success probability of $\mathcal{A}^Q$ as his advantage to break the Rank One-Wayness of the underlying encryption scheme, $\mathcal{ES}$. Similar to Experiment 1, the adversary, $\mathcal{A}^Q$, is given the plaintext domain, $\mathcal{X}$, a ciphertext rank, $Rank_y$, and additionally the database query logs, $Q_{DB}$. In the end, $\mathcal{A}^Q$ is asked to return the underlying plaintext, $x$. The ROW advantage of $\mathcal{A}^Q$ is formulated as:

$$Adv^{\text{ROW}}_{\text{ES}}(\mathcal{A}^Q) = Pr[Exp^{\text{QLA}}_{\text{ES}}(\mathcal{A}^Q) = 1] \tag{3}$$

**Experiment 3 . $\text{Exp}^{\text{ROW}}_{\text{ES}}(\mathcal{A}^Q)$**

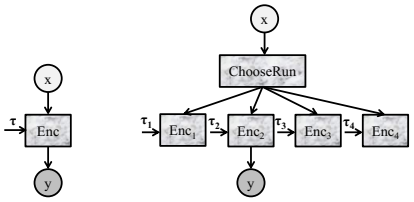
1: $K \xleftarrow{\$} \mathcal{K}eys; x \xleftarrow{\$} \mathcal{X}$
2: $y \leftarrow \mathcal{E}nc(K, x)$
3: $Rank_y \leftarrow rank(y)$
4: $x' \xleftarrow{\$} A^Q(\mathcal{X}, Rank_y, Q_{DB})$ ▷ note the additional argument $Q_{DB}$
5: **if** $x = x'$ **then return** 1
6: **else return** 0

## 4 Deterministic RPE

This section presents the deterministic *Randomly Partitioned Encryption* scheme (Det-RPE). The key idea of RPE is to take an existing weak encryption method such as OPE as in [1], [6], [7], [20], [34] as a building block and to enhance its security by applying it separately on different random partitions of the data called *Runs*. In contrast to other

partially order preserving encryption schemes, such as [26], [17] where they bin the plaintext domain in random-length (or fixed ranged) partitions, RPE creates partially ordered partitions called *Runs* by randomly assigning each domain value to a *Run*; thereby creating more uncertainty.

Figure 2a shows the workings of a traditional OPE encryption function. It receives a plaintext, $x$, and produces a ciphertext, $y$, and $\tau$ is the set of input parameters that $\mathcal{E}nc$ takes (e.g. a secret key). Figure 2b shows how RPE composes this traditional scheme to become more secure and have a number of additional operational advantages (e.g., support for updates). Instead of a single $\mathcal{E}nc$ function per domain, RPE makes use of $U$ encryption functions per domain, $\mathcal{E}nc_1, \mathcal{E}nc_2, \ldots, \mathcal{E}nc_U$, where $U$ is the number of *Runs*. These $U$ functions possibly all have the same structure (e.g. order-preserving using MOPE [6]) and just differ in the secret key they use. Given a plaintext, $x$, *ChooseRun* generates a number between 1 and $U$ and encrypts $x$ using the corresponding $\mathcal{E}nc$ function, i.e. $y = \mathcal{E}nc_{ChooseRun(x)}(x)$.




(a) OPE (b) RPE

**Fig. 2.** RPE Principle

Depending on the *ChooseRun* function and the structure of the $\mathcal{E}nc$ function, the composed encryption scheme of Figure 2b can have different properties. The performance overhead of RPE (as compared to no encryption) depends on the number of runs, $U$; In the extreme case of $U = 1$, RPE is the same as $\mathcal{E}nc$ which is typically a weak, yet high performance encryption scheme. In the other extreme, $U = \infty$, RPE is the same as *random* which corresponds to a strong yet low performance encryption scheme.

Deterministic RPE (abbreviated as Det-RPE) is a deterministic encryption scheme because a deterministic *ChooseRun* and a deterministic $\mathcal{E}nc$ function are used. Determinism of the *ChooseRun* function can be achieved by making its output depend on the plaintext, $x$, i.e. by using a *pseudo-random* function. Our Det-RPE construction has to generate two sets of keys, one set for the runs and the other for the *ChooseRun* function. The encryption function of Det-RPE then composes the *ChooseRun* function with the encryption function of a typical OPE scheme. Det-RPE can be composed with any OPE scheme from [1, 6, 7, 20, 33, 34].

**Table 2.** Det-RPE Example: $U = 2$

| Clear Text | Run 1 | Run 2 | Code ($\langle$ run, y $\rangle$) |
|---|---|---|---|
| Beatles | | 1 | $\langle 2, 1 \rangle$ |
| Beatles | | 1 | $\langle 2, 1 \rangle$ |
| Beatles | | 1 | $\langle 2, 1 \rangle$ |
| Elton John | 1 | | $\langle 1, 1 \rangle$ |
| Madonna | | 2 | $\langle 2, 2 \rangle$ |
| Madonna | | 2 | $\langle 2, 2 \rangle$ |
| Metallica | | 3 | $\langle 2, 3 \rangle$ |
| Nelly Furtado | 2 | | $\langle 1, 2 \rangle$ |
| Tina Turner | 3 | | $\langle 1, 3 \rangle$ |

**Construction 1.** *Let $\mathcal{OPE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ be a deterministic order-preserving encryption scheme. We define a deterministic RPE scheme, $\mathcal{D}et\text{-}\mathcal{RPE}(\mathcal{K}_{dRPE}, \mathcal{E}nc_{dRPE}, \mathcal{D}ec_{dRPE})$, as follows:*

- *$\mathcal{K}_{dRPE}$ runs $\mathcal{K}$ independently for each run and returns $U$ keys, namely $(K_1, ..., K_U)$. Also, it runs $\mathcal{K}$ independently to generate $K_{map}$ for the $\mathcal{C}hooseRun$ function.*

- $\mathcal{E}nc_{dRPE}$ *takes* $K_u$ *and* $x$*, as input where* $u = \mathcal{C}hooseRun(K_{map}, x)$*. Then it returns* $u$ *and* $y = \mathcal{E}nc(K_u, x)$*, i.e.* $(u, y)$*.*
- $\mathcal{D}ec_{dRPE}$ *takes* $(u, y)$ *as input and returns* $x = \mathcal{D}ec(K_u, y)$*.*

Table 2 gives an example of a set of customer names encrypted with *Det-RPE* with $U = 2$ using an OPE scheme where $\mathcal{U}$ denotes the set of runs and $|\mathcal{U}| = U$.

### 4.1 Analysis of Domain Attack

An ordinary OPE scheme such as in [1], [6], [7], [20], [23], [33], [34] allows the domain adversary, $\mathcal{A}^D$, to efficiently break the encryption by solely using sorting. In other words: $Adv^{\text{ROW}}_{\text{OPE}}(\mathcal{A}^D) = 1$. On the other hand, a Modular OPE scheme, as proposed in [6], is resilient against a Domain Attack as shown in Lemma 1.

**Lemma 1.** *ROW-advantage of* $\mathcal{A}^D$ *on Modular OPE is:*

$$Adv^{ROW}_{MOPE}(\mathcal{A}^D) = Pr[Exp^{ROW}_{MOPE}(A) = 1] = \frac{1}{X} \tag{4}$$

*Proof.* In order to win Experiment 1 on MOPE, the adversary needs to know the modular offset. Since offset is chosen randomly from $\mathcal{X}$, the adversary will win the game with a probability of $\frac{1}{X}$.

Nevertheless, MOPE is vulnerable to the Known Plaintext Attacks, where the adversary additionally has one or more plaintext-ciphertext pairs. In that case, MOPE collapses into OPE.

The Probabilistic OPE schemes such as in [11], [18], [35] have the following ROW advantage against a Domain Attack:

**Lemma 2.** *(proof in [25]) ROW-advantage of* $\mathcal{A}^D$ *on Prob-OPE is:*

$$Adv^{ROW}_{Prob\text{-}OPE}(\mathcal{A}^D) = Pr[Exp^{ROW}_{Prob\text{-}OPE}(\mathcal{A}^D) = 1] \quad = \frac{\binom{Rank_y - 1}{Rank_x - 1}\binom{Y - Rank_y}{X - Rank_x}}{\binom{Y-1}{X-1}} \tag{5}$$

RPE amends all variants of OPE schemes to a great extent by randomly partitioning the domain into *Runs*, thereby breaking the total order into $U$ partial orders. According to [5, 13], the problem of finding the right total order out of $U$ partial orders is classified as inapproximable.

**Lemma 3.** *(proof in [25]) ROW-advantage of* $\mathcal{A}^D$ *on Det-RPE is defined as:*

$$Adv^{ROW}_{Det\text{-}RPE}(\mathcal{A}^D) = Pr[Exp^{ROW}_{Det\text{-}RPE}(\mathcal{A}^D) = 1] \quad = \frac{\binom{Rank_x - 1}{Rank_{(y,r)} - 1}\binom{X - Rank_x}{\frac{X}{U} - Rank_{(y,r)}}}{\binom{X}{\frac{X}{U}}} \tag{6}$$

The probability distribution conforms to a negative hypergeometric probability distribution which applies to sampling without replacement from a finite population, in our case the domain, $\mathcal{X}$. As random selections are made from the population, each subsequent draw decreases the population causing the probability of success to change with each draw. The detailed proofs of Lemma 2 and 3 are quite involved and can be found in our technical report [25].

An important observation to make from Equations 5 and 6 is that the ROW advantage not only depends on the domain size but also on the rank of the plaintext in the domain. Consequently, extreme values of the domain (e.g. "AAA" or "ZZZ") are breaking the uniformity of the probability distribution. In order to fix this problem, Det-RPE can be composed with Modular OPE [6] to form a ring structure to hide these extreme values.

In Figure 3 the ROW advantage is plotted for each encryption scheme. As a baseline we plot the advantage of an adversary that outputs a random $x$ regardless of the domain. This is considered to be "ideal" and is exactly what Modular OPE [6] achieves. We see that Det-RPE helps an OPE scheme (in that case ROPE from [7]) with high ROW advantage to get close to the "ideal" threshold by increasing the number of runs.

The equations presented in this section can be tuned to meet the *user's negligibility requirement*. For example, in Equation 6, depending on the domain size, the number of runs can be tuned to meet the security/performance requirements of the system. The higher the number of runs, the closer one gets to the "ideal" threshold, but also the bigger is the performance overhead.

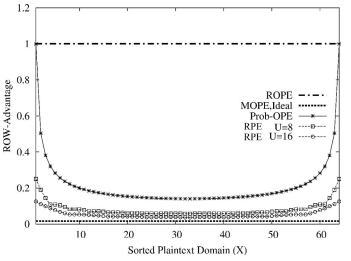



**Fig. 3.** ROW advantage under Domain Attack

**Table 3.** Prob-RPE with $U = 2$

| Customer Name | Run 1 | Run 2 | Prob-Code ($\langle$ run, y $\rangle$) |
|---|---|---|---|
| Beatles | | 1 | $\langle 2, 1\rangle$ |
| Beatles | | 2 | $\langle 2, 2\rangle$ |
| Beatles | 1 | | $\langle 1, 1\rangle$ |
| Elton John | | 3 | $\langle 2, 3\rangle$ |
| Madonna | 2 | | $\langle 1, 2\rangle$ |
| Madonna | | 4 | $\langle 2, 4\rangle$ |
| Metallica | 3 | | $\langle 1, 3\rangle$ |
| Nelly Furtado | 4 | | $\langle 1, 4\rangle$ |
| Tina Turner | | 5 | $\langle 2, 5\rangle$ |

## 5 Probabilistic RPE

In this section we present the probabilistic variant of Randomly Partitioned Encryption (Prob-RPE). RPE is made probabilistic in two ways: (a) by using a probabilistic order preserving encryption scheme within each run, such as proposed in [35] and (b) by assigning the same plaintext value to different runs to guarantee database dynamism and improve security.

In *Probabilistic RPE*, *ChooseRun* is defined as a *random function*. For each plaintext element, $x$, *ChooseRun* randomly selects a run, $u$. Afterwards, $x$ is to be encrypted in $u$. Table 3 gives an example of such a Prob-RPE encryption scheme where the same value can even have multiple codes within a single run. For instance, `Beatles` has two codes in *Run 2* and one code in *Run 1*, in other words $Beatles = \{\langle 2, 1\rangle, \langle 2, 2\rangle, \langle 1, 1\rangle\}$.

We formally construct Prob-RPE as follows:

**Construction 2.** *Let $\mathcal{P}rob\text{-}\mathcal{OPE}(\mathcal{K}_p, \mathcal{E}nc_p, \mathcal{D}ec_p)$ be a probabilistic order-preserving encryption scheme from [35]. We define a probabilistic RPE scheme, $\mathcal{P}rob\text{-}\mathcal{RPE}(\mathcal{K}_{pRPE}, \mathcal{E}nc_{pRPE}, \mathcal{D}ec_{pRPE})$, as follows:*

- *$\mathcal{K}_{pRPE}$ runs $\mathcal{K}_p$ independently for each run and returns $U$ keys, namely $(K_1, ..., K_U)$.*
- *$\mathcal{E}nc_{pRPE}$ takes $K_u$ and $x$, as input where $u = \mathcal{C}hooseRun()$. Then it returns $u$ and $y = \mathcal{E}nc_p(K_u, x)$, i.e. $(u, y)$.*
- *$\mathcal{D}ec_{pRPE}$ takes $(u, y)$ as input and returns $x = \mathcal{D}ec_p(K_u, y)$.*

The query rewrite mechanisms for RPE have been thoroughly explained in [25]. To give an intuition on how queries are rewritten using RPE, consider a query that asks for all customers with `name LIKE ''M%''` in the example of Table 3 for Prob-RPE. This query must be rewritten to:

```
SELECT * FROM Customer
  WHERE (run = 1 AND ( 1 < y < 4))
     OR (run = 2 AND ( 3 < y < 5))
```

### 5.1 Analysis of Frequency Attack

A known weakness of a deterministic encryption scheme is its inability to hide the original frequency distribution of the plaintext domain, specially if the plaintext domain has a skewed frequency distribution. Since OPE, Modular OPE and Det-RPE are all deterministic encryption schemes, the FOW-advantage introduced in Section 3.2 of $\mathcal{A}^{DF}$ depends on the plaintext frequency distribution, namely $F_{\mathcal{X}}$. In other words, if the original frequency distribution is uniform then the adversary's advantage is also uniform, i.e. best security. Nevertheless, if the original frequency distribution is skewed, then the adversary's advantage is different and depends on the number of elements having the same frequency.

**Corollary 1.** *Let $y = \mathcal{E}nc(K, x)$ and $G = \{w | w \in \mathcal{X} \wedge freq_{\mathcal{X}}(w) = freq_{\mathcal{Y}}(y)\}$ be the set of distinct plaintext values having the same frequency as $y$. Then, the FOW-advantage of the $\mathcal{A}^{DF}$ adversary on a deterministic encryption scheme is defined as his winning probability in Experiment 2:*

$$Adv_{DET}^{FOW}(\mathcal{A}^{DF}) = Pr[Exp_{DET}^{FOW}(A) = 1] = \frac{1}{|G|} \tag{7}$$

**Remark 1.** *A deterministic encryption scheme is optimally safe against a Frequency Attack if and only if $F_{\mathcal{X}}$ is uniform. A uniform $F_{\mathcal{X}}$ implies a maximum $|G|$ where $|G| = X$, i.e. $|G|$ equals the domain size.*

**Remark 2.** *Probabilistic OPE and Prob-RPE as described in this section, take care of a skewed probability distribution by each time creating a new ciphertext for a plaintext value. This way the skewed frequency distribution of the plaintext domain is mapped to a uniform frequency distribution in the ciphertext space i.e. the FOW advantage under frequency attack is optimal which is $\frac{1}{X}$.*

## 6 Fixed Range Query Rewrite

In Section 3.3, we have introduced an attack that is based on the information that an attacker can extract from the query logs. The query logs are said to be dangerous, when they carry more information about the underlying encryption scheme than the encrypted data in the database.

To see what type of queries can be dangerous for each of the encryption schemes presented in this paper, we introduce the concept of a *Query Simulator*. A *Query Simulator* generates queries by looking at the encrypted data. Thus, if we are dealing with a query which can be simulated just by looking at the encrypted data, that query is called *simulatable*. Otherwise, the query is dangerous and can be exploited to break the guarantees of the underlying encryption scheme. Typically, in both, probabilistic and partially order preserving schemes, such as in [17], [26], [35] to support efficient query processing, the queries reveal essential information about the certain initial values. Therefore, RPE conceals the total order which can be reconstructed by query log analysis. Table 4 presents a summary of the simulatable SQL operators in combination with the encryption schemes presented in this paper.

**Table 4.** SQL Operator Simulatability for different Encryption Schemes

| SQL-Operator | OPE | MOPE [6] | Prob-OPE [18], [35] | POP [17], [26] | Det-RPE | Prob-RPE | *-RPE-FR |
|---|---|---|---|---|---|---|---|
| WHERE (=, !=)/IN | ✓ | ✓ | × | × | ✓ | × | ✓ |
| WHERE (<, >)/LIKE(Prefix%) | ✓ | ✓ | × | × | × | × | ✓ |
| Equi-Join | ✓ | ✓ | × | × | ✓ | ✓ | ✓ |
| TOP N | ✓ | × | × | × | ✓ | × | ✓ |
| ORDER BY (sort) | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| MIN/MAX | ✓ | × | × | × | ✓ | × | ✓ |

Table 4 shows that all SQL Operators are simulatable for OPE. Nevertheless, OPE is not safe against *Query Log Attack* because the adversary has domain knowledge and can therefore launch a *Domain Attack*. Probabilistic OPE and Partially OPE schemes seem at first sight to be more secure, but on the other hand queries are not simulatable in those schemes, i.e. query log attack breaks them.

Our approach, RPE, is most of the time simulatable except for range queries. Thus, there are two solutions to this problem: (1) Suppress the range queries, (2) Using *Fixed Range Query Rewrite* mechanisms. In this section we will elaborate the second option.

The idea of *Fixed Range Query Rewrite* is to divide the ciphertext space, $\mathcal{Y}$ into $k$ disjoint fixed-sized sub-ranges of size $r$, $fr_i$ where $i = 1, ..., k$ and $|fr_i| = r$. Whenever a range query is asked, the smallest units that are returned are those *fixed ranges* that contain the user's results.

**Table 5.** Two fixed ranges of size 3, $r = 3$

| Fixed Range | Customer Names | Run 1 | Run 2 | Code ($\langle$ run, y $\rangle$) |
|---|---|---|---|---|
| $fr_1$ | Beatles | | 3 | $\langle 2, 3\rangle$ |
| | Elton John | 1 | | $\langle 1, 1\rangle$ |
| | Madonna | | 5 | $\langle 2, 5\rangle$ |
| $fr_2$ | Metallica | | 9 | $\langle 2, 9\rangle$ |
| | Nelly Furtado | 6 | | $\langle 1, 6\rangle$ |
| | Tina Turner | 7 | | $\langle 1, 7\rangle$ |

As an example, consider Table 5 where deterministic RPE has been used for encryption. The client submits the query: `SELECT name FROM customer WHERE name > 'Lady Gaga'`. Using Det-RPE the query should be rewritten to: `SELECT name FROM customer WHERE (run = 1 AND 3 < code < 10) OR (run = 2 AND 3 < code < 10)`.

However, since this query is revealing some information about the partition relationships, we will use fixed range query rewrite mechanisms. Thus, the original query will be rewritten as follows to cover both fixed ranges, $fr_1$ and $fr_2$ that contain the result: `SELECT name FROM customer WHERE (run = 1 AND 0 < code < 10) OR (run = 2 AND 0 < code < 10).`

This is the whole table in this example. The result includes two additional false positives, namely, "Beatles" and "Elton John" that will be filtered out during the post-processing. Obviously, using *Fixed Ranged Query Rewrite* returns a super-set of the result that needs to be post-filtered and leads to performance loss. On the other hand, the security guarantees and analysis will deviate from what we have discussed earlier in Section 4.

### 6.1 Analysis of Query Log Attack

According to Table 4, OPE schemes are safe against query analysis, however Modular OPE cannot safely handle min/max and rank queries. Probabilistic OPE cannot handle range queries without revealing the borders, so using *Fixed Ranged Query Rewrite* protects the encryption scheme from query log analysis. In case of RPE, range queries are problematic because they help to reconstruct the total order, which is exactly what RPE is trying to hide. Hence, *Fixed Ranged Query Rewrite* helps again to limit the ROW advantage under query log attack. Assuming we have a uniform code distribution within a range in all runs, Equation 6 changes into:

$$Adv^{\text{ROW}}_{\text{Det-RPE-FR}}(\mathcal{A}^Q) = Pr[Exp^{\text{ROW}}_{\text{Det-RPE-FR}}(\mathcal{A}^Q) = 1] \quad = \frac{\binom{Rank^{fr}_x - 1}{Rank^{fr}_{(y,r)} - 1}\binom{r - Rank^{fr}_x}{\frac{r}{u} - Rank^{fr}_{(y,r)}}}{\binom{r}{\frac{r}{U}}} \tag{8}$$

Informally speaking, the randomness is no more distributed throughout the domain, but is only available within the fixed range. From Equation 8 we reach the following conclusions:

- If $U = r$ then we have a uniform probability distribution. This means although no range query is possible within a fixed range (like having AES in each range). Among different fixed ranges, range queries are possible.
- Getting better security is achievable by both increasing the range size and the number of runs. However they both come at the cost of performance.
- Range size is the main parameter whereas the domain size does not play a role anymore.

## 7 Database Functionality

After the security assessment of RPE, we also want to compare its supported functionality to the state-of-the-art. Table 6 summarizes which SQL operators can be efficiently implemented for which encryption technique. We can see that the RPE variants support most of the desired SQL operators, which allows them to be used in practice while

semantically secure encryption schemes (like e.g. AES in CBC mode or fully homomorphic encryption schemes[12] become inefficient as soon as we want to ask `range`, `in`, `order-by` or `group-by` queries. RPE also allows indexes to be build on top of it as usual. For a full description on *query rewrite* and *referential integrity*, we refer the curious reader to our Technical Report [15].

**Table 6.** Supported SQL Operators: State of the Art vs. RPE

| SQL-Operator | OPE | MOPE[6] | POP[17, 26] | Det-RPE | Prob-RPE | FHE [12] | AES-CBC [19] |
|---|---|---|---|---|---|---|---|
| DISTINCT | ✓ | ✓ | × | ✓ | × | × | × |
| WHERE (=, !=) | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| WHERE (<, >) | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| LIKE(Prefix%) | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| LIKE(%Suffix) | × | × | × | × | × | × | × |
| IN | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| Equi-Join | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| Non Equi-Join | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| TOP N | ✓ | × | × | ✓ | × | × | × |
| ORDER BY | ✓ | ✓ | × | ✓ | ✓ | × | × |
| SUM | × | × | × | ✓ | ✓ | ✓ | × |
| MIN/MAX | ✓ | × | × | × | × | × | × |
| GROUP BY | ✓ | ✓ | × | ✓ | × | × | × |

## 8 Performance Analysis and Experimental Results

In order to measure to what extent missing SQL operator influence query performance, we implemented the TPC-H benchmark on the architecture shown in figure 1. We have executed a number of experiments with different varying parameters, e.g. the number of runs or the number of codes per value in a single run for Prob-RPE. We measured total response time, post-processing time, network cost and query compilation time. For space reasons we only present the most important results and reference to our technical report [25] for the complete numbers and explanations.

All experiments were conducted on two separate machines for client and server. The client was written in Java, ran on a machine with 24 GB of memory and communicated to the database server using JDBC. The server machine had 132 GB of memory available and hosted a MySQL 5.6 database. Both machines had 8 cores and ran a Debian-based Linux distribution. We measured end-to-end response time for all queries in separation, thereby using a scaling factor of 10 (which means that the size of the plaintext data set is 10 GB). Metrics used in aggregate functions (e.g., volume of orders) and surrogates (e.g., order numbers) were left unencrypted while all other (sensitive) attributes, such as names, dates, etc. were encrypted. Whenever SQL operators on encrypted data were not supported, the entire data was shipped and then aggregated and filtered during the post-processing step at the encryption layer. Wherever the TPC-H benchmark defines an index on an attribute $a$, we created the same index on the encrypted value of $a$ as well as a composite index on $(a\_run, a)$.

Figure 4 shows the response time for different encryption functions compared to *Plain*, which is the response time for query processing on unencrypted data. *Det-OPE* is a deterministic OPE scheme as proposed by [7] and was used as a gold standard to compare against two different RPE variants that both used fixed-range query rewrite. The deterministic RPE version is denoted by *Det-RPE-FR*, while *Prob-RPE-FR* stands

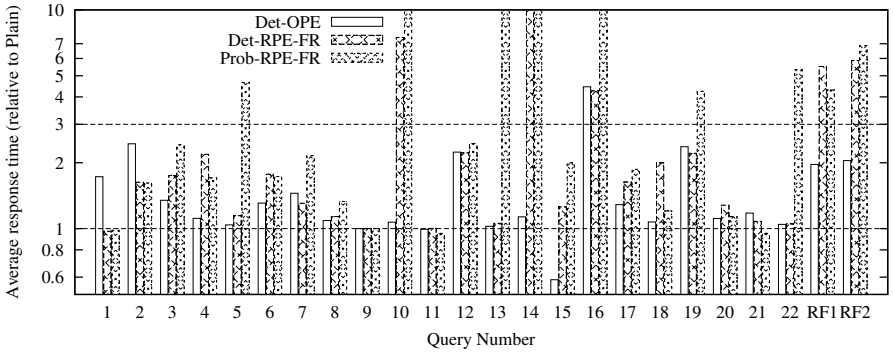



**Fig. 4.** TPC-H response times of different encryption functions relative to Plain, scaling factor 10

for the probabilistic version. We can see that for a majority of queries and RPE variants, response time is at most three times higher than *Plain* and twice as high as *Det-OPE*. Sometimes, it is even smaller than the baselines because the partitioning (together with the composite indexes) allows for parallelized query processing, which was employed in the presence of Top-N queries. As expected, *Prob-RPE-FR* can add significant overhead. However, we believe this overhead to be reasonable as the security gain is substantial as shown in the previous sections. We also measured the performance of deterministic AES-ECB [19], but did not include the results in the graph because most queries did not finish withing 30 minutes and therefore had an overhead ranging from about 20x to 200x. AES-CBC [19], a representative of semantically secure encryption schemes, would perform even worse.

The number of runs, $U$, is an important parameter of all RPE schemes. Figure 5 shows how the running time of Det-RPE-FR increases with the number of runs for the first six TPCH queries and relative to *Plain*. We chose these queries because they cover a wide spectrum of different operators (range predicates, aggregates, Top N, sub-selects, etc.). Overall, it can be seen that the curves are fairly flat. Only for Q6, there is a significant (but still linear) increase in the response time with a growing number of runs. Q2 and Q3 are Top N queries and are therefore executed with a degree of parallelism equal to $U$, which result in a decreasing response time. Q1, Q4 and Q5 have a similar behavior like most TPCH-queries: their curves stay fairly flat. This shows that partitioning does generally not hurt RPE.

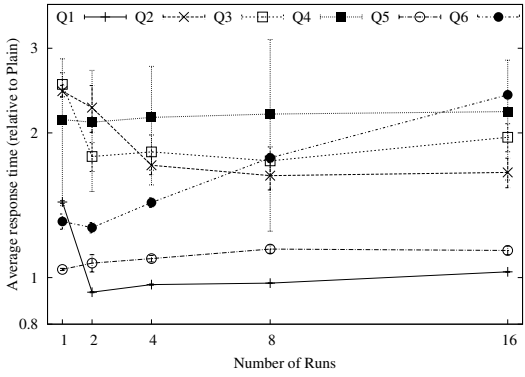



**Fig. 5.** Response Time of TPC-H Queries 1 to 6 for Det-RPE-FR, vary $U$ from 1 to 16

## 9 Related Work

To fill the gap between no security/high performance and high security/low performance, there have been a number of proposals for encryption techniques that support query processing without decrypting the data. Examples are [2], [9], [14]. However, these approaches return a superset of the desired result and lose a lot of time into decrypting and filtering out the false positives.

On the other hand, the goal of fully homomorphic encryption (FHE) [30], [10], [12], [22], [28] is to strongly encrypt the data and process it directly without decryption. [32] shows, at a conceptual level, how FHE can be used in databases. However, when taken to practice, databases explode in size because of the huge keys required by FHE. Therefore, the practicality of FHE in databases is an open question and no performance evaluation has been published so far.

Another important class of encryption techniques for databases are variants of order-preserving encryption (OPE), first introduced by Agrawal et al. [1]. Examples include *Random (Modular) OPE* [6], [7], *Mutable OPE* [23], *Indistinguishability-based OPE* [20], *Generalized OPE* [33], [34], *Structure Preserving Database Encryption* [11], *Probabilistic OPE* [35], *OPE with Splitting and Scaling* [31], *Multivalued OPE* [18], *Multivalued partial OPE* [17], and *Chaotic OPE* [26]. RPE is based on Random OPE and Modular Random OPE from [6], [7], but very different from all the other OPE derivatives in the sense that it either solves a different problem (i.e. addresses different attacks) or uses different techniques. Systems that exploit OPE are *CryptDB* [24] and *Monomi* [29]. RPE can be integrated into these systems to make them more secure.

Secure hardware has recently been exploited in the *TrustedDB* [4] and *Cipherbase* [3] projects. They are an interesting approach, but how to reach good performance, especially for analytical workloads, is still open. Again, RPE can be plugged-in into these systems in order to allow for faster, but still secure query processing.

Most commercial database products support strong encryption using AES [19], e.g. Oracle [21] and Microsoft SQL Server [16]. Unfortunately, they only support encryption for *data at rest* at the disk level, which means that these approaches do not address attacks issued by a curious database administrator or any other party that can access to the data in memory.

## 10 Conclusion

This paper presented different variants of *Randomly Partitioned Encryption*, a set of novel methods for encrypting cloud databases, thereby addressing real world attacker scenarios like *Domain*, *Frequency* and *Query Log* attacks. To the best of our knowledge, this is the first time that these attacks were formally defined and their success probability with respect to both, existing order-preserving encryption schemes, and the proposed RPE variants, analyzed. Moreover, the paper showed that the additional performance cost introduced by these new encryption schemes is reasonably small and gave a detailed overview of similar and related work in the literature.

## References


[1] Agrawal, R., et al.: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 563–574. ACM (2004)

[2] Agrawal, R., et al.: Privacy-preserving data mining. ACM Sigmod Record 29(2), 439–450 (2000)

[3] Arasu, A., et al.: Orthogonal Security with Cipherbase. In: CIDR. Citeseer (2013)

[4] Bajaj, S., et al.: TrustedDB: a trusted hardware based database with privacy and data confidentiality. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp. 205–216. ACM (2011)

[5] Berger, B., et al.: Approximation alogorithms for the maximum acyclic subgraph problem. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 236–243. Society for Industrial and Applied Mathematics (1990)

[6] Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: Improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011)

[7] Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009)

[8] Chow, R., et al.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, pp. 85–90. ACM (2009)

[9] Damiani, E., et al.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 93–102. ACM (2003)

[10] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)

[11] Elovici, Y., Waisenberg, R., Shmueli, E., Gudes, E.: A structure preserving database encryption scheme. In: Jonker, W., Petković, M. (eds.) SDM 2004. LNCS, vol. 3178, pp. 28–40. Springer, Heidelberg (2004)

[12] Gentry, C.: A fully homomorphic encryption scheme. PhD thesis. Stanford University (2009)

[13] Guruswami, V., et al.: Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In: IEEE 49th Annual IEEE Symposium on Foundations of Computer Science, pp. 573–582. IEEE (2008)

[14] Hacigümüş, H., et al.: Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 216–227. ACM (2002)

[15] Hildenbrand, S., et al.: Query processing on encrypted data in the cloud. Tech. rep. 735. Department of Computer Science, ETH Zurich (2011)

[16] Hsueh, S.: Database encryption in SQL server 2008 enterprise edition. Microsoft, SQL Server Technical Article (2008)

[17] Kadhem, H., et al.: A Secure and Efficient Order Preserving Encryption Scheme for Relational Databases. In: KMIS, pp. 25–35 (2010)

[18] Kadhem, H., et al.: MV-OPES: Multivalued-order preserving encryption scheme: A novel scheme for encrypting integer value to many different values. IEICE Transactions on Information and Systems 93(9), 2520–2533 (2010)

[19] Katz, J., et al.: Introduction to modern cryptography: principles and protocols. CRC Press (2007)
[20] Malkin, T., et al.: Order-Preserving Encryption Secure Beyond One-Wayness. Tech. rep. Citeseer (2013)
[21] Nanda, A.: Transparent Data Encryption. Oracle Magazine (2005)
[22] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
[23] Popa, R.A., et al.: An ideal-security protocol for order-preserving encoding. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 463–477. IEEE (2013)
[24] Popa, R.A., et al.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 85–100. ACM (2011)
[25] Sanamrad, T., et al.: POP: a new encryption scheme for dynamic databases. Tech. rep. 782. Department of Computer Science, ETH Zurich (2013)
[26] Seungmin, L., et al.: Chaotic order preserving encryption for efficient and secure queries on databases. IEICE Transactions on Information and Systems 92(11), 2207–2217 (2009)
[27] Sion, R.: Secure data outsourcing. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 1431–1432. VLDB Endowment (2007)
[28] Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010)
[29] Tu, S., et al.: Processing analytical queries over encrypted data. In: Proceedings of the 39th International Conference on Very Large Data Bases, pp. 289–300. VLDB Endowment (2013)
[30] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
[31] Wang, H., et al.: Efficient secure query evaluation over encrypted XML databases. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 127–138. VLDB Endowment (2006)
[32] Wang, S., et al.: Is Homomorphic Encryption the Holy Grail for Database Queries on Encrypted Data? Technical report, Department of Computer Science, UCSB (2012)
[33] Wozniak, S., et al.: Beyond the ideal object: towards disclosure-resilient order-preserving encryption schemes. In: Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop, pp. 89–100. ACM (2013)
[34] Xiao, L., et al.: A Note for the Ideal Order-Preserving Encryption Object and Generalized Order-Preserving Encryption. In: IACR Cryptology ePrint Archive 2012, p. 350 (2012)
[35] Yang, Z., Zhong, S., Wright, R.N.: Privacy-preserving queries on encrypted data. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 479–495. Springer, Heidelberg (2006)