Collusion-Resistant Processing of SQL Range Predicates

Manish Kesarwani[1], Akshar Kaul[1], Gagandeep Singh[1], Prasad M. Deshpande[2], and Jayant R. Haritsa[3]

[1]IBM India Research Lab [2]KENA Labs [3]Indian Institute of Science
{manishkesarwani, akshar.kaul, gagandeep_singh}@in.ibm.com
prasadmd@acm.org haritsa@iisc.ac.in

Abstract. Prior solutions for securely handling SQL range predicates in outsourced cloud-resident databases have primarily focused on *passive* attacks in the Honest-but-Curious adversarial model, where the server is only permitted to *observe* the encrypted query processing. We consider here a significantly more powerful adversary, wherein the server can launch an *active* attack by clandestinely issuing specific range queries via *collusion* with a few compromised clients. The security requirement in this environment is that data values from a plaintext domain of size N should not be leaked to within an interval of size H. Unfortunately, all prior encryption schemes for range predicate evaluation are easily breached with only $O(log_2\psi)$ range queries, where $\psi = N/H$. To address this lacuna, we present SPLIT, a new encryption scheme where the adversary requires *exponentially more* – $\mathbf{O}(\psi)$ – range queries to breach the interval constraint, and can therefore be easily detected by standard auditing mechanisms.
The novel aspect of SPLIT is that each value appearing in a range-sensitive column is first segmented into two parts. These segmented parts are then independently encrypted using a *layered composition* of a Secure Block Cipher with the Order-Preserving Encryption and Prefix-Preserving Encryption schemes, and the resulting ciphertexts are stored in separate tables. At query processing time, range predicates are rewritten into an equivalent set of table-specific sub-range predicates, and the disjoint union of their results forms the query answer. A detailed evaluation of SPLIT on benchmark database queries indicates that its execution times are well within a factor of *two* of the corresponding plaintext times, testifying to its efficiency in resisting active adversaries.

1 Introduction

Cloud computing has led to the emergence of the "Database-as-a-Service" (DBaaS) model for outsourcing databases to third-party service providers (e.g., Amazon RDS, IBM Cloudant). Accordingly, considerable efforts have been made over the last decade to devise encryption mechanisms that organically support query processing without materially compromising on data security. Here, we investigate this issue specifically with regard to *range predicates*, the core building blocks of decision-support (OLAP) queries on data warehouses.

Security Architecture A typical DBaaS setup consists of the entities shown in Figure 1, including: (i) a Service Provider (SP), who maintains the cloud infrastructure; (ii) a Data Owner (DO), who is the data source; (iii) a set of Query Clients (QC), who are authorized to issue queries over the data stored by DO on SP's platform, and (iv) a Security Agent (SA), who acts as the bridge connecting the DO and QC with the SP.

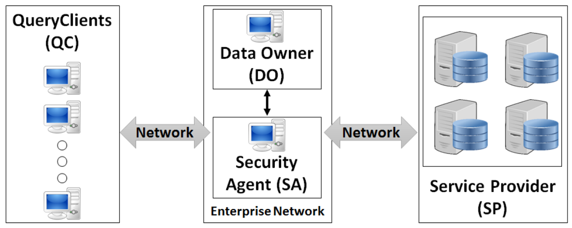


Fig. 1. System Entities in DBaaS model

The SA is a *trusted* entity, and could be a simple proxy in the DO's enterprise network. Alternatively, it could be located at the SP, implemented using secure threads or secure co-processors. Although all queries pass through the SA, it is a light-weight component since it is responsible only for query rewriting and decryption of the final results.

Adversary Model The SP, on the other hand, is always untrusted and treated as the primary adversary. We assume that the SP is only interested in deciphering the encrypted data, and not in affecting the functionality of the database system. That is, the query processing engine is in pristine condition, and all client queries are answered correctly and completely. Further, the SP maintains compliance with the standard access control and auditing mechanisms.

The Query Clients (QC) can either be trusted or untrusted, giving rise to the following alternative adversarial models:

(a) **Honest-but-Curious (HBC)**, in which the clients are trusted. Here, only *passive* attacks by the SP are possible – that is, the SP can try to breach the plaintext values solely by *observing* the encrypted data, and the computations executed by the database engine on this data. This model has been widely considered in the literature (e.g. [1,5,14,18,12,13,17]).
(b) **Honest-but-Curious with Collusion (HCC)**, in which the SP can unleash *active* attacks through *collusion* with a few compromised clients – specifically, the SP can *inject* range queries of its choice through the compromised QC, and then observe how these queries are processed by the database engine hosted at its site. Further, these injected queries can be constructed *adaptively*, using the results of previous queries. This powerful attack model was also recently considered in [8], as an *adaptive semi-honest* adversary.

CustName	LoanAmt	Collateral
Alice	50000	40000
Bob	24576	25000
Charlie	32000	28000
Dave	10000	8000

(a) Plaintext LOAN Table

CustName (AES)	LoanAmt (OPE)	Collateral (OPE)
Wsg^5j	5000340	173364
Sg*2js	1634009	35463
Uywhs@	4237461	65463
h7F&a1	738263	12073

(b) Encrypted LOAN_OPE Table

Fig. 2. Plaintext and OPE Banking Database

1.1 Example Security Breach under HCC

Consider a bank that has outsourced its relational database to the Cloud. Let the schema include a table LOAN *(CustName, LoanAmt, Collateral)* capturing the loans taken by customers, and the collaterals furnished to obtain these loans, as shown in Figure 2a. In order to simultaneously maintain security on the Cloud and support range query processing, the current practice is to employ one of the contemporary range encryption schemes – e.g. OPE [5] – on the sensitive *LoanAmt* and *Collateral* data columns, as shown in Figure 2b[1].

Assume that the bank provides a form-based interface to third-parties, such as auditors, analysts, etc. to query the encrypted data. For instance, a form to generate a report that lists all the loans of a customer (say *Alice*) in a given range – say [15000 : 40000], and the associated collaterals in another range – say [13000 : 33000]. The corresponding plaintext SQL query that is internally generated from the Web form is shown in Figure 3.

```
SELECT * FROM Loan WHERE
LoanAmt BETWEEN 15000 AND 40000 AND
Collateral BETWEEN 13000 AND 33000 AND CustName = 'Alice';
```

Fig. 3. Form-based SQL query with range predicates

Now suppose the HCC adversary comprises of the SP and the authorized auditors of customer *Alice*. In this setting, the security goal is to protect the adversary from learning the plaintext values of *LoanAmt* (and *Collateral*) for an *unrelated* customer from the encrypted LOAN_OPE table. However, the OPE-based encryption scheme can be easily breached for any target cell with just a few injected queries by *Alice's* auditors on LOAN_OPE. For instance, say the adversary selects the shaded tuple in LOAN_OPE as the target cell – corresponding to customer *Bob*. Then the attack proceeds as follows:

– The adversary first injects a query Q_1, similar to that of Figure 3, with the *LoanAmt* range set to [OPE(32768):OPE(65535)], *Collateral* range set to

[1] The CustName column is encrypted with AES for additional security.

[OPE(40000):OPE(40000)] [2] and CustName set to [AES('*Alice*')]. When Q_1 is processed by the database engine, the SP observes whether or not *Bob*'s encrypted LoanAmt lies in this range (note that the SP has unrestricted read access over encrypted data).

– Since it happens to lie outside the range, the adversary injects Q_2, which is identical to Q_1 except that the *LoanAmt* range is now set to [OPE(16384):OPE(32767)]. When Q_2 is executed, the SP finds that Bob's encrypted *LoanAmt* lies in the target range.
– The adversary then injects another similar query, Q_3, with *LoanAmt* now set to [OPE(24576):OPE(32767)].
– Since OPE(24576) is equal to Bob's encrypted *LoanAmt* value in LOAN_OPE, the HCC adversary learns that Bob's loan amount is 24576.

The above process is representative of an injection-based *binary search attack* (BSA) that becomes feasible via collusion. As explained in [20], it is also the *strongest* feasible attack in the HCC environment, and applicable to all security systems that store the encryption of a plaintext table in a single ciphertext table.

1.2 Range Predicate Security(RPS)

Before we address the above weakness, it is necessary to formalize the security definition in the HCC model. In this scenario, a plausible security formulation for SQL range predicates is that data values from a plaintext domain of size N should not be leaked to within an interval of size H on this domain. For instance, the bank may require that no loan amount should be leaked to within an interval of size 15000 from its actual value. Note that setting H to 1 corresponds to the special case where a security breach occurs only if a plaintext is *fully* leaked – this typically applies to identificatory attributes such as Social Security numbers.

Unfortunately, as highlighted in the BSA attack example, all previous schemes for range security can be breached under HCC with a sequence of only $\mathbf{O}(\mathbf{log_2}\psi)$ range queries, where $\psi = N/H$. To address this lacuna, we present here a new encryption scheme, called **SPLIT**, in which the HCC adversary requires *exponentially more* – i.e. $\mathbf{O}(\psi)$ – range queries to breach the interval constraint. Such extended query patterns can be easily detected by standard auditing mechanisms, or incur impractically long durations to achieve covertly, thereby effectively satisfying the interval security requirement.

We present a detailed evaluation of SPLIT on benchmark databases, and demonstrate that its execution times are always within *twice* the corresponding plaintext times, thus providing an attractive security-performance tradeoff against an extremely strong adversary. Further, while SPLIT does incur large storage overheads, the extremely low resource costs on the Cloud allow it to retain viability. Finally, SPLIT is attractive from a deployment perspective also since it can be implemented as a security layer over existing database engines, without necessitating internal changes.

[2] The *Collateral* range is fixed to a single value since the objective is to breach *LoanAmt*. A similar exercise can be carried out to break the *Collateral* column.

Organization The rest of the paper is organized as follows: We begin with the formal problem framework in Section 2. The new SPLIT encryption scheme, and its associated range query processing technique, are described in Sections 3 and 4, respectively. The security of SPLIT is analysed in Section 5, and the experimental results are presented in Section 6. Related work is reviewed in Section 7, and our conclusions are summarized in Section 8.

2 Problem Framework

As mentioned previously, the OPE and PPE schemes are currently in vogue for the secure handling of range queries, and are defined as follows:

Order-Preserving Encryption [**5**]: An order-preserving encryption function E_o is a one-to-one function from $A \subseteq \mathbb{N}$ to $B \subseteq \mathbb{N}$ with $|A| \leq |B|$, such that, for any two plaintext numbers $i, j \in A$, $E_o(i) > E_o(j)$ iff $i > j$.

Prefix-Preserving Encryption [**18**]: A prefix-preserving encryption function E_p is a one-to-one function from $\{0,1\}^n$ to $\{0,1\}^n$ such that, given two plaintext numbers a and b sharing a k-bit prefix, their corresponding ciphertexts $E_p(a)$ and $E_p(b)$ also share a k-bit prefix.

2.1 Adversary Objective

In accordance with the DBaaS model, the DO provides authorized access to portions of the data stored on the Cloud to individual QCs, using an access control mechanism and fixed query form templates. Further, the DO also defines the interval constraint size H. Given this environment, the adversary (i.e. SP + colluding QC) chooses to attack a *target cell* from an encrypted tuple which is *outside* of its authorized access, with the objective of breaching the Range Predicate Security (RPS) interval constraint H on this target cell.

Formally, the adversary $\mathcal{A}$ is given a set M^* consisting of m ciphertexts, and the interval constraint size H. $\mathcal{A}$ selects a challenge ciphertext $x^* \in M^*$ and its objective is to identify a plaintext interval (a, b) containing x^* such that $|b - a| < H$. In its attack, $\mathcal{A}$ is allowed to issue a polynomial(λ) number of range queries and observe their computations and results – here λ is the security parameter, corresponding to the bit-lengths of the plaintext values.

In the full version of this paper [20], the above attack model is formalized in the form of a *game* between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ for a deterministic encryption scheme $\mathcal{SE}$ that supports range query execution. We hereafter refer to this game as **Chosen Range Attack (CRA)**.

2.2 Notations

The following notations are used in the remainder of this paper:

- $x_p x_{p+1} \cdots x_q$ denotes extraction of bits p through q from the (big-endian) binary representation of x.

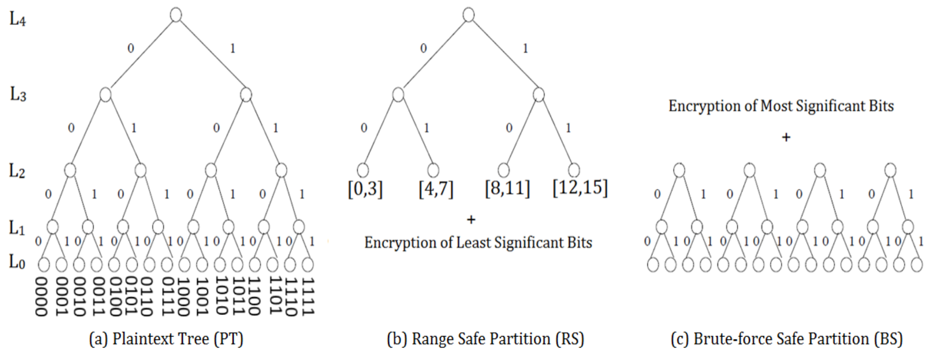
- $x_1||\cdots||x_k$ denotes the concatenation of bits $x_1, \cdots, x_k$, from which each x_i is uniquely recoverable.
- $\mathcal{P}$ denotes the plaintext domain. Further, given a plaintext value x, its encrypted version is denoted by x^*.
- N denotes the size of the plaintext domain, and H represents the size of the RPS interval constraint specified by the Data Owner. The *normalized* plaintext domain size is denoted by $\psi = \frac{N}{H}$.

3 Database Encryption with SPLIT

In this section, we present the design of the SPLIT encryption scheme, which is conceptually based on two main ideas of *splitting* and *layered encryption*. Subsequently, we describe how a plaintext database is converted to an encrypted database, followed by a rationale for the design choices.

3.1 Splitting of Data

If we consider plaintexts sourced from an n-bit integer domain, the entire set of these plaintexts can be represented by a complete binary tree of height n, referred to as the **Plaintext Tree (PT)**. The leaf level containing 2^n nodes is denoted as L_0, the level above it is denoted as L_1, and so on. For example, consider the plaintext tree for *4-bit* integers shown in Figure 4(a). In this case, n is 4 and PT contains nodes at 5 different levels, L_0 through L_4. Every node at the leaf level of PT is associated with *n-bits* of information characterizing its path from the root to level L_0.



L4
L3
L2
L1
L0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
(a) Plaintext Tree (PT)
[0,3]
[4,7]
[8,11]
[12,15]
+
Encryption of Least Significant Bits
(b) Range Safe Partition (RS)
Encryption of Most Significant Bits
+
(c) Brute-force Safe Partition (BS)

Fig. 4. Basic SPLIT Scheme

SPLIT partitions the levels of the PT into two contiguous groups, referred to as **Range Safe (RS)** and **Brute-force Safe (BS)**, respectively, and associated encrypted tables RS and BS are created based on this partitioning. The RS partition consists of the *top* levels of PT. For example, in Figure 4(a), levels

L_2 through L_4 belong to the RS partition, and the bits corresponding to these levels are encrypted for range query processing (this procedure is explained later in Section 3.2). Thus, in the encrypted RS table, for each plaintext value, the upper bits are encrypted for range query processing and the remaining bits are blinded using a Secure Block Cipher (SBC). Hence, in this example, nodes at level L_2 effectively serve as leaf nodes and the associated range for every such node is of granularity 2^2 integers, as shown in Figure 4(b).

The BS partition is comprised of the remaining levels of PT from level L_0 up to the level where the RS partition ends. In the current example, levels L_0 through L_2 are assigned to the BS partition, and the bits corresponding to these levels are encrypted for range query processing. Thus, in the encrypted BS table, the lower bits are encrypted for range query processing while the upper bits are blinded using SBC. This represents a set of trees, with the prefixes blinded, as shown in Figure 4(c).

3.2 Layered Encryption

SPLIT uses three encryption schemes as black boxes, namely, Secure Block Cipher ($\mathcal{E}_{SBC}$), Order Preserving Encryption ($\mathcal{E}_{OPE}$) and Prefix Preserving Encryption ($\mathcal{E}_{PPE}$). The SPLIT encryption scheme for plaintext domain $\mathcal{P}$ is constructed as a tuple of polynomial-time algorithms SPLIT $=$ $(KeyGen, \mathcal{E}_{BS}, \mathcal{E}_{RS}, \mathcal{E}_{SBC}, \mathcal{D}_{BS}, \mathcal{D}_{RS}, \mathcal{D}_{SBC})$, where $KeyGen$ is probabilistic and the rest are deterministic.

Key Generation $[\boldsymbol{sk \leftarrow KeyGen(\lambda, w, d)}]$ $KeyGen$ is a probabilistic algorithm that takes the following as input: The security parameter λ, the total number of table columns w, and the number of columns on which range predicates can be simultaneously applied d. It then outputs the secret key sk, which consists of $d * 2^d$ equi-length secret keys $(K_O^1, K_O^2, ..., K_O^{d*2^d})$ of the OPE encryption algorithm ($\mathcal{E}_{OPE}$), $d * 2^d$ equi-length secret keys $(K_P^1, K_P^2, ..., K_P^{d*2^d})$ of the PPE encryption algorithm ($\mathcal{E}_{PPE}$) and $w * 2^d$ equi-length secret keys $(K_S^1, K_S^2, ..., K_S^{w*2^d})$ of a Secure Block Cipher ($\mathcal{E}_{SBC}$).

Encryption Algorithms SPLIT incorporates two encryption algorithms $\mathcal{E}_{BS}$ and $\mathcal{E}_{RS}$. Both the algorithms are deterministic and take the following as input: the plaintext data item m, key for OPE encryption K_O, key for PPE encryption K_P, key for SBC K_S and number of bits u in the RS partition. The $\mathcal{E}_{BS}$ algorithm outputs the BS ciphertext (c^*_{BS}) while $\mathcal{E}_{RS}$ outputs the RS ciphertext (c^*_{RS}) corresponding to message m encrypted under the given keys. Let $l = n - u$, $m' = m_{n-1}m_{n-2}\cdots m_l$ and $m'' = m_{l-1}m_{l-2}\cdots m_0$, thus, $m = m'||m''$. Then,

– *Encryption for* BS $[\mathcal{E}_{\mathbf{BS}}(\mathbf{m}, \mathbf{K_O}, \mathbf{K_P}, \mathbf{K_S}, \mathbf{u})]$

$$c^*_{BS} \leftarrow \mathcal{E}_{OPE}^{K_O}(\mathcal{E}_{PPE}^{K_P}(\mathcal{E}_{SBC}^{K_S}(m')||m'')) \tag{1}$$

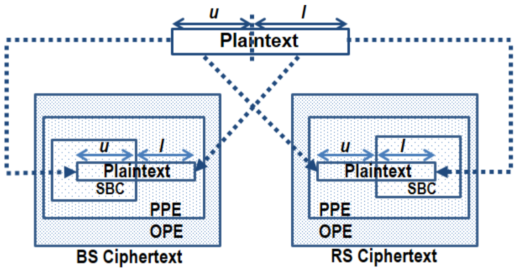


Fig. 5. SPLIT Ciphertext Construction

– *Encryption for* RS $[\mathcal{E}_{\mathbf{RS}}(\mathbf{m}, \mathbf{K_O}, \mathbf{K_P}, \mathbf{K_S}, \mathbf{u})]$

$$c^*_{RS} \leftarrow \mathcal{E}^{K_O}_{OPE}(\mathcal{E}^{K_P}_{PPE}(m' || \mathcal{E}^{K_S}_{SBC}(m''))) \tag{2}$$

The entire set of data encryption steps for a given plaintext value, as described above, is pictorially shown in Figure 5. The coressponding decryption algorithm is comprised of similar equations and is presented in [20].

3.3 Data Transformation

Consider a plaintext table with w columns, from which we wish to support range predicates on d columns. The plaintext values for each of the d columns are independently encrypted 2^{d-1} times using $\mathcal{E}_{BS}$ and $\mathcal{E}_{RS}$ each, thus creating 2^d ciphertext columns. Further, 2^d encrypted tables are created by capturing all BS and RS combinations of these columns. The remaining columns in the plaintext table – on which range queries will not be issued, are simply encrypted using an SBC.

We illustrate this data transformation process with the help of an example. Say our plaintext table is *Loan* with schema as enumerated in Figure 2a – then, $w = 3$. Assume that range predicates can only be asked on *LoanAmt* and *Collateral* columns, i.e. $d = 2$. First, we call $KeyGen(\lambda, 3, 2)$, which returns secret keys consisting of eight $(2*2^2)$ OPE keys $(K^1_O, K^2_O, \ldots, K^8_O)$, eight $(2*2^2)$ PPE keys $(K^1_P, K^2_P, \ldots, K^8_P)$, and twelve $(3*2^2)$ SBC keys $(K^1_S, K^2_S, \ldots, K^{12}_S)$. Next, we create four encrypted tables, as shown in Figure 6, which contain all combinations of the BS and RS partitions of *LoanAmt* and *Collateral*. Further, the physical row orderings of the tables are *randomized* to prevent *position-based* linkages across their tuples.

3.4 Design Rationale

The motivation for row randomization and layered encryption in SPLIT is to *prevent linkages* of tuples across the various encrypted tables. For example, there should be no linkage between tuples in LOAN_RS_RS and LOAN_BS_RS, both of which correspond to the RS partition of *Collateral*. If such a linkage

CustName	LoanAmt	Collateral
$E_{SBC}(K^1_S)$	$E_{BS}(K^1_O, K^1_P, K^2_S)$	$E_{BS}(K^2_O, K^2_P, K^3_S)$
Kjhd*&	7981328	18718
Rhwe#5	8374237	43628
Ywtw^2	237282	876213
iduhu7	918237	63782

(a) Loan_BS_BS

CustName	LoanAmt	Collateral
$E_{SBC}(K^1_S)$	$E_{BS}(K^1_O, K^1_P, K^2_S)$	$E_{BS}(K^2_O, K^2_P, K^3_S)$
&3y9W2	81927347	82723
ye@3^5	173687111	276372
tsfgU7	23193821	72376
lwUn7e	5362819	92836

(b) Loan_BS_RS

CustName	LoanAmt	Collateral
$E_{SBC}(K^1_S)$	$E_{BS}(K^1_O, K^1_P, K^2_S)$	$E_{BS}(K^2_O, K^2_P, K^3_S)$
&weu7w	7643	92837
Hwe^2h	2387	73648
7Shsu%	38272	27381
Sah#8s	2938	46372

(c) Loan_RS_BS

CustName	LoanAmt	Collateral
$E_{SBC}(K^1_S)$	$E_{BS}(K^1_O, K^1_P, K^2_S)$	$E_{BS}(K^2_O, K^2_P, K^3_S)$
Uye7^y	736473	83827
82&^ey	546378	74812
usgE6&	738272	328363
hs&6Hj	283749	83636

(d) Loan_RS_RS

Fig. 6. SPLIT Banking Database

exists, it can be used to connect the tuples on the *Collateral* column in the two tables, thereby enabling a binary search attack by keeping this column fixed, and searching on the other *LoanAmt* column.

Further, the *Collateral* values are encoded using the same RS *Encrypt* function, but with different keys in Loan_RS_RS and Loan_BS_RS. This is where the layered encryption, using OPE and PPE, plays a role. In both these columns, the lower l bits are blinded using an SBC with different keys, so it is not possible to link tuples based on the lower bits. However, if no further encryption is used, i.e. the upper u bits are kept as plaintext, it would be possible to link the tuples based on the upper bits. So, further encryption that enables range queries based on the upper u bits is necessary. Clearly, OPE and PPE are possible schemes that can be used. However, OPE by itself is not sufficient. Consider a set of values $\mathcal{V}$ encrypted using OPE with two different keys giving sets $\mathcal{V}_1$ and $\mathcal{V}_2$. Since OPE preserves order, the order of encrypted values in $\mathcal{V}_1$ and $\mathcal{V}_2$ is identical. Thus, by sorting these sets, one could link their values.

Similarly, PPE by itself is not secure since it preserves the structure of the tree corresponding to the binary representation. In some cases, it may be possible to map nodes across two PPE trees by using the structure. For example, if in the plaintext domain, there is a single value with bit $n-1$ as 1 and all others have bit $n-1$ as 0, then this value can be linked across different PPE trees, irrespective of whether bit $n-1$ gets flipped or not.

In a nutshell, the advantage of OPE is that it destroys the structure of the tree and the advantage of PPE is that it destroys the order information. Thus, by combining OPE with PPE, we remove both order and structure-based linkages.

4 Range Query Processing

In this section, we explain how a range query is executed over a SPLIT-encrypted database. The main idea is to transform the query range into a disjoint set of prefix ranges of the form $b_{n-1}b_{n-2}\cdots b_j*$, where each b_i is a bit taking value 0 or 1, and $*$ can match any value. Smaller ranges, corresponding to $j < l$, are answered from the BS tables and the larger ranges from the RS tables. Formally Range Query Processing consists of two main steps – Range Query Mapping and Range Query Execution, as described below.

4.1 Range Query Mapping

The steps to map range predicates from the plaintext domain to the RS and BS partitions are shown in RQM Algorithm 1. The mapping process starts by converting the input range r into a set of ranges $\mathcal{R}$ represented by prefixes (Line 1). The maximum number of such ranges is $2 * (n - 1)$, where n is the number of bits used for representing the attribute values [18]. For each prefix in $\mathcal{R}$, a value with that prefix is chosen – the remaining unspecified bits are set to 0 (Line 4). Then, depending on the size of the range represented by the prefix, it is mapped to either the RS or the BS partition. For a BS range, the higher order bits are encrypted with the SBC (Line 7). Then the value is encrypted with PPE encryption (Lines 8, 10). The lower and upper bounds of the range in the PPE encrypted domain are computed by replacing the remaining lower j bits by all 0 and by all 1 (Lines 12 – 13). Finally, these lower and upper bits are further encrypted using OPE encryption with the appropriate keys and the range is added to R_{BS} or R_{RS}, depending on the size of the range (Lines 14 – 20). It can be seen that due to the prefix-preserving property of PPE and the order preserving property of OPE, this mapping produces the correct range on the encrypted domain. The ranges in R_{RS} are answered from the RS partition, and those from R_{BS} are answered from the BS partition.

The above walkthrough shows the range mapping for a single column. If there are ranges on multiple columns, each range is split into prefixes and the set of all combinations of prefixes together represents the full range of the original query. Each combination is answered from the table corresponding to the range types. For example, a BS range on the *LoanAmt* column combined with BS range on the *Collateral* column is answered from the LOAN_BS_BS table.

4.2 Range Query Execution

The next step is to execute the ciphertext queries at SP. We illustrate this process through the example plaintext query specified in Figure 3. The following steps are performed to evaluate this query in SPLIT :

1. QC sends the plaintext query to the SA.
2. SA calls RQM Algorithm 1 and identifies sub-ranges over ciphertext tables.
3. Using output of Step 2, SA creates ciphertext sub-queries and sends them to SP.
4. SP executes the sub-queries and sends (encrypted) result tuples to the SA.
5. SA computes the union of the tuples returned from each sub-query, and then decrypts the result tuples. (The union is efficiently computable because it is apriori known that the sub-queries access *disjoint* sets of tuples.)

5 Security Analysis of SPLIT

In this section, we evaluate the Range Predicate Security offered by the SPLIT scheme against a Honest-but-Curious with Collusion adversary mounting a Chosen Range Attack. Specifically, in a binary search attack as the range is refined,

Algorithm 1 ***Range Query Mapping (RQM)***

```
Input: Range $r$ on plaintext attribute. OPE keys $K_O^1$ and $K_O^2$, PPE keys $K_P^1$ and $K_P^2$,
    SBC keys $K_S^1$ and $K_S^2$ for RS and BS partition respectively. The number of bits in
    RS partition '$u$'
Output: Set of ranges on RS partition $R_{RS}$, set of ranges on BS partition $R_{BS}$
 1: Convert $r$ into a set of ranges $\mathcal{R}$ of form $b_{n-1}b_{n-2}\cdots b_j*$ {using technique in [18]}
 2: Let $l = n - u$
 3: for all ($r_i = b_{n-1}b_{n-2}\cdots b_j*$) in $\mathcal{R}$ do
 4:     $v \leftarrow b_{n-1}b_{n-2}\cdots b_j0\cdots 0$ {set lower bits to 0}
 5:     $v_U \leftarrow v_{n-1}v_{n-2}\cdots v_l$ ; $v_L \leftarrow v_{l-1}v_{l-2}\cdots v_0$
 6:     if ($j < l$) then {BS range}
 7:         $v^* \leftarrow \mathcal{E}_{K_S^2}(v_U)||v_L$
 8:         $e_v^* \leftarrow \mathcal{E}_{K_P^2}(v^*)$
 9:     else {RS range}
10:         $e_v^* \leftarrow \mathcal{E}_{K_P^1}(v)$
11:     end if
12:     Let $c_n c_{n-1}\cdots c_0$ be the bit representation of $e_v^*$
13:     $r_L \leftarrow c_{n-1}c_{n-2}\cdots c_j0\cdots 0$; $r_U \leftarrow c_n c_{n-1}\cdots c_j1\cdots 1$
14:     if ($j < l$) then
15:         $r_L^* \leftarrow \mathcal{E}_{K_O^2}(r_L)$ ; $r_U^* \leftarrow \mathcal{E}_{K_O^2}(r_U)$
16:         Add $(r_L^*, r_U^*)$ to $R_{BS}$
17:     else
18:         $r_L^* \leftarrow \mathcal{E}_{K_O^1}(r_L)$ ; $r_U^* \leftarrow \mathcal{E}_{K_O^1}(r_U)$
19:         Add $(r_L^*, r_U^*)$ to $R_{RS}$
20:     end if
21: end for
22: return $R_{RS}$, $R_{BS}$
```

the table from which the query is answered is switched from RS to BS according to the RQM Algorithm 1. So, a target RS cell cannot be guessed to a range of size less than 2^l. And there is no way to reach the corresponding target cell in BS table in $log(\psi)$ steps unless the rows in the tables can be linked. Without linkage, binary searches over all the ψ sub-trees in the BS partition will be needed. We prove that the table rows cannot be correlated in the following discussion.

For ease of understanding, a diagrammatic view of the layered SPLIT encryption scheme is shown in Figure 7. [3] The various ways in which RPS for the *LoanAmt* column can be breached are highlighted through the numbered dotted lines, which are explained below – a similar reasoning holds for the *Collateral* column. The SPLIT scheme protects against all these breaches, as explained in the remainder of this section.

To begin with, the HCC adversary is unable to independently break the BS and RS ciphertexts (dotted lines 1 and 2, respectively) because these were generated by SBC-encrypting the upper and lower half bits of the plaintext

[3] For visual clarity, *CustName* is not shown in the figure, but its encrypted form, *CustName_Enc*, is present in all four tables.

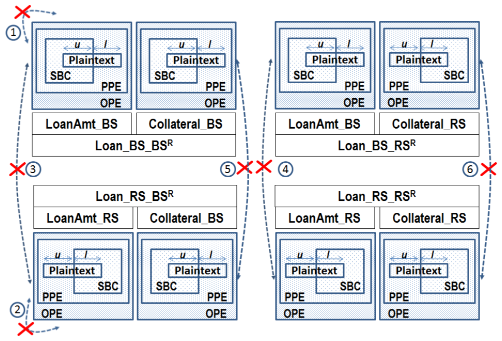


Fig. 7. Ensuring Security of *LoanAmt* values

value, respectively. Secondly, the BS and RS ciphertexts (dotted lines 3 and 4) corresponding to a given *LoanAmt* plaintext value, cannot be associated, because there is no value linkage between these ciphertexts – again due to the blinding of the lower half bits in the RS table and the upper half bits in the BS table using a SBC. Further, the linkages of row locations between these tables have been removed due to the randomization (denoted by R in the figure) of the physical row orderings of the tables. Preventing this association ensures a break in the chain of attack queries.

Apart from these direct attacks on *LoanAmt*, there could also be *indirect* attacks launched on it via the sibling *Collateral* attribute. Specifically, the linkage between a pair of BS ciphertexts corresponding to a *Collateral* plaintext value (dotted line 5), or a pair of RS ciphertexts corresponding to a *Collateral* plaintext value (dotted line 6), could be used to launch a BSA on *LoanAmt*. This is prevented because physical randomization ensures the absence of row linkages between the encrypted *Collateral* columns, while value linkages are eliminated by the three-layered SBC-PPE-OPE encryption, using different keys for each table, as described in Section 3.

In a nutshell, the security of the SPLIT encryption scheme is established based on the following points (the complete set of formal claims and proofs are available in [20]):

1. The BS and RS encryptions are independently secure (dotted lines 1 and 2 in Figure 7) .
2. For any plaintext table, there is no linkage between the corresponding BS and RS ciphertext tables (dotted lines 3 and 4 in Figure 7).
3. For any plaintext table, there is no linkage between a pair of corresponding BS (or two RS) ciphertext tables (dotted lines 5 and 6 in Figure 7).

6 Experimental Evaluation

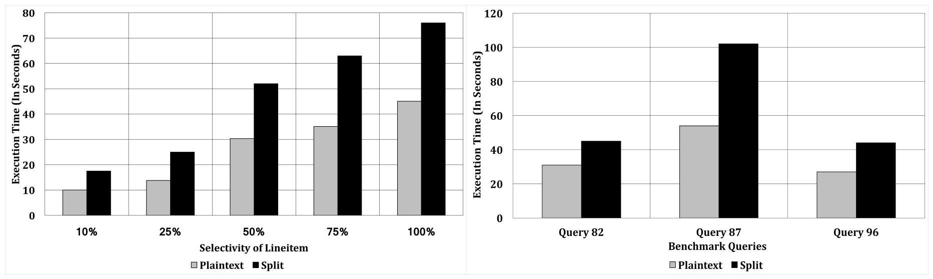
The importance of range predicates in OLAP environments can be gauged from the fact that more than half the queries in the TPC-H and TPC-DS decision support benchmarks feature such predicates. In this section, we move on to empirically evaluating SPLIT's efficiency with regard to handling range predicates in the encrypted domain.

Our experimental setup consisted of two identical server machines, with one representing the SP hosting the DO's encrypted data, and the other representing the SA interfacing with the QCs. PostgreSQL 9.4 was used as the database engine on the SP server, and all queries were issued through a Java program, which converted the plaintext queries to their SPLIT ciphertext equivalents.

The experiments were carried out on 10 GB versions of the TPC-H and TPC-DS benchmark databases. For TPC-H, the queries having range predicates on 4 attributes were constructed, with a range of selectivities on LINEITEM, the largest table in the TPC-H schema with 60 million rows. For TPC-DS, the standard benchmark tables sizes [21] were used and three benchmark queries (Query 82, Query 87 and Query 96) were executed to evaluate the performance.

6.1 Query Execution Time

The execution times taken for range query processing by the SPLIT and plaintext algorithms on the TPC-H and TPC-DS databases, as per the above experimental framework, are captured in Figures 8a and 8b, respectively, The results in these figures consistently show that the performance of SPLIT is within a factor of *two* of the plaintext query execution. For instance, in Figure 8a at 50% selectivity, the plaintext query takes around 30 seconds while SPLIT completes in 52 seconds. Similarly, in Figure 8b, Query 82 takes 32 seconds in the plaintext environment, and is computed in 45 seconds with SPLIT encryption.



(a) 10GB TPC-H database (b) 10GB TPC-DS database

Fig. 8. Query Execution Time on Benchmark Databases

At first look it may seem that SPLIT will incur a performance slowdown equal to the storage blowup. However such worst case scenario will require a query containing multi dimensional range predicate where each predicate has high selectivity requiring a full table scan. In general cases, if indexes are present and are chosen by the optimizer, then the number of tuples fetched from the disk will be equal to the size of the final result set. In these cases the performance overhead will be within two times since the ciphertext size is twice the size of the plaintext. Further note that since the query rewriting leads to multiple queries, each with predicates having lesser selectivity, the probability that the optimizer decides to use indexes is higher.

Note that the good performance of SPLIT is *inspite* of the large number of sub-queries in the transformed query. This is because each sub-query accesses a *disjoint* set of tuples, meaning that the total work done is almost equivalent to that of the single query in the plaintext domain, particularly if indexes are used in the query plan. This points to the practicality of the SPLIT scheme.

An important observation here is that the SPLIT implementation in these experiments lacked any *parallelization*. However, the many sub-queries (one per encrypted table) in the transformed query over the encrypted database can, in principle, all be executed in parallel. If this optimization were to be implemented, the time overheads will be further reduced.

6.2 Storage Cost

The size of the plaintext TPC-H database with indexes is 21 GB, whereas the corresponding SPLIT encrypted database is 335 GB. This is because we are handling 4D range predicates, resulting in the encrypted database being roughly 16 times the size of the plaintext database. Though this blowup is certainly large, the overall impact on the system *dollar cost* is substantively lower, since storage is relatively cheap. For instance, Table 1 shows the monthly costs for attaining same throughput with both the plaintext and SPLIT schemes, estimated using the rates charged by Amazon's AWS service [19] for machines similar to our experimental configuration. Since the execution time of SPLIT is within twice of the plaintext execution time, and the resource cost is dominated by the VM rental duration, the overall monetary investment in the SPLIT scheme is also within a factor of *two* with respect to the plaintext scheme. Further, various workload-dependent optimizations to reduce the storage overheads are also described in [20].

Scheme	Size (GB)	\$/VM	\$/GB	\$(VM)	\$(Storage)	\$(Total)
Plaintext	21	288	0.045	288	0.945	288.945
SPLIT	335	288	0.045	576	15.075	591.075

Table 1: Monthly Dollar Cost of Cloud Platforms

7 Related Work

Several schemes have been proposed over the last decade for securely processing range predicates over outsourced encrypted databases. The most prominent among them have been **OPE** [1,5,6,14,11] and **PPE** [18,12], which inevitably leak order-based and structure-based characteristics respectively, of plaintext data. In **PBtree** [13] the authors have proposed an encrypted tree-based index structure, but this scheme requires significant changes to the underlying database engine which may hinder its adoption by industry.

Subsequently, alternative tree-based encryption schemes have been proposed in [7,8] and **Bucketing Schemes** are proposed in [9,10]. These schemes provide stronger security guarantees than **OPE** schemes in Honest-but-Curious model. However the fundamental problem is that, these schemes return *false positives* in the query results.

Another line of research [15,16,3,2,17] has focused on building complete systems which support secure execution of entire SQL queries over encrypted databases. In **CryptDB** [15], multiple encryption schemes are used to encrypt the data in an "onion"-style layering. At query processing time, the outer layers of the appropriate onions are removed as dictated by the query predicates. **MONOMI** [16] also uses multiple encryption schemes, albeit without the onion-based layering. It assumes instead that the clients also have a local database engine, and each query is split into two parts – the first part is executed on the encrypted data at the Cloud server, and its result is transferred to the client, decrypted and loaded into the local database. The second part of the query is then run on this local plaintext database.

Systems such as **TrustedDB** [3] and **Cipherbase** [2] assume the availability of trusted hardware at the server, which can be used to decrypt and process the data in a secure manner. In TrustedDB, the whole database engine runs inside the trusted hardware, whereas in Cipherbase, the database engine is aware of the encryption requirements and integrates tightly with trusted hardware.

The common limitation of all the above systems is that they are susceptible to a CRA attack in the HCC model, as described in detail in [20].

8 Conclusions

In this paper we considered a Honest-but-Curious with Collusion adversary on Cloud-resident databases. This model represents a significantly more powerful attack than the traditional HBC adversary, and is capable of easily launching Chosen Range Attack to breach the encrypted data. We proposed the SPLIT encryption scheme to securely process range predicates in the presence of such adversaries, with the key features being splitting of data values and layered encryption. With this scheme the adversary requires exponentially more queries to breach the data, making the attack unviable in practice. SPLIT was implemented and evaluated on benchmark environments, and the experimental results demonstrate that its strong security guarantees can be supported without incurring more than a doubling of the plaintext response time, even under sequential execution. When parallel execution is implemented, these performance overheads will be much smaller.

In the full version of this paper [20], we have shown how SPLIT can be extended to handle updates and other database operators, as well as serve as a potent and efficient replacement for OPE in complete systems such as Cipherbase. Therefore, in an overall sense, SPLIT promises to be a viable and desirable component for securely handling OLAP queries.

In our future work, we plan to compare the efficiency of our work with other solutions in the HBC model (ex. **PBtree** [13]), and to design encryption schemes to securely handle additional SQL operators (ex. θ join) against HCC adversaries.

References

1. R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, "Order-Preserving Encryption for Numeric Data", *Proc. of ACM SIGMOD Conf., 2004.*
2. A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, R. Venkatesan, "Orthogonal security with cipherbase", *Proc. of CIDR Conf., 2013.*
3. S. Bajaj and R. Sion, "Trusteddb: A trusted hardware based outsourced database engine", *PVLDB, 4(12), 2011.*
4. M. Bellare, T. Ristenpart, P. Rogaway, T. Stegers, "Format-Preserving Encryption", *Proc. of Selected Areas in Cryptography Conf., 2009.*
5. A. Boldyreva, N. Chenette, Y. Lee, A. ONeill, "Order-preserving symmetric encryption", *Proc. of EUROCRYPT Conf., 2009.*
6. A. Boldyreva, N. Chenette, A. O'Neill, "Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions", *Proc. of CRYPTO Conf., 2011.*
7. J. Chi, C. Hong, M. Zhang, Z. Zhang, "Fast Multi-dimensional Range Queries on Encrypted Cloud Databases", *Proc. of DASFAA Conf., 2017.*
8. I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, M. Garofalakis, "Practical private range search revisited", *Proc. of ACM SIGMOD Conf., 2016.*
9. H. Hacigümüs, B. R. Iyer, C. Li, S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model", *Proc. of ACM SIGMOD Conf., 2002.*
10. B. Hore, S. Mehrotra, G. Tsudik, "A privacy-preserving index for range queries", *Proc. of VLDB Conf., 2004.*
11. F. Kerschbaum, "Frequency-Hiding Order-Preserving Encryption", *Proc. of CCS Conf., 2015.*
12. J. Li, E. R. Omiecinski, "Efficiency and Security Trade-Off in Supporting Range Queries on Encrypted Databases", *Proc. of DBSec Conf., 2005.*
13. R. Li, A. X. Liu, A. L, Wang, B. Bruhadeshwar, "Fast range query processing with strong privacy protection for cloud computing", *PVLDB, 7(14), 2014.*
14. R. A. Popa, F. H. Li, N. Zeldovich, "An Ideal-Security Protocol for Order-Preserving Encoding", *Proc. of IEEE Symp. on Security and Privacy, 2013.*
15. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: processing queries on an encrypted database", *Comm. of the ACM, 55(9), 2012.*
16. S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data", *PVLDB, 6(5), 2013.*
17. W. K. Wong, B Kao, D. W. Cheung, R. Li, S. Yiu, "Secure query processing with data interoperability in a cloud database environment", *Proc. of ACM SIGMOD Conf., 2014.*
18. J. Xu, J. Fan, M. H. Ammar, A. B. Moon, "Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme", *Proc. of ICNP Conf., 2002.*
19. *https://aws.amazon.com/ec2/pricing/*
20. *http://dsl.cds.iisc.ac.in/publications/report/TR/TR-2016-01.pdf*
21. *http://www.tpc.org/tpcds/*