Data Science and Engineering (2018) 3:323–340
https://doi.org/10.1007/s41019-018-0081-5

Collusion-Resistant Processing of SQL Range Predicates

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

Received: 1 August 2018 / Revised: 29 October 2018 / Accepted: 10 November 2018 / Published online: 20 November 2018

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 its efficiency in resisting active adversaries.

Keywords Security · SQL · Range · Cloud

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 Fig. 1, including: (1) a service provider (SP), who maintains the Cloud infrastructure; (2) a data owner (DO), who is the data source; (3) a set of query clients (QC), who are authorized to issue queries over the data stored by DO on SP's platform, and (4) a security agent (SA), who acts as the bridge connecting the DO and QC with the SP.

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 lightweight component since it is responsible only for query rewriting and decryption of the final results.

✉ Manish Kesarwani
manishkesarwani@in.ibm.com

Akshar Kaul
akshar.kaul@in.ibm.com

Gagandeep Singh
gagandeep_singh@in.ibm.com

Prasad M. Deshpande
prasadmd@acm.org

Jayant R. Haritsa
haritsa@iisc.ac.in

[1] IBM Research, Bangalore, India

[2] Kena Labs, Bangalore, India

[3] Indian Institute of Science, Bangalore, India

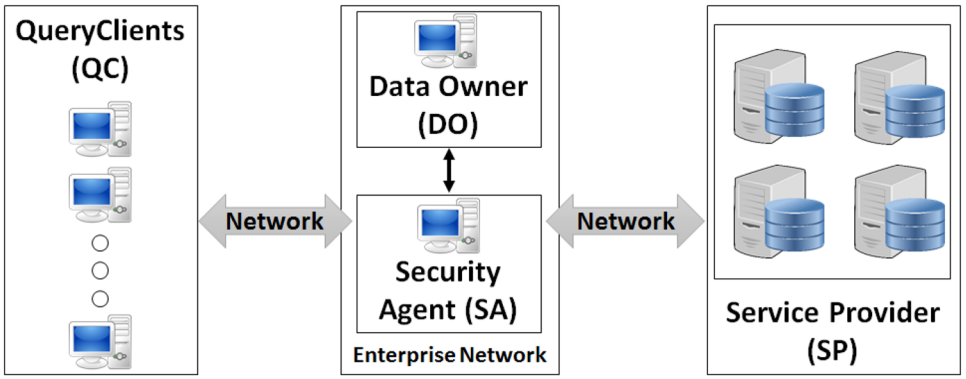

Fig. 1 System entities in DBaaS model

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

(a)

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

(b)

Fig. 2 Plaintext and OPE banking database. **a** Plaintext LOAN table. **b** Encrypted LOAN_OPE table

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

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 these data. This model has been widely considered in the literature (e.g., [1, 5, 15, 16, 18, 21, 22]).

(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 recently considered in [9], as an *adaptive semi-honest* adversary.

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 Fig. 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 Fig. 2b.[1]

Assume that the bank provides a form-based interface to third parties, such as auditors and analysts, 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 [15,000:40,000], and the associated collaterals in another range—say [13,000:33,000]. The corresponding

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

plaintext SQL query that is internally generated from the Web form is shown in Fig. 3.

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, assume that 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 Fig. 3, with the *LoanAmt* range set to [OPE(32,768):OPE(65,535)], *Collateral* range set to [OPE(40,000):OPE(40,000)],[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(16,384):OPE(32,767)]. 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(24,576):OPE(32,767)].
- Since OPE(24,576) is equal to Bob's encrypted *LoanAmt* value in LOAN_OPE, the HCC adversary learns that Bob's loan amount is 24,576.

The above process is representative of an injection-based *binary search attack* (BSA) that becomes feasible via collusion. As proved in [14], if the data distribution is not known, then no search algorithm that is based on comparison of data items can exhibit better worst-case performance than binary search. Therefore, BSA is also the *strongest* feasible attack in the HCC environment—it is applicable to all security systems that store the encryption of a plaintext table in a single ciphertext table and allow comparisons of ciphertexts in the encrypted domain.

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 15,000 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 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 require impractically long durations to achieve covertly or can be easily detected by standard auditing mechanisms. Therefore, the interval security requirement is effectively satisfied in either case.

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 trade-off 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.

Organization

The rest of the paper is organized as follows: We begin with the formal problem framework in Sect. 2. The new SPLIT encryption scheme and its associated range query processing technique are described in Sects. 3 and 4, respectively. The security of SPLIT is analyzed in Sect. 5, and the formal proofs are derived in Sect. 6. The experimental results are presented in Sect. 7, while deployment and integration issues are discussed in Sect. 8. Related work is reviewed in Sect. 9, and our conclusions are summarized in Sect. 10.

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 $A \subseteq \mathbb{N}$

[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.

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 [22] 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 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 $\psi = \frac{N}{H}$.
- $[x]_m \xleftarrow{\$} \mathcal{P}$ denotes a set of m plaintexts selected uniformly at random from the domain $\mathcal{P}$.
- The security parameter is denoted by λ—for our purposes here, it corresponds to the bit lengths, denoted by n, of the plaintext values.

Negligible Success Probability (Definition 3.5 of [17]) Any encryption scheme is said to be secure if for any probabilistic polynomial-time adversary,[3] there exists an integer N such that for all integers $\lambda > N$ the probability that the adversary succeeds in breaking the scheme is $f(\lambda) < 1/p(\lambda)$, where $p()$ is any polynomial in λ. That is, for every constant c, the adversary's success probability is smaller than λ^{-c} for large enough values of λ. A function that grows smaller than any inverse polynomial is called negligible and $f(\lambda)$ denotes this negligible success probability of the adversary.

2.2 Data and Query Model

We assume that the encrypted information stored on the Cloud corresponds to a data warehouse, and the underlying plaintext values in the range-sensitive columns are from high cardinality domains. The queries issued against this database are OLAP-style decision support queries submitted through form interfaces.

[3] Adversaries running for polynomial number of steps in the security parameter λ.

2.3 Adversary Model

As highlighted in Sect. 1, the data owner (DO) and the security agent (SA) are trusted entities in the HCC model, whereas the service provider (SP) is the untrusted entity who can collude with a few compromised query clients (QCs) to breach data security.

We assume that the adversary can observe all the encrypted data hosted at the SP site, and monitor the computations carried out by the database engine. He is also privy to the choice of encryption schemes, with only the specific keys being kept secret. Further, the adversary can issue a sequence of range queries (via the set of compromised QC). In formulating each query in the sequence, the adversary is given the power to observe the results of previous queries in the sequence—it is this ability to issue adaptive queries which makes HCC a very powerful adversary. Specifically, using the form-based interface, the adversary can issue range queries with varying parameters and observe the corresponding computations on the SP site.

Adversary's Objective In accordance with the DBaaS model, the DO gives access to portions of the data stored on the Cloud to 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's objective is to breach the range predicate security (RPS) interval constraint. As an aside, we note that RPS is a variant of *Message Recovery (MR)* under an adaptive adversary, a commonly used security requirement in practice [4].

We assume that the adversary chooses a *target cell* of an encrypted tuple and desires to gain more information on its plaintext value.[4] The adversary's quantitative goal is to identify an interval (a, b) in which the plaintext value of this encrypted cell lies such that $|b - a| < H$. Note that, in the HCC model, each query will *compulsorily leak some information about the cells*, since the adversary knows the plaintext values of the range limits through collusion with the client. Therefore, the goal of our scheme is not to completely prevent leakage; rather, we aim to minimize the *leakage bandwidth*, measured in terms of the number of queries required to breach the security constraint for the target cell.

2.4 Adversary Attack Model

We now move on to describing the *Chosen Range Attack* model (CRA) unleashed by a HCC adversary. For ease of

[4] While the specific mechanism to choose a target cell is outside the scope of this paper, it is possible that the HCC adversary may use secondary sources such as metadata information and access frequencies to make this identification.

presentation, we use $\mathcal{A}$ to represent the adversary against the deterministic encryption scheme $\mathcal{SE} = (KeyGen, Enc, Dec)$.

The CRA attack model is applicable only to encryption schemes that allow range predicates to be evaluated in the encrypted domain. In this model, the adversary $\mathcal{A}$ has access to a *Range Check Algorithm*, $RCA(x^*,\text{RI})$. *RCA* takes two inputs, x^* being a ciphertext and RI being the encrypted *Range Information* corresponding to a plaintext range. *RCA* outputs 1 if the plaintext value of x^* belongs to the underlying range specified by RI, else it outputs 0. *RCA* corresponds to the SP observing the query processing at the Cloud server. $\mathcal{A}$ also has access to a *Transformation Oracle* ($\mathcal{TO}$). The $\mathcal{TO}$, on input of a plaintext pair (*lvalue*, *hvalue*), returns the encrypted range information RI such that $RCA(x^*,\text{RI})$ outputs 1 if and only if $x \in [lvalue, hvalue]$, where $x^* \leftarrow Enc(x)$. $\mathcal{TO}$ corresponds to the SA rewriting the plaintext query into the equivalent query over the encrypted database. 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 his objective is to identify a plaintext interval (a, b) for x^* such that $|b - a| < H$. Also, $\mathcal{A}$ is allowed to issue a polynomial (in λ, the security parameter) number of range queries to the $\mathcal{TO}$ and observe their computations and results.

The above attack model can be formalized in the form of a game between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ for the deterministic encryption scheme $\mathcal{SE}$:

The Chosen Range Attack Game $Exp^{CRA}_{\mathcal{SE}}(\lambda)$

1. $\mathcal{C}$ computes key $K \leftarrow \text{KeyGen}(\lambda)$, chooses a set $M = \{x_1, \cdots, x_m\}$, where $[x]_m \xleftarrow{\$} \mathcal{P}$ from plaintext domain $\mathcal{P}$, and computes the corresponding ciphertext set $M^* = \{x_1^*, \cdots, x_m^*\}$, where $x_i^* \leftarrow Enc_K(x_i) \forall i \in \{1, m\}$.
2. $\mathcal{A}$ is given *RCA*, H and M^* as input and selects a ciphertext $x^* \in \{x_1^*, \cdots, x_m^*\}$ as its challenge ciphertext.
3. Now $\mathcal{A}$ adaptively asks the encryption of polynomial number of ranges from $\mathcal{TO}$ and obtains the corresponding encrypted range information RI. $\mathcal{A}$ uses *RCA* any number of times and finally outputs a plaintext range (a, b).
4. The output of the experiment is defined to be 1 if $x \in (a, b)$ and $|b - a| < H$, where x is the plaintext corresponding to the challenge ciphertext x^*, and 0 otherwise.

The *CRA advantage* of the adversary $\mathcal{A}$ against $\mathcal{SE}$ is defined as

$$Adv^{CRA}_{\mathcal{SE},\mathcal{A}}(\lambda) = |Pr[Exp^{CRA}_{\mathcal{SE},\mathcal{A}}(\lambda) = 1]| \quad (1)$$

$\mathcal{SE}$ is said to be secure against a *CRA adversary* running in polynomial time if the *advantage is negligible*. The goal of the SPLIT scheme is to provide security against the CRA by the HCC adversary, if the plaintext domain is sufficiently large.

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*. The motivation for splitting stems from the limitation of current security systems (like OPE) that allow range query processing directly over encrypted data. Such systems are easily susceptible to a CRA attack in the HCC model, as shown in Sect. 1.1. Specifically, the adversary is able to perform a simple binary search as it is able to refine its search range with every subsequent query. Hence, to stop the adversary from continuing its binary search, we divide the searchable data in separately encrypted data tables. Secondly, the motivation for layered encryption is to prevent linkages of tuples *across* these various encrypted tables.

The above concepts are elaborated in Sects. 3.1 and 3.2, respectively. 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 Fig. 4a. 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.

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

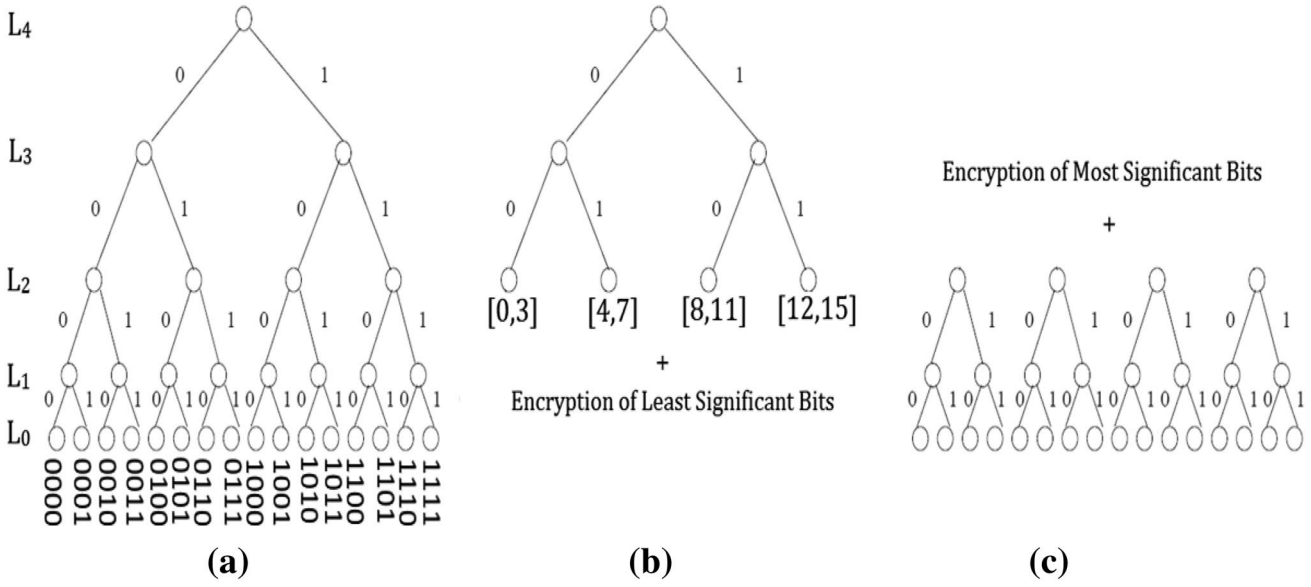

Fig. 4 Basic SPLIT scheme. **a** Plaintext tree (PT). **b** Range safe partition (RS). **C** Brute-force safe partition (BS)

and the associated range for every such node is of granularity 2^2 integers, as shown in Fig. 4b.

The BS partition, on the other hand, is comprised of the remaining PT levels 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 Fig. 4c.

In general, the number of bits in the BS or RS partitions is a configurable parameter for every column and can be set based on the application requirements. For example, if the DO has defined the RPS constraint to be H, then the number of bits in the BS partition are set to be $l = \lceil \log_2(H) \rceil$, and in the RS partition to be $u = n - l$. Now, a binary search over the RS table reveals values at granularities of size $2^l \geq H$. So, to decrypt a value from the BS table, the adversary has to make $O(2^{n-l}) = O(\psi)$ brute-force queries.

Note that there is a trade-off between the H constraint and number of queries required to breach the constraint. For example, if the DO is comfortable in revealing data to a granularity of $H = 2^{24}$, then considering $n = 32$ results in $u = 8$ and $l = 24$. Thus, $2^8 = 256$ queries are sufficient to breach this interval constraint. On the other hand, if H was 2^{20}, then $2^{12} = 1024$ queries would be needed to breach the constraint. In the coming sections, we will assume, without loss of generality, that the number of bits in the RS partition is equal to the number of bits in the BS partition, i.e., the PT is divided into two equal halves ($u = l = \frac{n}{2}$). This leads to a balance between the RPS constraint and the number of queries needed to breach the constraint.

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 $[sk \leftarrow KeyGen(\lambda, w, d)]$

$KeyGen$ 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

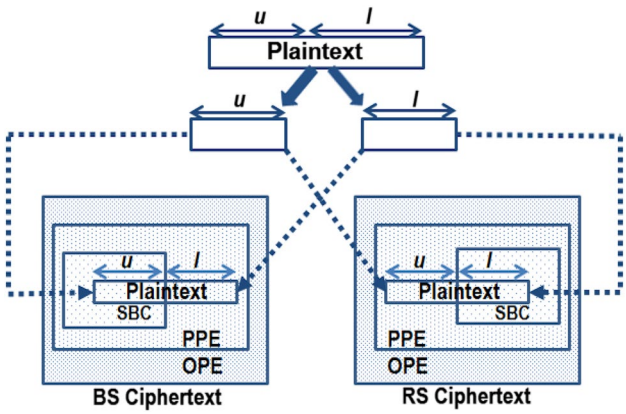

Fig. 5 SPLIT ciphertext construction

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 $m'' = m_{l-1}m_{l-2}\cdots m_0$, and thus, $m = m'||m''$. Then,

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

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

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

$$c^*_{RS} \leftarrow \mathcal{E}^{K_O}_{OPE}(\mathcal{E}^{K_P}_{PPE}(m'||\mathcal{E}^{K_S}_{SBC}(m''))) \tag{3}$$

The entire set of data encryption steps for a given plaintext value, as described above, is pictorially shown in Fig. 5.

Decryption Algorithms

SPLIT incorporates two decryption algorithms $\mathcal{D}_{BS}$ and $\mathcal{D}_{RS}$. Both the algorithms are deterministic and take as input the BS and RS ciphertexts, respectively, along with the key for OPE encryption K_O, key for PPE encryption K_P, key for SBC K_S, and the number of bits u in the RS partition. Let $l = n - u$. Then,

- *Decryption of* BS $[\mathcal{D}_{\mathbf{BS}}(\mathbf{c^*_{BS}}, \mathbf{K_O}, \mathbf{K_P}, \mathbf{K_S}, \mathbf{u})]$

$$m^* \leftarrow \mathcal{D}^{K_P}_{PPE}(\mathcal{D}^{K_O}_{OPE}(c^*_{BS})) \tag{4}$$

$$m \leftarrow \mathcal{D}^{K_S}_{SBC}(m^*_{n-1}\cdots m^*_l)||m^*_{l-1}\cdots m^*_0 \tag{5}$$

- *Decryption of* RS $[\mathcal{D}_{\mathbf{RS}}(\mathbf{c^*_{RS}}, \mathbf{K_O}, \mathbf{K_P}, \mathbf{K_S}, \mathbf{u})]$

$$m^* \leftarrow \mathcal{D}^{K_P}_{PPE}(\mathcal{D}^{K_O}_{OPE}(c^*_{RS})) \tag{6}$$

$$m \leftarrow m^*_{n-1}\cdots m^*_l||\mathcal{D}^{K_S}_{SBC}(m^*_{l-1}\cdots m^*_0) \tag{7}$$

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 these data transformation process with the help of an example. Assume the plaintext table is

Fig. 6 SPLIT banking database. **a** Loan_BS_BS. **b** Loan_BS_RS. **c** Loan_RS_BS. **d** Loan_RS_RS

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

(a)

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

(b)

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

(c)

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

(d)

Loan with schema as enumerated in Fig. 2a—then, $w = 3$. Assume that range predicates can only be asked on the *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 $(K_O^1, K_O^2, \ldots, K_O^8)$, eight $(2 * 2^2)$ PPE keys $(K_P^1, K_P^2, \ldots, K_P^8)$, and twelve $(3 * 2^2)$ SBC keys $(K_S^1, K_S^2, \ldots, K_S^{12})$. Next, we create four encrypted tables, as shown in Fig. 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 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, respectively. 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, their values can be linked.

Similarly, PPE by itself is not secure since it preserves the structure of the tree corresponding to the binary representation. That is, 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 or not bit $n - 1$ gets flipped.

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, while larger ranges are answered from the RS tables. Functionally, the range query processing consists of two main steps—range query mapping and range query execution, which are 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 [22]. 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 (Lines 8, 10). The lower and upper bounds of the range in the PPE encrypted domain are computed by replacing all the remaining lower j bits with 0 and 1, respectively (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 a BS range on the *Collateral* column is answered from the Loan_BS_BS table.

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}

Convert r into a set of ranges $\mathcal{R}$ of form $b_{n-1}b_{n-2}\cdots b_j*$ {using technique in [22]}
Let $l = n - u$
for all $(r_i = b_{n-1}b_{n-2}\cdots b_j*)$ in $\mathcal{R}$ **do**
  $v \leftarrow b_{n-1}b_{n-2}\cdots b_j 0\cdots 0$ {set lower bits to 0}
  $v_U \leftarrow v_{n-1}v_{n-2}\cdots v_l$; $v_L \leftarrow v_{l-1}v_{l-2}\cdots v_0$
  if $(j < l)$ **then** {BS range}
    $v^* \leftarrow \mathcal{E}_{K_S^2}(v_U)||v_L$
    $e_v^* \leftarrow \mathcal{E}_{K_P^2}(v^*)$
  else {RS range}
    $e_v^* \leftarrow \mathcal{E}_{K_P^1}(v)$
  end if
  Let $c_n c_{n-1}\cdots c_0$ be the bit representation of e_v^*
  $r_L \leftarrow c_{n-1}c_{n-2}\cdots c_j 0\cdots 0$; $r_U \leftarrow c_n c_{n-1}\cdots c_j 1\cdots 1$
  if $(j < l)$ **then**
    $r_L^* \leftarrow \mathcal{E}_{K_O^2}(r_L)$; $r_U^* \leftarrow \mathcal{E}_{K_O^2}(r_U)$
    Add (r_L^*, r_U^*) to R_{BS}
  else
    $r_L^* \leftarrow \mathcal{E}_{K_O^1}(r_L)$; $r_U^* \leftarrow \mathcal{E}_{K_O^1}(r_U)$
    Add (r_L^*, r_U^*) to R_{RS}
  end if
end for
return R_{RS}, R_{BS}

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 Fig. 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 the output of Step 2, SA creates ciphertext sub-queries and sends them to SP.
4. SP executes the sub-queries and sends the (encrypted) result tuples to 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, 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 the 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 formally 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 Fig. 7.[5] 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

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

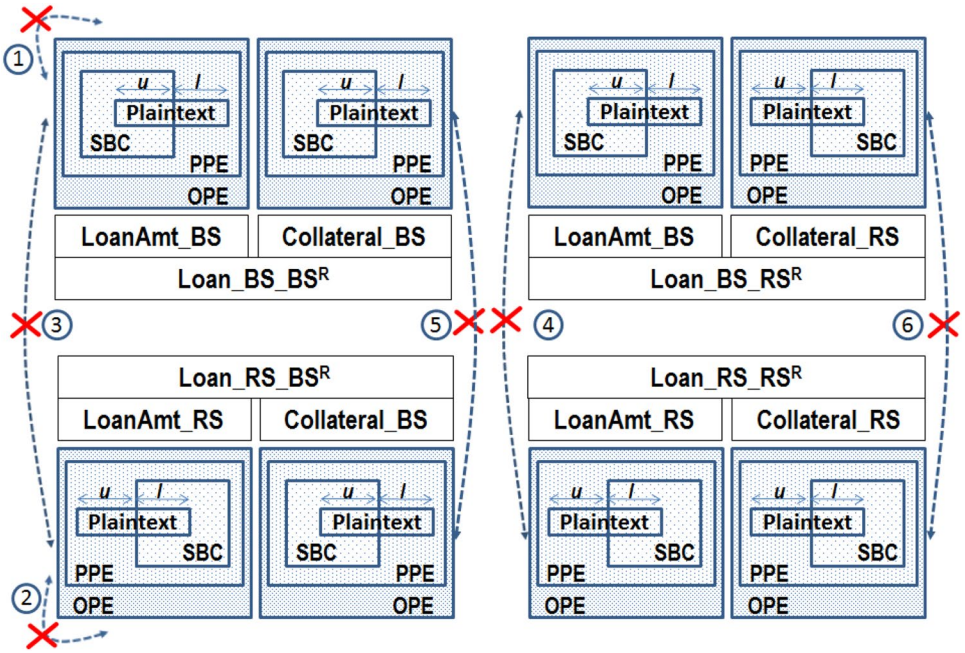

Fig. 7 Ensuring security of *LoanAmt* values

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 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 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 Sect. 3.

In a nutshell, the security of the SPLIT encryption scheme is established based on the following points:

1. The BS and RS encryptions are independently secure (dotted lines 1 and 2 in Fig. 7). The associated security proofs are presented in Sect. 6.
2. For any plaintext table, there is no linkage between the corresponding BS and RS ciphertext tables (dotted lines 3 and 4 in Fig. 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 Fig. 7).

6 Security Guarantees

In this section we present the formal proofs for the security of the BS and RS encryption schemes. The full suite of proofs for the security of SPLIT are available in the technical report [24].

6.1 Formal Security Proof For BS Encryption

Claim *If SBC is a secure block cipher, then any probabilistic polynomial-time adversary $\mathcal{A}_{BS}$ will have negligible advantage in the Chosen Range Attack experiment $Exp^{CRA}_{\mathcal{E}_{BS},\mathcal{A}_{BS}}(\lambda)$ against the $\mathcal{E}_{BS}$ encryption scheme.*

Proof We will prove the claim by contradiction. Specifically assume that there exists a CRA adversary $\mathcal{A}_{BS}$ against the $\mathcal{E}_{BS}$ scheme with some non-negligible advantage ϵ. Then we can show that there exists an adversary $\mathcal{A}_{SBC}$ against the *SBC* scheme with a non-negligible advantage ϵ'. This contradicts the assumption that *SBC* is a secure block cipher, and therefore, the adversaries $\mathcal{A}_{SBC}$ and $\mathcal{A}_{BS}$ cannot exist. The nihility of $\mathcal{A}_{SBC}$ and $\mathcal{A}_{BS}$ is shown in the remainder of this section.

Let $\mathcal{C}_{SBC}$ be a challenger for the SBC scheme. Given a set of *SBC* ciphertexts, the objective of $\mathcal{A}_{SBC}$ is to learn some plaintext bits of a target *SBC* ciphertext. Further $\mathcal{A}_{SBC}$ is allowed to ask the encryption of plaintexts (polynomial in λ) from $\mathcal{C}_{SBC}$. In the BS security game shown below, $\mathcal{A}_{SBC}$ acts as the challenger ($\mathcal{C}_{BS}$) of the $\mathcal{E}_{BS}$ scheme and uses $\mathcal{A}_{BS}$ to achieve its objective. Further, $\mathcal{C}_{BS}$ makes at most 2 SBC queries for every query from its adversary, since the ranges that map to the BS table using Algorithm 1 can have at most two different *u bit* prefixes. Let $\mathcal{A}_{BS}$ be able to breach the interval constraint H, and guess the target ciphertext to a plaintext range $H' \leq H$. It can be seen that the *u bit* prefix of all the points in the interval H' can take at most two values, since $H' \leq H = 2^l$. $\mathcal{A}_{SBC}$ will randomly pick one of these *u bit* prefixes (by choosing one of the two endpoints of the range and taking its prefix) as its guess for the target *SBC* ciphertext. If $\mathcal{A}_{BS}$ makes q queries and succeeds with non-negligible probability ϵ, $\mathcal{A}_{SBC}$ will make $2 * q$ queries and succeed with the non-negligible probability $\frac{\epsilon}{2}$, which contradicts the assumption that *SBC* is a secure block cipher. Hence, the adversary $\mathcal{A}_{BS}$ with non-negligible advantage against the $\mathcal{E}_{BS}$ encryption scheme cannot exist, and $\mathcal{E}_{BS}$ is secure against CRA (proof for dotted Line 1 in Fig. 7).

The security game between $\mathcal{C}_{SBC}$, $\mathcal{A}_{SBC}$, and $\mathcal{A}_{BS}$ has following steps:

1. The challenger $\mathcal{C}_{SBC}$ has a set $\mathcal{S}$ of $\frac{n}{2}$ *bit* integers. $\mathcal{C}_{SBC}$ first generates the key $K_S^1 \leftarrow KeyGen_{SBC}$ for the SBC and encrypts each number in set $\mathcal{S}$ to compute the set $\mathcal{S}^* = \{s^* | \exists s \in \mathcal{S} : s^* \leftarrow \mathcal{E}_{K_S^1}(s)\}$. $\mathcal{C}_{SBC}$ forwards $\mathcal{S}^*$ to the SBC adversary $\mathcal{A}_{SBC}$.
2. The SBC adversary $\mathcal{A}_{SBC}$ also acts as the challenger $\mathcal{C}_{BS}$ for the $\mathcal{E}_{BS}$ encryption scheme. It receives the set $\mathcal{S}^*$ and performs below steps to generate BS ciphertexts:
   - It generates the key $K_O^1 \leftarrow KeyGen_{OPE}$ for OPE and key $K_P^1 \leftarrow KeyGen_{PPE}$ for PPE.
   - It generates a set of all the $\frac{n}{2}$ *bit* numbers $\mathcal{R}$. It then picks every number in set $\mathcal{S}^*$ one by one and appends it with all the numbers in set $\mathcal{R}$ to compute the set $\mathcal{S}' = \{s' | \exists s^* \in \mathcal{S}^*, \exists u^* \in \mathcal{R} : s' \leftarrow s^* || u^*)\}$
   - It further computes a set $\mathcal{S}'' = \{s'' | \exists s' \in \mathcal{S}' : (s'' \leftarrow \mathcal{E}_{K_O^1}(\mathcal{E}_{K_P^1}(s'))\}$

   If we denote the set created by appending numbers to the elements in set $\mathcal{S}$ as $\mathcal{D}$, it can be seen that the elements in set $\mathcal{S}''$ are the BS ciphertexts that would have been produced by $\mathcal{E}_{BS}$ on $\mathcal{D}$. $\mathcal{C}_{BS}$ stores the values in the set $\mathcal{S}''$ as a single column BS table (say *D_BS*) at the SP's site which also acts as the CRA adversary ($\mathcal{A}_{BS}$) against the $\mathcal{E}_{BS}$ encryption scheme.
3. Next, the adversary $\mathcal{A}_{BS}$ selects a tuple from the BS table *D_BS* and declares it as its target ciphertext (say c^*) for decryption. Further, $\mathcal{A}_{SBC}$ declares the *SBC* ciphertext (say c') used to form c^* as its target. $\mathcal{A}_{BS}$ can now issue range predicate queries over *D_BS*.
4. $\mathcal{A}_{BS}$ sends its plaintext range predicate query to $\mathcal{C}_{BS}$.
5. $\mathcal{C}_{BS}$ identifies the prefix based sub-ranges as specified in Line 1 of Algorithm 1. The ranges represented by larger prefixes (more than $\frac{n}{2}$ *bits*) are answered from the *D_BS* table by $\mathcal{C}_{BS}$.
6. For each BS range, $\mathcal{C}_{BS}$ needs to perform the steps corresponding to BS in Algorithm 1 (Lines 7, 8, 15 and 16) to map the input range to a range on the encrypted domain. While the PPE and OPE encryptions can be directly performed by it, this is not the case for the *SBC* encryption in Line 6 since it does not have the secret key of *SBC*. Instead, it sends these higher-order $\frac{n}{2}$ *bits* to $\mathcal{C}_{SBC}$ to obtain their *SBC* encryptions.

   The rewritten query with the mapped ranges is executed over *D_BS*, and the received tuples are decrypted and forwarded to $\mathcal{A}_{BS}$. The tuples can be decrypted by $\mathcal{C}_{BS}$ since it knows the OPE and PPE keys and all result tuples corresponding to a range have the same $\frac{n}{2}$ *bits* prefix, whose *SBC* decryption corresponds to the $\frac{n}{2}$ *bits* prefix of the input range on the plaintext domain.
7. The adversary $\mathcal{A}_{BS}$ can ask for more range predicate queries (polynomial in the security parameter). For each of these, $\mathcal{C}_{BS}$ constructs a response as shown in the steps above.
8. At the end, $\mathcal{A}_{BS}$ outputs an interval H' as its guess of the interval in which the plaintext for its target ciphertext c^* belongs.
9. $\mathcal{C}_{BS}$ forwards to the *SBC* challenger $\mathcal{C}_{SBC}$, the $\frac{n}{2}$ *bits* of the prefix of one of the two endpoints (randomly chosen) in H' as the plaintext corresponding to the target ciphertext c'.

Further in Step 6 above, the ranges that map to the BS table using Algorithm 1 can have at most two different $\frac{n}{2}$ *bit* prefixes. This is because if the granularity of any sub-range for the BS table overlaps more than 2 sub-trees, then that sub-range can be further divided into 1 sub-range for the RS tables and 2 sub-ranges for the BS table. For example, consider the BS and RS divisions of the plaintext tree shown in Fig. 4—here, the sub-range [7, 12] which touches three sub-trees from the BS table can be rewritten

into sub-ranges {[8, 11]} for the RS table and {[7, 7], [12, 12]} for the BS table. □

6.2 Formal Security Proof For RS Encryption

Claim *If SBC is a secure block cipher, then any probabilistic polynomial-time adversary $\mathcal{A}_{RS}$ will have negligible advantage in the Chosen Range Attack experiment $Exp^{CRA}_{\mathcal{E}_{RS},\mathcal{A}_{RS}}(\lambda)$ against the $\mathcal{E}_{RS}$ encryption scheme.*

Proof The proof for this claim is similar to that of Claim 6.1. One difference in the RS game is that C_{RS} does not need to perform any *SBC* encryption queries from C_{SBC}. Further, the adversary $\mathcal{A}_{SBC}$ can only guess some of the l plaintext bits, depending on the granularity of the range guessed by the adversary $\mathcal{A}_{RS}$. Further, as in the BS game, the advantage is $\frac{\epsilon}{2}$, since C_{SBC} has to pick one of two possible prefixes, which contradicts the assumption that *SBC* is a secure block cipher. Hence, the adversary $\mathcal{A}_{RS}$ with non-negligible advantage against $\mathcal{E}_{RS}$ cannot exist, and therefore, $\mathcal{E}_{RS}$ is secure against CRA (proof for dotted Line 2).

The security game between C_{SBC}, $\mathcal{A}_{SBC}$ and $\mathcal{A}_{RS}$ has following steps:

1. The challenger C_{SBC} has a set $\mathcal{S}$ consisting of all $\frac{n}{2}$ *bit* integers. C_{SBC} first generates a key $K^1_S \leftarrow KeyGen_{SBC}$ for the SBC and then encrypts each number in the set $\mathcal{S}$—this results in the set $\mathcal{S}^* = \{s^* | \exists s \in \mathcal{S} : s^* \leftarrow \mathcal{E}_{K^1_S}(s)\}$.

   C_{SBC} forwards $\mathcal{S}^*$ to the SBC adversary $\mathcal{A}_{SBC}$.
2. The SBC adversary $\mathcal{A}_{SBC}$ also acts as the challenger C_{RS} for the $\mathcal{E}_{RS}$ encryption scheme. It receives the set $\mathcal{S}^*$ and performs the following actions to generate RS ciphertexts:
   - It generates key $K^1_O \leftarrow KeyGen_{OPE}$ for OPE and key $K^1_P \leftarrow KeyGen_{PPE}$ for PPE.
   - It generates a set of all the $\frac{n}{2}$ *bit* numbers $\mathcal{R}$. It then picks every number in set $\mathcal{R}$, one by one, and appends it with all the numbers in set $\mathcal{S}^*$ to compute the set $\mathcal{S}' = \{s' | \exists s^* \in \mathcal{S}^*, \exists u^* \in \mathcal{R} : s' \leftarrow u^* || s^*)\}$.
   - It further computes a set $\mathcal{S}'' = \{s'' | \exists s' \in \mathcal{S}' : (s'' \leftarrow \mathcal{E}_{K^1_O(\mathcal{E}_{K^1_P}(s'))}\}$.

   If we denote the set of all n *bit* numbers as $\mathcal{D}$, it can be seen that the elements in the set $\mathcal{S}''$ are the RS ciphertexts that would have been produced by $\mathcal{E}_{RS}$ on $\mathcal{D}$. C_{RS} stores the values in the set $\mathcal{S}''$ as a single column RS table (say *D_RS*) and host this table at SP's site, which also acts as the CRA adversary ($\mathcal{A}_{RS}$) against the $\mathcal{E}_{RS}$ encryption scheme.
3. Next, the adversary $\mathcal{A}_{RS}$ selects a tuple from the RS table *D_RS* and declares it as the target ciphertext (say c^*) for decryption. Further, $\mathcal{A}_{SBC}$ declares the SBC ciphertext (say c') used to form c^* as its target. $\mathcal{A}_{RS}$ can now issue range predicate queries over *D_RS*.
4. $\mathcal{A}_{RS}$ sends its plaintext range predicate query to C_{RS}, and let the result tuples be from the range $[a, b]$.
5. C_{RS} identifies the prefix based sub-ranges as specified in Line 1 of Algorithm 1. The ranges represented by smaller prefixes (less than $\frac{n}{2}$ *bits*) are answered from the *D_RS* table by C_{RS}.
6. For each RS range, C_{RS} needs to perform the steps corresponding to RS in Algorithm 1 (specifically Lines 10, 18 and 19) to map the input range to a range on the encrypted domain. All these steps can be directly performed since it has the keys for OPE and PPE and no block cipher encryptions are needed.

   The rewritten query with the mapped ranges is executed over *D_RS* to obtain the encrypted result tuples. The decrypted results correspond to all integers between a and b (the query range), since $\mathcal{D}$ consists of all the n *bit* integers. The encrypted result tuples, along with the decrypted set, are forwarded to $\mathcal{A}_{RS}$.
7. The adversary $\mathcal{A}_{RS}$ can ask more range predicate queries (polynomial in the security parameter). For each of these queries, C_{RS} constructs a response as shown in steps 5-6 above.
8. At the end, $\mathcal{A}_{RS}$ outputs an interval H' as its guess of the interval in which the plaintext for its target ciphertext c^* belongs.
9. C_{RS} computes the prefixes of the endpoints in H' and forwards to SBC challenger C_{SBC} the bits after the most significant $\frac{n}{2}$ *bits* of the prefix, representing its guess of some plaintext bits corresponding to its target ciphertext c'.

In the RS game, $C_{SPLIT_{RS}}$ does not need to perform any AES encryption queries from C_{AES}. Thus, if $\mathcal{A}_{SPLIT_{RS}}$ has a non-negligible advantage ϵ in the CRA experiment with $C_{SPLIT_{RS}}$, this advantage carries over and $\mathcal{A}_{AES}$ will win the AES challenge with the same non-negligible ϵ probability, i.e.,

$$Pr[success\ of\ \mathcal{A}_{AES}] = \epsilon \tag{8}$$

which contradicts the assumption that AES is a secure block cipher. Hence, the adversary $\mathcal{A}_{SPLIT_{RS}}$ with non-negligible advantage against $SPLIT_{RS}$ cannot exist. □

7 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

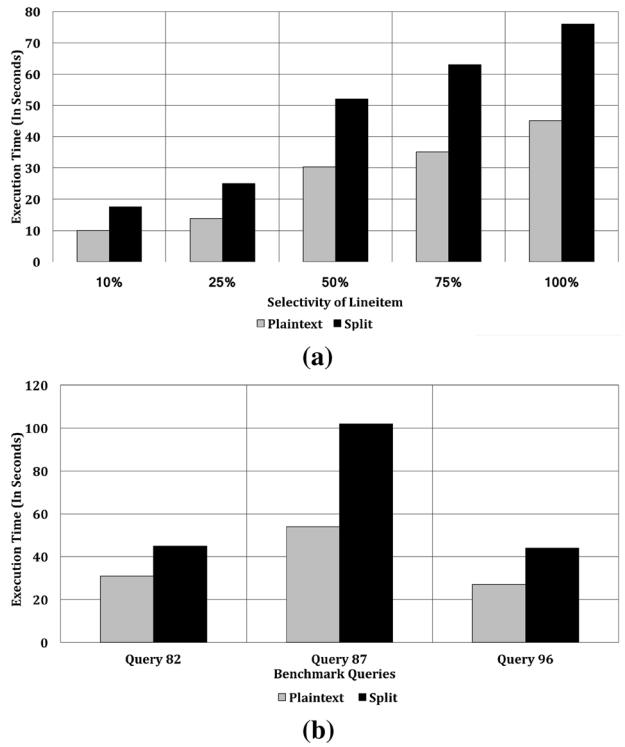

Fig. 8 Query execution time on benchmark databases. **a** 10GB TPC-H database. **b** 10GB TPC-DS database

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, we constructed the queries on LINEITEM, the largest table in the TPC-H schema (60 million rows). Each query had range predicates on four attributes, with the predicates covering the entire spectrum of selectivities. The reason we used synthetically created queries is because the native queries were either not rich in range predicates, or contained aggregate operators which we currently do not handle. On the other hand, for TPC-DS, we directly invoked three benchmark queries (queries Q82, Q87, and Q96) which contained only range predicates, and could therefore be fully executed on a SPLIT encrypted version of the database. The listings of these queries are provided in [24].

7.1 SPLIT Encryption Time

A legitimate concern about SPLIT could be regarding the time it takes to encrypt databases, especially in light of its multiple splitting and layered encryption operations. However, our experiments have shown that encrypting the 10 GB TPC-H database takes less than *12* h and for the 10 GB TPC-DS database it took around *13* h using 8 concurrent threads.

From the above, it is evident that data encryption takes only a few hours, which appears to be a reasonable one-time overhead that is amortized over the stream of range queries subsequently executed over the encrypted database. Further, we have found that the final result set decryption times are negligible in comparison with the data encryption or query execution times, due to the small size of these results, and hence, they are added to the query execution times reported in Sect. 7.2.

7.2 Query Execution Time

The execution times for range query processing by the SPLIT and plaintext algorithms are captured in Fig. 8a, b for the above-mentioned TPC-H and TPC-DS databases, 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 Fig. 8b at 50% LINEITEM selectivity, the plaintext query takes around 30 seconds while SPLIT completes in 52 seconds. Similarly, in Fig. 8b, Query 82 takes 32 seconds in the plaintext environment and is computed in 45 seconds with SPLIT encryption.

At first glance, it may seem that SPLIT may incur a performance slowdown commensurate with the storage blowup. However, such worst-case scenarios require a query containing multi-dimensional range predicates, where each predicate has a sufficiently coarse selectivity to result in a full table scan. In general, if indexes are present and are chosen by the optimizer, then the number of tuples fetched from the disk will be proportional 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 smaller selectivities, there is a higher likelihood that the optimizer decides to use indexes.

Note that the good performance of SPLIT is *despite* 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 effectively equivalent to that of the single query in the plaintext domain,

Table 1 Monthly dollar cost of Cloud platforms

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

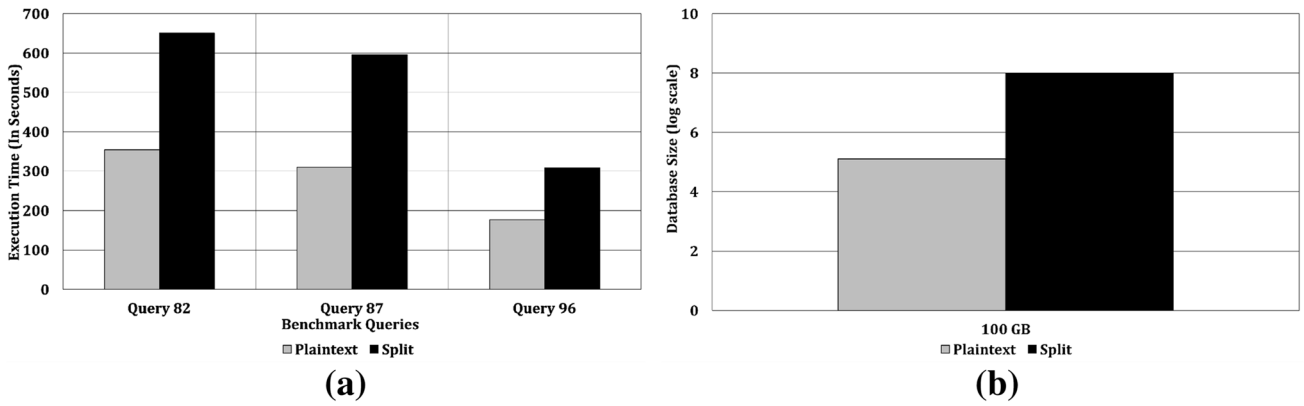

Fig. 9 **a** Query execution time and **b** database size for 100 GB TPC-DS benchmark database

particularly if indexes are used in the query plan. This feature points to the practicality of the SPLIT scheme.

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

7.3 Storage Cost

The storage blowup factor of SPLIT depends only on the number of attributes and is independent of the database size. Also, since indexes are used by both the plaintext and SPLIT algorithms, we expect their relative performance to be similar for different database sizes. In the above experiments, the size of the plaintext TPC-H database with indexes was 21 GB, whereas the corresponding SPLIT encrypted database is 335 GB. On the other hand, for TPC-DS the plaintext database size with indexes is 22 GB, whereas the corresponding SPLIT encrypted database is 359 GB. These values are an outcome of our incorporating *four* range predicates in the queries, resulting in the encrypted database being roughly 16 times the size of the plaintext database. Though the blowup is certainly large, the overall impact on the system *dollar cost* is much lower, since storage is relatively cheap. For instance, Table 1 shows the monthly costs for attaining similar throughputs with the plaintext and SPLIT schemes, estimated using the rates charged by Amazon's AWS service [23] 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. Finally, various workload-dependent optimizations to reduce the storage overheads are described later in Sect. 8.1.

7.4 Scaling Experiment

In the previous experiments, we evaluated SPLIT using 10 GB versions of the benchmark databases. But in order to

better understand the overheads and strengthen our performance claims, we now present results from scaleup experiments. Specifically, we created a 100 GB TPC-DS database and again executed queries Q82, Q87 and Q96. The times taken for processing these queries over the plaintext and SPLIT encrypted databases are shown in Fig. 9a, and it is evident that SPLIT is within a factor of *two* of the plaintext query execution even on these large databases.

For the same environment, the storage costs of the plaintext and SPLIT encrypted databases are shown (on a log scale) in Fig. 9b. Again, since we are handling four range predicates and have created indexes over both the plaintext and SPLIT databases, the encrypted database size turns out to be roughly 16 times the size of the plaintext database.

8 Deployment and Integration

We now turn our attention in this section to discuss deployment and integration issues of the SPLIT algorithm. Specifically, we first present a strategy for materially reducing its space overheads. Then we describe the mechanisms for deploying SPLIT as a component of secure database systems that support a rich set of SQL primitives.

8.1 Workload-based Reductions of Storage Requirement

The discussion of storage overheads in the experimental section was from the financial perspective. We now consider *workload-based* optimizations to reduce the space requirement of the SPLIT scheme.

Let $C = \{C_1 \cdots C_t\}$ represent the set comprising of all the columns in the database. Firstly, if the DO can identify a subset of columns S that will not be used as range predicates in any query, then they need not be considered for the SPLIT scheme. Instead, these columns can simply be encrypted using a secure deterministic encryption scheme, such as AES. Only the set $C' = C - S$ needs to be considered for SPLIT encryption. Let the size of C' be d. Further, suppose that the DO can partition C' into a number of disjoint classes based on the query workload, such that columns in a class *do not simultaneously occur* in range predicates of any query. Then, the columns in a class are treated like a composite column while generating the BS and RS combinations during encryption. Thus, if there are e such disjoint classes the total blowup due to table replication reduces to 2^e from 2^d.

The identification of disjoint classes from the set C' can be reduced to the minimum vertex coloring problem, following the steps shown in Algorithm 2. The minimum vertex coloring $\mathcal{I}$ represents the desired disjoint classes since no two adjacent vertices in the graph $\mathcal{G}$ are given the same color, which implies that none of the column pairs corresponding to vertices in $\mathcal{I}$ with the same color appear simultaneously in range predicates in a SQL query. Further, the chromatic number of the minimum vertex coloring gives a bound on the size of the encrypted database.

Since the minimum vertex coloring problem is NP-hard, we use the approximation algorithms presented in [8, 10] for the efficient identification of the color assignments.

Algorithm 2 *Minimum Number of Disjoint Classes*

Input: The set of columns C' that needs to be considered for SPLIT encryption,
The query workload $\mathcal{Q}$
Output: The e disjoint classes $\mathcal{I}$
1: Initialize an empty graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
2: Add a vertex v_i in $\mathcal{G}$ for every column $C'_i \in C'$
3: **for** every query $q \in \mathcal{Q}$ **do**
4: **for** every pair of columns C'_i and C'_j that appear in range predicates in q **do**
5: Create an edge e_{ij} between the corresponding vertices v_i and v_j
6: $\mathcal{E} = \mathcal{E} \cup e_{ij}$
7: **end for**
8: **end for**
9: Identify a minimum vertex coloring $\mathcal{I}$ for graph $\mathcal{G}$ [8,10]
10: **return** $\mathcal{I}$

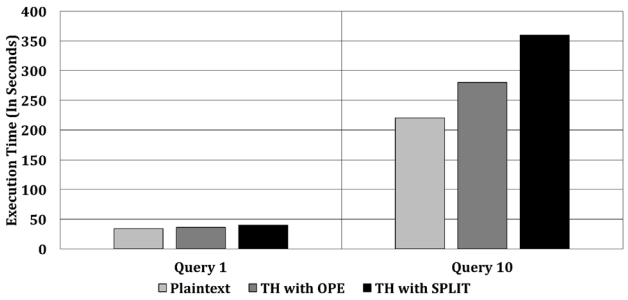

Fig. 10 Query execution time (10GB TPC-H database)

We expect that the above optimization is likely to be very effective in real deployments since Cloud databases are typically accessed through standard form interfaces, which restrict the types of queries issued to the underlying warehouse. For example, in TPC-H, the total number of columns in the database is 61 of which only 6 columns appear in range predicates, hence $d = 6$. Further, by analyzing the benchmark query set, we can partition these 6 columns into 3 disjoint classes, and therefore, $e = 3$. Thus, the SPLIT encrypted TPC-H database will be 8 times the plaintext database.

Similarly, with TPC-DS, the total number of columns is 429, of which only 34 columns appear in range predicates, hence $d = 34$. By analyzing the benchmark query set, we can partition these 34 columns into 4 disjoint classes, and therefore, $e = 4$. Thus, the SPLIT encrypted TPC-DS database is 16 times the plaintext database.

The above discussion, which indicates that 4 classes are sufficient for both TPC-H and TPC-DS, was the basis for deploying queries with 4 range predicates in our experiments.

8.2 Handling Updates

In principle, a SPLIT encrypted database can handle updates, insertions, and deletions to the original database. All these operations require updating of all the ciphertext tables corresponding to the plaintext table being updated. However, if these operations are done one row at a time, then they allow the adversary to correlate the corresponding rows across the tables. To prevent this correlation leakage, we resort to batch updates. That is, all pending updates are kept at the security agent (SA) until a full batch is accumulated. Then this batch update is applied at one go by making random passes over the set of updates in the batch for each encrypted table. Batch updates are common in databases designed for analytical workloads, since they are populated periodically using ETL scripts.

8.3 Extending SPLIT to Other Operators

As mentioned earlier, SPLIT can also be extended to handle additional operators that commonly occur in SQL queries such as Equality predicates, Equi-joins, Group By, and Having clauses.

The Equality and Equi-join predicates are handled by treating these operators as range predicates with a range of size 1. Further, since SPLIT is a deterministic encryption scheme, it directly supports the Group By operator. However, there is a possibility that a particular group may appear in the results of different sub-queries being executed on different tables. These groups can be easily merged by the SA after decrypting the independent results. For example, for a Group By with Count aggregate, the SA needs to simply add the counts for a given group from different result sets before returning the final answer to the QC. The Having clause can also be supported similarly as a filter applied by the SA on the Group By results.

8.4 Integrating SPLIT in a Complete System

To support the full power of SQL, including sub-queries and aggregates, SPLIT needs to be embedded in a complete data processing system. For instance, we can substitute SPLIT in place of OPE in systems such as Cipherbase [2] or Monomi [20], thereby enhancing their security at the cost of a modest increase in query execution times.

A collateral side benefit of incorporation in a full-fledged system is that it also allows us to handle situations where the DO has laid down a limit on the storage budget. For instance, consider the case where the DO has fixed the encrypted storage budget to be at most 8 times the size of the plaintext database. This constraint implies that we need to divide the potential range predicate columns into at most 3 equivalence classes, hence $e = 3$. In the event that this restriction cannot be met, we can still handle the situation if SPLIT is part of a larger system, such as Cipherbase. Specifically, we can rewrite the query by choosing a subset of predicates such that there is at most one column from each equivalence class. This reduced query is executed over the encrypted database, whereas the remaining predicates are applied in a post-filtering step, in the trusted hardware of the Cipherbase system.

Performance of SPLIT Component

To study the effectiveness of SPLIT when part of a larger system, we conducted sample experiments on the TPC-H environment described earlier. In these experiments, the complete benchmark queries were executed, not just the range predicates. To facilitate this, the input query was divided into two parts—the first part containing equi-joins, equality and range filter predicates, which were executed directly on the encrypted database, and the second part

	Scheme	OPE	PPE	PBTree	SDB	SPLIT
Adversary	Attack					
HBC	Ciphertext Only	✓	✓	✓	✓	✓
	Known Plaintext	✓	✓	✓	×	✓
HCC	Chosen Range	×	×	×	×	✓

Fig. 11 Comparison of SPLIT with prior systems

containing the remaining predicates which were processed after decryption in a trusted hardware setup. A Java program was used to simulate the trusted hardware processing.

Specifically, we built two secure trusted hardware systems, **TH-OPE** and **TH-SPLIT**, employing the OPE and SPLIT encryption schemes, respectively, to evaluate range predicates. All the columns not involved in range predicates were encrypted using AES. The SPLIT encrypted database was constructed so as to support the 3 disjoint classes (as described in the previous subsection) of range predicates arising in the TPC-H benchmark. Non-clustered indexes were created on range predicate columns in both the plaintext and ciphertext databases.

We executed TPC-H queries Q1 and Q10 to evaluate the performance of SPLIT when used as a replacement for OPE, and the resulting query execution times are shown in Fig. 10. These results clearly indicate that for complex queries, the performance differences between SPLIT and OPE are likely to reduce due to the predominant processing in the trusted hardware, which is identical for SPLIT and OPE. Overall, it indicates that SPLIT maintains reasonably good performance while delivering a much stronger security guarantee than OPE.

9 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, 13, 18] and **PPE** [15, 22], which inevitably leak order-based and structure-based characteristics, respectively, of the plaintext data. More pertinently, these solutions can be easily breached in the HCC model, as shown in Sect. 1.1.

An encrypted tree-based index structure, called **PBtree**, was proposed in [16], but it requires significant changes to the underlying database engine which may hinder its adoption by industry. In contrast, SPLIT can be noninvasively deployed with currently relational systems. Subsequently, alternative tree-based encryption schemes were presented in [7, 9], while **Bucketing Schemes** were analyzed in [11, 12]. These schemes provide stronger security guarantees than **OPE** schemes in the Honest-but-Curious model. However, a fundamental problem is that they return *false positives* in the query results. For instance in [11], the whole data domain is partitioned into buckets, each having a unique ID that is stored at the server along with the encrypted data. The input range query is mapped to bucket IDs and all rows which fall into these buckets are returned—this results in false positives if the bucket boundaries do not match exactly with the range boundary. SPLIT, on the other hand, always returns accurate results.

Another line of research [2, 3, 19–21] has focused on building complete systems which support secure execution of entire SQL queries over encrypted databases. For instance, in **CryptDB** [19], 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** [20] 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 and decrypted and loaded into the local database. The second part of the query is then run on this local plaintext database. Since both CryptDB and MONOMI incorporate the OPE encryption scheme for efficient query processing, they also inherit its security limitations.

Another group of secure systems, such as **TrustedDB** [3] and **Cipherbase** [2], assume the availability of trusted hardware at the server for decrypting and processing the data in a secure manner. In TrustedDB, the whole database engine runs inside the trusted hardware and all data is brought to it for processing. But this is a very expensive proposition in terms of both infrastructure and processing overheads. On the other hand, in Cipherbase, the database engine is aware of the encryption requirements and integrates tightly with

the trusted hardware. However, its use of OPE for efficient query processing results in the attendant security limitations.

Again these systems use OPE encryption scheme to enhance their performance and hence inherit its limitations too.

Finally, the **SDB** system [21] is based on an asymmetric secret-sharing scheme with a set of interoperable operators over the encrypted data. This design facilitates processing of a wide range of SQL queries at the SP, but is again susceptible to a CRA attack in the HCC model.

A summary comparison of the security properties of SPLIT with respect to the prior literature is shown in Fig. 11.

10 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 are expected to be much smaller. 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 alternative solutions in the HBC model (e.g., PBtree [16]) and to design encryption schemes to securely handle additional SQL operators (e.g., θ join) against HCC adversaries.

References

1. Agrawal R, Kiernan J, Srikant R, Xu Y (2004) Order-preserving encryption for numeric data. In: Proceedings of ACM SIGMOD conference
2. Arasu A, Blanas S, Eguro K, Kaushik R, Kossmann D, Ramamurthy R, Venkatesan R (2013) Orthogonal security with cipherbase. In: Proceedings of CIDR conference
3. Bajaj S, Sion R (2011) Trusteddb: a trusted hardware based outsourced database engine. In: PVLDB, vol 4, No. 12
4. Bellare M, Ristenpart T, Rogaway P, Stegers T (2009) Format-preserving encryption. In: Proceedings of selected areas in cryptography conference
5. Boldyreva A, Chenette N, Lee Y, ONeill A (2009) Order-preserving symmetric encryption. In: Proceedings of EUROCRYPT conference
6. Boldyreva A, Chenette N, O'Neill A (2011) Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Proceedings of CRYPTO conference
7. Chi J, Hong C, Zhang M, Zhang Z (2017) Fast multi-dimensional range queries on encrypted cloud databases. In: Proceedings of DASFAA conference
8. Cutello V, Nicosia G, Pavone M (2003) A hybrid immune algorithm with information gain for the graph coloring problem. In: Proceedings of genetic and evolutionary computation conference
9. Demertzis I, Papadopoulos S, Papapetrou O, Deligiannakis A, Garofalakis M (2016) Practical private range search revisited. In: Proceedings of ACM SIGMOD conference
10. Dowsland KA, Thompson JM (2008) An improved ant colony optimisation heuristic for graph colouring. Proc Discrete Appl Math 156(3):313–324
11. Hacigümüs H, Iyer BR, Li C, Mehrotra S (2002) Executing SQL over encrypted data in the database-service-provider model. In: Proceedings of ACM SIGMOD conference
12. Hore B, Mehrotra S, Tsudik G (2004) A privacy-preserving index for range queries. In: Proceedings of VLDB conference
13. Kerschbaum F (2015) Frequency-hiding order-preserving encryption. In: Proceedings of CCS conference
14. Knuth DE (1998) Sorting and searching. The art of computer programming, vol 3, 2nd edn. Addison-Wesley, Reading
15. Li J, Omiecinski ER (2005) Efficiency and security trade-off in supporting range queries on encrypted databases. In: Proceedings of DBSec conference
16. Li R, Liu AX, Wang AL, Bruhadeshwar B (2014) Fast range query processing with strong privacy protection for cloud computing. PVLDB 7(14):1953–1964
17. Lindell Y, Katz J (2014) Introduction to modern cryptography. Chapman and Hall/CRC, Boca Raton
18. Popa RA, Li FH, Zeldovich N (2013) An ideal-security protocol for order-preserving encoding. In: Proceedings of IEEE symposium on security and privacy
19. Popa RA, Redfield CMS, Zeldovich N, Balakrishnan H (2012) CryptDB: processing queries on an encrypted database. Commun ACM 55(9):103–111
20. Tu S, Kaashoek MF, Madden S, Zeldovich N (2013) Processing analytical queries over encrypted data. PVLDB 6(5):289–300
21. Wong WK, Kao B, Cheung DW, Li R, Yiu S (2014) Secure query processing with data interoperability in a cloud database environment. In: Proceedings of ACM SIGMOD conference
22. Xu J, Fan J, Ammar MH, Moon AB (2002) Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. In: Proceedings of ICNP conference
23. https://aws.amazon.com/ec2/pricing/
24. http://dsl.cds.iisc.ac.in/publications/report/TR/TR-2016-01.pdf