ABSTRACT

Privacy-preserving range queries allow encrypting data while still enabling queries on ciphertexts if their corresponding plaintexts fall within a requested range. This provides a data owner the possibility to outsource data collections to a cloud service provider without sacrificing privacy nor losing functionality of filtering this data. However, existing methods for range queries either leak additional information (like the ordering of the complete data set) or slow down the search process tremendously by requiring to query each ciphertext in the data collection. We present a novel scheme that only leaks the access pattern while supporting amortized poly-logarithmic search time. Our construction is based on the novel idea of enabling the cloud service provider to compare requested range queries. By doing so, the cloud service provider can use the access pattern to speed-up search time for range queries in the future. On the one hand, values that have fallen within a queried range, are stored in an interactively built index for future requests. On the other hand, values that have not been queried do not leak any information to the cloud service provider and stay perfectly secure. In order to show its practicability we have implemented our scheme and give a detailed runtime evaluation.

Keywords

Encrypted Database; Secure Computation; Searchable Encryption

1. INTRODUCTION

Cloud computing allows a data owner to outsource her data while enabling her to access this data collection with arbitrary devices anytime. Even devices with small computation power can be used to access an enormous data collection. This is possible by delegating computational expensive operations like searching to the cloud service provider. Then only a small subset matching the search query is processed directly by the client’s device.

In order to preserve data privacy, the outsourced data must be encrypted. However, standard encryption schemes are not suitable for this scenario since they prevent processing encrypted data. As a result, the complete encrypted data collection must be transferred to the client’s device and decrypted and processed locally. Advanced encryption schemes allow the cloud to perform search operations like exact pattern matching or range queries on ciphertexts. In more detail, the data owner can encrypt his files augmented with additional information (e.g., keywords, timestamps). The data owner transfers the ciphertexts created by this scheme to the cloud service provider. Using the secret key the data owner can create a search token (e.g., for exact pattern matching of a keyword, for a range the timestamp should fall within) and pass it to the cloud service provider. Using this search token the cloud service provider can filter for all ciphertexts that match with the search token.

All previous schemes providing this functionality, either have linear search time or leak the complete order of all outsourced values thus are vulnerable to simple yet effective attacks presented recently on property preserving encryption by Naveed et al. in [25]. In this paper we present a novel approach for implementing privacy-preserving range queries with poly-logarithmic search time that only leaks the access pattern, hence prevent such powerful attacks. In our scheme we enable the cloud service provider to compare range tokens that have already been queried in previous search requests. This enables the cloud service provider to decrease its amortized search time for range queries. While initial search time for a range query is linear in the number of indexed files we can speed up future queries as follows: In the first, initial search the cloud service provider learns the result set of the range query; given a range query in a second search request that is a subrange of the already queried range in the first step, it is sufficient to scan this previously learned result set. This downsampling of the possible search space results in a tremendous speed-up for the search operation. Furthermore, using this approach for every new range query the cloud service provider can construct and update an encrypted search index in an interactive protocol between the client and the server. As a result, the scheme achieves decreased search time. In addition, ciphertexts that have never fallen within any queried range are not contained in any access pattern, hence, using a suitable encryption scheme, these unqueried ciphertexts do not leak any information at all.

By implementing a prototype in Python 3 we demonstrate the performance benefits of our construction after a short period of queries. Furthermore, by changing parameters that influence how our index is organized we can decrease com-
putational effort for the client, but increase it on the server side. This combination of different trade-off parameters allows suitable deployments for different use cases. We contribute a new encryption scheme for privacy preserving range queries, whose properties can be summarized as follows:

**secure**: We define and proof security using a simulation-based approach in a widely accepted formal model. In more detail, we define leakage functions that give an upper bound for information that is leaked by our construction.

**efficient**: Our scheme has amortized poly-logarithmic runtime. This is achieved by interactively building a search index. The implementation shows the benefits of this change already after a short period of queries.

**modular**: We build our scheme on a black box interface for functional encryption for secure inner product evaluation. Hence, we can profit from any performance improvements in this active research area. To evaluate this approach we have implemented our scheme based on different functional encryption schemes.

This paper is structured as follows. We give an overview of related work in Section 2. In Section 3 we give a definition of the problem, present two naive solutions with their drawbacks and define the security we want to achieve. Then we present the actual implementation including a proof for its security in Section 4. We go on with a practical evaluation in Section 5 and conclude in Section 6.

2. RELATED WORK

The problem of secure data outsourcing while still enabling computation can be addressed using fully homomorphic encryption [11]. However, due to performance shortcomings of this universal solution, a variety of algorithms and protocols for specific use-cases have been published, e.g., benchmarking [16, 17, 22], RFID tracking [20], reputation systems [18], e-commerce [7].

In this work we focus on search over encrypted data as first proposed by Song et al. in [29]. A scheme for a similar scenario in the public key setting was presented in [5]. Although deterministic encryption can be used and the same functionality has been proposed in [2], searchable symmetric encryption provides better security properties. The main reason why implementations of encrypted databases like CryptDB [26] nevertheless use deterministic encryption is the low deployment overhead. Especially, indexing techniques provided by the database engine can result in huge search time speed-up. Goh published the first scheme using indexing techniques for searchable encryption in [12]. Further improvements for indexing searchable encrypted data are presented in [8, 9]. Recently an idea has been published by Hahn and Kerschbaum in [14] where the index for exact pattern matching is constructed in an incremental way by using information of already searched tokens. From a high level perspective we extend their idea from privacy-preserving exact pattern matching to range queries. For the functionality of range queries a similar trade-off between security and processing time is possible by building search indexes or additional information leakage. The idea of order-preserving encryption was introduced by Agrawal et al. in [1]. In more detail, this kind of encryption has the following characteristic: given two plaintexts \( x \) and \( y \) with property \( x \geq y \), the same property \( \text{Enc}(x) \geq \text{Enc}(y) \) holds for their corresponding ciphertexts. A first concrete implementation of order-preserving encryption was introduced by Boneh et al. [6] and optimized in [21]. However, privacy properties of such encryption schemes might be questionable for highly sensitive data. Although addressed by work like [19], recent work published by Naveed et al. [25] demonstrate concrete attacks on order-preserving encrypted values with low entropy in practice.

The paradigm of searchable encryption for exact pattern matching, i.e. hide as much information as possible by only unveiling tokens corresponding to requested predicates, can be transferred into encryption schemes supporting secure range queries. One solution in the secret key setting has been published in [27]. Solutions for the public key setting exist and have been published in [6, 28]. In [10] this construction has been revisited and realized with searchable encryption for exact pattern matching. This leads to faster execution time but leakage from queries increases e.g. information about the covered subranges is unveiled. The first approach of building search indexes for range queries has been introduced by Lu in [23], however, this index reveals the order of all indexed elements. A trade-off between privacy and performance for range queries is proposed in [15] by using bucketization of indexed ciphertexts. Other tree index approaches have been published by Wang et al. in [30], however, again bucketization of indexed ciphertexts is leaked. In [31] an encryption is used that leaks the relative distance of all indexed ciphertexts to build an R-tree as index for ciphertexts. The leakage of all these indexes results in the vulnerability to the before mentioned attacks as those published by Naveed et al. A comparison of different approaches for secure range queries is presented in Table 1.

3. DEFINITIONS

Let \( \mathbb{N} \) denote the set of natural numbers. We denote \( [i, j] \) with \( i \leq j \) and \( i, j \in \mathbb{N} \) as the set of integers starting at \( i \) and including \( j \), i.e., the set \( \{i, \ldots, j\} \). The output \( z \) of a (possible probabilistic) algorithm \( A \) is written as \( z \leftarrow A \). Throughout, \( \lambda \) denotes the security parameter. A function \( f: \mathbb{N} \rightarrow \mathbb{R} \) is called negligible (in \( x \)) if for every positive polynomial \( p(\cdot) \) there exists a \( x_0 \) such that for all \( x > x_0 \) it holds that \( f(x) < 1/p(x) \). Given matrix \( M \), we denote \( M[i] \) as the \( i \)-th row, and \( M[i][j] \) as the \( j \)-th element in the \( i \)-th row. Further on, message \( m \) consists of a tuple \((f, v)\) where \( f \) is an arbitrary file (i.e., a document or image) and a value point \( v \) for indexing purpose. We assume each message \( m \) has a unique identifier \( ID(m) \) and each file \( f \) can be accessed given the associated identifier \( ID(f) \). This allows us to decouple the actual file \( f \) from the value point \( v \) and identifier \( ID(m) \): one can encrypt \( f \) using an arbitrary encryption scheme, e.g. AES. For range \( Q = [q^{(a)}, q^{(b)}] \) we

| Scheme                      | Sublinear Search Time | Index Leakage |
|-----------------------------|-----------------------|---------------|
| Boneh, Waters [6]          | no                    | n/a           |
| Shi et al. [28]            | no                    | n/a           |
| Shen et al. [27]           | no                    | n/a           |
| Lu [23]                    | yes                   | Order         |
| Wang et al. [30]           | yes                   | Bucketization |
| Wang et al. [31]           | yes                   | Distance      |
| Demertzis et al. [10]      | yes                   | -             |
| This paper                 | yes                   | -             |

Table 1: Comparison of different schemes for privacy-preserving range queries.
define $ID_Q$ as the set of file identifiers indexed under values $v$ with $v \in Q$, i.e., $ID_Q = \{ID(m) : m = (f, v) \text{ with } q^v \leq \gamma \}

3.1 Problem description

A scheme for secure and efficient range queries is composed of the following (partly probabilistic) polynomial-time algorithms: SRQ-Setup, SRQ-Enc, SRQ-IndexFiles, SRQ-Token, SRQ-Search. In the initial step, the data owner creates public parameters and a master key for a desired value domain by running SRQ-Setup. We assume the public parameters are known by all parties and omit them for the sake of simplicity in the remainder of the work. In the next step a message collection is encrypted and indexed under given value points by running SRQ-Enc; each value point has to lie in the value domain used in the initial setup step. The result consisting of an encrypted index and a ciphertext collection is transferred to a server using SRQ-IndexFiles. From this moment on the data owner holding the master key $mk$ is able to create range tokens by calling SRQ-Token. Given this range token to the server he can run SRQ-Search to filter all (encrypted and indexed) messages associated with value points falling within the requested range.

**Definition 1.** The scheme SRQ for secure range queries consists of the following (probabilistic) polynomial-time algorithms:

- $mk \leftarrow$ SRQ-Setup$(1^\lambda, [0, D - 1])$: a probabilistic algorithm that takes a security parameter $1^\lambda$ and value domain $[0, D - 1]$ as input and outputs a master key $mk$.
- $c \leftarrow$ SRQ-Enc$(mk, m)$: a probabilistic algorithm that takes a master key $mk$ and message $m$ as input. Message $m$ is a tuple $m = (f, v)$ of file $f$ and value point $v \in [0, D - 1]$. A ciphertext $c$ is output.
- $\gamma, C \leftarrow$ SRQ-IndexFiles$(ID(m), c_1, n)$: a deterministic algorithm that takes $n$ tuples $(ID(m_i), c_i)$ as input. A secure search index $\gamma$ and a ciphertext collection $C$ is output.
- $\tau_Q \leftarrow$ SRQ-Token$(mk, Q)$: a probabilistic algorithm that takes master key $mk$ and range $[q^v, q^{v'}] = Q \subseteq [0, D - 1]$ as input and outputs a search token $\tau_Q$ for range $Q$.
- $ID_Q \leftarrow$ SRQ-Search$(\tau_Q, \gamma)$: a deterministic algorithm that takes a search token $\tau_Q$ for range $Q$ and index $\gamma$ as input and outputs $ID_Q$.

3.2 OPE and RPE

In the following we describe two solutions for building an encrypted search index that supports range queries as described before: i) sorting all indexed ciphertexts beforehand with order-preserving encryption allows logarithmic search time or ii) scan all indexed ciphertexts linearly using range predicate encryption.

In more detail, the first solution utilizes a scheme OPE = (OPE-Setup, OPE-Enc, OPE-Dec) where OPE-Enc$(x) \leq$ OPE-Enc$(y)$ if and only if $x \leq y$. All index entries of the form $c_i = (OPE-Enc(v_i), ID(m_i))$ are sorted by the OPE encrypted values. For search queries for range $[q^v, q^{v'}]$ a range token is implemented as a tuple $\tau_Q = (OPE-Enc(q^v), OPE-Enc(q^{v'}))$. Given $\tau_Q$ to the server storing the search index she is able to obtain the set $\{OPE-Enc(v_i), ID(m_i) : OPE-Enc(q^v) \leq OPE-Enc(v_i) \leq OPE-Enc(q^{v'})\}$ in logarithmic time by running binary search. However, even indexed but not queried points can be compared with all other indexed (queried and not queried) points. This can be exploited for concrete attacks and result in a total data breach in the worst case as demonstrated by Naveed et al. in [25].

One approach mitigate this attack vector, i.e. to hide the information about the order is Range Predicate Encryption (RPE) introduced by [28] in the public key setting. Later RPE has been transformed to the private key setting by [23] using techniques from [27]. In our work we utilize the approach of range predicate encryption, hence we describe its design and security properties in more detail in this paragraph. An RPE scheme consists of the following algorithms:

- $k \leftarrow$ RPE-Setup$(1^\lambda, [0, D - 1])$ on input of a security parameter $1^\lambda$ and a domain range $[0, D - 1]$ outputs a key $k$.
- $c \leftarrow$ RPE-Enc$(k, v)$ on input of a key $k$ and an attribute value $v$ outputs a ciphertext $c$.
- $tk_Q \leftarrow$ RPE-Token$(k, Q)$ on input of key $k$ and range $Q$ outputs range token $tk_Q$.
- $\{0, 1\} \leftarrow$ RPE-Match$(tk_Q, c)$ on input of range token $tk_Q$ and ciphertext $c = RPE-Enc(k, v)$ outputs 1 if $v \in Q$ and 0 otherwise.

Security for an RPE scheme guarantees plaintext privacy (cf. Definition 2) on the one hand, and predicate privacy (cf. Definition 3) on the other hand.

**Definition 2.** Let RPE be a range predicate encryption scheme. Consider the following security game between attacker $A$ and a challenger consisting of the phases described below:

- **Init:** $A$ submits two values $v_0, v_1 \in [0, D - 1]$ where it wishes to be challenged.
- **Setup:** The challenger generates a secret key $k$ by running RPE-Setup$(1^\lambda, [0, D - 1])$.
- **Query Phase 1:** $A$ adaptively issues queries, where each query is one of two types:
  1. **Token query:** On the $i$-th query, $A$ submits query $Q_i \subseteq [0, D - 1]$ with the following condition: either $(v_0 \notin Q_i \land v_1 \notin Q_i)$ or $(v_0 \in Q_i \land v_1 \in Q_i)$. The challenger generates a token by running $tk_{Q_i} \leftarrow$ RPE-Token$(k, Q_i)$ and outputs it.
  2. **Ciphertext query:** On the $i$-th query, $A$ submits a value $z_i$. The challenger value point $z_i$ by running RPE-Enc$(k, z_i)$ and returns the output.
- **Challenge:** The challenger flips a random coin $b \leftarrow \{0, 1\}$ and outputs RPE-Enc$(k, v_b)$.

**Query Phase 2:** $A$ adaptively issues further queries with the same restrictions as in Phase 1.

- **Guess:** $A$ outputs a guess $b'$ of $b$.

We say RPE has selective secure plaintext privacy, if for all probabilistic polynomial-time attackers $A$ running this security game, it holds that

$$\Pr[b = b'] - \frac{1}{2} \leq \epsilon$$

where $\epsilon$ is negligible in $\lambda$.

**Definition 3.** Let RPE be a scheme for range predicate encryption. Consider the following security game between attacker $A$ and a challenger consisting of the phases described below:
\textbf{Init:} A submits two ranges $R_0, R_1 \subset [0, D - 1]$ where it wishes to be challenged.

\textbf{Setup:} The challenger generates a secret key $k$ by running 
\text{RPE-Setup}(1^\lambda, [0, D - 1])

\textbf{Query Phase 1:} A adaptively issues queries, where each query is one of two types:

1. Token query: On the $i$-th query, $Q_i \subset [0, D - 1]$ is submitted. The challenger generates a token by running $\tau_Q \leftarrow \text{RPE-Token}(k, Q_i)$ and outputs $\tau_Q$.

2. Ciphertext query: On the $i$-th query, value point $z_i$ is submitted such that $z_i \in R_0 \cap z_i \not\in R_1$. The challenger encrypts value point $z_i$ by running $\text{RPE-Enc}(k, z_i)$ and returns the output.

\textbf{Challenge:} The challenger flips a coin $b \leftarrow \{0, 1\}$ and outputs $\text{RPE-Token}(k, R_b)$.

\textbf{Query Phase 2:} A adaptively issues further queries with the same restrictions as in Phase 1.

\textbf{Guess:} A outputs a guess $b'$ of $b$.

We say RPE has selective secure predicate privacy, if for all probabilistic polynomial-time attackers $A$ running this security game, it holds that

$$\Pr[b = b'] - \frac{1}{2} \leq \varepsilon$$

where $\varepsilon$ is negligible in $\lambda$.

Given an RPE scheme with such security properties one can construct a scheme with small leakage but linear runtime. More particular, for message $m_1 = (f_i, v_i)$ the attribute $v_i$ is encrypted to $c_i = \text{RPE-Enc}(v_i)$ and the tuple $(c_i, ID(m_i))$ is indexed. For a range query of range $Q$ a token $\tau_Q$ is created by the data owner holding the master key using \text{RPE-Token}. Given this token, the server creates $ID_Q$ by return all entries $ID(m_i)$ with $\text{RPE-Match}(\tau_Q, c_i) = 1$. Note that it is necessary to scan the complete index, hence runtime is linear in the number of all indexed files.

3.3 Security definition

In order to increase search speed, messages have to be indexed in a suitable way but we want this index to leak as little information as possible. In the next definition present a framework to formalize leakage using the simulation-based definition as introduced by Curtmola et al. in [9].

**Definition 4.** Given a scheme for secure range queries $\text{SRQ} = (\text{SRQ-Setup, SRQ-Enc, SRQ-IndexFiles, SRQ-Token, SRQ-Search})$ and security parameter $\lambda \in \mathbb{N}$, we consider the following probabilistic experiments with adversary $A$, simulator $S$ and leakage functions $L_1, L_2$.

**$\text{Ideal}_{A,S}(\lambda)$:** the simulator sets up its internal environment for domain $[0, D - 1]$. The adversary $A$ sends an $f$-tuple of messages $M = (m_1, \ldots, m_f)$ where $m_i = (f_i, v_i)$ with $v_i \in [0, D - 1]$ and $f_i$ is a file for all $i \in [0, f]$ and a $q$-tuple of queries $Q = (Q_1, \ldots, Q_q)$ with $Q_i \subset [0, D - 1]$ for all $i \in [1, q]$ and the simulator is given the appropriate leakage $L_1(M)$ for message tuple and $L_2(Q)$ for query tuple. Simulator $S$ returns an $f$-tuple $C$ and a $q$-tuple $TK$ to the adversary. Finally, $A$ returns a bit $b$ that is output by the experiment.

We say $\text{SRQ}$ is $(L_1, L_2)$-secure against non-adaptive chosen-range attacks if for all probabilistic polynomial-time algorithms $A$ there exists a probabilistic polynomial-time simulator $S$ so that advantage of $A$ defined as

$$\Pr[\text{Ideal}_{A,S}(1^\lambda) = 1] - \Pr[\text{Ideal}_{A,S}(1^\lambda) = 1]$$

is negligible in $\lambda$.

4. DESIGN

Now we are ready to describe how to organize the search index in order to increase search speed but minimize the leakage of the indexed encrypted values. We tackle these contradictory requirements by updating the index every time the server learns new information. This knowledge, leaked in form of the access pattern and the corresponding search token is then used to refine the encrypted search index for future searches. First, we explain our ideas and design decision on plain data and transfer this on encrypted values in the upcoming section.

4.1 Searching on plaintexts . . .

Search index $\gamma$ consists of the following two components:

**The point list** denoted as $P$ is a linear list of all indexed points. This list enables the server to answer all queries in linear time.

**The tree list** denoted as $T$ is a list of search trees, each tree covering one coherent and already searched range. Whenever a new search is executed, existing trees are updated or a new tree is added to the list. This enables the server to answer range queries that are subranges of already queried ranges in logarithmic time.

Tree list $T$ contains $R$-trees [13]. Each $R$-tree $\Gamma$ covers one coherent range completely. More particular, each inner node holds up to $t$ entries. Every entry has the form $(p, R_p)$, where $R$ is a range and $p$ is a pointer to another node (either an inner node or a leaf) covering this range; hence pointer $p$ points to a subtree. We denote $\Gamma[p]$ as the subtree of $\Gamma$ pointed to by $p$. For simplicity we write $\Gamma \subseteq S$ for a range $S$, if the covered range of $\Gamma$ is a subset of $S$ and vice versa $S \subseteq \Gamma$. In addition, for any two entries $(p_1, R_1)$ and $(p_2, R_2)$ of the same node it holds that $R_1 \cap R_2 = \emptyset$, i.e., the ranges in one node do not overlap. For every entry $(p, R)$ it holds that the subtree rooted at the node pointed to by $p$ covers range $R$, i.e., $\Gamma[p] = R$. Furthermore, all leaves consist of up to $t$ entries, every entry has the form $(obj, R)$, where $R$ is a range and $obj$ points to $ID_{obj}$. Queried range $Q = [q^{(0)}, q^{(1)}]$ the server holding a $R$-tree $\Gamma$ covering a superset of $Q$ (i.e., $Q \subseteq \Gamma$) can calculate $ID_Q$ by using Algorithm 1 in logarithmic time.

A example is given in Figure 1: The initial search index $\gamma$ consisting of point $P$ and tree list $T$ contains one tree $\Gamma$. 

26
covering [4, 16] as depicted in Figure 1a and 1b. Lets assume, the next query is range [17, 23]. In the initial step, the server checks if there exists a tree $\Gamma \in T$ that covers the queried range [17, 23] $\subset \Gamma$ to search in logarithmic time. Since this is not the case, the server scans all entries in $P$ linearly to construct $ID[17,23]$. This new information is then added to the search index for future queries and results an updated version of $\Gamma$ covering [4, 23] as depicted in Figure 1c.

4.2 ... and doing so on encrypted values

Note that all functionality needed for such range queries is the following: first, checking if range $R$ and range $Q$ intersect and second, checking if range $Q$ is a subrange of range $R$. This functionality can be provided by a slightly modified RPE scheme and hence every range query can also be answered over trees that consist of ranges encrypted by this modified RPE scheme. Every token for range $Q$ created by $RPE-Token$ must be augmented with encrypted limiting points (that is start and end point) encrypted using $RPE-Enc$ additionally. This modified version of RPE combined with the idea presented in the previous Section can define an SRQ scheme with poly-logarithmic runtime formally analyzed in Section 4.4 and small leakage formally defined in Section 4.5 as follows:

Given an RPE scheme consisting of algorithms (RPE-Setup, RPE-Enc, RPE-Token, RPE-Match), an IND-CCA secure scheme $\Pi_1 = (Gen^{IND-CCA}, Enc^{IND-CCA}, Dec^{IND-CCA})$ and a second encryption scheme $\Pi_2 = (Gen, Enc, Dec)$ we construct an SRQ scheme as follows:

- $mk \leftarrow SRQ-Setup(1^\lambda, [0, D - 1])$ on input of security parameter and value domain $[0, D - 1]$ create keys $k_1 \leftarrow RPE-Setup(1^\lambda, [0, D - 1])$ and $k_2 \leftarrow Gen^{IND-CCA}(1^\lambda)$ and $k_3 \leftarrow Gen(1^\lambda)$; set $mk = (k_1, k_2, k_3)$. Output master key $mk$.
- $c \leftarrow SRQ-Enc(mk, m)$: on input of master key $mk = (k_1, k_2, k_3)$ and message $m = (f, v)$ do the following:
  - Encrypt $c_1 \leftarrow RPE-Enc(k_1, v)$.
  - Encrypt $c_2 \leftarrow Enc^{IND-CCA}(k_2, f)$.
- Finally, output $c = (c_1, c_2)$.

- $\gamma, C \leftarrow SRQ-IndexFiles((ID(m_i), c_i)_{i \in [1,n]}); Initialize an empty search index $\gamma = (P, T)$ that contains an empty point list $P$, an empty tree list $T$ and an empty ciphertext collection $C$. For each $i \in [1,n]$: parse $c_i = (c_{i0}, c_{i1})$ and add tuple $(ID(m_i), c_{i1})$ to $C$. Furthermore, add tuple $(ID(m_i), c_{i1})$ to point list $P$. Output ciphertext collection $C$ and secure search index $\gamma$.

- $\tau_Q \leftarrow SRQ-Token(mk, Q)$: on input of master key $mk = (k_1, k_2, k_3)$ and range $Q = [q^{(s)}, q^{(e)}]$ flip a coin $b \left\{ 0, 1 \right\}$ and use RPE-Enc for encrypting the limiting points $c_Q^{(b)} = RPE-Enc(k_1, q^{(s)})$ and $c_Q^{(1-b)} = RPE-Enc(k_1, q^{(e)})$. Furthermore create range token $tk_Q = RPE-Token(k_1, Q)$. In addition, encrypt $Q$ to $c_Q = Enc(k_3, Q)$ to enable the client to decrypt this range token. Output $\tau_Q = (c_Q^{(b)}, c_Q^{(1-b)}, tk_Q, c_Q)$ as range token.

- $IDQ \leftarrow SRQ-Search(\tau_Q, \gamma)$ given index $\gamma$ and range token $\tau_Q = (c_Q^{(b)}, c_Q^{(1-b)}, tk_Q, c_Q)$ for range $Q$ check for all index trees in $T$ if they cover the queried values completely or partly. Initialize a list $\tilde{T} = \{G | G \subseteq Q\}$ of trees that lie completely in the queried range $Q$. Then do the following for the boundary points $c_Q^{(b)}$ and $c_Q^{(1)}$:
  1. Check if there exists one tree $G_i \in T$ containing $c_Q^{(b)}$ and $c_Q^{(1)}$. If this is the case, get $ID_Q$ by calling Algorithm 1 and set $\Gamma^{(s)} = \Gamma^{(e)} = G_i$.
  2. Otherwise, check if there exists a tree covering the queried range partly. In more detail, sort tree $\Gamma^{(s)} \in T$ with $c_Q^{(b)} \in \Gamma^{(s)}$ and $\Gamma^{(e)} = \perp$ otherwise. Do the same with $\Gamma^{(b)}$ and $\Gamma^{(1)}$.
  3. Else set $\Gamma^{(s)} = \Gamma^{(e)} = \perp$.

If case 1 does not occur scan all ciphertexts $(ID_Q, c_{i1}) \in P$ using $RPE-Match(tk_Q, c_{i1}) = r_i$ and store $ID_{Q,i}$ in the result set $ID_Q$ iff $r_i = 1$. In order to maintain logarithmic search time for future queries that are a subrange of already queried ranges call an interactive procedure.
SRQ-UpdateIndex($\tau_Q, ID_Q, \Gamma_Q, \Gamma_R, \mathbf{T}$) (described in Section 4.3). Finally, output $ID_Q$ as result.

Given these algorithms it is possible to outsource encrypted data but still support range queries: The initial algorithm SRQ-Setup creates a master key and defines a possible value domain. Next the data owner encrypts his file collection by calling SRQ-Enc, each file is indexed under a value point. The encrypted files and value points are transferred to the server and added to the index via SRQ-IndexFiles. Later, the data owner holding the master key can create search tokens for ranges by calling SRQ-Token. Note that the server can compare different range tokens without knowing the master key. The server can profit from this capability to speed-up future requests by storing previously queried range tokens together with the corresponding result set in an encrypted index structure. More precisely, given two tokens $\tau_Q = (c_Q^{(0)}, c_Q^{(1)}, tk_Q, c_Q)$ and $\tau_R = (c_R^{(0)}, c_R^{(1)}, tk_R, c_R)$ the server is able to check for the following properties:

1. ranges $R$ and $Q$ intersect if RPE-Match($tk_Q, c_R^{(0)}$) = 1 or if RPE-Match($tk_R, c_Q^{(i)}$) = 1 for $i \in \{0, 1\}$.
2. range $R$ is a subrange of range $Q$ if RPE-Match($tk_Q, c_R^{(0)}$) = 1 and RPE-Match($tk_R, c_Q^{(i)}$) = 1.
3. Ranges are equal if $R$ is a subrange of $Q$ and $Q$ is a subrange of $R$.

Using SRQ-Search the server getting a range token $\tau_Q$ for range $Q = [q^{(s)}, q^{(e)}]$ searches for all files associated with values falling within the range $Q$. In the initial step the server checks if he has extracted enough information from previous queries to answer the current query and if that is not to case decides how to update the search index; each tree $\Gamma_i \in \mathbf{T}$ is tested for being a subrange of $Q$ or intersecting with $Q$: All entries $((p_i, \tau_R_i), \ldots, (p_{m_i}, \tau_R_{m_i}))$ contained in the root of any tree $\Gamma_i$ are compared with range $Q$, where $\Gamma_i \subset Q$ if all $R_1, \ldots, R_{m_i}$ are subranges of $Q$. A list $\mathbf{T} = \{\Gamma_i | \Gamma_i \subset Q\}$ of all trees covering a subrange of $Q$ is created, $\Gamma_i$ intersects with $Q$ if at least one range $R_j$ intersects range $Q$. So partial intersections of indexed search trees and the new queried range are computed: Denote $\Gamma_Q$ as the tree containing range $\hat{R}$ covering $c_Q^{(0)}$, i.e., $\hat{R} \in \Gamma_Q$ so that RPE-Match($\hat{R}, c_Q^{(0)}$) = 1. If no such tree was found set $\Gamma_Q = \bot$. The same is done for the encrypted end value $c_Q^{(1)}$ resulting in a tree $\Gamma_Q \in \mathbf{T}$ and $\Gamma_Q = \bot$ otherwise. Depending on the result the are multiple update strategies for SRQ-UpdateIndex described in Section 4.3 in more detail:

1. One tree covers the complete queried range $Q$, that is $\Gamma_Q = \Gamma_Q$, so $Q \subset \Gamma$. If this is the case, the server does not need to perform a search over the complete point list $\mathbf{P}$ but searching over the value points indexed by $\Gamma$ is sufficient. This is done by Algorithm 1. Finally, SRQ-UpdateIndex has to refine indexed ranges by using information gained from the current range query.
2. No intersection of the current range query and previously queried ranges, so $\Gamma_Q = \Gamma$ = $\bot$ and $\mathbf{T} = \emptyset$. If this is the case, the server does not know anything about the current range query. As a result, the server has to scan all points indexed in point list $\mathbf{P}$.
3. Only a part of the queried range is covered by indexed search trees. Either $\Gamma_Q = \bot$ or $\Gamma_Q = \bot$. In this case, the server cannot know if there are values in point list $\mathbf{P}$ falling within $Q$ but are not covered by $\Gamma_Q$ resp. $\Gamma$. As a result, the server has to scan all points indexed in point list $\mathbf{P}$.

Finally, SRQ-UpdateIndex has to extend the one tree covering the queried range partly (the tree that is not $\bot$).

4. The values fall within different trees, that is $c^{(0)} \in \Gamma_Q$ and $c^{(1)} \in \Gamma_Q$ where $\Gamma_Q \neq \Gamma_Q$. If this is the case, the server cannot be sure that there is no “not indexed gap” between the two trees, i.e., there could be values in $\mathbf{P}$ falling neither within $\Gamma_Q$ nor $\Gamma_Q$ but that fall within range $Q$. As a result, the server has to scan all points indexed in point list $\mathbf{P}$.

Finally, SRQ-UpdateIndex has to merge these two trees $\Gamma_Q$ and $\Gamma_Q$ since the gap has been closed by the current range query.

### 4.3 Updating the encrypted index

From a high-level perspective, a new range token contains new information given to the server, namely the result set $ID_Q$ and the set relation to all previous result sets. This newly gained information is implicit in the search token and access pattern. Note that all efficient searchable encryption schemes leak this information and we use this leakage to update the encrypted search index for accelerating future queries. For a formal security analysis of this additional knowledge given to the server we refer to Section 4.5.

As noted in previous Section 4.2 four different update situations SRQ-UpdateIndex can occur, where the server has to either refine one tree, create a new tree, extend one tree, or merge trees. In addition, trees that are covered completely by $Q$ (i.e., contained in $\mathbf{T}$) are composed using a combination of tree extension and tree merges.

Algorithm 2: Rebalancing a tree.

```
RebalanceTree
Input: Tree $\Gamma$, modified leaf $l$
Output: Rebalanced tree $\Gamma'$
Set $cur_{node} = l$;
while $cur_{node} > t$ do
    Send all $c_i = (p_i, \tau_R_i)$ of $cur_{node}$ to client;
    Sort $\{c_i\}$ according to $R_i$, create two tokens
    $\tau_U, \tau_V$ where $U = \bigcup_{i=0}^{m-1} R_i$, $V = \bigcup_{i=0}^{m-1} R_i$ and set nodes $N_U = \{c_i | R_i \subseteq U\}$, $N_V = \{c_i | R_i \subseteq V\}$;
    Send back $N_U, N_V, \tau_U, \tau_V$ to server;
    if $cur_{node}$ is not root then
        Replace $cur_{node}$ with $L_U, L_U$ indexed with tokens $\tau_U, \tau_U$
        in parent of $cur_{node}$;
        Set $cur_{node}$ to parent of $cur_{node}$;
    else
        Replace $cur_{node}$ with $L_V, L_U$;
        Create new root consisting of tokens $\tau_U, \tau_V$
        pointing to $L_V, L_U$;
    end
end
```

Since most operations make it necessary to create new range tokens for encrypted trees and this creation is only possible with the master key, these updates are interactive.
can be achieved using tree extension as described before. Followed by the start point covered by the other tree. This detail the end point covered by one tree must be directly both intersect with the newly queried range. Note that these after tree extension since at least one leaf got a new entry, tree requires the creation of new range tokens, so this also requires to be an interactive protocol.

Refine a tree: The server sends the new range token and previous range tokens that intersect with this new token to the data owner asking for help. The data owner decrypts the range tokens creates (up to) four not intersecting, but more refined ranges and sends back their tokens generated by SRQ-Token. Now the server can replace the old range tokens with the new, more refined tokens and the indexed file lists are segmented according to these new tokens. For a formal description see Algorithm 3. Since this replacement increases the entries in a node, the server finally runs RebalanceTree.

Create a new tree: If \( \Gamma^{(s)} = \Gamma^{(e)} = \perp \) and \( \mathbf{T} \) is empty the server has to create a new tree: The server creates a new, tree \( \Gamma \) with one entry \( \tau_Q \) and indexed item \( ID_Q \). This tree \( \Gamma \) is added to tree list \( \mathbf{T} \).

Extend a tree: A tree should be extended if a new range token intersects partially with a tree, i.e., the range token intersects with the tree, but at least one limiting point of this newly queried range does not. This is started by the server sending the newly learned range token and the root node to the data owner. The data owner decrypts all ranges to reconstruct the whole range currently covered by this tree. A new range token for the gap between the range covered by the tree and the boundary points of the new range token lying outside the tree range is created and added to the tree’s leaf. Furthermore, the tree’s inner nodes (up to the root) are updated, that is, the indexed range of all inner nodes must be replaced by an extended version. See Algorithm 4 for a formal description. The resulting tree must be rebalanced after tree extension since at least one leaf got a new entry, \( \Gamma \).

Merge two trees: Two trees should be merged if they both intersect with the newly queried range. Note that these two trees must not have a value gap between them. In more detail the end point covered by one tree must be directly followed by the start point covered by the other tree. This can be achieved using tree extension as described before.

In order to be able to merge trees in logarithmic time we integrate the tree \( \Gamma \) with the lower height into the tree \( \Gamma \) with greater height. So, a new entry in an inner node of \( \Gamma \) is created pointing to the root of \( \Gamma \). This newly covered range must then be propagated through the inner nodes up to the root. See Algorithm 5 for a formal description. Again, rebalancing the resulting tree is the final step.

Merge multiple trees: If a range token has been queried where multiple trees fall within, we combine the steps of tree extension and tree merging. In more detail, all roots in \( \mathbf{T}, \Gamma^{(s)}, \Gamma^{(e)} \) and the newly queried range token \( \tau_Q \) are sent to the client. The client decrypts all roots and gets ranges \( R_i = [r_i^{(s)}, r_i^{(e)}] \) covered by tree \( \Gamma_i \), sorted according to their range start point \( r_i^{(s)} \). Now the client chooses two trees \( \Gamma_j, \Gamma_{j+1} \) she wants to merge. Without loss of generality, lets assume \( \Gamma_j \) has greater height, so we extend \( \Gamma_j \) to cover \( [r_j^{(s)}, r_{j+1}^{(s)} - 1] \) using Algorithm 4. Now \( \Gamma_j \) and \( \Gamma_{j+1} \) can be merged using Algorithm 5 and the number of different trees is reduced by one. This is done repeatedly until exactly one search tree is left covering the complete queried range.

4.4 Runtime

For simplicity we have assumed a range is not queried multiple times so far. As a result, every token contains new information the server can use for updating index \( \gamma \). Given a value domain with \( D \) elements and \( n \) indexed items, there exist \( \sum_{i=0}^{D} i = \frac{D(D+1)}{2} = O(D^2) \) different coherent ranges that can be queried. So after \( D^2 \) different queries all pos-

1 range of size \( D, \ldots, D \) ranges of size 1.
The expected update time is bound by \( n \), since any update operation adds at least one new

\[
\frac{n \cdot n \log^2(D)}{x} = \log^2(D)
\]

We conclude that last after \( n^2 \) searches we have achieved amortized poly-logarithmic search time.

### 4.5 Security

In this section we give a rigorous security analysis for our protocol. We can decouple encryption of the payload from the encrypted attribute value by using an arbitrary semantic secure encryption scheme. First, the security of tokenized queries using SRQ-Token is examined. Finally, we analyze the whole protocol in a simulator-based framework as introduced in [9].

Before we give a security proof according Definition 4 we define the leakage functions \( L_1, L_2 \) as follows:

\[
L_1(M) = \{(ID, m_i, \text{len}(f_j))\}_{i \in [1, f]} \quad L_2(Q) = \{ID_Q = (ID_{Q_1}, \ldots, ID_{Q_q}), \text{RR}(Q)\}
\]

where \( \text{RR}(Q) \) is a \( q \times q \) range relation matrix, each element is in the set \( \{\emptyset, \cap, \subset, \supset, \subseteq, \supseteq\} \). Here an element in row \( i \) and column \( j \) indicates the relation of ranges \( Q_i \) and \( Q_j \) given in queries \( i \) and \( j \). \( \emptyset \) denotes no intersection, \( \cap \) denotes the equality of two ranges, \( \subset \) denotes a subset relation with one limiting point in common, \( \supset \) denotes a superset relation, and \( \subseteq \) denotes a subset relation with no limiting point in common, and the other way round \( \supset \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \supseteq \) denotes a subset relation with one limiting point in common, and the other way round \( \subseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \supset \) denotes a set relation with no limiting point in common, and the other way round \( \subseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common.

Before we give a security proof according Definition 4 we define the leakage functions \( L_1, L_2 \) as follows:

\[
L_1(M) = \{(ID, m_i, \text{len}(f_j))\}_{i \in [1, f]} \quad L_2(Q) = \{ID_Q = (ID_{Q_1}, \ldots, ID_{Q_q}), \text{RR}(Q)\}
\]

where \( \text{RR}(Q) \) is a \( q \times q \) range relation matrix, each element is in the set \( \{\emptyset, \cap, \subset, \supset, \subseteq, \supseteq\} \). Here an element in row \( i \) and column \( j \) indicates the relation of ranges \( Q_i \) and \( Q_j \) given in queries \( i \) and \( j \). \( \emptyset \) denotes no intersection, \( \cap \) denotes the equality of two ranges, \( \subset \) denotes a subset relation with one limiting point in common, \( \supset \) denotes a subset relation with no limiting point in common, and the other way round \( \supset \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \subseteq \) denotes a subset relation with one limiting point in common, and the other way round \( \supseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common.

Before we give a security proof according Definition 4 we define the leakage functions \( L_1, L_2 \) as follows:

\[
L_1(M) = \{(ID, m_i, \text{len}(f_j))\}_{i \in [1, f]} \quad L_2(Q) = \{ID_Q = (ID_{Q_1}, \ldots, ID_{Q_q}), \text{RR}(Q)\}
\]

where \( \text{RR}(Q) \) is a \( q \times q \) range relation matrix, each element is in the set \( \{\emptyset, \cap, \subset, \supset, \subseteq, \supseteq\} \). Here an element in row \( i \) and column \( j \) indicates the relation of ranges \( Q_i \) and \( Q_j \) given in queries \( i \) and \( j \). \( \emptyset \) denotes no intersection, \( \cap \) denotes the equality of two ranges, \( \subset \) denotes a subset relation with one limiting point in common, \( \supset \) denotes a subset relation with no limiting point in common, and the other way round \( \supset \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \subseteq \) denotes a subset relation with one limiting point in common, and the other way round \( \supseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \subset \) denotes a subset relation with no limiting point in common, and the other way round \( \supseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common.

Before we give a security proof according Definition 4 we define the leakage functions \( L_1, L_2 \) as follows:

\[
L_1(M) = \{(ID, m_i, \text{len}(f_j))\}_{i \in [1, f]} \quad L_2(Q) = \{ID_Q = (ID_{Q_1}, \ldots, ID_{Q_q}), \text{RR}(Q)\}
\]

where \( \text{RR}(Q) \) is a \( q \times q \) range relation matrix, each element is in the set \( \{\emptyset, \cap, \subset, \supset, \subseteq, \supseteq\} \). Here an element in row \( i \) and column \( j \) indicates the relation of ranges \( Q_i \) and \( Q_j \) given in queries \( i \) and \( j \). \( \emptyset \) denotes no intersection, \( \cap \) denotes the equality of two ranges, \( \subset \) denotes a subset relation with one limiting point in common, \( \supset \) denotes a subset relation with no limiting point in common, and the other way round \( \supset \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \subseteq \) denotes a subset relation with one limiting point in common, and the other way round \( \supseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \cap \) denotes the equality of two ranges, \( \subset \) denotes a subset relation with one limiting point in common, and the other way round \( \supseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common, \( \supset \) denotes a subset relation with no limiting point in common, and the other way round \( \subseteq \) denotes that range \( Q_i \) is a subset of \( Q_j \) but no limiting points are in common.
Informal, we remove these restrictions by giving the simulator access to this information in form of the access pattern and the range relation matrix. This is needed to show security of a real whole protocol run, where fulfilling the restrictions of the security games cannot be guaranteed. On the other hand, given two range token sequences with the same range relation matrix (for their ranges), no attacker can distinguish between these range token sequences.

**Theorem 1.** Assume $SRQ$ that is built upon an RPE scheme with selective secure plaintext privacy (cf. Definition 2) and selective secure predicate privacy (cf. Definition 3). Given a domain $[0, D - 1]$, two query sequences $(Q_1, \ldots, Q_n) \subseteq Q \neq R = (R_1, \ldots, R_m)$ with $Q_1, \ldots, Q_n \subseteq [0, D - 1]$ and $RR(Q) = RR(R)$ the probability for any master key $mk \leftarrow SRQ\text{-Setup}(1^\lambda, [0, D - 1])$ the corresponding token tuples $TK_Q = (SRQ\text{-Token}(mk, q_1), \ldots, SRQ\text{-Token}(mk, q_n))$ and $TK_R = (SRQ\text{-Token}(mk, r_1), \ldots, SRQ\text{-Token}(mk, r_n))$

$$\Pr[A(TK_Q) = 1] - \Pr[A(TK_R) = 1]$$

is negligible for any distinguisher $A$.

**Proof.** Denote $\varepsilon_1$ as the probability of an attacker $A$ breaking the used IND-CCA secure encryption scheme, denote $\varepsilon_2$ as the probability of an attacker $A$ winning the RPE plaintext privacy security game and $\varepsilon_2$ as the probability of an attacker $A$ winning the RPE predicate privacy game. Given negligible $\varepsilon_1, \varepsilon_1$ and $\varepsilon_2$ it is possible to extend, shrink and move the ranges, so that the probability of any attacker $A$ to distinguish between a token $\tau_Q$ and token $\tau_R$ that is a extended, shrunk or moved version of $Q$ is negligible.

First, given a range token sequence $\tau_Q = (c_Q^{(0)}, c_Q^{(1)}, t_kQ, c_Q)$, it is possible to extend range $Q$ to range $\bar{Q}$ as long as there is no other range $R$ for which a token $\tau_R$ is known, with $R \cap Q = \emptyset$ but $R \cap \bar{Q} = \emptyset$. In a first step, assume no such range $R$ exists, we later show how to move this range $R$. We present a series of games, and show that the probability of any attacker $A$ to distinguish two games is negligible.

In $G_0$ the original token $\tau_Q$ is given.

In $G_1$ replace $c_Q$ with encryption $c_Q = Enc_{IND-CCA}(\bar{Q})$. $A$ can distinguish between $G_0$ and $G_1$ with probability $\varepsilon_1$.

In $G_2$ we replace $t_kQ$ with this new RPE token $t_kQ = RPE\text{-Token}(\bar{Q})$. Note that $q^{(s)} \notin Q$ and $q^{(s)} \notin \bar{Q}$ still holds. Hence, attacker $A$ can distinguish between $G_1$ and $G_2$ with probability $\varepsilon_2$.

In $G_3$ we move the limiting point $c_Q^{(s)}$ that encrypts $q^{(s)}$: Replace $c_Q^{(s)} = RPE\text{-Enc}(q^{(s)})$ with $c_Q^{(s)} = RPE\text{-Enc}(q^{(s)})$. $A$ can distinguish between $G_2$ and $G_3$ with probability $\varepsilon_1$.

After $G_3$ we have a valid token $\tau_Q$ for the new range $\bar{Q}$. Putting it altogether, attacker $A$ can distinguish between these tokens with probability $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_1$.

Shrinking a range $Q$ to a range $\bar{Q}$ can be done in a similar way, as long as there is no other range $R$ for which token $\tau_R$ is known, with $R \cap Q \neq \emptyset$ but $R \cap \bar{Q} = \emptyset$. We only have to swap $G_3$ and $G_2$. As a result, attacker $A$ can distinguish between a token $\tau_Q$ and a token for a shrunk range $\tau_{\bar{Q}}$ with probability $\varepsilon_1 + \varepsilon_2 + \varepsilon = \tilde{\varepsilon}$.

Combining these two techniques we can move a range $Q = [q^{(s)}, q^{(e)}]$ to a new range $\bar{Q} = [q^{(s)} + x, q^{(e)} + x]$, as long as there is no other range $R$ with $r^{(s)} > q^{(s)}$ but $r^{(s)} < (q^{(s)} + x)$ (otherwise, this range $R$ must be moved before). First, extend $Q$ to a range $Q'$ = $[q^{(s)}, q^{(e)} + x]$, then shrink $Q'$ to range $\bar{Q} = [q^{(s)} + x, q^{(e)} + x]$.

Finally, we can proof Theorem 1: w.l.o.g. first assume $\max_{Q_e \in Q}(q^{(e)}) < \max_{R_e \in R}(r^{(e)})$. First extend the token for $Q$, with the greatest limiting point $q^{(e)}$ to a token for range $[q^{(s)} + x, q^{(e)} + x]$ (using the techniques described before). Repeating this technique for all ranges in descending order of their end point, the complete range sequence $Q$ is modified to an extended range sequence $Q'$ with the same end points as $R$. Last, all ranges in the extend range sequence $Q'$ are shrunk to be identical to range sequence $R$. As shown before, an attacker can distinguish each of these extending an shrink modifications with probability $\tilde{\varepsilon}$ which is negligible. Hence, a combination of polynomial many modifications is still negligible.

Given this Theorem we are now ready to prove the security of our protocol in a formal way using leakage based Definition 4 as it has been introduced by [9] together with the defined leakage functions $\Pi_1, \Pi_2$ at the beginning of this section.

**Theorem 2.** If the used RPE scheme has selective secure plaintext privacy based on an RPE scheme with selective secure predicate privacy and $\Pi_1$ is an IND-CCA secure encryption scheme, then $SRQ$ as described in Definition 4.2 is $(\Pi_1, \Pi_2)$-secure against non-adaptive chosen-range attacks.

**Proof.** We present a PPT simulator $S$ for which the advantage of any PPT adversary $A$ to distinguish between the $Real_A$ and $Ideal_A,S$ experiments from Definition 4 is negligible. For this, we describe $S$ setting up the environment and simulating range tokens $TK$ and ciphertexts $C$ using leakage $\Pi_1$ and $\Pi_2$.

**Setting up the environment:** $S$ internally runs setup algorithm $SRQ\text{-Setup}(1^\lambda, [0, D - 1])$ and receives a master key $mk$.

**Simulating $TK$:** $S$ extracts clusters of ranges that form one big coherent range using Algorithm 6.

Each cluster is a separate $R$-Tree in the implementation presented in Section 4.2. For every cluster $C$ simulates ranges with the same range relation matrix as the actual given range relation matrix $BR(Q)$. In more detail, for every cluster simulator $S$ transforms the range relation matrix $BR(Q)$

| $\sim$ | $\cap$ | $\subset$ | $\subset^*$ | $\emptyset$ |
|---|---|---|---|---|
| $R$ | $Q$ | $R$ | $Q$ | $R$ |
| $R$ | $Q$ | $R$ | $Q$ | $R$ |

Table 2: Illustration and formal representation of different relationships between ranges.
Algorithm 6: Algorithm for extracting range clusters.

Init empty list of lists clusters;
Init empty list of used indexes U;
while \(|U| < q\) do
  Init two empty lists \(N, G\);
  Add random index from \([0, q]\ \setminus U\) to \(N\);
  while \(N\) not empty do
    choose random row index \(i\) from \(N\);
    for \(0 < j < q\) do
      if \(\mathsf{RR}(Q) \neq \emptyset\) and \(j \notin U\) then
        Add \(j\) to \(N\);
      end
    end
    Add \(i\) to \(U\) and \(G\);
    Remove \(i\) from \(N\);
  end
  Add \(G\) to \(clusters\);
end

5. EVALUATION

For evaluating our SRQ scheme we implemented a prototype in Python 3 using bindings for the PBC library (version 0.5.14) [24]. The runtime benchmarks are executed on a machine running Ubuntu 14.04 with 8GB RAM and an Intel Xeon 1230v3 CPU 3.3GHz.

We count the average number of comparisons for one range query, i.e., how often the server must run \(\mathsf{RPE-Match}\). All files are indexed under random value point sets distributed among the complete value domain. In addition, every range query is generated randomly with a size between 1 and a defined upper limit, starting from an arbitrary point in the domain. The upper limit is given as a factor of the complete domain size, for example given a domain size 1000 and a query factor \(10^{-1}\) the range for one query may have a length between 1 and 100. By varying the range size we can modify the probability for two ranges to intersect, hence we can influence the probability of merging and extending trees. Furthermore, we analyzed the number of trees our index consists of.

Next update costs on the client side are analyzed. For this we counted the number of range decryption operations and range creations.

Finally, we present micro benchmarks for encrypting one data point by running \(\mathsf{SRQ-Enc}\), creating a search token by \(\mathsf{SRQ-Token}\) and checking two tokens generated for intersection depending on the domain size and security parameter. Since we use range predicate encryption as a black box, we can change the underlying implementation without modifying our construction. For our demonstration we implemented the values for secure inner-product evaluation from [27] and [3] and utilized them for \(\mathsf{RPE}\) as described in [23].

Searching and updating trees: All measurements presented here are repeated five times and we use the mean values. For this section we assumed a domain \(D = 2^{26}\) and a number of index files of \(2^{26}\). Furthermore we grouped 50 values for one data point, e.g., 50 successive search queries are represented by one data point in Figure 2a. By modifying the maximum size of one range query we also modify the probability for range intersections. As a result, the number of merge operations vary, hence also the number of trees are stored in the search index vary. This trade off is summarized in Table 3, where the number of indexed trees is given as a function of the number of already searched ranges and the query factor. These trees index a smaller average range, hence the server must scan the complete file list more often resulting in more \(\mathsf{RPE-Match}\) calls, as depicted in Figure 2a. Here it does not matter if \(\mathsf{RPE-Match}\) is called for comparing two range tokens or checking if an indexed file falls within the queried range. In the worst case, there is a huge amount of trees, each covers only a small range. Now given a new small range token, all these indexed trees must be searched. If no match was found, the complete point list must be scanned additionally, resulting in even more searches than a linear scan of all files would.

| Number of preceding searches | 100 | 1000 | 5000 | 20000 | 40000 |
|-----------------------------|-----|------|------|-------|-------|
| query factor \(2^{-11}\)    | 98.8| 784.2| 1467 | 149.4 | 3.2   |
| query factor \(2^{-10}\)    | 100 | 993.2| 4213.8| 1713.2| 2944.8|

Table 3: Mean number of trees after five runs.

Furthermore, the number \(t\) of entries one node holds, influences the number of \(\mathsf{RPE-Match}\) calls: A greater threshold
results in more RPE-Match calls per node and consequently in the overall number of RPE-Match calls as depicted in Figure 2b. On the other hand, we can decrease the probability of calling the interactive protocol RebalanceTree by increasing the number of entries one node can hold. As a result, the server asks for help less often, hence the number of token generations can be decreased as presented in Figure 2c.

**Microbenchmarks:** In our SRQ implementation we used the construction from [28] utilizing functional encryption for inner products. For the secret key setting such a scheme was presented in [27] based on pairings and already used in [23]. In addition, we implemented schemes providing such functionality that have been published recently in [3]. We denote our implementation using the scheme from [27] as SRQ-BJK and the scheme from [3] as SRQ-SSW. Note that SRQ-BJK avoids pairings, however, this construction leaks the actual range \( R \) given a range token \( \tau_R \). Two parameters affect the runtime: the used security parameter benchmarked in Table 4; the possible domain size, benchmarked in Table 5. In SRQ-Enc we omitted the actual file encryption operation using an IND-CCA secure encryption scheme. Its runtime depends on the file size and the used encryption scheme is a well studied problem.

| \( |S| \)     | 80 bits | 128 bits | 256 bits |
|-------------|---------|----------|----------|
| SRQ-Enc      | 18 ms   | 41 ms    | 141 ms   |
| SRQ-Enc-BJK  | 385 ms  | 854 ms   | 2466 ms  |
| BJK Token intersection | 230 ms | 531 ms | 1997 ms |
| SRQ-Enc-BJK  | 383 ms  | 1117 ms  | 3898 ms  |
| SRQ-Token    | 9660 ms | 34045 ms | 143173 ms|
| SSW Token intersection | 1553 ms | 40625 ms | 144121 ms|

Table 4: Microbenchmarks for domain size \( 2^{32} \).

| \( |S| \)     | \(2^1\) | \(2^{20}\) | \(2^{32}\) |
|-------------|--------|--------|--------|
| SRQ-Enc      | 16 ms  | 26 ms  | 141 ms |
| SRQ-Enc-BJK  | 114 ms | 363 ms | 2406 ms|
| BJK Token intersection | 56 ms | 220 ms | 1897 ms |
| SRQ-Enc-BJK  | 943 ms | 9435 ms| 11477 ms|
| SRQ-Token    | 11685 ms | 15071 ms | 34045 ms |
| SSW Token intersection | 11685 ms | 14301 ms | 40625 ms |

Table 5: Benchmark for fixed security parameter 128 bits.

**Putting it all together:** Finally, we present 5 runs of real searches. We implemented the RPE-scheme using the BJK with 80 Bits security parameter. Here we encrypted \( 2^{16} \) files and indexed them under value points, sampled randomly out of domain \( D = [0, 2^{32} - 1] \). Figure 3 shows the mean values of all runs, where five searches are aggregated in one bar. We measured the pure search time that is performed merely on the server side. Additionally, the needed update time was measured; here the index is updated in the interactive way, hence the client and the server are involved. By adding these times we get the complete execution time for one search. Furthermore, the duration of one linear scan of all files is depicted as a dashed line. As we can see, already after 5 search operations the execution time that includes building an index is lower than the linear search time, so we can profit from this index construction.

6. CONCLUSION

In this paper we proposed a novel approach for performing range queries. The server can decrease search time for future queries by updating a search index using the access patterns learned from past queries. We analyzed this effect on the runtime theoretically and empirically and have presented a simulation based security proof as it is state of the art for searchable encryption. Our leakage is tremendously smaller compared with previous schemes for privacy-preserving range queries with polylogarithmic runtime. Furthermore, our construction utilized functional encryption for inner product evaluation as a black-box functionality, so one can exchange the underlying algorithm without modifying our scheme. As a result, our construction profits from all future improvements in this research area. By implementing our scheme we demonstrate its feasibility and point out different parameters to adjust search time and complexity on the client side. This adjustment enables us to deploy our scheme in varying scenarios.

**Acknowledgments**

This work has received funding from the European Union’s Seventh Framework Programme and Horizon 2020 Research and Innovation Programme under grant agreements No. 609611, No. 644579 and No. 644412 of the PRACTICE, ESCUDO-CLOUD and TREDISEC projects.
7. REFERENCES

[1] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. Order preserving encryption for numeric data. In Proceedings of the ACM International Conference on Management of Data (2004), SIGMOD.

[2] Bellare, M., Boldyreva, A., and O’Neill, A. Deterministic and efficiently searchable encryption. In Proceedings of the 27th International Conference on Advances in Cryptology (2007), CRYPTO.

[3] Bishop, A., Jain, A., and Kowalczyk, L. Function-hiding inner product encryption. In Advances in Cryptology (2015), ASIACRYPT.

[4] Boldyreva, A., Chenette, N., Lee, Y., and O’Neill, A. Order-preserving symmetric encryption. In Proceedings of the 28th International Conference on Advances in Cryptology (2009), EUROCRYPT.

[5] Boneh, D., Di Crescenzo, G., Ostrovsky, R., and Persiano, G. Public key encryption with keyword search. In Advances in Cryptology (2004), EUROCRYPT.

[6] Boneh, D., and Waters, B. Conjunctive, subset, and range queries on encrypted data. In Proceedings of the 4th Theory of Cryptography Conference (2007), TCC.

[7] Catrina, O., and Kerschbaum, F. Fostering the uptake of secure multiparty computation in e-commerce. In Proceedings of the third International Conference on Availability, Reliability and Security (2008), ARES.

[8] Chang, Y.-C., and Mitzenmacher, M. Privacy preserving keyword searches on remote encrypted data. In Proceedings of the International Conference on Applied Cryptography and Network Security (2005), ACNS.

[9] Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. Searchable symmetric encryption: improved definitions and efficient constructions. In Proceedings of the 13th ACM Conference on Computer and Communications Security (2006), CCS.

[10] Demertzis, I., Papadopoulos, S., Papapetrou, O., Deligiannakis, A., and Garofalakis, M. Practical private range search revisited.

[11] Gentry, C. A fully homomorphic encryption scheme. PhD thesis, Stanford University, 2009.

[12] Goh, E.-J. Secure indexes. IACR Cryptology ePrint Archive, 216 (2003).

[13] Guttman, A. R-trees: a dynamic index structure for spatial searching. In Proceedings of the ACM International Conference on Management of Data (1984), SIGMOD.

[14] Hain, F., and Kerschbaum, F. Searchable encryption with secure and efficient updates. In Proceedings of the 21st ACM Conference on Computer and Communications Security (2014), CCS.

[15] Hose, B., Mehratosa, S., and Tsudik, G. A privacy-preserving index for range queries. In Proceedings of the 32nd International Conference on Very Large Data Bases (2004), VLDB.

[16] Kerschbaum, F. Building a privacy-preserving benchmarking enterprise system. Enterprise Information Systems 2, 4 (2008).

[17] Kerschbaum, F. Practical privacy-preserving benchmarking. In Proceedings of the IFIP International Information Security Conference (2008), SEC, pp. 17–31.

[18] Kerschbaum, F. A verifiable, centralized, coercion-free reputation system. In Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society (2009), WPES.

[19] Kerschbaum, F. Frequency-hiding order-preserving encryption. In Proceedings of the 22nd ACM Conference on Computer and Communications Security (2015), CCS.

[20] Kerschbaum, F., and Oertel, N. Privacy-preserving pattern matching for anomaly detection in rfid anti-counterfeiting. In International Workshop on Radio Frequency Identification: Security and Privacy Issues (2010), RFIDSec.

[21] Kerschbaum, F., and Schröpfer, A. Optimal average-complexity ideal-security order-preserving encryption. In Proceedings of the 21st ACM Conference on Computer and Communications Security (2014), CCS.

[22] Kerschbaum, F., and Terzidis, O. Filtering for private collaborative benchmarking. Emerging Trends in Information and Communication Security (2006).

[23] Lu, Y. Privacy-preserving logarithmic-time search on encrypted data in cloud. In Proceedings of the 19th Network and Distributed System Security Symposium (2012), NDSS.

[24] Lynn, B. PBC library - the pairing-based cryptography library. https://crypto.stanford.edu/pbc.

[25] Naveed, M., Kamara, S., and Wright, C. V. Inference attacks on property-preserving encrypted databases. In Proceedings of the 22nd ACM Conference on Computer and Communications Security (2015), CCS.

[26] Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. Cryptdb: protecting confidentiality with encrypted query processing. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (2011), SOSP.

[27] Shen, E., Shi, E., and Waters, B. Predicate privacy in encryption systems. In Proceedings of the 6th Theory of Cryptography Conference (2009), TCC.

[28] Shi, E., Bethencourt, J., Chan, H. T.-H., Song, D. X., and Perrig, A. Multi-dimensional range query over encrypted data. In Proceedings of the 2007 Symposium on Security and Privacy (2007), S&P.

[29] Song, D. X., Wagner, D., and Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the 21st IEEE Symposium on Security and Privacy (2000), S&P.

[30] Wang, B., Hou, Y., Li, M., Wang, H., and Li, H. Maple: Scalable multi-dimensional range search over encrypted cloud data with tree-based index. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (2014), ASIA CCS.

[31] Wang, P., and Ravishankar, C. Secure and efficient range queries on outsourced databases using R-trees. In Proceedings of the 30th IEEE International Conference on Data Engineering (2013), ICDE.