Hierarchical Range Queries on Encrypted Data

Master's Thesis

September 16, 2013

Olga Havlíčková

Supervisors:
Dr. Zekeriya Erkin
Prof. Dr. W.J. Fokkink
Dr. ir. Thijs Veugen

VU VRIJE UNIVERSITEIT AMSTERDAM

TU Delft Delft University of Technology

Contents

1 Introduction

Nowadays, tracking systems monitoring the location of vehicles or people are widely used and the services based on tracking systems offer a number of useful applications. Information collected from a vehicular tracking system can be used for analyzing road congestion and consequent improvement of traffic forecasting and road planning. Systems for tracking people can be used for detecting anomalous behaviour in sensitive areas [14].

Although the benefits of the services based on tracking systems are indisputable, tracking systems are often considered the ultimate "big brother" and their usage creates legitimate privacy concerns. Exposing location data to malicious parties may pose significant privacy risks. From personal location data, one can, for instance, make conclusions about the user's medical condition if the person is often seen in a hospital, or one can deduce information about the person's political activity, in case the person visits certain political events. Therefore, location data have to be adequately protected in order to prevent any data exploitation.

Typically, the entity running the tracking system would like to store the collected location data for future use and analysis and potentially allow authorized clients to access the data. However, the place of the data storage may not be under full control of the operator of the tracking system. Outsourcing location data to external data centers raises a question whether the stored information is safe and how to minimize the threat of exposing the data to an entity with iniquitous intentions.

There are two straightforward and easy-to-employ solutions for protecting sensitive outsourced data. One is to define a privacy policy between the data creator and the entity storing the data. Another one is to present an access control mechanism for the data manipulation. After taking a closer look at these methods, none of them provides sufficient guarantees of protecting the stored information from being handled in an undesired manner [16]. In both cases, the data is stored in its plaintext form. The storage provider may secretly diverge from the established rules, or may not be able to protect the stored data from leakage to an external malicious party [8].

Cryptography gives an answer to this problem. If we encrypt the dataset before sending it for storage, the jeopardy of breaching the data confidentiality disappears. Unfortunately, when we employ conventional encryption techniques the ability to efficiently search over the encrypted data is lost. Downloading and decrypting all the stored data and performing the search locally is impractical and goes directly against the idea of outsourcing the data. To maintain the ability of efficient search over the encrypted data a special encryption method, searchable encryption, needs to be used [22].

The main goal of this work is to develop a hierarchical range query mechanism for encrypted databases. Consider a situation where the output of a tracking system is stored in an outsourced database and several clients are eligible to search in the dataset. To preserve the privacy to the largest possible extent, the clients are allowed to learn the location data only with a pre-defined precision. The pre-defined precision may be lower than the original precision of the location information returned by the tracking system. The hierarchical range query mechanism guarantees that unauthorized entities are not

able to learn the content of the database and each client eligible to access the database can retrieve only the location data of the precision assigned to this particular client.

The rest of the thesis is structured as follows. We discuss the related work in Section 2. In Section 3, we define the research problem in detail and present an application scenario. In Section 4, we describe the existing encryption schemes needed for our constructions, present two constructions of a hierarchical range query mechanism and discuss the implementation details and analyze the performance. Finally, in Section 5, we outline the directions for further work.

2 Related Work

Searchable encryption schemes provide an efficient way of searching in the encrypted domain. Such cryptographic schemes were proposed in both symmetric and asymmetric settings.

In the symmetric setting, only a single entity, who is in possession of the secret symmetric key, is able to generate searchable content, issue valid queries and decrypt the results. The concept of searchable encryption and the first searchable encryption scheme in a symmetric setting was presented by Song et al. in [21]. The basic principle of the scheme is to divide the data into blocks, encrypt them independently and then base the search on the blocks' content. Its main disadvantage is the need of a sequential scan of the entire database, which can lead to impractical search times for large datasets. Other searchable encryption schemes in the symmetric setting are described in [3], [13] and [11].

In the asymmetric setting, the client publishes his public key and everyone with access to this public key is able to contribute searchable content to the database. Only the owner of the private key has the capability to query the database and decrypt corresponding results. Boneh et al. presented the first searchable encryption scheme in the asymmetric setting in [7]. In this scheme every encrypted item is appended with a set of keywords encrypted with the public key. To perform a search for a particular keyword, a trapdoor is needed. The private key is used for computing the trapdoor, therefore only the owner of the private key can generate valid queries. More constructions in the asymmetric setting can be found in [1], [6] and [15].

The research on searchable encryption at first covered only exact-match queries. Over time schemes evolved for supporting more advanced types of queries such as conjunctive and disjunctive keyword queries [4, 15, 17], subset queries [4] and range queries [4, 20].

There are two seminal constructions supporting multidimensional range queries on encrypted data. Boneh et al. proposed a new cryptographic primitive called Hidden Vector Encryption [4]. This primitve can be implemented to support several types of queries: next to multidimensional range queries also subset and conjunctive queries. Another scheme developed by Shi et al. [20] uses binary trees to represent ranges in multiple dimensions. A thorough survey on searchable encryption techniques can be found in [22].

3 Problem Definition

Imagine an airport with a security system that allows identifying the location of people within the area of the airport. Benefits of such a system are for example speeding up the boarding process by checking the location of a passenger who did not arrive at the gate on time, or optimizing the number of staff at security checkpoints.

The system is owned and run by a single administrator. We call this administrator the authority. The authority would like to store the collected data in a database in order to allow third parties in justifiable cases to either learn the position of a particular person, or identify people who were at some location during a specific time interval. Entities eligible to query the database can be the local police department or the airport administration. We call these parties clients.

Assume the authority does not possess an appropriate facility for storing the data and handling the clients' queries and wants to outsource the database. Since location data are of a sensitive nature, it is necessary to prevent the database manager from revealing the database content. Therefore the authority has to encrypt the database records before sending them to the database manager for storage.

In situations where a number of clients is able to query the database, it may be desirable to lower the location data precision in results returned to some clients. For instance, the police will have access to the records of the highest possible precision, while the airport administration will be allowed to get only less precise results.

In Section 3.1 the parties engaged in the application scenario are described. In Section 3.2, the research objective is discussed. Finally, necessary security requirements are given in Section 3.3.

3.1 Involved Parties

Our model includes four parties. Without loss of generality we follow the example of the airport tracking system for presenting the involved entities and their mutual relationship. Those entities are:

- **Passengers**
  Passengers play a passive role in our model – their location data are collected by the tracking system and further processed by the authority. They fully trust the authority will not handle their location data in an improper manner.

- **Authority**
  The authority is the entity in possession of the tracking system. It has the exclusive capability to collect and encrypt the location data of passengers and to subsequently send them to the database manager for storage. Furthermore, the authority is responsible for distributing search capabilities amongst clients in such a way that each client is granted access to location information of pre-defined precision and it is infeasible for the client to obtain more precise data.

- **Database Manager**
  The database manager stores the encrypted database and performs the search on request of a client. We assume the database manager to be honest-but-curious. It means that he does not deviate from the protocol, yet he is interested in the database records and may analyze the content of the database and communication between the clients and himself in order to deduce any useful information about the stored data. The honest-but-curious model fits our application – we assume the database manager has economic interest in following the protocol as agreed on, but on the other hand is not fully trusted by the authority.

- **Clients**
  Clients are external entities interested in the records stored in the database such as the local police department or airport security. After they obtain search capabilities from the authority they are able to send a valid query to the database manager and consequently obtain matching database records.

3.2 Research Objective

In this section, we describe the actual structure of the database and two types of supported queries. For simplicity we show them on the plaintext preimage of the database, but the reader should keep in mind that in our future scheme the database must be encrypted to prevent the database manager from accessing the stored data.

We assume that the raw output data received from the tracking system have the form

$$\langle ID, x, y, t \rangle,$$

where ID uniquely identifies a person, x and y are the coordinates defining the person's exact location and t is the timestamp. The tuple $\langle ID, x, y, t \rangle$ can be split into two logical parts – the person's identifier ID and the triplet of spatiotemporal information $\langle x, y, t \rangle$. Based on this division we can organize the database in two different manners. Either the index consists of the tuples $\langle x, y, t \rangle$ and the corresponding IDs form the content of the record, or the set of IDs form the searchable index and each record is the sequence of spatiotemporal points where ID was tracked. We call the associated two types of queries *Region Inspection* and *Object Location*, respectively.

- **Region Inspection**

  The first option is to form the searchable index from triplets $\langle x_i, y_i, t_i \rangle$. Then the record corresponding to $\langle x_i, y_i, t_i \rangle$ consists of the identifier ID_i of the person who was at time t_i at location $\langle x_i, y_i \rangle$. Therefore the database looks as follows:

Index	Record
$\langle x_1, y_1, t_1 \rangle$	ID_1
$\langle x_2, y_2, t_2 \rangle$	ID_2
$\vdots$	$\vdots$
$\langle x_n, y_n, t_n \rangle$	ID_n

Such database layout supports queries where the client sends to the database manager a spatiotemporal range $\langle [x_\ell, x_u], [y_\ell, y_u], [t_\ell, t_u] \rangle$ and the database manager returns identifiers ID_i of all passengers whose spatiotemporal data fall within the queried range:

Query: $\langle [x_\ell, x_u], [y_\ell, y_u], [t_\ell, t_u] \rangle$

Result: $\{ID \,|\, \langle ID, x, y, t\rangle \in DB, x \in [x_\ell, x_u], y \in [y_\ell, y_u], t \in [t_\ell, t_u]\}$

Assume we assigned to each client a maximum precision level on which he is able to send queries to the database manager. Our aim is to build query mechanism that guarantees this property and makes it impossible for the client to send more precise queries. As an additional requirement, we would like the clients to obtain the answer during one communication round with the database manager and avoid the need of iterating the search from the lowest to the desired precision level. It is a reasonable assumption that in many cases the clients are interested in the data recently added to the database rather than in the records created a long time ago – recall the situation mentioned at the beginning of Section 3 when a passenger is late for boarding. Therefore the authority has to be able to create and send encrypted records for storage in short time intervals. Moreover, the authority should be able to distribute the search capabilities to clients at the setup stage and not having to grant permission for every query, since that would lead to an undesirable communication overhead.

- **Object Location**

  The second way how to organize the database is to form the searchable index from the passengers' identifiers ID_i. The corresponding record is then a sequence of spatiotemporal points $\langle x_{i,1}, y_{i,1}, t_{i,1}\rangle, \cdots, \langle x_{i,n}, y_{i,n}, t_{i,n}\rangle$ defining the ID_i's movement in time. The database looks as follows:

Index	Record
ID_1	$\langle x_{1,1}, y_{1,1}, t_{1,1}\rangle, \cdots, \langle x_{1,n}, y_{1,n}, t_{1,n}\rangle$
ID_2	$\langle x_{2,1}, y_{2,1}, t_{2,1}\rangle, \cdots, \langle x_{2,n}, y_{2,n}, t_{2,n}\rangle$
$\vdots$	$\vdots$
ID_m	$\langle x_{m,1}, y_{m,1}, t_{m,1}\rangle, \cdots, \langle x_{m,n}, y_{m,n}, t_{m,n}\rangle$

  In this scenario, the client sends to the database manager the ID of the person whose location the client wishes to learn. The database manager returns a sequence of all spatiotemporal points associated with ID:

  Query: ID

  Result: $\{\langle x, y, t\rangle \,|\, \langle ID, x, y, t\rangle \in DB\}$

The requirements on the functionality of the scheme for Object Location are analogous to the requirements on the scheme for Region Inspection. Again, we assume each client is given a pre-defined precision level for each of the three dimensions x, y and t and he must not be able to obtain the passengers' location data with a higher resolution in any dimension. Also, the client should obtain the answer from the database in one communication round. Regarding the authority, there are two main requirements. It must be feasible to add new records to the database dynamically and granting search capabilities should be a one-time action.

3.3 Security Objective

Security objectives specify what requirements the constructed scheme has to fulfil to provide demanded functionality and yet prevent all actors in our model from finding out more about the database content than they are allowed to learn. The security requirements are concerning the database and the queries.

- **Database requirements**

  We want the database content to stay confidential. In other words, it must be computationally infeasible to reveal the plaintext version of database records when given access to the encrypted database.

- **Queries requirements**

  1. Only eligible clients are able to generate valid queries for the database.
  2. The content of the query is concealed from ineligible entities.
  3. Given capabilities for a spatiotemporal region of pre-defined resolution, the client is not able to derive capabilities for a spatiotemporal region of higher resolution, i.e. clients are not able to circumvent the hierarchical mechanism.

4 Hierarchical Range Queries on Encrypted Data

One of the two query types for outsourced databases with location data presented in Section 3.2 deals with the situation when a querying client would like to know who was, during a given time interval, at a particular area within the airport. The client sends to the database manager a query defined by a range $\langle [x_\ell, x_u], [y_\ell, y_u], [t_\ell, t_u] \rangle$ and the database manager returns identifiers of all passengers who were tracked by the surveillance system at a location and time which falls within the queried spatiotemporal range. We called this type of query *Region Inspection*.

The second type of query from Section 3.2 referred to the dual scenario, when a client wants to know the path of one particular passenger in the airport. The client's query is defined by the passenger's identifier ID and the result returned by the database manager consists of all spatiotemporal entries $\langle x, y, t \rangle$, where the passenger ID was spotted by the tracking system. We named this query type *Object Location*.

In Section 4.1 we present a scheme for multidimensional range queries, give a detailed description of the protocol for Region Inspection based on this scheme and eventually discuss the space and time consumption of the protocol's implementation. Similarly, in Section 4.2 we first describe a symmetric searchable scheme and highlight the techniques used for two proposed protocols for Object Location, then we point out the protocols' benefits and disadvantages and finally we discuss the performance of the implementations.

4.1 Region Inspection

In the Region Inspection query type every plaintext record M has several attributes of exact value. The goal is to encrypt M with its attributes in such a way that it is possible to efficiently perform range queries on these encrypted attributes. That is, given the decryption key for a particular range we can decrypt all records M with attributes within the given range.

We give a more formal definition of this property. Assume M is encrypted with its D attributes $(a_1, \ldots, a_D)$. Let the ith attribute a_i be an element of range $[1, T_i]$. We would like to release decryption keys, where each key corresponds to a specific D-dimensional range

$$\langle [x_1, y_1], [x_2, y_2], \ldots [x_D, y_D] \rangle.$$

Given the decryption key corresponding with these range parameters one can successfully decrypt all the records encrypted with attributes $(a_1, a_2, \ldots, a_D)$ such that

$$a_1 \in [x_1, y_1], a_2 \in [x_2, y_2], \ldots, a_D \in [x_D, y_D].$$

If at least one attribute does not match its dimension, the decryption will be unsuccessful. In our application, the plaintext M equals to the identifier ID and the number of dimensions D is three – two dimensions represent spatial coordinates x and y, one dimension represents the temporal data t.

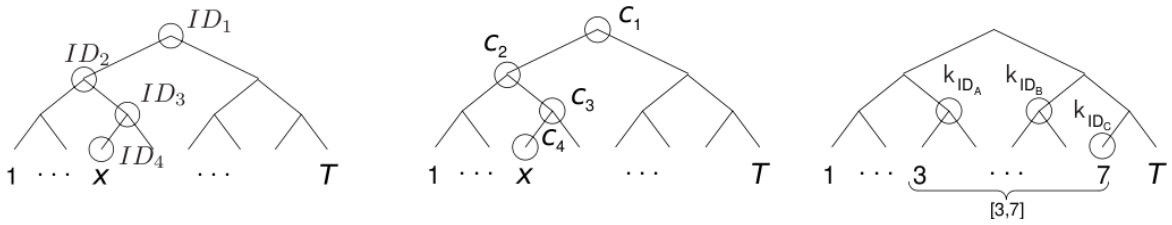

Figure 1: **(Left)** The path $\mathbb{P}(x) = \{ID_1, ID_2, ID_3, ID_4\}$. **(Middle)** Set of ciphertexts $C = \{c_1, c_2, c_3, c_4\}$ of the encryptions with path $\mathbb{P}(x)$. **(Right)** Decryption keys for the range $[3, 7]$. [20]

4.1.1 Range Query Mechanism

In this section we introduce a mechanism for supporting multidimensional range queries over encrypted data (MRQED), which was proposed by Shi et al. [20]. We use the MRQED scheme as the starting point for our solution.

There are three types of parties considered in the MRQED scheme. Firstly, an authority who creates the records and is able to grant search competences for the encrypted database. Then an untrusted repository storing the encrypted records. And lastly, several clients, who request decryption keys for a region from the authority.

We first describe the construction for one-dimensional queries and then extend the mechanism to multiple dimensions. The basic idea is to build D binary trees, one for every dimension, and assign each node in the tree a range. Assigning the range follows a simple principle: leaf nodes represent an exact value and every internal node represents the union of ranges of its child nodes.

The MRQED is a public-key encryption scheme. Each node has a unique identifier which can be used for deriving the node's public key for identity based encryption. Identity-based encryption is a public key encryption scheme in which the public key can be an arbitrary string and a master key is needed for deriving the private key. In MRQED the owner of the secret master key is the authority. Since identity-based encryption is not part of the end solution as presented in this thesis we will omit further details of this encryption scheme. The interested reader is referred to [5] and [9].

To encrypt a message M under an exact value x we identify the path $\mathbb{P}(x)$ from the root to the node representing x and encrypt M separately with all public keys of the nodes in the path $\mathbb{P}(x)$. The resulting ciphertext C sent to the repository for storage is the sequence of component ciphertexts. An example is shown in Figure 1 – to encrypt under the value $x \in [1, T]$ we find the path $\mathbb{P}(x) = \{ID_1, ID_2, ID_3, ID_4\}$. Each node in $\mathbb{P}(x)$ contributes to the final sequence of ciphertexts $C = \{c_1, c_2, c_3, c_4\}$ with one component ciphertext.

To decrypt a sequence of cipertexts C under a range $[s, t]$ the client first finds the smallest set of nodes $\Lambda(s, t)$ covering $[s, t]$ and asks the authority to send him the private keys

of these nodes. If the authority approves the request and sends the keys for decryption, the client goes sequentially through the database and tries to decrypt every component ciphertext with every decryption key he received from the authority. In case the attribute x of a ciphertext C falls in the range $[s,t]$, the set of nodes in $\Lambda(s,t)$ and the set of nodes in $\mathbb{P}(x)$ intersect in exactly one node. Hence, if the client tries all decryption keys on all component ciphertexts that C consists of, he successfully decrypts C. Figure 1 shows an example for the range $[3,7]$.

The MRQED for multiple dimensions works by employing the MRQED for each dimension separately. An example for two dimensions is shown in Figure 2. To encrypt M under the attribute pair $(3,5)$ we first identify in the first tree the path leading from the root to the leaf node with value 3 and then we find in the second tree the path leading to the leaf with value 5. We encrypt M with keys of nodes in path $\mathbb{P}_1(3)$ and get component ciphertexts c_1, c_2, c_3 and c_4. In the second dimension we encrypt with keys of nodes in path $\mathbb{P}_2(5)$ and obtain ciphertexts c_5, c_6, c_7 and c_8. Following the principle of decryption in one dimension, the decryption under two-dimensional range $[2,6] \times [3,7]$ requires revealing keys $\{k_{ID_A}, k_{ID_B}, k_{ID_C}\}$ in the first dimension and keys $\{k_{ID_D}, k_{ID_E}, k_{ID_F}\}$ in the second dimension, as $\{ID_A, ID_B, ID_C\}$ is the smallest set of nodes in the first dimension covering the range $[2,6]$ and $\{ID_D, ID_E, ID_F\}$ is the smallest set of nodes in the second dimension covering the range $[3,7]$.

Testing whether a ciphertext $C = \{c_1, \ldots, c_8\}$ decrypts successfully with the keys $\{k_{ID_A}, \ldots, k_{ID_F}\}$ works as follows: at first, we choose one key from each dimension. Following the example in Figure 2 it means one key from the set $\{k_{ID_A}, k_{ID_B}, k_{ID_C}\}$ and one key from the set $\{k_{ID_D}, k_{ID_E}, k_{ID_F}\}$. Let those be, for instance, keys k_{ID_A} and k_{ID_E}. We pick from the ciphertext C component ciphertexts, which correspond to the chosen keys in each dimension and level. In our case these are c_4 and c_7. We try to decrypt the component ciphertexts with the keys. We repeat this process for all combinations of keys. If the ciphertext is an encryption under a point lying inside the range which the keys correspond to, then one decryption will be successful. In the example in Figure 2 the value $x = (3,5)$ lies inside the queried range $[2,6] \times [3,7]$ and the key pair $\{k_{ID_B}, k_{ID_E}\}$ successfully decrypts component ciphertext pair $\{c_3, c_7\}$. We omit the technical details of the construction, since it relies on pairing-based cryptography and we will not employ this cryptographic primitive in our hierarchical range query mechanism.

We have to be aware of the fact that such a construction leads to the possibility of a collusion attack. We illustrate the attack on two dimensional queries (Figure 2). Assume the authority revealed to a client keys $\{k_{x1}, k_{y1}\}$ corresponding to a region R_1 and keys $\{k_{x2}, k_{y2}\}$ corresponding to region R_4. Then this client has access also to regions R_2 and R_3, because the decryption keys for the region R_2 are $\{k_{x2}, k_{y1}\}$ and the keys for the region R_3 are $\{k_{x1}, k_{y2}\}$, which the client already knows. The solution of the problem is described in [20]. Since our specific application is not vulnerable to collusion attacks, we will not present more details of this solution.

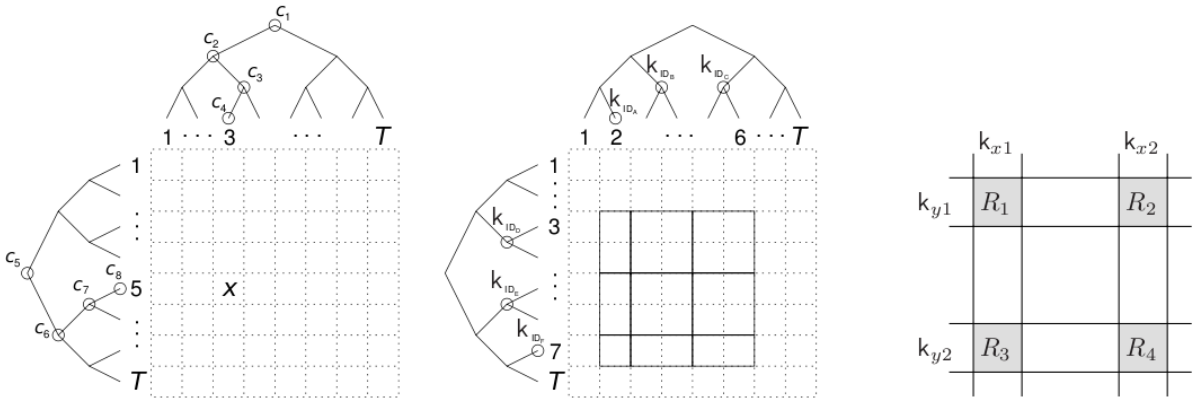

Figure 2: **(Left)** Encryption under the point $(3, 5)$. **(Middle)** Decryption under the range $[2, 6] \times [3, 7]$. **(Right)** Collusion attack in two dimensions. [20]

4.1.2 Protocol Description

For our scheme we adopt from MRQED the principle of attribute hierarchy and build for every dimension a tree with nodes representing a range in the dimension. However, we are able to modify the original scheme due to different assumptions in the research objective. Namely, we identify the following main differences:

- **Queried range**

  Clients in the MRQED model can search for arbitrary ranges if it is allowed by the authority, while each client in our model has the query range limited to a pre-defined level in the tree. In this aspect, we can consider our scheme to be a special case of MRQED.

- **Distribution of search capabilities**

  The authority in the MRQED model releases keys on every client's demand, range by range. The authority in our model distributes keys to all clients at the setup stage and later does not intervene with the actual search in which only a client and the database manager are involved.

- **Collusion attack**

  We give to each client keys for the whole range of every tree level that the client is allowed to query. Thus we do not have to solve the problem of avoiding collusion attacks.

- **Plaintext concealing**

In MRQED finding a match reveals the record content to the database manager. However, in our scenario, we wish to conceal the plaintext record from the database manager when a matching record is found.

Based on these differences mentioned above we change the MRQED scheme in two ways. We modify the tree structure and we use a different cryptographic primitive.

As for the tree corresponding to a dimension, we do not build a binary tree. Instead, we use a tree where the number of children of each node is equal for a particular level, but not necessarily equal for nodes on different levels. As opposed to a binary tree, this limits the range the clients are able to query, but since we do not aim for supporting queries of an arbitrary range this is not a problem in our application. Changing the tree structure in this way also leads to decreasing the total number of nodes in the tree and consequently to lower demands on the database storage.

With regard to the used cryptographic primitive, we use block ciphers instead of pairing-based cryptography as the basic building block of our mechanism. This is because we do not have to deal with the threat of collusion attacks as described in Section 4.1.1. The protocol for Region Inspection consists of several stages, which we describe in the remainder of this section. Full description of the Region Inspection protocol can be found on page 17.

Setup

During the initial setup stage the authority generates the data needed for the system functionality. It defines the structure of the trees, generates the keys required for creating records and performing the search and it distributes the search capabilities to clients.

At first, the authority defines trees T_1, T_2 and T_3, one for each element of the attributes triplet $\langle x, y, t \rangle$. That is, tree T_1 corresponds to the coordinate x, tree T_2 to coordinate y and tree T_3 to the temporal attribute t. By defining the trees we mean setting the number of nodes in each level and assigning each node the range it represents. While the maximum range of the trees defining the location are limited by the area where the tracking system is installed, the time is a continuous quantity. Therefore we assume the full range of the time interval is bounded by some logical unit, e.g. one day.

Each node in each tree is assigned a unique node key. Unlike in MRQED, in our construction it is a block-cipher key. The node keys are used for creating records of the database and they are distributed to clients to grant them access to the database. When we move lower in the tree structure, the number of nodes in a level, and therefore also the number of node keys, increases, and can become significantly large in the bottom levels of a tree. The node keys serve as the means for granting search capabilities to the clients and have to be sent to each client. To minimize the communication costs between the authority and the clients we want to find a better solution than transmitting all the single node keys. The solution to this problem is to generate some secret value from which all the keys on a particular level can be derived. Then the authority needs to transmit only this secret to the client and he can compute all the node keys locally.

The authority generates one random seed s for each tree and level and derives the keys as hashed values of the seed concatenated with a counter value which increases by one for ev-

ery new key. The first key of a specific level is generated as $h(s\|1)$, the second as $h(s\|2)$ and so on. An additional improvement is to generate a random seed for the lowest tree level and compute the seed for the level above as the hash of the seed of the lower level. That is, the seed for the lowest level equals s, the seed for the level above is computed as $h(s)$, the seed of two levels above as $h(h(s))$ etc. Then each client needs to get from the authority only the seed for the maximum precision level he is allowed to query and then he can compute all the node keys for this level and all levels above. Under the assumption that h is pre-image resistant, the client is not able to compute from the secret seed for level ℓ the seed for level $\ell + 1$.

Apart from the node keys, the authority needs to generated one more set of keys. Those keys are used to pre-encrypt the value of ID when the authority creates a new record. The pre-encryption guarantees that the database manager does not learn the actual ID when he finds in the database a record matching to a client's query. One pre-encryption key is needed for each tree level.

The last step in the setup stage is generating for each tree T_i a permutation of the set $\{0, 1, \ldots, d_i\}$ where d_i denotes the depth of the tree T_i. The permutation is used to mask the order of component ciphertext when a new record for the database is computed. Its purpose is to prevent the database manager from learning information about the queried depth, when the database manager performs the search on behalf of a client.

Creating a Record

The authority is the only entity eligible to add records to the database, hence this stage is completely in the hands of the authority. Assume the authority received $\langle ID, x, y, t\rangle$ as the output of the tracking system and now wants to encrypt ID under the attributes $\langle x, y, t\rangle$. We will describe what happens for a single attribute x, because the process for y and t is analogous. This stage consists of several steps.

1. *Path Identification*

   The authority identifies in the tree T_1 the path $\mathbb{P}_1$ leading from the root to the leaf node representing the value of x. With each level ℓ in T_1 is associated a pre-encryption key $K_{1,\ell}$ and with each node in $\mathbb{P}_1$ a node key. As $\mathbb{P}_1$ contains $d_1 + 1$ nodes, there are $d_1 + 1$ node keys.

2. *Randomization and Pre-encryption*

   The authority generates a random number r consisting of n random bits, appends it to ID and encrypts $d_1 + 1$ copies of $(ID\|r)$ with keys $K_{1,0}$, $K_{1,1}$, ..., K_{1,d_1}. The random number prevents record linking. Consider we did not use this randomization and sent to the database manager two records computed from two tuples $\langle ID, x, y, t\rangle$, where ID and x are equal. Then the database manager would not be able to read the exact value of ID, but would recognize that the ID and x are equal in both records. When we use the randomization such linking is impossible.

3. *Encryption under Node Keys*

   The authority appends to the resulting ciphertext m zeros. Appended zeros enable the database manager to distinguish matching records during the search phase. The authority takes $(E_{K_{1,0}}(ID\|r)\|0^m)$ and encrypts it with the key of the first node in $\mathbb{P}_1$ (i.e. with the root key) as $c_{1,0}$. Consequently, the authority encrypts $(E_{K_{1,1}}(ID\|r)\|0^m)$ with the node key of the second node in $\mathbb{P}_1$ as $c_{1,1}$ and continues similarly for all the nodes remaining in $\mathbb{P}_1$.

4. *Permutation of Component Ciphertexts*

   In the previous step the authority created component ciphertexts $\{c_{1,0}, c_{1,1}, \ldots, c_{1,d_1}\}$. In the last step the authority permutes component ciphertexts with π_1. The final record sent to the database manager for storage including the component ciphertexts for attributes y and t looks as follows:

$$\begin{aligned}&\pi_1\{c_{1,0}, c_{1,1}, \ldots, c_{1,d_1}\},\\ &\pi_2\{c_{2,0}, c_{2,1}, \ldots, c_{2,d_2}\},\\ &\pi_3\{c_{3,0}, c_{3,1}, \ldots, c_{3,d_3}\}.\end{aligned}$$

Request for Search

Assume that the client is allowed to query the database on maximum level ℓ_1 in T_1, level ℓ_2 in T_2 and level ℓ_3 in T_3. The client received during the setup stage seeds $s_{1,\ell_1}, s_{2,\ell_2}, s_{3,\ell_3}$ from the authority. He derives the node keys for the required region. Let d'_1, d'_2 and d'_3 denote the depth of the region in trees T_1, T_2 and T_3 respectively. As the next step, the client computes indexes $\pi_1(d'_1), \pi_2(d'_2), \pi_3(d'_3)$ defining on which place in the encrypted record a potentially matching item is placed. Then the client sends the keys and indexes to the database manager.

Search in the Database

The database manager goes sequentially through all records in the database and decrypts $c_{1,\pi_1(d'_1)}$ with the first key from the client. If the result ends with m zeros, he decrypts $c_{2,\pi_2(d'_2)}$ with the second key. In case he obtains a result ending with m zeros, he continues to the third dimension. If this one again ends with m zeros, the database manager deletes in each result the last m zero bits and sends the records to the client.

Result Decryption

The client decrypt the three ciphertexts with keys $K_{1,d'_1}, K_{2,d'_2}, K_{3,d'_3}$ and discards the last n bits of each result. If the results are equal, he considers this the correct answer to the query.

Complexity

The complexity of the Region Inspection protocol is summarized in Table 1. As up to now, d_1, d_2 and d_3 denote the maximum depth of the trees T_1, T_2 and T_3, but for the sake of clarity of the summary we start counting the depths from one (therefore the depth of the root is one), while in the remainder of the text the depths are counted from zero (the depth of the root is zero).

The overall amount of operations during the stages Create Record and Perform Search grows linearly with the total number of records. The number of operations during the Decrypt Results stage grows linearly with the number of results returned to the client.

Protocol Stage	**Number and Type of Operations**
Setup	$d_1 + d_2 + d_3 - 3$ hash operations
Create Record (per record)	$2 \times (d_1 + d_2 + d_3)$ block cipher encryptions $d_1 + d_2 + d_3$ hash operations
Send Query	3 hash operations
Perform Search (per record)	1–3 block cipher decryptions
Decrypt Results (per result)	3 block cipher decryptions

Table 1: Complexity overview of the Region Inspection protocol.

Region Inspection Protocol – Full Description

Protocol Setup :: Authority

1. Build the structure of trees T_1, T_2, T_3 and set d_1, d_2 and d_3 as the depth of the tree T_1, T_2 and T_3, respectively.
2. Generate random keys $K_{1,0}, \ldots, K_{1,d_1}, K_{2,0}, \ldots, K_{2,d_2}, K_{3,0}, \ldots, K_{3,d_3}$.
3. Select a hash function h, generate random seeds s_{1,d_1}, s_{2,d_2}, s_{3,d_3} and compute $s_{i,j-1} := h(s_{i,j})$ for $i = 1, 2, 3$ and $j = 1, \ldots, d_i$.
4. Generate random permutations π_1, π_2, π_3, where π_i is a permutation of the set $\{1, 2, \ldots, d_i\}$.
5. Send each client $\mathcal{C}\ell$ a description of the trees T_1, T_2, T_3 and the maximum precision levels ℓ_1, ℓ_2, ℓ_3 of queries allowed for the client $\mathcal{C}\ell$ in dimensions x, y and t.
6. Send π_1, π_2, π_3, $s_{1,\ell_1}, s_{2,\ell_2}, s_{3,\ell_3}$ and $K_{j,0}, \ldots, K_{j,\ell_j}$ for $j = 1, 2, 3$ to every client $\mathcal{C}\ell$ in a secure manner.

Create Record$(\langle ID, x, y, t\rangle)$ :: Authority

1. Generate a random number r consisting of n bits.
2. Compute $c' := E_{K_{i,j}}(ID \| r) \| 0^m$ for $i = 1, 2, 3$, $j = 0, 1, \ldots, d_i$.
3. For $i = 1, 2, 3$ identify the path $\mathbb{P}_i$ leading in the tree T_i from the root node to the node representing the value x, y and t, respectively. For $i = 1, 2, 3$, $d = 0, 1, \ldots, d_i$, set $ctr_{i,d}$ as the position of the node at the depth d in path $\mathbb{P}_i$.
4. For $i = 1, 2, 3$, $d = 0, 1, \ldots, d_i$, compute $k_{i,d} := h(s_{i,d} \| ctr_{i,d})$.
5. Compute component ciphertexts $c_{1,0} := E_{k_{1,0}}(c'), \ldots, c_{1,d_1} := E_{k_{1,d_1}}(c')$,
   $c_{2,0} := E_{k_{2,0}}(c'), \ldots, c_{2,d_2} := E_{k_{2,d_2}}(c')$,
   $c_{3,0} := E_{k_{3,0}}(c'), \ldots, c_{3,d_3} := E_{k_{3,d_3}}(c')$.
6. Permute component ciphertexts $\pi_1(c_{1,0}, c_{1,1}, \ldots, c_{1,d_1})$, $\pi_2(c_{2,0}, c_{2,1}, \ldots, c_{2,d_2})$, $\pi_3(c_{3,0}, c_{3,1}, \ldots, c_{3,d_3})$.
7. Create final ciphertext C as a sequence of permuted component ciphertexts and send C to the database manager for storage.

Send Query$(d'_1, d'_2, d'_3, ctr_{1,d'_1}, ctr_{2,d'_2}, ctr_{3,d'_3})$ :: Client $\mathcal{C}\ell$

1. For $j = 1, 2, 3$, set $s_j := s_{j,\ell_j}$ and repeat $\ell_j - d_j$ times $s_j := h(s_j)$.

2. Compute $k_1 := h(s_1 \| ctr_{1,d'_1})$, $k_2 := h(s_2 \| ctr_{2,d'_2})$ and $k_3 := h(s_3 \| ctr_{3,d'_3})$.

3. Send k_1, k_2, k_3 and $\pi_1(d'_1), \pi_2(d'_2), \pi_3(d'_3)$ to the database manager.

Perform Search$(k_1, k_2, k_3, d'_1, d'_2, d'_3)$:: Database manager

1. Initialize $\mathcal{C} = \emptyset$.

2. For each database record C:
   if $D_{k_1}(c_{1,d'_1}) = c'_1 \| 0^m$ for some c'_1 then
   if $D_{k_2}(c_{2,d'_2}) = c'_2 \| 0^m$ for some c'_2 then
   if $D_{k_3}(c_{3,d'_3}) = c'_3 \| 0^m$ for some c'_3 then $\mathcal{C} = \mathcal{C} \cup \{c'_1, c'_2, c'_3\}$.

3. Send $\mathcal{C}$ to the client $\mathcal{C}\ell_i$.

Decrypt Result(c'_1, c'_2, c'_3) :: Client $\mathcal{C}\ell$

1. Compute $p_1 = D_{K_{1,d'_1}}(c'_1)$, $p_2 = D_{K_{2,d'_2}}(c'_2)$, $p_3 = D_{K_{3,d'_3}}(c'_3)$.

2. If $p_1 = p_2 = p_3$ then remove the last n bits of p_1 and consider this the answer to the query.

4.1.3 Performance Analysis

The Region Inspection protocol was implemented to measure the time needed for executing the protocol steps. The code was written in C++ and the Crypto++ library [10] was used to provide the implementation of block ciphers, hash functions and a pseudorandom generator.

Regarding the cryptographic primitives and its parameters, AES with 128-bit key [18] was chosen for both the plaintext pre-encryption and the encryption of the pre-encrypted text using a node key. Further, SHA-256 from the SHA-2 family of cryptographic hash functions [19] was used for deriving the individual node keys. For randomization, instead of appending random bits to the plaintext input, the most commonly used mode of operation for a block cipher, the CBC-mode [12], with a random initialization vector was used for the AES encryption.

The following system parameters were chosen:

- **Tree structures**

  As for the maximum ranges, the area of Prague International Airport was chosen for the purpose of practical illustration. The maximum ranges for x and y were chosen as 750 and 200 metres, respectively, as the airport's approximate area. The full range for the tree storing temporal information was chosen to be one day, as a natural time unit. Each tree has depth four, where depth is counted from zero. The details on the number of nodes and the range they represent are given in Table 2.

Tree T_1					
Depth	0	1	2	3	4
Number of Nodes	1	2	4	8	40
Node Range	200 m	100 m	50 m	25 m	5 m
Tree T_2					
Depth	0	1	2	3	4
Number of Nodes	1	3	15	30	150
Node Range	750 m	250 m	50 m	25 m	5 m
Tree T_3					
Depth	0	1	2	3	4
Number of Nodes	1	24	96	1440	17280
Node Range	24 h	1 h	15 min	1 min	5 sec

Table 2: The structures of the trees T_1, T_2 and T_3.

- **Number of records**

  For demonstration purposes, the total number of records was chosen to be 100.000. Altogether 100 random identifiers were generated and for each identifier 1000 random triplets $\langle x, y, t \rangle$ were generated.

- **Identifier**

  Following the example of the BSN number, a nine-digit number was used as an identifier.

Time Consumption

The time to perform the individual steps of the protocol was measured. That is, we measured the duration of the setup stage, the time to populate the database and finally to perform the search. The resulting times are shown in Table 3.

Protocol Stage	Measured Time
Setup	60.0 μs
Database population	186.6 s
Search (per query)	10.6 s

Table 3: Time consumption.

The Prague Airport handles approximately 30.000 passengers in one day. Assume that the average time spent on the airport is one hour. The number of spatiotemporal records collected for one passenger during one hour is 720. For 30.000 passengers that means 21.600.000 database records. As the time grows linearly with the number of database records, searching in a database of such size would take around 35 minutes.

The time needed for database population is in principle the time needed for encrypting the plaintext records. Since the data collection and encryption happens in batches throughout the whole day and not at one time for the entire dataset, the linear increase in the time required for the database population would not pose a problem.

The search time measured during the performance tests is acceptable in case the results are not required immediately or the size of the database is rather small. If the results are needed immediately, e.g. when we need to locate a passenger who did not arrive for boarding, the time for searching becomes impractical.

The time consumption can be improved by further optimization. Checking successful decryption of a record is independent on checking successful decryption of other records, therefore it is possible to parallelize the computations. The workload can be divided over several servers responsible for searching in the corresponding part of the database.

Space Consumption

We calculated the storage requirements for the database and the auxiliary data such as the key sets. Encrypting a nine-digit number as the initial input leads to 32 bytes long component ciphertext, i.e. two AES blocks. The reasoning is following:

- The plaintext record of 9 digits corresponds to 9 bytes. These 9 bytes are padded to the AES block size, that is 16 bytes, and encrypted.

- Several zero bytes for denoting successful search are appended to the 16-byte text. We used 8 bytes, thus now we have a text of 24 bytes.
- Eventually, the pre-encrypted 24-byte text is encrypted with AES. Before encryption the ciphertext is padded in order to have length of a multiple of the block size. The closest multiple of a block size is $2 \times 16 = 32$ bytes. Therefore, the length of one component ciphertext is 32 bytes.

There are three dimensions and for every dimension five component ciphertexts have to be calculated. Hence, one complete database record consists of 15 component ciphertexts and the size of one complete database record is $15 \times 32 = 480$ bytes.

However, there are some additional storage requirements on the side of the database manager to provide full functionality of the system.

- Initialization vectors (IV) for the pre-encryption have to be stored. When a match is found the corresponding IV is returned to the querying client who needs the IV to recover the plaintext ID from the result. The database manager does not use these data for any computation. One IV of 16 bytes is needed per record.
- Furthermore, initialization vectors for the encryption are stored. When the database manager looks for a match he uses these IVs for trial decryption of the records. Again, one IV of 16 bytes per record is needed.

The space needed for both IVs is hence 32 bytes per record.

Storage requirements for the authority are negligible. The authority has to store the pre-encryption keys. There is one pre-encryption key of 16 bytes for every tree and every depth, together 15×16 bytes $= 240$ bytes. Besides the pre-encryption keys, the authority stores also seeds used as the input for the cryptographic hash function. We use a 32 bytes long input, therefore the authority needs 15×32 bytes $= 480$ bytes to store the seeds. The overview of storage requirements is given in Table 4.

Constituent Data	**Storage Requirements**
Per database	
Pre-encryption keys	240 bytes
Seeds	480 bytes
Per record	
Initialization vectors	32 bytes
Component ciphertext	32 bytes
Complete database record (excluding IVs)	480 bytes
Complete database record (including IVs)	512 bytes

Table 4: Space consumption.

For a database with 21.600.000 records the total storage needed at the side of the database manager is approximately 10.3 GB, which is a satisfactory result.

4.2 Object Location

In the Object Location query type the client is interested in learning the trajectory of a person with the identifier ID. To support such queries, every item in the database consists of two parts – an encrypted personal identifier ID and ID's encrypted trajectory. When a client wants to search the database for ID's trajectory he sends to the database manager the encrypted identifier. If a matching item is found in the database, ID's encrypted trajectory is sent back to the client as the result.

Similarly to Region Inspection, we assume there are several precision levels of the spatiotemporal data and every client is assigned a maximum precision level on which he is able to obtain records from the database. The aim of the encryption scheme is to guarantee that the client does not learn the trajectory with higher precision than he is allowed.

In order to efficiently represent the points of the trajectory on multiple resolution levels, the spatiotemporal points of the trajectory are represented as ranges. Before sending a new encrypted record for storage in the database, the authority converts the exact spatiotemporal point $\langle x, y, t\rangle$ to several records of the form $\langle [x_\ell, x_u], [y_\ell, y_u], [t_\ell, t_u]\rangle$. There exists one record for every precision level.

A result returned to the client is then the encryption of the following data:

$$\langle [x_{1,\ell}, x_{1,u}], [y_{1,\ell}, y_{1,u}], [t_{1,\ell}, t_{1,u}]\rangle, \ldots, \langle [x_{n,\ell}, x_{n,u}], [y_{n,\ell}, y_{n,u}], [t_{n,\ell}, t_{n,u}]\rangle,$$

where n is the number of spatiotemporal points in the trajectory and the ranges correspond to the maximum precision level assigned to the querying client.

4.2.1 Symmetric Searchable Scheme

The state-of-the-art symmetric searchable encryption schemes employ several recurrent techniques. Plaintext records are masked with random bits and the ciphertexts are then stored at a random location of the storage space.

The seminal work of Curtmola et al. [11] provides an illustration of both principles. In their model a collection of encrypted files $f_1, f_2, \ldots, f_n$ is to be stored in an untrusted repository and the goal is to efficiently search for files containing a keyword from a keyword set $W = \{w_1, w_2, \ldots, w_\ell\}$. Two data structures, an array A and look-up table T, are created to achieve the desired functionality.

A linked list L_w is constructed for every keyword w. Every node of L_w is encrypted with a symmetric key K_i and the nodes of L_w are stored at randomly chosen locations of the array A. The i-th node N_i has the form

$$\langle f_w, \mathrm{addr}(N_{i+1}), K_{i+1}\rangle,$$

where f_w is a unique identifier of a file containing the keyword w, $\mathrm{addr}(N_{i+1})$ determines the position of the next node of L_w inside the array A and K_{i+1} is the symmetric key for decrypting the following node.

The position of the head of L_w and the key to recover its content is kept in a look-up table T. One item in the look-up table corresponds to one keyword and the items in T are encrypted symmetrically.

The client eligible for search has knowledge of the look-up table T and knows the keys to decrypt the items in the table. Then the client just has to identify the location of the item corresponding to the desired keyword w and send to the repository the key to decrypt the item in T. Consequently, the repository manager decrypts the item in T, gets access to the first node of L_w stored in A and therefore also to all following nodes of L_w. When he finds the identifiers he sends to the client the encrypted documents containing w. The keyword w stays concealed from the repository manager.

4.2.2 Protocol Description

We adopt the representation of ranges from the scheme for Region Inspection. Trees T_1, T_2 and T_3 are built over dimensions x, y and t. Every level in a tree corresponds to one precision level of the attribute. That is, the root node represents the whole range and the leaves stand for the ranges of the finest resolution. To represent the value of x from the spatiotemporal point $\langle x, y, t\rangle$ on different precision levels the authority finds in T_1 the node corresponding to the range containing x. The authority encrypts the counter holding the position of the node within the level and repeats the actions for y and t.

The major challenge is to add new database records dynamically and at the same time protect the data to the highest possible extent. We propose two protocols for Object Location and compare their advantages and disadvantages.

Object Location Protocol 1

As stated at the beginning of Section 4.2, the database can be logically divided into two parts, the searchable index and encrypted trajectories. The index consists of encrypted identifiers which are stored in the form of

$$(ID \oplus F_K(ID)),$$

where F is a pseudorandom function and K is a secret symmetric key generated by the authority. The authority sends the key to the eligible clients during the setup stage of the protocol.

For simplicity we assume that the trees T_1, T_2 and T_3 have the same depth d. To add an encrypted spatiotemporal point to ID's trajectory a tuple of $3\times(d+1)$ items is appended to ID's encrypted identifier. The tuple looks as follows:

$\langle r_{1,0}, ctr_{1,0} \oplus G_{K_{1,0}}(r_{1,0})\rangle$	$\langle r_{2,0}, ctr_{2,0} \oplus G_{K_{2,0}}(r_{2,0})\rangle$	$\langle r_{3,0}, ctr_{3,0} \oplus G_{K_{3,0}}(r_{3,0})\rangle$
$\langle r_{1,1}, ctr_{1,1} \oplus G_{K_{1,1}}(r_{1,1})\rangle$	$\langle r_{2,1}, ctr_{2,1} \oplus G_{K_{2,1}}(r_{2,1})\rangle$	$\langle r_{3,1}, ctr_{3,1} \oplus G_{K_{3,1}}(r_{3,1})\rangle$
$\vdots$	$\vdots$	$\vdots$
$\langle r_{1,d}, ctr_{1,d} \oplus G_{K_{1,d}}(r_{1,d})\rangle$	$\langle r_{2,d}, ctr_{2,d} \oplus G_{K_{2,d}}(r_{2,d})\rangle$	$\langle r_{3,d}, ctr_{3,d} \oplus G_{K_{3,d}}(r_{3,d})\rangle$

G denotes a pseudorandom function, $r_{i,j}$ is a random value chosen anew for every component item and $ctr_{1,j}, ctr_{2,j}$ and $ctr_{3,j}$ represent x, y and t on the precision level j.

The authority permutes the positions of the items within the record before sending them to the database manager for storage. The keys for the function G are distributed by the authority during the setup stage. Every client obtains the keys $K_{1,\ell_1}, K_{2,\ell_2}$ and K_{3,ℓ_3}, where ℓ_1, ℓ_2 and ℓ_3 are the maximum permitted precisions for dimensions x, y and t. Every client is also given the position of the items with precision ℓ_1, ℓ_2 and ℓ_3 in the permuted record.

Now assume the client wants to look for the records of some passenger ID. Since he knows the key K he can calculate $(ID \oplus F_K(ID))$ and send it to the database manager. The client sends also the position of the data of requested precision within the tuple, as the database manager does not return the whole tuple of $3\times(d+1)$ items, but sends back only the items of the requested precision. The database manager goes through the index and either says he did not find any match for this person or he sends back all the items

$$\langle r_{1,\ell_1}, ctr_{1,\ell_1} \oplus G_{K_{1,\ell_1}}(r_{1,\ell_1})\rangle, \langle r_{2,\ell_2}, ctr_{2,\ell_2} \oplus G_{K_{2,\ell_2}}(r_{2,\ell_2})\rangle, \langle r_{3,\ell_3}, ctr_{3,\ell_3} \oplus G_{K_{3,\ell_3}}(r_{3,\ell_3})\rangle.$$

The client uses $r_{1,\ell_1}, r_{2,\ell_2}$ and r_{3,ℓ_3} with the corresponding keys as the input for G and recovers the plaintexts $ctr_{1,\ell_1}, ctr_{2,\ell_2}$ and ctr_{3,ℓ_3}.

This scheme allows the authority to add new records to an existing database dynamically. The database manager does not learn anything about the queried precision levels and thanks to randomization can not distinguish when two records store the same information. An additional advantage of this scheme is that the database manager can order the encrypted identifiers in the index and thus make the search faster.

Unfortunately, this scheme suffers from one major drawback. When a client queries for some specific precision level in the database, information about the higher precision of the returned location data may leak. The source of the problem is that multiple copies of equal plaintext records are returned back to the client. Different outputs $\langle ID, x_1, y_1, t_1\rangle$ and $\langle ID, x_2, y_2, t_2\rangle$ can lead to the same representation

$$\langle ctr_{x_1,\ell_x}, ctr_{y_1,\ell_y}, ctr_{t_1,\ell_t}\rangle = \langle ctr_{x_2,\ell_x}, ctr_{y_2,\ell_y}, ctr_{t_2,\ell_t}\rangle,$$

especially in the higher levels of the trees.

We will give an example of the data leakage for one dimension, but the same principle can be extended to more dimensions. Consider a tree where level 4 is the most precise level, every leaf represents a 5-meter range and all records on level 4 are unique, since that is the granularity of the tracking system output. Assume a client is eligible for queries on level 3, where every node represents a 10-meter range. The client asks the database manager for the trajectory of ID. After decrypting the results he recovers two records

$$\langle ID, [10, 20]\rangle,\ \langle ID, [10, 20]\rangle.$$

Then the client can deduce that ID must have been located in both underlying higher-precision ranges. That is,

$$\langle ID, [10, 15]\rangle,\ \langle ID, [15, 20]\rangle,$$

because only these records on level 4 can lead to the doubled record on level 3. Therefore the client learned ID's location with higher precision than he was allowed to.

Object Location Protocol 1 – Full Description

Protocol Setup$(p \in \mathbb{N}, k \in \mathbb{N}, \ell \in \mathbb{N})$ :: Authority

1. Build the structure of trees T_1, T_2 and T_3 of equal depth d.
2. Choose a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^p$ and generate a key $K \in \{0,1\}^k$.
3. Choose a pseudorandom function $G : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ and generate keys $K_{1,0}, \ldots, K_{1,d}, K_{2,0}, \ldots, K_{2,d}, K_{3,0}, \ldots, K_{3,d} \in \{0,1\}^k$.
4. Generate permutations π_1, π_2, π_3 of the set $\{0, \ldots, d\}$.
5. Send to every client $\mathcal{C}\ell$:
   1. ℓ_1, ℓ_2 and ℓ_3 denoting the maximum precision levels of results in dimensions x, y and t allowed for the client.
   2. $K_{1,\ell_1}, K_{2,\ell_2}, K_{3,\ell_3}$.
   3. $\pi_1(\ell_1), \pi_2(\ell_2), \pi_3(\ell_3)$.

Create Record$(\langle ID, x, y, t\rangle)$ :: Authority

1. Compute $\widetilde{ID} = (ID \oplus F_K(ID))$.
2. Identify the nodes $ctr_{1,0}, \ldots, ctr_{1,d}$ in T_1, $ctr_{2,0}, \ldots, ctr_{2,d}$ in T_2 and $ctr_{3,0}, \ldots, ctr_{3,d}$ in T_3 representing the values x, y and t on every precision level.
3. Generate $r_{1,0}, \ldots, r_{1,d}, r_{2,0}, \ldots, r_{2,d}, r_{3,0}, \ldots, r_{3,d} \in \{0,1\}^\ell$.
4. For $i = 1, 2, 3, j = 0, \ldots, d$, compute $C_{i,j} = ctr_{i,j} \oplus G_{K_{i,j}}(r_{i,j})$.
5. For $j = 0, \ldots, d$ compute $\pi_1(C_{1,j}), \pi_2(C_{2,j}), \pi_3(C_{3,j})$ and send

   $$\langle \widetilde{ID}, \pi_1(C_{1,0}), \ldots, \pi_1(C_{1,d}), \pi_2(C_{2,0}), \ldots, \pi_2(C_{2,d}), \pi_3(C_{3,0}), \ldots, \pi_3(C_{3,d})\rangle$$

   to the database manager for storage.

Send Query(ID) :: Client $\mathcal{C}\ell$

1. Compute $\widetilde{ID} = (ID \oplus F_K(ID))$.
2. Send $\widetilde{ID}$ and $P_1 = \pi_1(\ell_1), P_1 = \pi_2(\ell_2), P_3 = \pi_3(\ell_3)$ to the database manager.

Perform Search$(\langle \widetilde{ID}, P_1, P_2, P_3 \rangle)$:: Database manager

1. Find in the index item $\widetilde{ID}$ and return items on positions P_1, P_2 and P_3 in corresponding records.

Decrypt Result$(\langle C_1, C_2, C_3 \rangle)$ :: Client $\mathcal{C}\ell$

1. For all returned results
    1. Parse C_1, C_2, C_3 to $\langle r_1, c_1 \rangle, \langle r_2, c_2 \rangle, \langle r_3, c_3 \rangle$.
    2. Compute and return $c_1 \oplus G_{K_{1,\ell_1}}(r_1), c_2 \oplus G_{K_{2,\ell_2}}(r_2), c_3 \oplus G_{K_{3,\ell_3}}(r_3)$.

Object Location Protocol 2

The Object Location Protocol 2 is a variation of the Object Location Protocol 1. Several changes are introduced to solve the problem of information leakage described above. We use again the trees T_1, T_2 and T_3 and their structure to represent the ranges of x, y and t and we assume the trees have equal depth d.

The index is constructed exactly as in the first scheme and consists of encrypted identifiers

$$(ID \oplus F_{\tilde{K}}(ID)).$$

Adding new points to ID's trajectory is done by appending tuples to the masked ID. In this protocol one tuple has $(d+1)^3$ component items, one item for every combination of precisions $\langle \ell_1, \ell_2, \ell_3 \rangle$. Since there are $(d+1)$ precision levels and three dimensions, the total number of possible combinations is $(d+1)^3$.

The cause of the information leakage in the previous protocol resides in retrieving multiple copies of the same plaintext. To fix the problem we make the trajectory encryption deterministic for every ID. Then, repeated combination of ID and a range of some precision can be distinguished and prevented from storing. Nevertheless, the encryption of the same range for two different IDs should vary to prevent the database manager from linking IDs that were seen at the same location.

Let $\langle ctr_{1,\ell_1}, ctr_{2,\ell_2}, ctr_{3,\ell_3} \rangle$ be the representation of $\langle x, y, t \rangle$ on precision combination $\ell = \langle \ell_1, \ell_2, \ell_3 \rangle$ and let (K, E, D) be a symmetric encryption scheme. A spatiotemporal point $\langle x, y, t \rangle$ of precision ℓ is stored as

$$E_{K_\ell}(ctr_{1,\ell_1}, ctr_{2,\ell_2}, ctr_{3,\ell_3}).$$

If we couple the key with the ID, then the encryption of $\langle ctr_{1,\ell_1}, ctr_{2,\ell_2}, ctr_{3,\ell_3} \rangle$ will result in equal ciphertexts for the same ID, but will be different for two different IDs. The authority assigns to every precision combination ℓ a key K'_ℓ. Hence, there are $(d+1)^3$ encryption keys K'_ℓ. The keys K_ℓ are computed as

$$K_\ell = G_{K'_\ell}(ID),$$

where G is a pseudorandom function.

We actively engage the database manager in storing the encrypted trajectory points. Every new record sent for storage consists of the masked identifier $\widetilde{ID} = (ID \oplus F_{\tilde{K}}(ID))$ and the encrypted location of $N = (d+1)^3$ precisions:

$E_{K_1}(ctr_{1,\ell_{1,1}}, ctr_{2,\ell_{2,1}}, ctr_{3,\ell_{3,1}})$
$E_{K_2}(ctr_{1,\ell_{1,2}}, ctr_{2,\ell_{2,2}}, ctr_{3,\ell_{3,2}})$
$\vdots$
$E_{K_N}(ctr_{1,\ell_{1,N}}, ctr_{2,\ell_{2,N}}, ctr_{3,\ell_{3,N}})$

When the database manager receives a new record for storage he finds $\widetilde{ID}$ in the index. If $\widetilde{ID}$ is not yet in the database he adds the complete record as sent by the authority. In case an entry $\widetilde{ID}$ already exists in the index he checks for every precision "bin" whether $E_K(ctr_{1,\ell_1}, ctr_{2,\ell_2}, ctr_{3,\ell_3})$ is stored in the database. If the item is in the database the database manager does not store another copy in this precision bin. This way we avoid multiple copies of the same plaintext.

Every client receives during the setup stage from the authority keys K'_ℓ for the precision combinations he is allowed to learn and information about the position of the precision bin ℓ within the record. To query the database the client first computes $K_\ell = G_{K'_\ell}(ID)$, then calculates $\widetilde{ID} = (ID \oplus F_{\tilde{K}}(ID))$ and sends $\widetilde{ID}$ and the position of the precision bin to the database manager, who returns all matching records.

The Object Location Protocol 2 does not leak information about the higher precision data to the client and at the same time retains most of the advantages of the Object Location Protocol 1. The authority can add records dynamically and the database manager can order the records and make the search more efficient. However, the database manager can deduce information about the resolution levels based on the length of the requested precision bin.

Object Location Protocol 2 – Full Description

Protocol Setup$(k \in \mathbb{N}, p \in \mathbb{N}, m \in \mathbb{N})$:: Authority

1. Build the structure of trees T_1, T_2 and T_3 of equal depth d.
2. Choose a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^p$ and generate a key $\tilde{K} \in \{0,1\}^k$.
3. Choose a pseudorandom function $G : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^k$ and generate keys $K'_1, \ldots, K'_{(d+1)^3} \in \{0,1\}^k$.
4. Send to every client $\mathcal{C}\ell$:
    1. $\ell = \langle \ell_1, \ell_2, \ell_3 \rangle$ denoting the maximum precision level of results in dimensions x, y and t allowed for the client.
    2. P_ℓ denoting the position of the item with precision ℓ within the record.
    3. keys $\tilde{K}, K'_\ell$.

Create Record$(\langle ID, x, y, t \rangle)$ :: Authority

1. Compute $\widetilde{ID} = (ID \oplus F_{\tilde{K}}(ID))$.
2. Identify the nodes $ctr_{1,0}, \ldots, ctr_{1,d}$ in T_1, $ctr_{2,0}, \ldots, ctr_{2,d}$ in T_2 and $ctr_{3,0}, \ldots, ctr_{3,d}$ in T_3 representing the values x, y and t on every precision level.
3. For every precision combination $\ell = \langle \ell_1, \ell_2, \ell_3 \rangle$ compute
    1. $K_\ell = G_{K'_\ell}(ID)$.
    2. $C_\ell = E_{K_\ell}(ctr_{1,\ell_1}, ctr_{2,\ell_2}, ctr_{3,\ell_3})$.
4. Send $\langle \widetilde{ID}, C_1, \ldots, C_{(d+1)^3} \rangle$ to the database manager for storage.

Store Record$(\langle \widetilde{ID}, C_1, \ldots, C_{(d+1)^3} \rangle)$ :: Database manager

1. Find index entry $\widetilde{ID}$ and for $i = 1, \ldots, (d+1)^3$ store item C_i on the i-th position if an equal item is not stored at the i-th position yet.

Send Query(ID) :: Client $\mathcal{C}\ell$

1. Compute $\widetilde{ID} = (ID \oplus F_{\tilde{K}}(ID))$.
2. Send $\widetilde{ID}$ and P_ℓ to the database manager.

Perform Search$(\langle \widetilde{ID}, P_\ell \rangle)$:: Database manager

1. Find index entry $\widetilde{ID}$ and return all items $\langle C_1, \ldots, C_n \rangle$ on position P_ℓ of the corresponding records.

Decrypt Result$(\langle C_1, \ldots, C_n \rangle)$ :: Client $\mathcal{C}\ell$

1. Compute $K_\ell = G_{K'_\ell}(ID)$.
2. For $i = 1, \ldots, n$ compute and return $D_{K_\ell}(C_i)$.

4.2.3 Performance Analysis

The Object Location Protocol 1 (OLP1) and Object Location Protocol 2 (OLP2) were implemented in C++ and the Crypto++ library was used. We used HMAC based on SHA-256 with 128-bit key as a pseudorandom function. As a block cipher in OLP2, we employed AES with 128-bit key in ECB mode [23] to guarantee deterministic behaviour.

The number of records, the type of identifiers and tree structures for representing ranges on different precision levels were adopted from the Region Inspection implementation. Specifically, the maximum ranges for x, y and t were chosen to be 200 m, 750 m and 24 hours, respectively. A detailed description of the tree structures can be found in Table 2 on page 19. The number of unique identifiers was 100 and 1000 random triplets $\langle x, y, t \rangle$ were generated for every identifier. Hence, the total number of database records was 100.000.

Time Consumption

Table 5 gives an overview of the time needed to perform the steps of both Object Location Protocols.

Protocol Stage	**OLP1**	**OLP2**
Setup	90.7 μs	155.7 μs
Database population	4.22 s	57.8 s
Search (per query)	37.3 μs	42.8 μs

Table 5: Time consumption.

The comparison of the computational requirements of OLP1 and OLP2 correspond to the measured times. OLP2 is expected to be computationally heavier – in OLP1 the number of keys to generate in the setup stage is $(3 \times (d+1))$, while in OLP2 it is $((d+1)^3+1)$; regarding the database population, in OLP1 $(3 \times (d+1))$ component items need to be calculated for every database record, in OLP2 it is $(d+1)^3$ items. Although not all the component items are eventually stored in OLP2, all items have to be computed. The search time for OLP1 and OLP2 is comparable, as in both protocols the index legth was 100 and performing the search only means going through the index and looking for a matching item.

In comparison with the Region Inspection Protocol, the time consumption of OLP1 and OLP2 is much lower. This is because more cryptographic operations are needed for computing a record in the Region Inspection Protocol than in both Object Location Protocols. Also the search in Object Location Protocols is several orders of magnitude faster, as the database manager only looks for an equal item in the index and does not perform any cryptographic computations.

In Section 4.1.3 was introduced a practical scenario with 30.000 passengers, each staying at the airport for one hour, which leads to a database of 21.600.000 records. Assuming the search time grows linearly with the size of the index, the search time of both Object

Location Protocols is in order of miliseconds. Therefore the search time is practical even for large datasets and no further optimization is needed. The database population is done in batches, hence the increased time for large datasets would not pose a problem in practical situations.

Space Consumption

The overview of space consumption of OLP1 and OLP2 is given in Table 6. We first clarify the space consumption for OLP1 and then explain the space requirements for OLP2.

The requirements for the authority in OLP1 are low. The authority stores together $(3 \times (d+1)+1) = 16$ keys for pseudorandom functions (PRF): one key for the PRF generating bits to mask the identifier and $(3 \times (d+1)) = 15$ keys for the PRF generating bits to mask the counters. The keys have length of 16 bytes, therefore 16×16 bytes $= 256$ bytes are needed to store the keys.

On the side of the database manager the requirements are as follows. One index item has 9 characters, which corresponds to 9 bytes. Each record consists of $(3 \times (d+1)) = 15$ pairs of the form $\langle r, ctr \oplus G_K(r) \rangle$. Length of r was chosen to be 10 bytes, and the output of G was shortened to 10 bytes. Thus, one item in the record is 20 bytes long and the complete record has 300 bytes.

In OLP2 the authority stores one key for the PRF returning bits for masking the identifier. Besides, the authority stores $(d+1)^3 = 125$ keys for the PRF generating the symmetric keys depending on the identifier. Together 126 keys of 16 bytes are needed, together it is 2016 bytes. The index is the very same as for OLP1, one item has 9 bytes. One database record consists of $(d+1)^3 = 125$ items with encrypted counters. The concatenated counters fit into one AES block of 16 bytes, therefore the encrypted item has also 16 bytes. One database record has 125 items of 16 bytes, i.e. one database record is 2000 bytes long. Recall though, that only the unique items of the records are eventually stored.

Constituent Data	**OLP1**	**OLP2**
Per database		
PRF keys	256 bytes	2016 bytes
Per index entry		
Index entry	9 bytes	9 bytes
Database record - component item	20 bytes	16 bytes
Database record - total size	300 bytes	2000 bytes

Table 6: Space consumption.

For the the practical example with 30.000 passengers and 21.600.000 records, the total space needed is 6.0 GB for OLP1. For OLP2 it is 40.2 GB to transmit, but the actual storage requirements would be lower, as the portion of the stored data depends on the number of unique records. Also, further lowering of the space consumption is possible by choosing only some precision combination from all $(d+1)^3$ possible precision combinations.

5 Discussion and Future Work

We developed a mechanism for supporting hierarchical range queries in encrypted databases with location data. In Section 3 we defined four security requirements on the system, namely database confidentiality, query unforgeability, query privacy and hierarchy compliance.

- Database confidentiality is satisfied, since the database records are stored encrypted and no entity can recover the original plaintext records without the knowledge of the keys used for encrypting the records.
- Query unforgeability is guaranteed through the key management. Clients are provided with the keys corresponding to the data precision permitted for their queries or results, respectively.
- Query privacy is preserved, because queries are either disguised with random bits or the queries are sent in the form of relevant keys.
- Hierarchy compliance holds with the exception of the Object Location Protocol 1 where the client may learn additional information about higher precision version of the returned data. However, the problem is fixed in Object Location Protocol 2.

As for the system's practical applicability, the developed query mechanism is functional and can be deployed in practical situations. From every participant's view the communication requirements and workload division are optimal. The authority is responsible only for the record creation and key distribution, i.e. the actions which necessarily have to stay in the hands of the authority. Outsourcing any of the mentioned tasks to the database manager would pose a risk of revealing the database records to external parties. Regarding the clients, receiving the search capabilities is a one-time action happening during the setup stage and there is no further need to contact the authority to get a permission for a query. Moreover, the clients eligible to retrieve high-precision location data obtain the result of desired precision in a single communication round with the database manager.

We identify two directions for future improvement and extension of the described protocols:

- **One database for both query types**

  In order to build an encrypted database supporting two query types we constructed a database consisting of two parts, where each part supports one type of query. While this structural division provides the desired functionality, it naturally leads to a question if it is possible to build a database supporting both query types on the same background data structure. The construction of such data structure remains an open problem.

- **Support of arbitrary ranges**

  In both the Region Inspection and Object Location Protocols the clients can only work with queries and results of pre-defined ranges. Support of queries of arbitrary ranges is an open challenge. Two issues have to be considered. The first is the possibility of deriving information about higher precision from two queries of lower precision – when the client learns about two overlapping low-precision regions he may be able to deduce information about the high-precision intersection of the regions. The second is the efficiency, since straigtforward extension of the described protocols would lead to impractical storage requirements.

6 References

[1] Bellare, Mihir, Alexandra Boldyreva, and Adam O'Neill. "Deterministic and efficiently searchable encryption." Advances in Cryptology-CRYPTO 2007. Springer Berlin Heidelberg, 2007. 535-552.

[2] Bellare, Mihir. "New proofs for NMAC and HMAC: Security without collision-resistance." Advances in Cryptology-CRYPTO 2006. Springer Berlin Heidelberg, 2006. 602-619.

[3] Boldyreva, Alexandra, et al. "Order-preserving symmetric encryption." Advances in Cryptology-EUROCRYPT 2009. Springer Berlin Heidelberg, 2009. 224-241.

[4] Boneh, Dan, and Brent Waters. "Conjunctive, subset, and range queries on encrypted data." Theory of cryptography. Springer Berlin Heidelberg, 2007. 535-554.

[5] Boneh, Dan, and Matt Franklin. "Identity-based encryption from the Weil pairing." Advances in Cryptology—CRYPTO 2001. Springer Berlin Heidelberg, 2001.

[6] Boneh, Dan, et al. "Public key encryption that allows PIR queries." Advances in Cryptology-CRYPTO 2007. Springer Berlin Heidelberg, 2007. 50-67.

[7] Boneh, Dan, et al. "Public key encryption with keyword search." Advances in Cryptology-Eurocrypt 2004. Springer Berlin Heidelberg, 2004.

[8] Chronology of data breaches. Privacy Rights Clearinghouse. Web. 14 September 2013. http://www.privacyrights.org/data-breach/

[9] Cocks, Clifford. "An identity based encryption scheme based on quadratic residues." Cryptography and Coding. Springer Berlin Heidelberg, 2001. 360-363.

[10] Crypto++®. Crypto++ Library. Web. 8 August 2013. http://www.cryptopp.com/

[11] Curtmola, Reza, et al. "Searchable symmetric encryption: improved definitions and efficient constructions." Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006.

[12] Ehrsam, William F., et al. "Message verification and transmission error detection by block chaining." U.S. Patent No. 4,074,066. 14 Feb. 1978.

[13] Goldreich, Oded, and Rafail Ostrovsky. "Software protection and simulation on oblivious RAMs." Journal of the ACM (JACM) 43.3 (1996): 431-473.

[14] Hu, Weiming, et al. "A survey on visual surveillance of object motion and behaviors." Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 34.3 (2004): 334-352.

[15] Hwang, Yong Ho, and Pil Joong Lee. "Public key encryption with conjunctive keyword search and its extension to a multi-user system." Pairing-Based Cryptography–Pairing 2007. Springer Berlin Heidelberg, 2007. 2-22.

[16] Kagal, Lalana, and Hal Abelson. "Access control is an inadequate framework for privacy protection." W3C Privacy Workshop. 2010.

[17] Katz, Jonathan, Amit Sahai, and Brent Waters. "Predicate encryption supporting disjunctions, polynomial equations, and inner products." Advances in Cryptology–EUROCRYPT 2008. Springer Berlin Heidelberg, 2008. 146-162.

[18] NIST, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," Federal Information Processing Standards Publication, n. 197, November 26, 2001.

[19] NIST, "Secure Hash Signature Standard", Federal Information Processing Standards Publication 180-2. Gaithersburg, MD. 2012.

[20] Shi, Elaine, et al. "Multi-dimensional range query over encrypted data." Security and Privacy, 2007. SP'07. IEEE Symposium on. IEEE, 2007.

[21] Song, Dawn Xiaoding, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data." Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000.

[22] Tang, Qiang. "Search in Encrypted Data: Theoretical Models and Practical Applications." IACR Cryptology ePrint Archive, 2012.

[23] Weiss, Jeffrey A. "Method and apparatus for synchronizing encrypting and decrypting systems." U.S. Patent No. 4,654,480. 31 Mar. 1987.