Hindawi
Security and Communication Networks
Volume 2021, Article ID 5317062, 12 pages
https://doi.org/10.1155/2021/5317062

Research Article

A Reconstruction Attack Scheme on Secure Outsourced Spatial Dataset in Vehicular Ad-Hoc Networks

Qing Ren,[1] Feng Tian ,[1] Xiangyi Lu,[1] Yumeng Shen,[1] Zhenqiang Wu,[1] and Xiaolin Gui[2]

[1]*School of Computer Science, Shaanxi Normal University, Xi'an 710062, Shaanxi, China*
[2]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, Shaanxi, China*

Correspondence should be addressed to Feng Tian; tianfeng@snnu.edu.cn

Received 11 June 2021; Accepted 14 August 2021; Published 8 September 2021

Academic Editor: Anjia Yang

In the cloud-based vehicular ad-hoc network (VANET), massive vehicle information is stored on the cloud, and a large amount of data query, calculation, monitoring, and management are carried out at all times. The secure spatial query methods in VANET allow authorized users to convert the original spatial query to encrypted spatial query, which is called query token and will be processed in ciphertext mode by the service provider. Thus, the service provider learns which encrypted records are returned as the result of a query, which is defined as the access pattern. Since only the correct query results that match the query tokens are returned, the service provider can observe which encrypted data are accessed and returned to the client when a query is launched clearly, and it leads to the leakage of data access pattern. In this paper, a reconstruction attack scheme is proposed, which utilizes the access patterns in the secure query processes, and then it reconstructs the index of outsourced spatial data that are collected from the vehicles. The proposed scheme proves the security threats in the VANET. Extensive experiments on real-world datasets demonstrate that our attack scheme can achieve quite a high reconstruction rate.

1. Introduction

At present, the vehicular ad-hoc network (VANET) has gained a lot of attention in the field of intelligent transportation. The VANET can be used to intelligently control the traffic process, such as real-time traffic information systems to ensure traffic efficiency, and vehicle safety systems, such as rear-end collision warning systems, to improve vehicle safety. However, the powerful function of the VANET is supported by information sharing between vehicle users, which will introduce serious data security threats [1, 2]. For example, exploiting the weakness of lacking physical proximity authentication, malicious attackers may infer the location of the vehicle during a specific period of time [2]. Due to the openness and mobility of VANET, the content delivery of VANET poses serious security threats, for which some countermeasures have been proposed, such as confidentiality, integrity, and authentication [3, 4].

As the data generated by users of VANET continue to grow and beyond the processing capacity of the data owners, the data need to be outsourced and stored in the cloud server to reduce data management overhead. To ensure the security of user data on the untrustworthy server, cryptographic techniques are employed, while still allowing efficient query processing on the cloud server. However, the privacy provided by the existing secure outsourced dataset systems is poorly considered. In the searchable encryption mechanism, the search process for encrypted files is as follows: First, the authorized user will submit the query token to the service provider, who will process the query through a series of calculations and return the query result to the user in the form of ciphertext. Then, the user decrypts the query result locally. It seems very safe because the entire query process is carried out in the ciphertext state, including query submission, query process calculations, and query results feedback.

Nonetheless, this process leaks access patterns. In other words, the service provider can observe which encrypted files in the dataset are accessed and returned to the authorized users. Therefore, the honest but curious service provider clearly knows the matching relationship between encrypted files and queries.

Existing studies [5–8] have shown that attackers can use leaked access patterns to recover user privacy information. Li et al. [5] demonstrated the hidden security threats caused by leaked access patterns with an encrypted patient medical dataset stored in a third-party server. Series of examples are given sequentially to illustrate how the patient's sensitive information is gradually inferred with the leakage of access patterns. With leaked access patterns, Quan et al. [6] implemented a range injection attack on the one-dimensional and discrete dataset. Exploiting the collusion of the service provider and the secondary user, a set of selected range queries are injected into the dataset, and through the access patterns of these queries, the dataset index is completely reconstructed. The attack method proposed by Islam et al. [7] does not require collusion, which is designed for text data. Considering the collected keyword co-occurrence matrix as prior knowledge, the service provider realizes an assignment of keywords to each query. Kallaris et al. [8] reconstructed the discrete one-dimensional data and completely restored the data index stored on the server, in which neither collusion nor prior knowledge is required.

Researchers have explored various types of attack methods for different types of datasets to prove the security threats caused by the leakage of access patterns, but there is no research to prove the potential security threats caused by the leakage of spatial data access patterns.

In this paper, we propose a reconstruction attack scheme on outsourced spatial dataset using access pattern leakage in VANET systems. The threat model considered is as follows. We take the honest but curious service provider as the attacker, who will process the query correctly and honestly, but will be curious about the dataset stored on the server. Our attack aims to completely reconstruct the dataset stored on the server without any deciphering, that is, to determine the spatial index of each record on the server.

Assuming that the service provider only has a little prior knowledge of the spatial dataset and the users will issue enough one-dimensional and uniform queries to the server, our reconstruction attack will be processed as the following four steps. Firstly, the data space will be discretized in accordance with the granularity that the attacker aims to achieve. Secondly, to improve the efficiency of attack, the collected access patterns will be simplified. Thirdly, we will determine the relative order for each row/column of records. Finally, the spatial index of each record will be recovered.

The contributions of this paper are summarized as follows:

(i) A reconstruction attack against secure outsourced spatial dataset in VANET systems is proposed, proving that the security threats caused by access pattern leakage are universal.

(ii) With spatial discretization, our scheme can support the attack for optional spatial granularity. Meanwhile, utilizing the statistic of the record co-occurrence, access patterns are simplified, which guarantees the attack efficiency.

(iii) Extensive experiments on real-world datasets demonstrate that our attack scheme can achieve quite a high reconstruction rate.

2. Related Work

2.1. Location Privacy Protection of VANET. The existing location privacy protection technologies of VANET mainly include three categories. The first category is a rule-based privacy protection method [9, 10], which restricts service providers from a legal perspective and prohibits the data and user information abused through privacy protection rules, standards, and detailed specifications acting on the server side. For example, IETF's GeoPriv [9] and W3C's P3P [10] stipulate that the authorization, integrity, and privacy requirements must be met when data are used. However, the security of such methods depends on legal supervision and public opinion, and the reliability of privacy protection depends on the implementation degree of the service provider.

The second category is based on generalization and obfuscation [11–17]. Spatial concealment technology [11, 12] forms a hidden area containing k real users for each user, making it difficult for service providers to determine the real identity and accurate spatial of the user from the hidden area. But in this type of method, it is difficult to achieve a balance between privacy protection and service quality in data-sparse areas. Spatial offset and obfuscation technology [13–16] protect user's privacy by moving the real spatial in a small area or replacing the real spatial with a certain area. For example, Andrés et al. [16] achieved the geo-indistinguishability by adding controlled random noise to user's spatial, which corresponds to differential privacy. Nevertheless, this kind of method reduces the spatial accuracy [17], so the returned service data may be untrustworthy.

The third type of privacy protection method is based on cryptography [18–22], through the processing of cryptographic technology to achieve the requirements of privacy protection. The general process is as follows: Authorized client encrypts the spatial information and sends it to the service provider, who processes the corresponding query in the ciphertext and returns matching encrypted result. Finally, authorized client decrypts locally to obtain the plaintext information. After encryption processing, the data sent by the client can meet stricter privacy protection requirements, but encryption and decryption bring huge computational overhead. In addition, although the data are encrypted before outsourcing, search process is also performed in ciphertext on the server; this method still has the problem of access pattern leakage (except for the ORAM scheme [23–25]); that is, the attacker can observe the correspondence between the encrypted query and the accessed encrypted documents. Furthermore, the ORAM solution protects the access pattern by shuffling and re-encrypting after each access of data, but its huge communication overhead makes using ORAM to protect the access pattern too expensive.

Therefore, in location privacy protection technology, the privacy security threats caused by leaked access patterns need to be further studied.

2.2. Attack with Access Patterns. In recent years, researchers have made arguments for the privacy security threats caused by the leakage of access patterns from the perspective of attacks, which are mainly divided into two categories.

One is active attacks [6, 26], including injection, tampering, and forgery, by which the attacker attacks server information actively. By injecting files that added selected keywords into the dataset on the server and observing the access patterns of the injected files, Zhang et al. [26] inferred the user's query information successfully. With the collusion of the service provider and secondary users, Quan et al. [6] injected a set of selected range queries and observed access patterns to infer the user's precise information. However, because the active attacks destroy the authenticity and integrity of the information, it is easy to be detected.

The other is passive attack [5, 7, 8, 27]. The attacker can infer the user's sensitive information through the monitored access pattern without affecting normal data communication, thereby undermining the confidentiality of data transmission. Islam et al. [7] proposed that by utilizing the statistical keyword co-occurrence matrix as prior knowledge and collecting access patterns, the attacker realizes an assignment of keywords to each query that achieves maximum matching from background knowledge. Assuming that the attacker has more prior knowledge (including the number of files corresponding to each keyword), Cash et al. [27] proposed an improved passive attack method based on IKK, and utilized the access pattern leakage to recover the query keywords. Without any prior knowledge, utilizing the continuity of range query, Kellaris et al. [8] assigned index to each file on the server and completely reconstructed the dataset. This type of attack is not easy to detect, because there is no direct impact on data transmission.

In summary, existing researches have proved the security threats caused by the access pattern leakage from different angles. However, the problem of spatial data access pattern leakage has not been studied in detail. And, most of the existing attack methods are active attacks, including injection or passive attacks, requiring much prior knowledge, so the attack conditions are subject to certain restrictions.

3. System Model

3.1. Spatial Data Outsourcing System. The system model of the outsourcing dataset system is shown in Figure 1, including three entities, namely, the data owner, the server, and the authorized user.

As the data generated by users of VANET continue to grow and beyond the processing capacity of the data owners (DO), the data need to be outsourced to the cloud server. For security concerns, encryption is usually performed before outsourcing. In order to facilitate user query, the encrypted index is also generated and uploaded to the cloud server at the same time.

Authorized users (AU) have secret keys, KI and KD. Encrypted query tokens will be generated by KI and uploaded to the cloud server. Ciphertext query results obtained from the server will be decrypted by the secret key KD.

The server stores the data and their corresponding query index for the data owner, and has powerful computing capabilities as well as searchable encryption algorithm so that it supports queries under ciphertext.

As shown in Figure 1, when an authorized user generates a query Q and encrypts and sends it to the cloud server, the service provider will perform the query operation and return the matching set of records R to the authorized user.

3.2. Access Pattern Leakage. In the data outsourcing system, authorized users usually generate spatial query tokens, which are processed in ciphertext mode on the server, and finally only the matching results are accessed and returned. So, the service provider learns which encrypted records match a query, which is defined as access pattern. Since only the correct query results that match the query tokens are accessed and returned, the service provider can observe the access pattern clearly when a query is launched. So, the process leads to the leakage of data access pattern. Such leakage is typical for current VANET systems based on symmetric searchable encryption. The queries on the server are continuous, so the service provider can sniff out a large number of access patterns quietly, which provides tremendous amount of data to the attacker.

In this paper, we define the access pattern leakage $\mathbf{L}_{\mathbf{access}}$ as: the correspondence between the ciphertext query q and the matching ciphertext query result $\mathbf{R}$.

$$\mathbf{L}_{\mathbf{access}} = \{(q_1, \mathbf{R}_1), \ldots, (q_n, \mathbf{R}_\mathbf{n})\}, \tag{1}$$

$$\mathbf{R}_\mathbf{n} = \{r_j | q_n(I_j) = 1, \quad I_j \in \mathbf{I}\}. \tag{2}$$

As shown in equation (1), the leaked access pattern contains multiple queries and their matching query records sets, where q_n represents a query token, and $\mathbf{R}_\mathbf{n}$ represents the matching records set of query q_n. As shown in equation (2), the index corresponding to each record on the dataset is calculated with query token q_n, and the matching records are returned as set $\mathbf{R}_\mathbf{n}$.

3.3. Attack Model. In this section, we describe the attack model of reconstruction attack with access pattern leakage. Here, the attacker has access to the communication channel, and thus observes a set of access patterns $\mathbf{L}_{\mathbf{access}} = \{(q_1, \mathbf{R}_1), \ldots, (q_n, \mathbf{R}_\mathbf{n})\}$. Let us define a week attacker as follows:

(1) The attacker is passive. The attacker follows the predefined storage and query rules and provides users with correct query results. The attacker will not perform illegal access, injection, or tampering to the dataset, but will process information stealing and collecting. Since the attack process does not involve

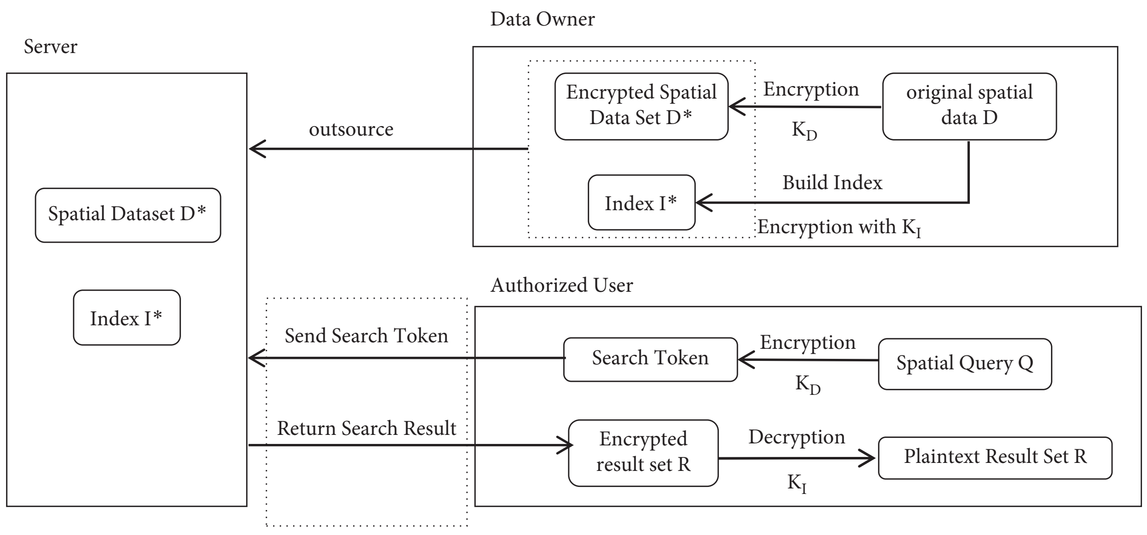

Figure 1: Spatial data outsourcing system.

data destruction, the legitimate users will not realize the attacker's activities at all.

(2) The attacker cannot decipher the query submitted by the authorized user through the secret key.

(3) The attacker has a very high probability of succeeding if it has a small amount of background knowledge. We assume that the attacker knows the underlying indexes for k of records in the dataset. That is, the attacker has access to the map $\mathbf{M}_{\mathbf{know}} = \{(I_j, r_j) | I_j \in \mathbf{I} \,\&\, r_j \in \mathbf{R}\}$, where $\mathbf{R}$ includes all records stored on the server and $\mathbf{I}$ includes all Index corresponding to R. Taking that $k = |\mathbf{M}_{\mathbf{know}}|$ and $l = |\mathbf{R}|$, then $k \ll l$.

What we need to be clear is that the attacker can observe the distribution of all records stored in the cloud. Although these records are usually encrypted before being uploaded to the server and the attacker cannot obtain the plaintext information of any record, the records can be distinguished easily.

Now, we assume that the attacker knows the encrypted dataset R, the access patterns $\mathbf{L}_{\mathbf{access}}$, and the prior knowledge $\mathbf{M}_{\mathbf{know}}$, and that the user will issue enough one-dimensional and uniform queries to the server. Then, the goal of the attack is to reconstruct the index of all records such that the statistical results as seen by access patterns fit the uniform query rule. In the following sections, we will describe how the attacker uses access pattern leakage to carry out reconstruction attacks on the two-dimensional spatial outsourcing dataset system.

4. Reconstruction Attack with Access Patterns

We propose a reconstruction attack on outsourced spatial dataset that exploits access pattern leakage, that is, the correspondence between encrypted queries and matching query results, which is very common in searchable symmetric encryption.

Assume that the dataset contains n records $r_1, r_2, \ldots, r_n$, which are, respectively, pointed to by spatial indexes $I_1, I_2, \ldots, I_n$. On the spatial dataset, the index is actually the spatial coordinate of each record, that is, $I_i = \{x_i, y_i\}$, which, respectively, represent the horizontal and vertical coordinates of the record. The ultimate goal of the reconstruction attack is to restore the corresponding position coordinates for each record r_1 in the dataset. Ideally, a complete plaintext index can be established. Assuming that the service provider only has little prior knowledge of the spatial dataset and the user will issue enough one-dimensional and uniform queries to the server, the service provider can utilize the observed access patterns to determine the index of each record.

In this section, we will describe the reconstruction attack process in detail, including spatial discretization, access pattern simplification, and determination of row and column indexes.

4.1. Spatial Discretization. In this section, we will discuss the problem of attack granularity. The ultimate goal of the reconstruction attack is to recover the index of each record on the two-dimensional spatial dataset. But for different application scenarios, the attacker hopes that the granularity of the spatial obtained by the attack is different.

As shown in Figure 2, we discretize the data space into $\Delta_2^2, \Delta_4^2, \Delta_8^2$, which means that both dimensions of space is divided into 2, 4, or 8 index spaces, respectively. Under different attack granularities, the attacker recovers the index of records in Figure 2, and the obtained index coordinates are (0,1), (1,2), (2,5), respectively. Combined with the size of the dataset known to the attacker, under different attack granularities, the index spatial of the record has different degrees of privacy leakage. It is undeniable that regardless of the granularity, reconstruction attacks will expose data privacy to a certain extent.

Therefore, we first need to determine the granularity of the data index that the attacker wants to achieve before formally attacking, which determines the granularity of the

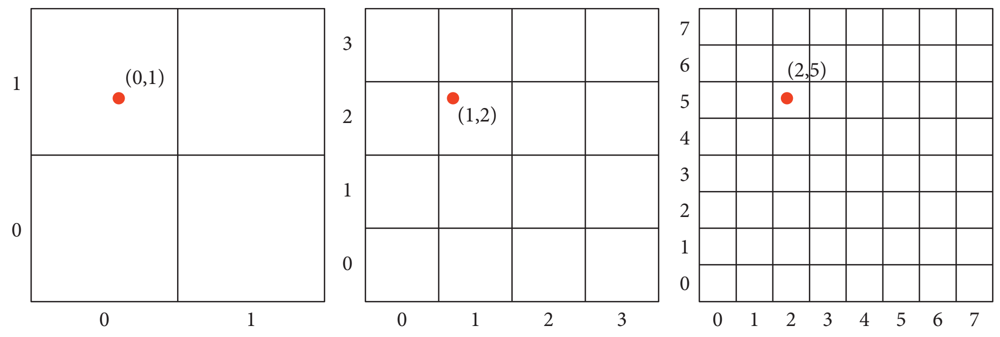

Figure 2: The record index obtained by the attack at different granularities.

space division. Discretize the data space according to the granularity to obtain a two-dimensional discrete data space. We assume that the data space is divided into Tx scales horizontally and T_y scales vertically, and the size of the data space is T_xT_y.

For each record in the dataset, it is pointed to by a certain position index $I_i = \{x_i, y_i\}$, where $x_i \in [0, T_x]$ and $y_i \in [0, T_y]$. The ultimate goal of our reconstruction attack is to recover the corresponding position coordinate $I_i' = \{x_i', y_i'\}$ for each record r_1 in the dataset.

4.2. Access Pattern Simplification. In section 4.1, we have introduced the process of spatial discretization. Assuming that the space has been discretized into n pieces of index space, the goal of reconstruction attack is to assign one piece of index space to each record stored on the server.

After a period of sniffing, the attacker (honest but curious service provider) collects enough access patterns to perform the attack. In order to improve the efficiency of our attack, we want to simplify these access patterns in this step. We will classify the records where the records belonging to the same piece of space are classified into one category and a representative record will be selected. By observing the access patterns and counting the co-occurrence of encrypted records, the records with co-occurrence rate of 100% are classified into the same category, and for each category, one record is taken as the representative. Then, the access pattern is simplified by replacing all records with the representative record of corresponding category in the collection of access patterns.

For example, with the attack granularity of 10×10, space is divided into 100 regions, and each region will be pointed to by the same index coordinate. Therefore, for the records that belong to the same region, we only take one record as the representive. If each region has records, we will get 100 representative records at most. Then, we simplify the observed access patterns. Because records belonging to the same category are pointed to by the same spatial index, we can simplify the collection of access patterns by keeping only representative records.

4.3. Determination of Row and Column Indexes. In this section, the row and column indexes will be determined, respectively, and we will introduce the process with 3 algorithms.

The Algorithm 1 is the main method, reconstructing the index of the dataset from the x-dimension and the y-dimension, respectively. Each dimension includes 2 steps. Firstly, the attacker utilizes the continuity of the range query to determine the relative position of records on a single dimension (Algorithm 2). Secondly, leveraging the uniformity of query and the rate of document co-occurrence in the access patterns, the attacker determines the specific index of each record (Algorithm 3). -

For the discretized two-dimensional space, we determine the x coordinate of each record line by line. Firstly, the attacker classifies the records according to the results of a single-row query. Encrypted records belonging to the same row query are grouped together. For this row of records, the attacker uses the collected access patterns set $\mathbf{L}_{\mathbf{access1}}$ (these access patterns are generated by uniform query) to determine the relative order of these records in the row through Algorithm 2 (line 5). Next, the attacker uses Algorithm 3 to get the index of these records in the x-direction (line 6). After the line-by-line process is over, the x-coordinates of each record are saved through the Map collection, the ID of the encrypted record is used as the key, and the X-coordinate as the value (line 7–9). Then, the attacker uses the same method to determine the Y coordinate of each record in the dataset (line 13–14), and saves it through the Map collection, with the ID of the encrypted record as the key, and the Y coordinate as the value (line 15–17).

4.3.1. Determination of Relative Order. In this section, we will introduce the procession of the relative order determination for a row/column of records. As a weak attacker defined in Section 3.3, the service provider can only continuously observe the user's query process and calculate access patterns. Assume that the attacker counts a lot of uniform one-dimensional spatial queries, which is enough to perform our reconstruction attack. The two-dimensional spatial dataset reconstruction attack can be converted into multiple one-dimensional attacks and the latter can adopt Generic Attack introduced in [8].

Given that the attacker collects only one-dimensional queries, it is easy to categorize each record stored in a two-dimensional space by row or column. For each row of

```
Input:attack granularity $T_x, T_y$, dataset, access patterns
Output: $x$ index and $y$ index of all records in the dataset
(1)  Map map Index$X$
(2)  Map map Index$Y$
(3)  FOR each row
(4)      get $\mathbf{L_{access1}}$
(5)      ordered Record_$X$←Get ordered ($\mathbf{L_{access1}}$)
(6)      Guess Index_$X$←Get Index ($\mathbf{L_{access1}}$, Ordered Record_$X$)
(7)      For each record in ordered Record
(8)          map index (record$ID$, $X$_Index)
(9)      END FOR
(10) END FOR
(11) FOR each column
(12)     get $\mathbf{L_{access2}}$
(13)     ordered Record_$Y$←Get ordered ($\mathbf{L_{access2}}$)
(14)     Guess Index_$Y$←Get Index ($\mathbf{L_{access2}}$, Ordered Record_$Y$)
(15)     For each record in ordered Record
(16)         map index (record$ID$, $Y$_Index)
(17)     END FOR
(18) END FOR
(19) RETURN map Index$X$, map Index$Y$
```

Algorithm 1: Reconstruction of the spatial dataset index.

```
Input: the collection of access patterns for a row/column $\mathbf{L_{access1}}$
    $\mathbf{L_{access1}} = \{(q_1, \mathbf{R_1}), \dots, (q_n, \mathbf{R_n})\}$
Output: the ordered set of records in the row/column Ordered Record
    Ordered Record $= (r_1, r_2, \dots r_m)$
(1)  $m = 0, R = \{\}$
(2)  FOR each $L_i$ in $\mathbf{L_{access1}}$ DO
(3)      IF $|R| > m$ THEN
(4)          $m = |R_i|, R = R_i$
(5)      END IF
(6)  END FOR
(7)  FOR each $L_i$ in $\mathbf{L_{access1}}$ DO
(8)      IF $|R| = m - 1$ THEN
(9)          ordered Record add ($R/R_i$)
(10)         BREAK
(11)     END IF
(12) END FOR
(13) FOR $j = 2$ to $m$ DO
(14)     FOR each $L_i$ in $\mathbf{L_{access1}}$ $\sqrt{a^2 + b^2}$ DO
(15)         IF $|R_i|$ = |ordered Record| + 1 and $R_i$Contains All (ordered Record) THEN
(16)             ordered Record add ($R_i$/ordered Record)
(17)             BREAK
(18)         END IF
(19)     END FOR
(20) END FOR
(21) RETURN ordered Record
```

Algorithm 2: GetOrder: Determination of the relative order for row/column records.

records, we first utilize the continuity of spatial query to determine the relative order of the records, through a process known as GetOrder, the details of which are as follows:

(1) Find the maximum set U of the row query in the access patterns

(2) Find the largest true subset S_1 of U, then the difference set between U and S_1 is the first record r_1

(3) Find the minimum superset S_1 of the set $\mathbf{R_{i-1}}$, where $\mathbf{R_{i-1}}$is formed by the confirmed records $\mathbf{R_{i-1}} = \{r_j \mid j \in [1, i-1]\}$. And, the difference set between S_i and $\mathbf{R_{i-1}}$ is the next record.

```
Input: The set of access patterns of uniform query for a row/column$\mathbf{L}_{\mathbf{access2}}$
   $\mathbf{L}_{\mathbf{access2}} = \{(q_1, R_1), \ldots, (q_n, R_n)\}$
   The row/column ordered record collection **Ordered Record**
   **Ordered Record** $= (r_1, r_2, \ldots r_m)$
Output: the index of the row record/the column record $I$
 (1)   sum First = 0
 (2)   FOR each$L_i$ in $\mathbf{L}_{\mathbf{access2}}$ DO
 (3)     IF **ordered Record** [1] = in$R_i$THEN
 (4)       sumFirst + +
 (5)     END IF
 (6)   END FOR
 (7)   $I[1] = \arg\min X\,(\text{SumFirst}/\mathbf{L}_{\mathbf{access2}}\text{length}2x\,(T - x + I)/T\,(T + 1))$
 (8)   Sum = {0, 0}
 (9)   Sum [1] = Sum First
(10)   FOR each $L_i$ in $\mathbf{L}_{\mathbf{access2}}$ DO
(11)     FOR $j = 2$ to $m$ DO
(12)       $R' = \{\textbf{ordered Record}\,[m]\} | 0 < m < j + 1$
(13)       IF $R'$ in $R_i$ THEN
(14)         Sum [$j$] + +
(15)       END IF
(16)     END FOR
(17)   END FOR
(18)   FOR $j = 2$ to $m$ DO
(19)     $I[j] = \arg\min x\,(\text{Sum}\,j/\mathbf{L}_{\mathbf{access2}}\text{length} - 2I\,[I]\,(T - x + 1)/T\,(T + I))$
(20)   END FOR
(21)   RETURM $I$
```

Algorithm 3: GetIndex: Determination of indexes for row/column records.

To explain the process of GetOrder, let us consider that the user makes a one-dimensional range query for the single-row record distribution, as shown in Figure 3.

Then, the maximum set of query results is $\{r_7, r_{12}, r_{17}, r_{20}\}$. The largest true subsets are $\{r_{17}, r_7, r_{12}\}$ and $\{r_7, r_{12}, r_{20}\}$. Without the distribution figure, only according to the set relation, we can easily know that the first record of this row is either r_{17} or r_{20}, which is consistent with actual distribution. Assuming that the first record is r_{17}, the minimum superset of $\{r_{17}\}$ in the access pattern set is $\{r_{17}, r_7\}$. Thus, the second record is determined as r_7. The superset of the confirmed records is deduced accordingly and the third and fourth records are r_{12} and r_{20}, respectively. If we assume that the first entry is r_{20}, what we will get is the reverse order of the records. According to the prior knowledge, we can determine whether to reverse this sequence. Finally, as shown in Figure 4, we know that this row stores 4 records, and the relative order of storage is $\{r_{17}, r_7, r_{12}, r_{20}\}$, and the next step is to match each record with the correct index.

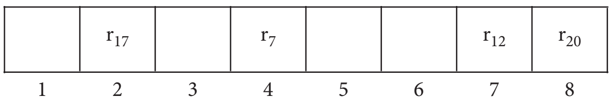

Figure 3: Records distribution of one row.

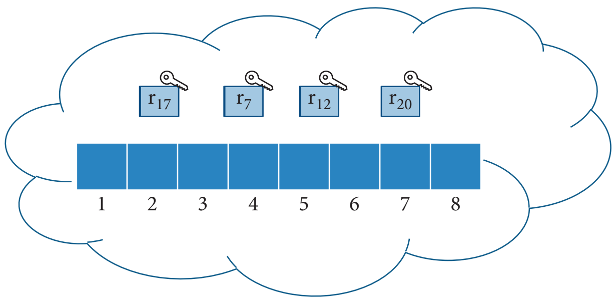

Figure 4: Relative order for the row of records.

The process of GetOrder has been summarized in Algorithm 2. The input of the algorithm is the access pattern set $\mathbf{L}_{\mathbf{access1}} = \{(q_1, \mathbf{R}_1), \ldots, (q_n, \mathbf{R}_\mathbf{n})\}$ of a certain line of query observed by the attacker. First traverse all access patterns to find the most number of record sets. Assuming that the attacker samples enough queries, $\mathbf{L}_{\mathbf{access1}}$ contains the access pattern accessing all the records in the row. From this, we get the row record set R and the record number m of this row (line: 1–6). After that, we determine the first item of the row's records (line: 7–12). Utilizing the continuity of range query, we find the largest proper subset R_1 of the complete set R of the row's records in the access pattern set, and then the difference between R_1 and R is the first/last item of the row of records. We assume that it is the first item, and then judge the row record after the row order is fully determined. If it is proved to be the last item, the row record only needs to be reversed. Finally, we sort the other encrypted records in this row one by one (lines: 13–20). Utilizing the continuity of range query, we find the minimum superset of the determined record in the access pattern set, and the difference between the minimum superset and the record set in the determined order is the next record. Determine the relative order of these m records one by one and store them in the queue **orderedRecord**, which is the order/reverse order of the row of records and the algorithm finally returns.

4.3.2. Determination of Indexes. Then, we utilize the uniformity of the query to determine the one-dimensional index value of each record, denoted as GetIndex. Assume that the index space of the row is N and the index coordinate of the i-th record is Z_i. By observing the access patterns of Q uniform queries and counting the number of access patterns containing record r_1 as q_1, we present the calculation of the first record index as an optimization problem by equation (3), and the calculation of the i-th record Index as an optimization problem by equation (4), where Qi represents the number of access patterns including the previous i records.

The result of this equation satisfying the optimization problem is an assignment of index Z_i to the record that achieves a minimum distance from the uniformity of query.

$$\arg\min \sum_{Z_1} \left(\frac{Q_1}{Q} - \left(\frac{2Z_1 (N - Z_1 + 1)}{N(N+1)} \right) \right)^2, \tag{3}$$

$$\arg\min \sum_{Z_1} \left(\frac{Q_i}{Q} - \left(\frac{2Z_1 (N - Z_i + 1)}{N(N+1)} \right) \right)^2. \tag{4}$$

To explain the model described in equation (3), let us consider the following example. Assume that the index space of a row is $[1, N]$, the record r_i is the i-th record of the row, and the x coordinate of r_i is Z_i. The row is uniformly queried, according to the permutation and combination, and the number of unique queries that can be generated is $N(N+1)/2$, the number of unique queries containing the first records is $Z_1(N - Z_1 + 1)$, and the number of unique queries containing the previous i records is $Z_1(N - Z_1 + 1)$. Now, for the first record r_1, an attacker can calculate the probability of the r_1 appearing in the uniform query by $\alpha_1 = 2Z_1(N - Z_1 + 1)/N(N+1)$.

For any given record r_i, $i > 1$, the attacker can calculate the probability of the previous i records $\{r_j | j \in [1, i]\}$ appearing together in the uniform query by $\alpha_1 = 2Z_1(N - Z_1 + 1)/N(N+1)$. Therefore, by observing access patterns, the attacker can calculate the probability of the previous i records $\{r_j | j \in [1, i]\}$ appearing together in the uniform query by $\beta = Q_i/Q$, where Q represents the total number of access patterns of uniform query and Q_i represents the number of access patterns including the previous i records. Naturally, the attacker will assign coordinate Z_i to the record r_i, if the calculated probability α is close to the observed probability β from the access pattern. This closeness can be measured by the arithmetic distance function $(\alpha - \beta)^2$, where a lower value of this function is preferred over a higher value. So, the goal of the attacker will be to assign an index to records such that this distance function is minimized.

A specific example is shown as follows to explain the process of GetIndex in detail. Assuming that the attack has determined the record order as shown in Figure 4, and the access pattern collected by the service provider is the smallest set that satisfies the attack conditions as shown in Figure 5, the attacker obtains the relative order $\{r_{17}, r_7, r_{12}, r_{20}\}$ and Q access patterns, where $Q = 36$.

Since the query is uniform, assuming that the x coordinate of r_{17} is $Z_1 \in [1, N]$, theoretically, the proportion of uniform queries including the first record r_{17} is $\alpha_1 = 2Z_1(N - Z_1 + 1)/N(N+1)$. Observing sampled access patterns, according to the statistics, the proportion of access patterns including the first record r_{17} is q_1/Q (i.e.14/36). Then, the difference between the actual value and the theoretical value is $q_1/Q - 2Z_1(N - Z_1 + 1)/N(N+1)$. We could find Z_1 that minimizes the absolute value of the difference and infer the value of Z_1 should be 2, so the x-coordinate of r_{17} is 2.

Determine the number of q_2 queries with $\mathbf{S}_2 = \{r_{17}, r_7\}$ included in the access patterns of the uniform query, where S_2 is the union of the records including records' determined position and the record of the position to be determined in this step. Then, in this example, q_2 is 10 and the proportion is q_2/Q. Since the query is uniform, assuming that the x coordinate of the second record r_7 is $Z_2 \in [1, \mathrm{N}]$, theoretically, the proportion of uniform queries including $\mathbf{S}_2 = \{\mathrm{r}_{17}, \mathrm{r}_7\}$ is $2Z_1(N - Z_2 + 1)/N(N+1)$, then the difference between the actual value and the theoretical value is $q_1/Q - 2Z_1(N - Z_1 + 1)/N(N+1)$, find Z_2 minimizes the absolute value of the difference and get Z_2 is 4, so the x coordinate of r_7 is 4. And, we can adopt the same measures for other records for their index values, and get the x coordinate of r_{12} and r_{20} as 7 and 8, respectively.

At this point, the attacker knows that there are four records $r_{17}, r_7, r_{12}, r_{20}$ in this row of the data space, and their x-coordinates are 2, 4, 7, and 8, respectively (Figure 6).

The process of GetIndex has been summarized in Algorithm 3. The input of the algorithm is the access pattern set $\mathbf{L_{access2}} = \{(q_1, R_1), \ldots, (q_n, R_n)\}$ of a uniform single-row query observed by the attacker, and the relative order queue of the row **orderedRecord** determined by algorithm GetOrder.

First, we count the number of access patterns including the first record and store in sumFirst (line: 1–6). Second, we determine the index number of the first record (line: 7). Utilizing the continuity of range query, assuming that a uniform query set is generated for the row of records, and the index number in the x-direction of the first record is x, theoretically, the probability that the query result contains the first record is $2x(T - x + 1)/T(T+1)$, where T is the data space size of the row. Then, traverse the access pattern set $\mathbf{L_{access2}}$, and according to statistics, the ratio of the access pattern including the first record is sumFirst/$\mathbf{L_{access2}}$.length. Find the x value that minimizes the difference between the theoretical value and the actual statistical value, which is the index number of the first item.

Finally, we determine the index of other records one by one (lines: 8–20). Utilizing the continuity of range query, assuming that a uniform query set is generated for the row of records, and the index number in the x-direction of the record **orderedRecord** $[j]$ is x, theoretically, the probability that the query result contains the first j records is

1.	{ }	11.	$\{r_{17},r_7\}$	21.	$\{r_7,r_{12},r_{20}\}$	21.	{ }
2.	$\{r_{17}\}$	12.	$\{r_{17},r_7\}$	22.	$\{r_7\}$	22.	$\{r_{12}\}$
3.	$\{r_{17}\}$	13.	$\{r_{17},r_7\}$	23.	$\{r_7\}$	23.	$\{r_{12},r_{20}\}$
4.	$\{r_{17},r_7\}$	14.	$\{r_{17},r_7,r_{12}\}$	24.	$\{r_7\}$	24.	$\{r_{12}\}$
5.	$\{r_{17},r_7\}$	15.	$\{r_{17},r_7,r_{12},r_{20}\}$	25.	$\{r_7,r_{12}\}$	25.	$\{r_{12},r_{20}\}$
6.	$\{r_{17},r_7\}$	16.	{ }	26.	$\{r_7,r_{12},r_{20}\}$	26.	$\{r_{20}\}$
7.	$\{r_{17},r_7,r_{12}\}$	17.	$\{r_7\}$	27.	{ }		
8.	$\{r_{17},r_7,r_{12},r_{20}\}$	18.	$\{r_7\}$	28.	{ }		
9.	$\{r_{17}\}$	19.	$\{r_7\}$	29.	$\{r_{12}\}$		
10.	$\{r_{17}\}$	20.	$\{r_7,r_{12}\}$	30.	$\{r_{12},r_{20}\}$		

Figure 5: Access pattern samples of uniform query.

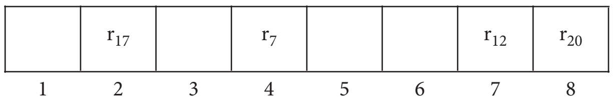

Figure 6: Index for the row of records.

$2x(T-x+1)/T(T+1)$, where i_1 is the index number of the first item and T is the data space size of the row. Then, we determine the statistical value of the probability. Traverse the access pattern set $\mathbf{L}_{\mathbf{access2}}$, and count the number of the access patterns including the first j records (**orderedRecord** [1], ... ,**orderedRecord** [j]) as sum[j] (lines: 8–17). Therefore, the ratio of each record is sum [j]/$\mathbf{L}_{\mathbf{access2}}$.length. Find the x value that minimizes the difference between the theoretical value and the actual statistical value, which is the index number of the **orderedRecord** [j] (lines: 18–20). At last, the index set I corresponding to the records in orderedRecord will be returned by the algorithm.

5. Experimental Results

The experiments are conducted on a laptop with limited resources (Intel Core i5 2.5 GHz CPU and 8 GB RAM).

We simulate three entities in this experiment, namely, DO, AU, and server. As shown in Figure 7, the AU stores a dataset to the server through the DO, and selects the spatial attribute as the data index. DO encrypts each record before storing it on the server. Then, the AU asks for a series of range queries on the spatial index, and DO retrieves the required encrypted records from the server, decrypts, and sends them to the user. In addition, we simulated a sniffer in Java on the server to observe data packets between the server and the DO for access pattern statistics. Finally, utilizing the observed access patterns, we performed the reconstruction attacks on the server side.

To evaluate the performance of our attack, we leverage a real-world spatial dataset, the distribution of North America Post Office including 175811 tuples [28]. We preprocess the raw dataset and normalize it to $[0, 1]^2$ before applying it to our experiment. The detailed distribution of this test dataset is shown in Figure 8. We encrypted and uploaded these tuples to the server and took spatial coordinates as their index.

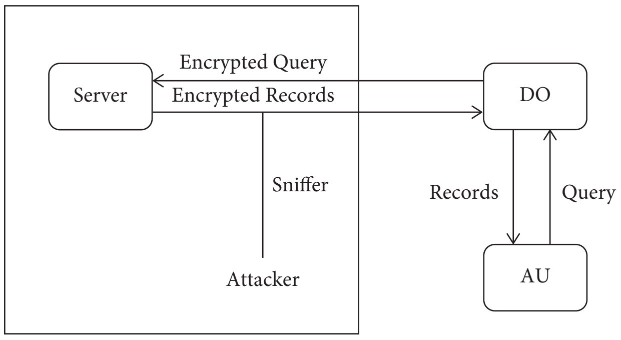

Figure 7: System implementation.

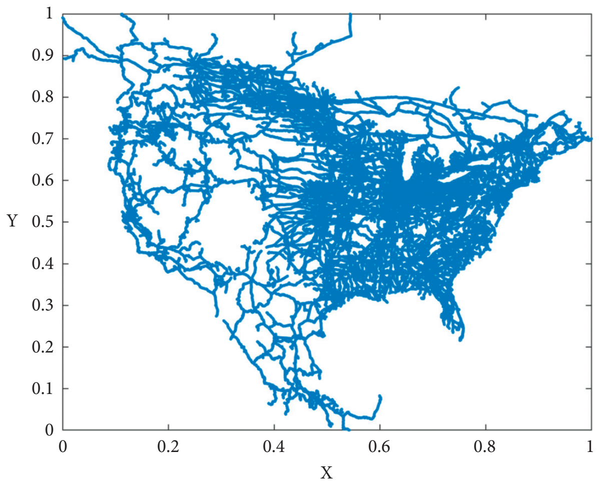

Figure 8: The distribution of test dataset.

According to the granularity that the attacker wants to achieve, we first discretize the dataset with different granularities. As shown in Table 1, column Attack Granularity describes the granularity of space discretization, and space is discretized into $10 \times 10 \cdots\cdots 100 \times 100$, while Index Number depicts the number of indexes in the different discretization conditions. And, Records Per Index represents the average number of records per position Index. Under different granularities, we will recover the index of each record through our attack.

Table 1: Space discretization.

Attack granularity	Index number	Records per index
10×10	100	1758.11
20×20	400	439.53
30×30	900	195.35
40×40	1600	109.88
50×50	2500	70.32
60×60	3600	48.84
70×70	4900	35.88
80×80	6400	27.47
90×90	8100	21.70
100×100	10000	17.58

After that, we gathered enough access patterns in order to run our attacks. The AU generates uniform range queries and issues to the DO. For each query, the DO retrieves encrypted matching records, decrypts them, and sends them back to the AU.

Due to the size of the query range, the network speed, and the number of users, the time of access patterns collection for our dataset will be very long. Therefore, in this experiment, we used a single user to simulate the process of query retrieval, and generate the minimum number of queries required for experimental conditions (Number of corresponding access patterns shown in Table 2). However, in the actual application scenario, when different users issue queries, the sniffer will sniff the access patterns from different users at the same time, and the collection speed will multiply.

Then, we preprocess the observed access patterns. By counting the co-occurrence of encrypted records in the access patterns, the records with co-occurrence rate of 100% are classified into categories, and for each category, one record is taken as the representative. The column Represent Number collects the number of representative records in the dataset under different attack granularities. For example, with the attack granularity of 10×10, the space is divided into 100 regions, each of which is pointed to by an index coordinate. By observing the access patterns, we classified 175,811 records, resulting in 68 representative records of 68 classes. Therefore, there are 32 regions with no record distribution under this attack granularity.

Finally, we preprocess the observed access patterns. Because records belonging to the same category are pointed to by the same position index, we can simplify the collection of access patterns by keeping only representative records.

Our reconstruction attack was performed on the preprocessed set of access patterns; Figure 9 summarizes our attack results. As shown in Figure 9, with the increase of attack granularity, our attack reconstruction rate showed a slight trend of decline, but it does not change much and the overall effect of the attack is good. Assuming that the data space is divided into $T_x \times T_y$, the higher granularity indicates that we divide the space more finely and that $T_x \times T_y$ is larger. As the granularity increases, the size of the index set increases, and it becomes more difficult for the attacker to assign indexes to records, so the reconstruction rate tends to decline.

The result curve looks a little wobbly because there are two main reasons that may affect the reconstruction rate of attack.

One is when some rows/columns do not conform to the prior knowledge, which will result in the reconstructed order being reversed during the process of GetOrder (Algorithm 2). In our reconstruction attack, we first determine the relative order of each row of records utilizing statistics on leaked access patterns. However, due to the symmetry of row query, we cannot determine whether the order we recover is in the positive or reverse order. If this is uncertain, the reconstruction rate of our attack will be very low. Therefore, we utilize prior knowledge $\mathbf{M}_{\mathbf{know}}$ to help us choose the correct option between the positive and reverse orders. If $\mathbf{M}_{\mathbf{know}}$ has no prior knowledge about any record of this row, the reconstruction rate of this row will be affected.

The other is when the first record in a row is in the second half of this row ($I_1 > T/2$), which will cause a recovery error during the process of GetIndex (Algorithm 3), and further lead to the error of other records in the same row. In the process of GetIndex, we determine the indexes for the records of each row/column. Similarly, due to the uniformity of row queries, when calculating the index of the first term of each row with equation (3), the optimal solution will be two indexes A and B, which are symmetric in the row. Assuming that A < B, then A is in the first half of the index, and B is in the second half. For dense datasets, the first record is naturally placed in the first half, so A is usually taken as the index of the first record. Therefore, when the first record in a row is in the second half of this row ($I_1 > T/2$), the reconstruction rate of this row will be affected.

Our attack includes four steps and Table 3 summarizes the running time of each step in our attack using access patterns.

(1) getRepresent: Count the co-occurrence probability of the encrypted records by the access patterns. Classify all records through co-occurrence probability and get a representative record for each category;

(2) Simplify access patterns: Simplify the access patterns, and keep only representative records;

(3) getRowIndex: Process simplified column access patterns and reconstruct the row index of records through the algorithm of GetOrder and GetIndex;

(4) getColIndex: Process simplified row access patterns and reconstruct the column index of records through the algorithm of GetOrder and GetIndex.

Table 2: The number of access patterns and represent records.

Attack granularity	Access pattern number	Represent record number
10×10	550×2	68
20×20	4200×2	224
30×30	13950×2	454
40×40	32800×2	756
50×50	63750×2	1105
60×60	109800×2	1488
70×70	173950×2	1931
80×80	259200×2	2433
90×90	368550×2	2927
100×100	505000×2	3474

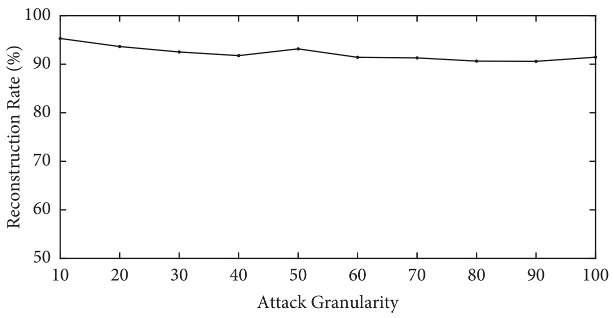

Figure 9: Attack reconstruction rate under different attack granularities.

Table 3: Attack time per step.

Attack granularity	GetRepresent (ms)	Simplification (ms)	GetRowIndex (ms)	GetColIndex (ms)
10×10	5	4	3	3
20×20	20	21	18	28
30×30	69	63	79	99
40×40	128	131	172	217
50×50	321	342	424	484
60×60	788	740	842	996
70×70	1708	1653	1379	1613
80×80	3358	3300	2535	2839
90×90	6086	6210	4336	4685
100×100	11229	10445	6987	7317

As shown in Table 3, GetRepresent represents the average time of finding representative records among 175811 records, while Simplification represents the average time of simplification for access patterns. GetRowIndex represents the average time to reconstruct the row index using these access patterns, while GetColIndex represents the average time to reconstruct the column index.

Table 3 shows that the time consumed for each step is up to only 11 seconds. Taking the most fine-grained granularity in this experiment as an example, when the space is discretized into 100×100 regions, we achieve that on average only 18 records are pointed to by the same position index, and the reconstruction rate of the record index reaches 91.45%. Under such fine-grained granularity and reconstruction rate, our attack time only needs a second level.

6. Conclusion

In this paper, a reconstruction attack on secure outsourced spatial dataset is proposed, proving that access pattern leakage will lead to security threats in VANET. With spatial discretization, our scheme can support the attack for optional spatial granularity. Meanwhile, utilizing the statistic of the record co-occurrence, access patterns are simplified, which improves the attack efficiency. Using the continuity and uniformity of the range query, the attacker determines the index for each record in the dataset. Extensive experiments on a real-world dataset demonstrate that our attack scheme can achieve a reconstruction rate of more than 90 percent even at a relatively fine-grained granularity. In future work, we will investigate the defense mechanism to address these security threats.

Data Availability

The parameters and datasets used to support the findings of this study are included within the paper.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported in part by National Natural Science Foundation of China under Grant nos. 61602290, 61902229, 61672334, 61802242, 61802241, and 61762058, Fundamental Research Funds for the Central Universities under Grant nos. GK202103090 and GK202103084, Natural Science Basic Research Plan in Shaanxi Province of China under Grant nos. 2017JQ6038 and No. 2020JM-288, and Foundation of Guizhou Provincial Key Laboratory of Public Big Data under Grant no. 2018BDKFJJ004.

References

[1] R. Kaur, T. P. Singh, and V. Khajuria, "Security issues in vehicular ad-hoc network(VANET)," in *Proceedings of the 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 884–889, Thirunelveli, India, May 2018.

[2] A. Yang, J. Weng, N. Cheng, J. Ni, X. Lin, and X. Shen, "DeQoS attack: degrading quality of service in VANETs and its mitigation," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4834–4845, 2019.

[3] M. Li, "Security in VANETs," *Vehicular Communications*, vol. 1, no. 2, 2014.

[4] A. Yang, J. Weng, K. Yang, C. Huang, and X. Shen, "Delegating authentication to edge: a decentralized authentication architecture for vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2020.

[5] S.-P. Li and M.-H. Wong, "Privacy-Preserving queries over outsourced data with access pattern protection," in *Proceedings of the 2014 IEEE International Conference on Data Mining Workshop*, pp. 581–588, Shenzhen, China, December 2014.

[6] H. Quan, H. Liu, B. Wang, M. Li, and Y. Zhang, "Randex: mitigating range injection attacks on searchable encryption," in *Proceedings of the 2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 133–141, Washington DC, DC, USA, June 2019.

[7] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: ramifification, attack and mitigation," in *Proceedings of the NDSS'12*, San Diego, CA, USA, February 2012.

[8] G. Kellaris, K. George, K. Nissim, and O. N. Adam, *Generic Attacks on Secure Outsourced Databases*, Association for Computing Machinery, New York, NY, USA, 2016.

[9] IETF, "Geographic spatial/privacy (geopriv) [EB/OL]," 2004, http://datatracker.ietf.org/wg/geopriv/charter/.

[10] World Wide Web Consortium (W3C), *Platform for Privacy Preferences (P3P) Project*, World Wide Web Consortium (W3C), Cambridge, MA, USA, 2000.

[11] K. Lim, K. M. Tuladhar, X. Wang, and M. Liu, "A scalable and secure key distribution scheme for group signature based authentication in VANET," in *Proceedings of the 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON) IEEE*, New York, NY, USA, October 2017.

[12] N. Guo, L. Ma, and T. Gao, "Independent mix zone for location privacy in vehicular networks," *IEEE Access*, vol. 6, pp. 16842–16850, 2018.

[13] M. L. Yiu, C. S. Jensen, J. Møller, and H. Lu, "Design and analysis of a ranking approach to private location-based services," *ACM Transactions on Database Systems*, vol. 36, no. 2, pp. 1–42, 2011.

[14] C. Ardagna, M. Cremonini, S. De Capitani Di Vimercati, and P. Samarati, "An obfuscation-based approach for protecting location privacy," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 1, pp. 13–27, 2011.

[15] C. Zhou, C. Ma, and S. Yang, "Research of LBS privacy preserving based on sensitive location diversity," *Journal on Communications*, vol. 36, no. 4, pp. 14–25, 2015.

[16] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, *Geo-Indistinguishability: Differential Privacy for Spatial-Based Systems*, Association for Computing Machinery, New York, NY, USA, 2013.

[17] W. Ni, X. Chen, and Z. Ma, "Location privacy preserving k nearest neighbor query method on road network in presence of user's preference," *Chinese Journal of Computers*, vol. 38, no. 4, pp. 884–896, 2015.

[18] R. Paulet, M. G. Kaosar, X. Xun Yi, and E. Bertino, "Privacy-Preserving and content-protecting location based queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1200–1210, 2014.

[19] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K. Tan, "Private queries in location based services: anonymizers are not necessary," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008*, ACM, Vancouver, Canada, June 2008.

[20] X. Yi, R. Paulet, E. Bertino, and V. Varadharajan, "Practical approximate k nearest neighbor queries with location and query privacy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1546–1559, 2016.

[21] D. Dawn Xiaoding Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pp. 44–55, Berkeley, CA, US, May 2000.

[22] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[23] M. Naveed, *The Fallacy of Composition of Oblivious RAM and Searchable Encryption*, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2015.

[24] S. Garg, P. Mohassel, and C. Papamanthou, *Tworam: Effificient Oblivious Ram in Two Rounds With Applications to Searchable Encryption*, Springer, New York, NY, USA, 2016.

[25] B. Fuller, M. Varia, A. Yerukhimovich et al., "SoK: cryptographically protected database search," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pp. 172–191, San Jose, CA, USA, May 2017.

[26] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: the power of file-injection attacks on searchable encryption," in *Proceedings of the USENIX Security*, pp. 707–720, Austin, TX, USA, August 2016.

[27] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the CCS'15*, Denver, CO, USA, October 2015.

[28] "Real datasets for spatial databases: road networks and points of interest," 2005, http://www.cs.utah.edu/lifeifei/SpatialDataset.htm.