Secure and Practical Searchable Encryption: A Position Paper

Shujie Cui✉, Muhammad Rizwan Asghar, Steven D. Galbraith, and Giovanni Russello

The University of Auckland, New Zealand
scui379@aucklanduni.ac.nz, {r.asghar, s.galbraith, g.russello}@auckland.ac.nz

Abstract. Searchable Encryption (SE) makes it possible for users to outsource the encrypted database and search operation to cloud service providers without leaking the content of the query and data to them. A number of SE schemes have been proposed in the literature; however, most of them leak a significant amount of information that could lead to inference attacks. To minimise information leakage, there are alternate solutions, such as Oblivious Random Access Memory (O-RAM) and Private Information Retrieval (PIR). Unfortunately, existing solutions are prohibitively costly and impractical. A practical scheme should support not only a lightweight user client but also a flexible key management mechanism for multi-user access.

In this position paper, we briefly analyse several leakage-based attacks, and identify a set of requirements for a secure cloud database that could resist against these attacks and ensure usability of the system simultaneously. We also discuss several possible solutions to fulfil the identified requirements.

1 Introduction

Cloud computing is a successful paradigm offering users virtually unlimited data storage and computational power at very attractive costs. Despite its merits, cloud computing raises privacy issues to users. Once the data is outsourced, it is exposed not only to third party intruders but also to careless or even potentially malicious Cloud Service Providers (CSPs). Standard encryption can protect the content of the outsourced data. However, it also prevents users from searching on encrypted data. If standard encryption is used, there is a trivial solution to perform search on encrypted data: if a particular piece of data is needed, the user has to download all the content to its local (trusted) environment, decrypt the data, and perform the search operation. If the database is very large, this trivial solution does not scale well. The matter becomes more complicated in multi-user settings, where multiple users could access the same data set.

The concept of Searchable Encryption (SE) provides a promising solution to protect outsourced data from unauthorised accesses by CSPs or external adversaries. Encrypted data is tagged with encrypted keywords (also called search tokens) in such a way that a CSP, which is given an encrypted search term, can check whether a record has keyword(s) that satisfies the search term. Such schemes allow the CSP to perform encrypted search on encrypted data without leaking the content of the query and the data.

Since the seminal paper by Song *et al.* [45], many SE schemes have been proposed. A long line of works, such as [2,7,9,10,14,18,20,22,23,25,29,30,35,38,44,45,47,50,53, 54, 56, 57], focus on investigating SE with complex functionality (*e.g.,* multi-keyword search, range queries, rank search, and fuzzy search) and improved performance. Although these schemes are secure under some cryptographic assumptions, we will see that the CSP is still potentially able to learn about the data by observing patterns in the results of insert, delete or update operations. For instance, the CSP is able to see which encrypted data is accessed by a given query by looking at the matching records (referred to as *access pattern leakage*). By comparing the matched records, the CSP can also infer if two or more queries are equivalent or not (referred to as *search pattern leakage*). Moreover, the CSP can simply log the number of matched records or files returned by each query (referred to as *size pattern leakage*).

When an SE scheme supports insert and delete operations, it is referred to as a *dynamic* SE scheme. Dynamic SE schemes might leak extra information if they do not support *forward privacy* and *backward privacy* properties. Forward privacy means that the CSP can not learn if newly inserted data or updated data matches previously executed queries. Backward privacy means that the CSP can not learn if deleted or stale data matches new queries. Supporting forward and backward privacy is fundamental to limit the power of the CSP to collect information on how the data evolves over time. Only a few of the existing schemes [5, 6, 10, 47] support forward privacy, but no scheme is able to support both forward and backward privacy simultaneously.

Some recent works [8, 27, 32, 34, 58] have shown that even a minor leakage could be exploited to learn sensitive information and break a scheme. In particular, given the plaintext of a small number of queries (*e.g.,* data, search habit or preference of users), a malicious CSP could recover a large fraction of the data and queries. Unfortunately, the majority of the existing SE solutions, such as [2, 7, 9, 10, 14, 18, 20, 22, 23, 25, 29, 30, 35, 38, 44, 45, 47, 50, 53, 54, 56, 57], are vulnerable to these attacks due to the leakage of search, access and size patterns, and lack of forward and backward privacy.

Privacy-sensitive applications, such as electronic healthcare systems, impose stringent security requirements when it comes to data outsourcing. The first step to meet those requirements is to minimise information leakage that could lead to inference attacks. Oblivious Random Access Memory (ORAM) [21,36,46] and Private Information Retrieval (PIR) [12, 13, 55] are two possible techniques to minimise information leakage. However, existing ORAM and PIR schemes are prohibitively costly and impractical. We are concerned with practical solutions where users can get the required data efficiently without incurring high storage, communication and computation overheads. Moreover, in multi-user settings, users could join or leave the system at any time, ideally without affecting the rest of the users. Unfortunately, neither ORAM nor PIR cover these aspects. Therefore, an important problem is to develop secure and practical SE schemes for privacy-sensitive applications. In this paper, we first define a set of requirements towards a secure cloud database. Then, we provide an extensive classification of the existing literature based on these requirements. Furthermore, we provide some research directions and possible approaches to ensure confidentiality without compromising on functionality and a practical user experience.

2 Requirements and Challenges

The recent proposed attacks on SE schemes, such as [8, 34, 58], put the outsourced data at risk. Almost all the attacks recover the data by leveraging private information leaked by size, search and access patterns. To minimise information leakage, the size, search and access patterns must be hidden from the CSP. In this section, for each pattern, we first briefly describe the typical attacks, and then we identify the requirements and challenges for protecting it. For a practical solution, a flexible key management mechanism is needed for applications requiring multi-user access. We also identify the requirements and challenges to manage the keys, but without leaking private information.

Size Pattern. The number of matched records for each query is called the size pattern or the frequency information [8, 27, 34, 45]. In particular, we say *Size Pattern Privacy (SzPP)* is achieved if the CSP is unable to learn the number of matched records of a query on encrypted data. In most of the existing SE schemes, there are two kinds of frequency information leakages. The first kind is a leakage from the encrypted data stored in the database; we call this *static frequency information*. For instance, in CryptDB [38], the searchable data is only protected with deterministic encryption (the outer layer encryptions are peeled off to support search operations). That is, the same data in the database has the same ciphertext. The frequency information in the database is exposed to the CSP directly. With the knowledge of publicly available information, like census data and hospital statistics, Naveed *et al.* [34] successfully recover more than 60% of patient records from electronic medical databases based on static frequency information without executing any search operation. In contrast, some other schemes [2] use semantically secure primitives to encrypt the data. In such schemes, the static frequency information is protected since all the encrypted data has different ciphertexts. However, after performing a query, the CSP could still count the number of matched records; we call this *dynamic frequency information*. With the knowledge of the frequency information of keywords in plaintext, attackers could recover the queries easily based on the number of matched records, which is called a *count attack* in [8].

Clearly, semantically secure encryption is not sufficient to protect the dynamic frequency information if the search operation is performed by the CSP. Protection against the count attack is required.

Access Pattern. We say that an SE scheme achieves *Access Pattern Privacy (APP)* if the CSP is unable to infer if two (or more) result sets contain the same data or not. A formal definition is given in [14]. A typical instance of access pattern based attack is the *file injection attack* introduced in [58], which is also referred to as the *chosen-document attack* in [8]. Technically, an active malicious CSP sends files with keywords of its choice, such as emails, to users who then encrypt and upload them to the CSP. Afterwards, the CSP tracks these injected files and checks if they match queries. Since all the keywords in these files are known to the CSP, given enough injected files, the CSP could recover the keywords included in queries. Specifically, the keywords included in the matched but not included in the unmatched injected files are the possible searched keywords.

ORAM is a primitive intended for hiding storage access patterns. However, it is difficult to get a practical SE scheme that is based on ORAM technique. We now explain the two main obstacles to using traditional ORAM in SE schemes. First, in ORAM, the store address of required data is known to users before fetching. However, in SE schemes, the CSP first needs to search over the database and get matched records or indexes of matched files. It is impractical to store all the addresses on the user side, especially for applications with thousands of users and resource-constrained devices. Second, despite significant recent improvements [16, 19, 40], ORAM incurs huge bandwidth, latency and storage overheads, making it impractical for SE schemes. According to the study by Naveed [33], the naive approach, downloading the whole database and search locally for each query, is even more efficient than ORAM.

PIR is another approach to hide the access pattern. It allows users to retrieve the data without leaking which data is retrieved to the CSP. Specifically, in PIR-based SE schemes, such as [15, 42], the CSP returns a much larger data set than required to the user. Although the access pattern is unknown to the CSP, the user has to perform some computation locally to extract the matched data. It is clear that the communication and computation overheads on the user are also huge in this approach.

Although both the ORAM and PIR techniques can protect the access pattern from the CSP, they may leak information to users. In an application with fine-grained access control policies, users should only get what they are allowed to learn. However, all the existing ORAM and PIR approaches do not ensure that all the returned data is authorised to the user. Therefore, a more practical method to protect the access pattern without leaking information to users should be proposed.

Search Pattern. We say *Search Pattern Privacy (SPP)* is achieved if the CSP is not able to distinguish if two (or more) encrypted queries feature the same keywords or not. In [32], Liu *et al.* show that given the search habit of users, the searched keywords could be recovered based on the search pattern. In [8], Cash *et al.* illustrate that given the plaintext of a small number of queries, the plaintext of other queries could be recovered easily if the adversary knows the search pattern. It is not trivial to protect the search pattern since it can be inferred not only from the encrypted queries, but also from the access and size patterns.

To protect the search pattern, first we should use a semantically secure encryption algorithm for the search tokens, which makes same queries look different once encrypted. Recall that to protect the static frequency information, it is also necessary to encrypt the data with a semantically secure algorithm. If both the query and the data are semantically secure, to the best of our knowledge, the only solutions in the literature use complex cryptographic primitives such as pairings or homomorphic encryptions. These primitives tend to be much slower than traditional symmetric encryption. So these methods do not scale well when processing the search operation over millions of records. A more efficient approach to test equality between semantically secure encrypted data is needed.

Furthermore, even if the encrypted queries are semantically secure, the CSP could still infer the search pattern by looking at the access pattern. That is, by looking at the physical locations of the encrypted data returned by a search, the CSP can infer that two queries are equivalent if the same result sets are returned, since generally only the same query gets exactly the same result set.

However, even if the matched data for all queries are different in terms of the storage location and ciphertext, the search pattern can still be inferred from the size pattern. If the database is static then equivalent queries will always return the same number of matched records or files. This fact can be used by the CSP to mount an attack: Although it is not always true that the two queries are logically equivalent when their result sets have the same size, it is true that they are different queries when their result set sizes are different.

Therefore, to conceal the search pattern, it is necessary to make the size, storage location and ciphertext of the search results variable even if the same query is executed twice.

Forward and Backward Privacy. Generally speaking, forward and backward privacy mean the CSP will learn nothing if it repeats a previously executed query (using the original search tokens) over newly added or updated data, or executes a new query over data that was supposed to have been deleted or updated. If the SE scheme cannot ensure forward and backward privacy, the CSP could recover all the queries with the file injection attack by executing all the previous queries again over the newly injected files. Similarly, if the CSP learns the plaintext of deleted files or records, then the queries could also be recovered by checking if they match deleted data.

To protect the search and access patterns, it is, in fact, necessary to also ensure forward and backward privacy. Recall that the storage locations and ciphertexts of searched data must be updated to protect the search and access patterns. However, if the CSP could execute the previous queries over the updated data and get a new set of matched data, it will infer the search pattern by comparing the result set of a new query with the result sets of previous queries. Likewise, the CSP can also infer the search pattern by caching the database and executing all the new queries over it.

Forward privacy is achieved in [5, 6, 10, 47]. Unfortunately, all of these proposals require the user to store a set of the latest keys, which will be used to encrypt queries. In multi-user settings, where multiple users could read and write to the database according to the access control policies, if one of the users inserts or updates a new record or file, the keys have to be updated, and then the new keys would have to be distributed to other users. Otherwise, with the stale keys, the other users cannot get the correct result set. These key management issues are impractical for multi-user applications. A more flexible approach is needed. Moreover, to ensure data confidentiality, backward privacy should also be guaranteed.

Key Management. Another issue with existing SE schemes is to have a very flexible key management mechanism for multi-user access. Many schemes, like [14], encrypt the data and queries with a key shared among all the users. Consequently, all the queries and search results issued by one user could be decrypted by all the other authorised users. Even worse, when one user is revoked, the single key has to be changed and the data has to be re-encrypted with the new key. In other schemes, such as [7] and [52], the keys are only known to the data owner. The users have to send the query and search result to the data owner to get the search tokens and cleartext results, which means that the data owner represents a bottleneck in the system. Both of these options are impractical in modern organisations, since a large number of users may access the data concurrently, or they may join and leave their position at any time. We call all such schemes *Single User (SU)*.

In a *Multi-User (MU)* scheme, users can submit queries to search or update the data uploaded by other users according to the access control policies, and no key regeneration and data re-encryption are needed for user revocation. Almost all the existing MU SE schemes, such as [2, 3, 17, 31, 39, 51], are based on proxy-encryption techniques. Basically, in these schemes, instead of sharing a single encryption key among all the users, each user has a unique key to encrypt data and queries. Moreover, the CSP stores another key for each user, with which the CSP could perform the equality check between the query and data encrypted by different users. However, all these schemes leak the search pattern since their query encryption algorithms are deterministic. Moreover, in [2] and [17], if a malicious user colludes with the CSP, they can recover all the data by putting their keys together. The schemes introduced in [3, 31, 39, 51] include several pairing operations, making the search operations computationally intensive. A more secure and practical MU key management mechanism is needed.

3 Literature Review

Table 1. A comparison of searchable encryption schemes.

Schemes	**Search pattern privacy**	**Access pattern privacy**	**Size pattern privacy**	**Forward privacy**	**Backward privacy**	**Key management**
Hang *et al.* [23]	×	×	×	×	×	○
Ferretti *et al.* [18]	×	×	×	×	×	○
Popa *et al.* [38]	×	×	×	×	×	●
Sarfraz *et al.* [44]	×	×	×	×	×	●
Sun *et al.* [50]	×	×	×	×	×	○
Yang *et al.* [56]	×	×	×	×	×	●
Asghar *et al.* [2]	×	×	×	×	×	●
Bao *et al.* [3]	×	×	×	×	×	●
Popa *et al.* [39]	×	×	×	×	×	●
Tang [51]	×	×	×	×	×	●
Kiayias *et al.* [31]	×	×	×	×	×	●
Curtmola *et al.* [14]	×	×	×	Static	Static	○
Jarecki *et al.* [28]	×	×	×	Static	Static	○
Kamara *et al.* [30]	×	×	×	×	×	○
Kamara *et al.* [29]	×	×	×	×	×	○
Hahn *et al.* [22]	×	×	×	×	×	○
Cao *et al.* [7]	✓	×	×	Static	Static	○
Wang *et al.* [52]	✓	✓	✓	Static	Static	○
Ishai *et al.* [26]	✓	✓	✓	Static	Static	○
Naveed *et al.* [35]	×	×	✓	×	×	○
Samanthula *et al.* [43]	✓	✓	✓	×	×	○
Stefanov *et al.* [47]	×	×	×	✓	×	○
Rizomiliotis *et al.* [41]	×	×	×	✓	×	○
Bost [5]	×	×	×	✓	×	○
Our objectives	✓	✓	✓	✓	✓	●

✓ and × indicate that the property is achieved or not, respectively.
○ represents a Single User (SU) scheme. ● represents a Multi-User (MU) scheme.
Static means the scheme does not support insert, update, or delete operations.

Since the seminal paper by Song *et al.* [45], many searchable schemes have been proposed and research in this area has been extended in several directions. In this section, we categorise the approaches presented in the literature based on information leakage and key management, and summarise their limitations.

Only several recent works tried to partially address the issue of information leakage. In [35], Naveed *et al.* achieve SzPP. The basic idea is to divide each document into a set of blocks. When a document is requested, a larger set of blocks will be downloaded and decrypted by the client, which aggravates the computational and storage overheads on the client side. Moreover, it fails to achieve SPP and APP, since the same query requests the same block set.

Samanthula *et al.* [43] present a query processing framework that supports complex queries. A homomorphic encryption algorithm is used to encrypt the data in their scheme. Thus, it supports more complex queries when compared to other schemes, and achieves SPP, APP, and SzPP. However, this scheme is single user and does not scale well for databases with a large number of attributes.

Cao *et al.* [7] design a scheme that supports a multi-keyword ranked search. The scheme ensures SPP by hiding the trapdoor linkability. Wang *et al.* [52] propose a public multi-keyword searchable encryption scheme based on Paillier [37], which achieves SPP, APP, and SzPP. More recently, in [26], Ishai *et al.* protect both the search and access patterns combining a PIR technique with a B-tree data structure. Although these three schemes provide different index structures for speeding up the search, the constructions are static and do not support insert, update, and delete operations.

In [47], Stefanov *et al.* design a dynamic sub-linear searchable construction based on an ORAM-like hierarchical structure and achieve forward privacy. Similarly, Rizomiliotis *et al.* [41] propose another dynamic ORAM-based scheme that achieves forward privacy and sub-linear search. More recently, the dynamic SE scheme introduced by Bost [5] also achieves forward privacy. Instead of using an ORAM-like structure, this scheme relies on a trapdoor permutation. However, it only ensures forward privacy until a new query is issued. A CSP could still learn if the new file contains the keywords searched previously, by comparing the access pattern of a new query with those of previous queries. Moreover, all these three schemes fail to ensure backward privacy.

Several works have concentrated on supporting multi-user access and simplifying key management. Curtmola *et al.* [14] introduce a multi-user (MU) scheme by combining a single user SE scheme with a broadcast encryption scheme, where only the authorised user can issue queries with the key received from the data owner. However, each time a user is revoked, the data owner has to generate a new key. Even worse, the data stored on the cloud server is encrypted with the key shared among all the users, which means the revoked users can still recover all the data if they collude with the cloud server. The MU SE scheme given by Jarecki *et al.* [28] has the same problem. That is, the data security against revoked users is achieved based on the assumption that there is no collusion between the cloud server and revoked users; otherwise, the key has to be updated and the data has to be re-encrypted with the new key. Moreover, in their scheme, the data owner has to be online to generate search tokens for all the authorised users.

Hang *et al.* [23] and Ferretti *et al.* [18] present two different collusion-resistant mechanisms that support multi-user access to the outsourced data. Although they support approaches to avoid key sharing among users, in both, after user revocation, it is necessary to generate a new key and re-encrypt the data.

CryptDB [38] is a multi-user scheme where each user has her own password, which is managed by a proxy between the user and the database server. Sarfraz *et al.* [44] revisit CrtypDB and also design a MU scheme with a fine-grained access control. Instead of assigning the keys to users, both [38] and [44] store them in a proxy. Since the users never know the underlying encryption key, they do not require to refresh the key when revoking a user. The problem is that these two mechanisms require the proxy to be online for performing operations on behalf of the users. As a result, the proxy represents a single point of failure: an attacker who compromises the proxy will gain access to all the logged-in users' keys and data.

Sun *et al.* [50] utilise a Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [4] mechanism to achieve a scalable SE scheme that supports multi-user read and write operations without sharing any key. However, for user revocation, the data has to be re-encrypted with a new access structure and secret keys of all the other users need to be updated with a new attribute set. Strictly speaking, this scheme is also a SU scheme.

In the literature, only the proxy-based encryption schemes, such as [2, 3, 31, 39, 51, 56], can support multi-user access, where each user has her own key and does not require any re-encryption when an authorised user is revoked.

Many other works also investigated approaches to increase search efficiency [14, 22, 30], or data integrity and reliability in the setting where the CSP is totally untrusted [11, 49]. Unfortunately, as shown in Table 1, none of the reviewed approaches are able to limit information leakage and support multi-user access.

4 Possible Solutions and Future Research

In this section, we propose several possible solutions and outline future research directions to meet the requirements and address the challenges to minimise information leakage and achieve a flexible key management.

Size Pattern. If the search operation is performed by the CSP, it is inevitable that the CSP could count the matched records or files. As mentioned in [8] and [14], introducing a set of dummy data into the database is an effective way to protect the size pattern without leaking private information to users. Basically, the dummy data should look exactly like the real data and even should match queries. In this way, the search result for each query will include a random amount of dummy data. Consequently, the size pattern is protected from the CSP. However, to ensure a lightweight overhead on the user end, it should be easy for users to filter out the dummy data before decrypting.

Furthermore, the amount of dummy data should be controllable since it affects the security level and performance of the system. The higher the percentage of dummy data with respect to real data, the harder for the CSP to infer the real size pattern. However, this also implies that more data should be searched by the CSP and more dummy data should be filtered out by the user for each query. The study by Cash *et al.* in [8] suggests that 1.6 is the minimum ratio between dummy and real data to resist against count attacks. A thorough security analysis for identifying a right balance between real and dummy data for achieving a sustainable level of security and performance should be investigated.

Another possible solution could be dividing the database into partitions and distributing these partitions over multiple non-colluding CSPs. For each query, each single CSP searches over its partition independently. Using this approach, each CSP only gets a small part of the search result. The total number of matched records is unknown to all the CSPs if they do not put their sub-results together.

Access Pattern. In the access pattern based attacks, the key point is that the CSP knows which injected records match the query and which do not. Therefore, to resist such attacks, we need to make storage locations of the injected data untraceable for the CSP. In fact, this can be achieved by generating and uploading dummy data when uploading real data. If the dummy and real encrypted data are indistinguishable to the CSP, it cannot learn if the search result contains the injected data or dummy data. Even so, the set of data injected at different time points is still distinguishable for the CSP. A technique that makes the data inserted at different times untraceable is to shuffle the database after executing each query, as we explain in the next sections.

Search Pattern. To protect the search pattern, first of all, the encrypted queries should be semantically secure. Moreover, we should break the link between the search pattern and access and size patterns. That is, we should ensure that the CSP will always see a new set of data being matched even if a query equivalent to the previous one is executed.

To break the link between the access and search patterns, the only choice is to shuffle the physical locations of the searched data after executing each query. Moreover, for making the data untraceable, the corresponding ciphertext should be re-randomised prior to moving to new locations. Even with the ORAM technique, the access pattern is protected by changing the data location and re-randomising its ciphertext. In this way, the CSP is unable to infer the search pattern from the access pattern, since the access patterns for all the queries are different. Note that the scheme is secure against file injection attacks, because the shuffling and re-randomisation operations make all the data untraceable whenever they are inserted.

To break the link between the size pattern and search pattern, when dummy data is introduced to hide the real size pattern, one possible solution is to ensure that the responses to all queries have a constant size. However, this potentially requires a huge number of dummy records (possibly exponentially many) if the database has large variability in its frequency information. A more practical solution is to vary the result size of each query. To do this, some of the dummy data should be deleted or updated, or some new dummy data should be inserted between any two queries. This ensures that the number of matched dummy records are different even if the same query is executed again. Alternatively, as mentioned in Section 2, if the database is divided into partitions and stored on multiple CSPs, the matched data together with a set of unmatched data in each partition should be re-randomised and moved across CSPs after executing each query. Due to the re-randomisation and re-location of the data, each CSP will only see a one-time match. That is, a CSP does not learn whether the data ever matched previous queries, or will match future queries.

In summary, to resist leakage-based attacks, a number of dummy records should be introduced and updated after executing each query, and the searched records should be re-randomised and shuffled after executing each query. All these operations affect the performance of the system. The fact is that there is always a trade-off between security and performance. It is impossible to achieve a higher level security without sacrificing performance. However, we aim to design a lightweight client for the user. It is impractical to ask the user to perform these operations. Basically, the dummy records increase the storage, bandwidth and computation overheads on the user end. From a security point of view, these operations should be hidden from the CSP. Otherwise, the CSP could learn more useful information and recover the data and queries. Inevitably, a third entity, or more entities should be involved to guarantee security and achieve efficiency. Specifically, the following two models can be considered:

- **Combining a Private Cloud with the CSP.** According to the latest report by Rightscale [1], the hybrid cloud computing approach is getting more popular among large enterprises. This model combines the public cloud service with a private cloud platform owned by the organisation. The private cloud could be considered as a trusted entity, because it is inherently managed by the organisation, where the sensitive data can be stored and executed without an extra layer of security. However, due to its limited storage and computational power, the bulk of the operations and storage should be delegated to the public CSP. To minimise information leakage, the private cloud could be leveraged to perform the shuffle, re-randomisation and dummy data refreshment operations after executing each query.
- **Combining Multiple CSPs.** The third entity could also be an untrusted public CSP. In fact, the idea of utilising multiple CSPs to reduce the load on users is already integrated into the ORAM technique. In [48], Stefanov and Shi have introduced a 2-cloud oblivious storage system that achieves APP and significantly reduced the bandwidth cost between the client and the CSP. Recently, Hoang *et al.* [24] also proposed a distributed encrypted data structure for SE schemes that could be deployed on two non-colluding CSPs. Their proposal achieves much higher security than traditional SE schemes. Unfortunately, both [48] and [24] suffer from the same problem as faced by traditional ORAM techniques. That is, an encrypted search operation is not considered and they can only protect the file access pattern. Moreover, in [48], the shuffle operation is performed before returning the data to users, which increases the latency on the user side.

  We could employ at least two non-colluding CSPs. However, we should also consider the search operation performed on index structures, and aim to achieve the index access pattern privacy. Specifically, one CSP stores the encrypted data and performs the search operation, and after executing each query, first it returns the result set to the user, and then sends the searched data to another CSP for shuffling, re-randomising and dummy data refreshing. In this case, the CSP that performs the queries never knows how the searched data is updated, and the CSP that performs rest of the operations cannot execute the query and never knows which of the records are matched. If the CSPs never collude together, all the patterns are protected from them. However, the cooperation between CSPs should be carefully designed, and the approaches to resist against the collusion between the CSPs should also be investigated.

To ensure a high level of security, it is possible to shuffle all the data in the database. However, this degrades the system performance. Although these operations do not affect the end-to-end latency from the user's point of view, the next query cannot be executed until the shuffle, re-randomisation, and dummy data refreshment operations have been finished. Hence, it affects the throughput of the system and should be completed efficiently. The more data is shuffled and re-randomised, the more difficult it is for the CSPs to infer the access pattern, but it is worse in terms of the system performance. It is an interesting research direction to investigate the required amount of data that should be shuffled and re-randomised for achieving a sustainable level of security and performance.

Forward and Backward Privacy. To achieve both forward and backward privacy, first of all, we should ensure the CSP cannot execute previous queries over newly added data or execute new queries over deleted data. Furthermore, to achieve SPP and APP, it is also necessary to ensure that the CSP cannot repeat the previous query after shuffling and re-randomising, or execute a new query over the snapshot of the data before shuffling and re-randomising. Therefore, not only the newly added data but also the shuffled and re-randomised data should include a new element (say a nonce) that should make them unmatched with the previous queries.

Likewise, a new query should include an element that makes it unmatched with stale (deleted or modified) data. However, the new queries must match with the latest data. To this end, the element included in the latest data should be stored somewhere and used to encrypt new queries, as done in [5, 6, 10, 47]. It is impractical to store the new element on the user side in multi-user settings. One possible solution is for a third entity to store and manage these query elements. In this case, all the queries should first be sent to the third entity and re-encrypted with the element included in the latest data. Moreover, the element should be updated when re-randomising the searched data. It is an open problem to achieve both forward and backward privacy in an efficient manner.

Index Structure. To achieve sub-linear search time, a number of works have investigated special index structures to narrow down the search range, such as the inverted index given in [14], the ORAM-like hierarchical structure designed in [47], the red-black tree based structure proposed by Kamara *et al.* [29], and the B-tree based scheme introduced in [26]. Unfortunately, the CSP could learn the search, access and size patterns from searching the index structure.

To improve search efficiency with minimised leakage, these index structures need to be redesigned. First, dummy data should be inserted into the structure to hide the size pattern. Second, both the encrypted nodes and queries should be semantically secure to hide the search pattern. Finally, to protect patterns and ensure forward and backward privacy, the searched nodes should be shuffled and re-randomised after executing each query.

However, it may be infeasible or inefficient to perform those operations on the proposed index structures. For instance, in the inverted index structure, encrypted linked lists are used to accelerate the searching. To hide the search and access patterns, the linked lists should be shuffled and re-randomised after executing each query. However, the shuffle operation will upset the linkability between nodes, and then the CSP cannot get the correct matched indexes. Therefore, one potential future work would be to investigate new sub-linear data structures that support equality check between semantically secure encrypted data and can be shuffled and re-randomised efficiently without leaking sensitive information.

Key Management. The existing proxy encryption based key management approaches, including [2, 3, 17, 31, 39, 51], could be used to ensure efficient user registration and revocation in multi-user settings. However, to ensure SPP, the query encryption should be replaced with a semantically secure primitive, and the equality check operation in these schemes should be changed accordingly. Moreover, as mentioned in [17], the third party could be leveraged to secure against collusion attacks between a user and the CSP. Meanwhile, an approach to avoid expensive pairing operations by making use of the third entity should be investigated.

Towards a more secure cloud database, there are many other security issues to be addressed, such as the accountability of the search result when the CSP is assumed to be totally untrusted and the access control for fine-grained access. Certainly, there is a long way to go to ensure confidentiality and privacy of the outsourced data.

5 Concluding Remarks

In this paper, we investigated the state of the art of SE schemes and some challenges for achieving a secure outsourced database. Almost all the existing SE schemes are vulnerable to inference attacks due to sensitive information leakage, which makes them unusable for privacy-sensitive applications. Based on these attacks, we identify a set of requirements for a cloud database that could be secure against them and ensure an efficient and practical user searching experience. We also briefly reviewed possible solutions to meet these requirements. To achieve a better balance between the security level and performance of the system, we finally outlined several future research directions. These directions will be developed in future work by the authors.

References

1. Rightscale 2016 state of the cloud report, `https://www.rightscale.com/lp/state-of-the-cloud`, last accessed: July 3, 2016
2. Asghar, M.R., Russello, G., Crispo, B., Ion, M.: Supporting complex queries and access policies for multi-user encrypted databases. In: Juels, A., Parno, B. (eds.) CCSW 2013. pp. 77–88. ACM (2013)
3. Bao, F., Deng, R.H., Ding, X., Yang, Y.: Private query on encrypted data in multi-user settings. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC 2008. Lecture Notes in Computer Science, vol. 4991, pp. 71–85. Springer (2008)
4. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: S&P 2007. pp. 321–334. IEEE Computer Society (2007)
5. Bost, R.: ∑οφος: Forward secure searchable encryption. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) SIGSAC 2016. pp. 1143–1154. ACM (2016)

6. Bost, R., Fouque, P., Pointcheval, D.: Verifiable dynamic symmetric searchable encryption: Optimality and forward security. IACR Cryptology ePrint Archive 2016, 62 (2016)
7. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. 25(1), 222–233 (2014)
8. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Ray, I., Li, N., Kruegel, C. (eds.) SIGSAC 2015. pp. 668–679. ACM (2015)
9. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Dynamic searchable encryption in very-large databases: Data structures and implementation. In: NDSS 2014. The Internet Society (2014)
10. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. Lecture Notes in Computer Science, vol. 3531, pp. 442–455 (2005)
11. Cheng, R., Yan, J., Guan, C., Zhang, F., Ren, K.: Verifiable searchable symmetric encryption from indistinguishability obfuscation. In: Bao, F., Miller, S., Zhou, J., Ahn, G. (eds.) ASIA CCS 2015. pp. 621–626. ACM (2015)
12. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)
13. Crescenzo, G.D., Cook, D.L., McIntosh, A., Panagos, E.: Practical private information retrieval from a time-varying, multi-attribute, and multiple-occurrence database. In: Atluri, V., Pernul, G. (eds.) DBSec 2014. Lecture Notes in Computer Science, vol. 8566, pp. 339–355. Springer (2014)
14. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) CCS 2006. pp. 79–88. ACM (2006)
15. Dautrich, J., Ravishankar, C.V.: Combining ORAM with PIR to minimize bandwidth costs. In: Park, J., Squicciarini, A.C. (eds.) CODASPY 2015. pp. 289–296. ACM (2015)
16. Devadas, S., van Dijk, M., Fletcher, C.W., Ren, L., Shi, E., Wichs, D.: Onion ORAM: A constant bandwidth blowup oblivious RAM. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A. Lecture Notes in Computer Science, vol. 9563, pp. 145–174. Springer (2016)
17. Dong, C., Russello, G., Dulay, N.: Shared and searchable encrypted data for untrusted servers. In: Atluri, V. (ed.) DBSec 2008. Lecture Notes in Computer Science, vol. 5094, pp. 127–143. Springer (2008)
18. Ferretti, L., Pierazzi, F., Colajanni, M., Marchetti, M.: Scalable architecture for multi-user encrypted SQL operations on cloud database services. IEEE Trans. Cloud Computing 2(4), 448–458 (2014)
19. Garg, S., Mohassel, P., Papamanthou, C.: TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. Lecture Notes in Computer Science, vol. 9816, pp. 563–592. Springer (2016)
20. Goh, E.: Secure indexes. IACR Cryptology ePrint Archive 2003, 216 (2003)
21. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM 43(3), 431–473 (1996)
22. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: Ahn, G., Yung, M., Li, N. (eds.) SIGSAC 2014. pp. 310–320. ACM (2014)
23. Hang, I., Kerschbaum, F., Damiani, E.: ENKI: access control for encrypted query processing. In: Sellis, T.K., Davidson, S.B., Ives, Z.G. (eds.) SIGMOD 2015. pp. 183–196. ACM (2015)
24. Hoang, T., Yavuz, A.A., Guajardo, J.: Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In: Schwab, S., Robertson, W.K., Balzarotti, D. (eds.) ACSAC 2016. pp. 302–313. ACM (2016)
25. Hwang, Y.H., Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. Lecture Notes in Computer Science, vol. 4575, pp. 2–22. Springer (2007)

26. Ishai, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. In: Sako, K. (ed.) CT-RSA 2016. Lecture Notes in Computer Science, vol. 9610, pp. 90–107. Springer (2016)
27. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS 2012. The Internet Society (2012)
28. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) SIGSAC 2013. pp. 875–888. ACM (2013)
29. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A. (ed.) FC 2013. Lecture Notes in Computer Science, vol. 7859, pp. 258–274. Springer (2013)
30. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) CCS 2012. pp. 965–976. ACM (2012)
31. Kiayias, A., Oksuz, O., Russell, A., Tang, Q., Wang, B.: Efficient encrypted keyword search for multi-user data sharing. In: Askoxylakis, I.G., Ioannidis, S., Katsikas, S.K., Meadows, C.A. (eds.) ESORICS 2016. Lecture Notes in Computer Science, vol. 9878, pp. 173–195. Springer (2016)
32. Liu, C., Zhu, L., Wang, M., Tan, Y.: Search pattern leakage in searchable encryption: Attacks and new construction. Inf. Sci. 265, 176–188 (2014)
33. Naveed, M.: The fallacy of composition of oblivious RAM and searchable encryption. IACR Cryptology ePrint Archive 2015, 668 (2015)
34. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Ray, I., Li, N., Kruegel, C. (eds.) SIGSAC 2015. pp. 644–655. ACM (2015)
35. Naveed, M., Prabhakaran, M., Gunter, C.A.: Dynamic searchable encryption via blind storage. In: SP 2014. pp. 639–654. IEEE Computer Society (2014)
36. Ostrovsky, R.: Efficient computation on oblivious RAMs. In: Ortiz, H. (ed.) STOC 1990. pp. 514–523. ACM (1990)
37. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999)
38. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Wobber, T., Druschel, P. (eds.) SOSP 2011. pp. 85–100. ACM (2011)
39. Popa, R.A., Zeldovich, N.: Multi-key searchable encryption. IACR Cryptology ePrint Archive 2013, 508 (2013)
40. Ren, L., Fletcher, C.W., Kwon, A., Stefanov, E., Shi, E., van Dijk, M., Devadas, S.: Constants count: Practical improvements to oblivious RAM. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 415–430. USENIX Association (2015)
41. Rizomiliotis, P., Gritzalis, S.: ORAM based forward privacy preserving dynamic searchable symmetric encryption schemes. In: Ray, I., Wang, X., Ren, K., Kerschbaum, F., Nita-Rotaru, C. (eds.) CCSW 2015. pp. 65–76. ACM (2015)
42. Rompay, C.V., Molva, R., Önen, M.: Multi-user searchable encryption in the cloud. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. Lecture Notes in Computer Science, vol. 9290, pp. 299–316. Springer (2015)
43. Samanthula, B.K., Jiang, W., Bertino, E.: Privacy-preserving complex query evaluation over semantically secure encrypted data. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014. Lecture Notes in Computer Science, vol. 8712, pp. 400–418. Springer (2014)
44. Sarfraz, M.I., Nabeel, M., Cao, J., Bertino, E.: Dbmask: Fine-grained access control on encrypted relational databases. In: Park, J., Squicciarini, A.C. (eds.) CODASPY 2015. pp. 1–11. ACM (2015)

45. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: S&P 2000. pp. 44–55. IEEE Computer Society (2000)
46. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) SIGSAC 2013. pp. 299–310. ACM (2013)
47. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS 2013. vol. 71, pp. 72–75 (2014)
48. Stefanov, E., Shi, E.: Multi-cloud oblivious storage. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) SIGSAC 2013. pp. 247–258. ACM (2013)
49. Sun, W., Liu, X., Lou, W., Hou, Y.T., Li, H.: Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In: INFOCOM 2015. pp. 2110–2118. IEEE (2015)
50. Sun, W., Yu, S., Lou, W., Hou, Y.T., Li, H.: Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In: INFOCOM 2014. pp. 226–234. IEEE (2014)
51. Tang, Q.: Nothing is for free: Security in searching shared and encrypted data. IEEE Trans. Information Forensics and Security 9(11), 1943–1952 (2014)
52. Wang, B., Song, W., Lou, W., Hou, Y.T.: Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. In: INFOCOM 2015. pp. 2092–2100. IEEE (2015)
53. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: INFOCOM 2014. pp. 2112–2120. IEEE (2014)
54. Wang, B., Hou, Y., Li, M., Wang, H., Li, H.: Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index. In: Moriai, S., Jaeger, T., Sakurai, K. (eds.) ASIA CCS 2014. pp. 111–122. ACM (2014)
55. Williams, P., Sion, R.: Usable PIR. In: NDSS 2008. The Internet Society (2008)
56. Yang, Y., Liu, J.K., Liang, K., Choo, K.R., Zhou, J.: Extended proxy-assisted approach: Achieving revocable fine-grained encryption of cloud data. In: Pernul, G., Ryan, P.Y.A., Weippl, E.R. (eds.) ESORICS 2015. Lecture Notes in Computer Science, vol. 9327, pp. 146–166. Springer (2015)
57. Yavuz, A.A., Guajardo, J.: Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. Lecture Notes in Computer Science, vol. 9566, pp. 241–259. Springer (2015)
58. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: USENIX Security 2016. pp. 707–720. USENIX Association (2016)