Forward Secure Searchable Symmetric Encryption

Muhammad Saqib Niaz
Databases & Software Engineering
Otto von Guericke University
Magdeburg, Germany
saqib.niaz@ovgu.de

Gunter Saake
Databases & Software Engineering
Otto von Guericke University
Magdeburg, Germany
gunter.saake@ovgu.de

Abstract—**Data outsourcing to third party clouds poses numerous data security threats. Access by unauthorized users is one of the security threat to the outsourced data. Unauthorized access can be avoided by encrypting the data before outsourcing. However, encrypting data before outsourcing renders it unsearchable to the data owner. Searchable encryption schemes are developed to specifically target this problem. A dynamic searchable encryption is the one that allows the data owner to add or delete a file after data outsourcing. Dynamic searchable encryption schemes are vulnerable to two specific security threats that are not applicable to the static searchable encryption schemes namely forward privacy and backward privacy. Forward privacy requires that the addition of a file should not reveal the presence of a previously searched keyword. Backward privacy requires that a search should not return the file identifier of a previously deleted file. In this paper, we propose a dynamic searchable scheme that guarantees forward privacy. It only uses the symmetric key algorithms hence reducing the requirements for storage and processing power on the client side. Furthermore, our proposed scheme is space reclaiming. After the deletion of a file, the redundant data nodes are also deleted from the secure index in the subsequent searches. Because of this space reclaiming capability of the scheme, the scheme is also partially backward private.**

Keywords-component; Dynamic Searchable Symmetric Encryption; Dynamic Searchable Encryption; Searchable Symmetric Encryption; Forward Privacy; Backward Privacy; Space Reclaiming

I. INTRODUCTION

Data outsourcing to the third-party clouds has seen a tremendous growth both in the personal and business setting. Despite the fact that the use of the third-party cloud storage is growing incredibly, one cannot deny the security threats involved in using a third-party cloud storage. One of the security threats is the unauthorized access by the malicious users. If the user data is stored without any encryption, the cloud administrators can simply access the data. Furthermore, in the case of a hack, the hackers could also access the user's plaintext data. In order to curb this problem, one simple solution is to encrypt all the files and then store them on a third-party cloud. However, encrypting the files with the conventional encryption schemes renders the data unsearchable. In order to search in such a setting, the user would have to download all the encrypted files to the local storage, decrypt all the files and then run the search query. This is infeasible and impractical as it eliminates almost all the benefits of a cloud storage.

Searchable encryption (SE) gives a user the ability to run a search query without decrypting the file contents. Basic idea behind an SE is to extract the keywords from the plaintext files. Based on the extracted keywords, create an index. Encrypt the files along with the associated index and send both to the cloud storage. Later on, when the user wants to search a keyword, he creates a trapdoor based on the search keyword so that the server does not learn the keyword being searched. This trapdoor is then sent to the server; server searches the trapdoor in the secure index and returns the corresponding encrypted files.

Initially proposed SE schemes were static in nature. Once the secure index was generated, user was unable to add or delete a file without regenerating the whole index. Later on, dynamic searchable encryption (DSE) schemes are developed that gives the data owner the ability to add or delete a file after creation of the secure index. Update of a file in these schemes does not require the regeneration of the secure index. There are some established security criteria for SEs. These criteria are also applicable to DSEs. Furthermore, two new security criteria namely forward privacy and backward privacy are introduced that are only applicable to DSEs [1]. Forward privacy requires that the addition of a new file should not reveal the presence of an already searched keyword. Backward privacy requires that a search should not return any file identifier of a previously deleted file.

In this paper, we present a forward secure searchable symmetric encryption abbreviated as FSE. The basic idea of how to make a forward privacy scheme is inspired from the [2]'s scheme. The idea is to insert the new nodes in a way that the server is unable to relate the newly inserted nodes with any of the already searched keywords. Later on, during the search, the server should be able to retrieve all the files corresponding to the search keyword including the newly inserted nodes. Our scheme presents numerous improvements over [2]'s scheme. Our scheme does not use any public key algorithms both on the server and the client-side. As the scheme only uses the symmetric key cryptography therefore the client-side storage and processing requirements are minimized. Our proposed scheme is also space reclaiming. In case of a file deletion, the redundant data nodes are deleted from the secure index in the subsequent searches. The space reclaiming feature does not require regeneration of the whole secure index. Due to the space reclaiming feature, our scheme partially fulfills the criteria of backward privacy.

978-1-908320/93/3/$31.00©2017 IEEE

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on November 14,2020 at 08:20:03 UTC from IEEE Xplore. Restrictions apply.

II. Related Work

First searchable encryption scheme was introduced in 2000 [3]. Every word of each document was separately encrypted hence making the scheme vulnerable to various statistical attacks. Encrypting every word separately also makes the files incompressible.

Goh proposed a bloom filters based scheme [4] but it presented weaker security guarantees. By design, bloom filters produce false positives which is considered a weakness in a searchable encryption scheme. The author also presented the first security definitions for a searchable scheme.

Chang et al. proposed a scheme in 2005 [5]. This scheme required generation of one index per document in the document collection. The main drawback of the scheme was the search overhead on the server side. Search algorithm had to go through each index to search a single keyword.

Curtmola et al. proposed an inverted index based scheme [6]. The authors achieved a sub-linear searching efficiency by storing all the file identifiers related to a keyword in a linked list. The authors also presented new security definitions for SEs.

Van Liesdonk et al. proposed a searchable encryption scheme based on public key encryption [7]. It was a multi-writer and single-reader scheme. This scheme allowed wildcard searches over any alphabet.

Kamara et al. proposed a scheme in 2010 [8]. It was an improved version of [6] and it was a static scheme. Kamara et al. proposed a dynamic scheme in 2012 [9]. This scheme had a sub-linear search complexity. It leaked the hashes of the keywords contained in the updated documents. Kamara et al. proposed an improvement in 2013 [10] resolving the hash leaking problem at the cost of extra storage at server-side.

Schemes proposed by Goh [4] and Van Liesdonk et al. [7] were dynamic in nature but Kamara et al. were the first one to propose a DSE with sub-linear search complexity [9].

Cash et al. introduced a scheme that was capable of running boolean queries on outsourced data [11]. Later on, they extended this work and proposed a DSE claiming to handle very large databases [12].

Yavuz et al. presented a DSE with efficient update complexity [13]. However, the search speed of the scheme came down to linear from sub-linear. Search algorithm has to process nodes equal to the total number of documents in the file database instead of only processing the nodes equal to the frequency of the search keyword.

Islam et al. [14] and Cash et al. [15] presented new attacks that become possible due to the leakage in SEs.

Stefanov et al. presented new security definitions for DSEs [1]. The authors also proposed a scheme that was forward privacy secure but the scheme suffered from computational and storage inefficiencies.

In 2016, Bost presented a scheme [2] that was also forward privacy secure. Bost's scheme offered some improvements over the [1]'s scheme.

III. Security Criteria For DSEs

Goh was the first one to introduce the security criteria for a searchable encryption scheme in 2003 [4]. According to his security definition, an adversary should not be able to deduce the document contents from the secure index. However, the security criteria do not require any security for the search trapdoors. The security definition presented two security criteria i.e. IND1-CKA and IND2-CKA. IND1-CKA requires that the indices of the equal sized documents should be indistinguishable to an adversary. IND2-CKA requires that the indices of the unequal sized documents should be indistinguishable to an adversary. These security criteria specifically deal with the schemes that produced one index per document.

Curtmola et al. proposed the new and improved security definitions for the searchable encryption schemes in 2006 [6]. The security definitions require that the remotely stored index and files should not leak any information to an adversary. Additionally, these definitions also require the security of the trapdoors. They presented two security criteria i.e. IND-CKA1 and IND-CKA2. INK-CKA1 requires the security against a non-adaptive attacker. Non-adaptive adversary is the one who generates all the queries at the beginning of the protocol. IND-CKA2 requires the security against an adaptive attacker. An adaptive adversary is the one who generates the subsequent queries dynamically based on the results of the previous queries.

Shi et al. proposed the two new security criteria especially for dynamic searchable encryption schemes in 2014 [1]. These security criteria are named as forward privacy and backward privacy. Forward privacy requires that the addition of a new file should not reveal the presence of an already searched keyword. Backward privacy requires that after a deletion operation, a search query should not return the file identifiers of the already deleted files.

IV. FSE Construction

FSE is a forward secure dynamic searchable symmetric encryption scheme. As the name describes it, our scheme is a forward secure scheme. In case of a file addition operation, the server is unable to link the nodes of the newly inserted file with any of the already searched keywords. It is also a space reclaiming scheme. After a file deletion operation, the redundant nodes are removed from the secure index in the subsequent searches. Due to the space reclaiming capability of the FSE, it is also a partially backward private scheme. Search complexity of the FSE is sub-linear. Sub-linear search means that the number of nodes processed by the server search algorithm is equal to the frequency of the keyword being searched. Addition and deletion operation for a file is executed in a single step on server side. FSE only uses the symmetric key cryptosystem and a keyed hash function. No public key crypto-algorithm is used at any stage on the client or the server side.

A. General Ideas

Generally, in the searchable encryption schemes, the nodes related to one keyword are linked. In search operation, the server receives a trapdoor and its associated decryption key from the client. The search algorithm at the server-side goes to the first node, decrypts it, gets the address of the next node and a file

978-1-908320/93/3/$31.00©2017 IEEE

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on November 14,2020 at 08:20:03 UTC from IEEE Xplore. Restrictions apply.

identifier. The search algorithm keeps on decrypting and collecting the file identifiers until the end of the list is reached. After completion of the search algorithm, the server knows all the nodes/file identifiers related to a trapdoor. Now, if the client performs a file addition operation. Client sends the encrypted file along with its associated trapdoors to the server. Server inserts those nodes at specific locations as dictated by the client's addition algorithm. At this point, if the server is able to relate the new nodes with the previously searched keyword/trapdoor lists then the scheme is not forward secure. In contrast, if server is unable to relate the newly added nodes to any of the already searched keyword/trapdoor lists then the scheme is forward secure.

Our proposed scheme is a forward privacy secure searchable encryption scheme. The technique is to generate the addition token in a way that the server cannot relate the newly added nodes to the list of nodes of an already searched keyword/trapdoor. But at a later time when that keyword is searched again, the server gets a new trapdoor and a new decryption key for the same keyword and at that point server is able to get all the related file identifiers to the searched keyword.

In FSE, the addition token contains a new node and an address. The server addition algorithm simply stores the new node data at the associated address. The node data contains the file identifier of the newly added file, the encryption key to the next node in the list and the address of the next node, here the next node refers to the first node for the keyword in question. Every new node is inserted before the first node of the link list for that specific keyword. The contents of the node data are encrypted with a newly generated secret key that stays private at the client-side. At the time of the search, the client retrieves the address and the secret key associated with the search keyword. Client sends this address and encryption key to the server, server goes to the address and decrypts the node to get the file identifier and from the node data server also gets the address and the decryption key of the next node in the list. In this way server navigates through the whole list. But after the addition and before a search, it is not possible for the server to know the linkage of the newly added node to any of the old lists of nodes for a keyword/trapdoor.

B. Preliminaries

File identifiers and the keywords are considered to be strings. Let λ be the security parameter. Poly(λ) and negl(λ) denote the polynomial and the negligible function in λ. The set of all binary strings of length λ is denoted as $\{0,1\}^{\lambda}$, and the set of all finite binary strings as $\{0,1\}^{*}$. x||y refers to the concatenation of the two strings x and y. $x \xleftarrow{\$} X$ means that the x is uniformly sampled from a finite set X. A *database* $DB = (ind_i, W_i)_{i=1}^{D}$ is a set of file-identifier/keyword pair with $ind_i \in \{0,1\}^{\lambda}$ and $W_i \subseteq \{0,1\}^{l}$. The set of keywords in the whole database can be represented by $W = \cup_{i=1}^{D} W_i$.

Our searchable encryption scheme is comprised of following algorithms:

- Setup is an algorithm that takes the parameter λ. It outputs a secret key of length λ, an encrypted database EDB and σ the client's state.
- AddFile is an algorithm that runs on the client-side. It takes as input the client state σ, the client secret key K_s and the new file to be inserted in the database. It updates the client state and outputs the encrypted file and the corresponding addition tokens.
- AddToken is an algorithm that runs on the server-side. It takes input the EDB, the encrypted file and the addition tokens. It outputs the updated EDB.
- DeleteFile is an algorithm that runs on the client-side. It takes as input the file identifier of the file to be deleted. It outputs a deletion token.
- DelToken is an algorithm that runs on the server-side. It takes input the EDB and the deletion token. Deletes the file and outputs the updated EDB.
- Trapdoor is an algorithm that runs on the client-side. It takes the keyword w and the client state σ as input. If the keyword exists in the client state, it outputs the address and the encryption key for the keyword.
- Search is an algorithm that runs on the server-side. It takes the encrypted database EDB and the trapdoor (address & encryption) for the search keyword as the input. It outputs the file identifiers of the files matching the trapdoor's criteria.

C. Basic Construction

Basic construction of FSE is forward private secure but it is not space reclaiming. After a file deletion operation, the extra node data corresponding to deleted file stays in the secure index.

Setup
$K_s \xleftarrow{\$} \{0,1\}^{\lambda}$
$W, T \leftarrow empty\ map$
$return\ (K_s, T, W)$
$T\ for\ server, K_s\ \&\ W\ for\ client$

1) Setup

Setup of the FSE is straightforward. A secret key is generated. This key always stays at the client-side. This key is used for encryption/decryption of the files. Two empty maps are generated. T can be any data structure that provides a functionality of mapping a key to a value. A key is associated with every node in the secure index. We call this key the 'address'. Each node in this T is inserted at a specific address. Later on, in search algorithm, each node is accessed using its corresponding address. Client-side map W is a structure that stores data based on the keywords. Each element of this map can be accessed using a keyword. For each keyword, two values are stored in the W array, first is the address (in the T map on server-side) of the first node containing this keyword and second is the corresponding decryption key.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on November 14,2020 at 08:20:03 UTC from IEEE Xplore. Restrictions apply.

AddFile:
Extract keywords from the file
Encrypt the file with K_s
Do following for each keyword:
$Generate\ K_w \xleftarrow{\$} \{0,1\}^{\lambda}$
$Generate\ address\ Add_w \leftarrow H(K_s, F_{identifier}||W)$
Search w in W array
If value found:
Create node containing $F_{identifier}$, Add_{w-1}, K_{w-1}
endif
If value not found
Create node containing $F_{identifier}$, null, null
endif
Put K_w and Add_w in W array at location w
Encrypt the node data with Kw
Output the encrypted node data and Add_w

2) AddFile

AddFile algorithm extracts the keywords from the file being added. It encrypts the file with client-side secret key. For each extracted keyword, it checks the existence of that keyword in client-side W array. If the keyword does not already exist in client state. It generates a random key for the keyword. It generates an address using a keyed hash function based on the file identifier and the keyword. The algorithm creates a data node with an empty next node pointer and encrypts this node data with the newly generated random key. It creates a data node for the client state containing the address for the server-side map and the encryption key for the node data. If the keyword already exists in the client state, then the AddFile algorithm creates the new node for an already existing keyword in such a way that server is unable to link the newly added data nodes with any of the already searched lists of nodes. AddFile algorithm gets the address and the encryption key of the previously stored first node of the list. It creates a new node with the new file identifier, puts the address and the encryption key of the already existing first node. Encrypts this node with a new randomly generated key. Keeps the new secret key with client and sends the encrypted node data along with the newly calculated address where server is supposed to store this data. In this way, link is created but the link information is kept secret from the server. Server is unable to link this newly generated node with any of the already existing lists of the searched keywords.

AddToken:
Goto location Add_w and store encrypted node data

3) AddToken

AddToken algorithm runs on the server-side. It receives an encrypted file and an address-node pair for each keyword contained in that file. Server stores the encrypted file in file storage. For each address-node pair, AddToken algorithm goes to the specific address and stores the node data on this location.

DeleteFile:
Outputs $F_{identifier}$ (to be sent to server)

DelToken:
Server deletes the corresponding file to $F_{identifier}$

4) DeleteFile & DelToken

File deletion is straight forward. Client sends the file identifier of the file to the server. Server simply deletes the encrypted file from the file storage. This does not affect the secure index.

$Trapdoor$:
$K_w, Add_w \leftarrow W[w]$
$if\ (K_w, Add_w)\ is\ null$
$return$
$Output\ K_w\ and\ Add_w\ (to\ be\ sent\ to\ server)$

5) Trapdoor

Trapdoor algorithm runs on the client side. It takes the search keyword as input. Searches the client-side W array, if the keyword is found it outputs the address and the corresponding decryption key.

$Search$:
$goto\ Add_w$
decrypt data with K_w
add $F_{identifier}$ to $list_{ident}$
$Add_{next} \leftarrow Add_{w-1}$
$Key_{next} \leftarrow K_{w-1}$
$while(Add_{next} \neq null)$
$goto\ Add_{next}$
$decrypt\ data\ node\ with\ Knext$
$add\ F_{identifier}\ to\ list_{ident}$
$Add_{next} \leftarrow Add_{w-1}$
$Key_{next} \leftarrow K_{w-1}$
$end\ while$
$Output\ files\ corresponding\ to\ list_{ident}$

6) Search

Search algorithm in the basic construction is simple as it does not care about the deleted files. That's why the basic construction is not space reclaiming. Search algorithm runs on server-side, it receives the address and the secret key for the search keyword. The search algorithm simply goes to the address and decrypts the data with the secret key. Upon every decryption, the server gets a file identifier, an address of the next node and the secret key for the next node. At the end of the algorithm a list of identifiers of the files is generated that contains the keyword being searched. Server sends back all the existing files corresponding to that list of identifiers.

D. Space reclaiming construction

This is an enhanced version of the basic construction. The only difference between the two constructions is the search algorithm. If a file is deleted from the server, the index stays unchanged. Later on, when a keyword is searched for which a file has already been deleted, server removes the deleted nodes from the secure index. It does not matter how many nodes for a

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on November 14,2020 at 08:20:03 UTC from IEEE Xplore. Restrictions apply.

specific keyword are deleted. On the first subsequent search after the deletions, all the extra nodes are removed from the server index.

```
Trapdoor:
K_w, Add_w ← W[w]
if (K_w, Add_w) is null
  return
Output K_w and Add_w
```

1) Trapdoor

Trapdoor algorithm of enhanced version is identical to the basic version. It takes a keyword and if the keyword exists in the secure index, it outputs the address and the secret key for the first node.

```
Search:
create empty list_ident
goto Add_w
decrypt with K_w
extract F_ident−w
check file existence for F_ident−w
if F_ident−w doesn't exist
  keep decrypting next nodes until valid F_ident is found
  encrypt all value of this node with K_w
  store this newly encrypted node at Add_w
  F_ident−w ← F_ident
  if no node with valid F_ident is found
    delete first node from Addw
    return null to client
  endif
end
add F_ident−w to list_ident
goto next address and start decrypting node data
if file exists add to list_ident
if encounters a non − existent file
  keep decrypting nodes until a valid node found
  update last node with valid identifier
  only update next node address and encrypt again
  if no node found with valid F_ident
    update last node with valid F_ident
      replace next node address with null
    encrypt again with node key
  endif
endif
return files from list_ident
```

2) Search

In the space reclaiming version, the search algorithm at the server-side starts decrypting the linked nodes and while decrypting the nodes and collecting the file identifiers, it also checks the existence of the corresponding files. If the server gets such a file identifier from a decrypted node for which the corresponding file does not exist, the search algorithm deletes that node from the list and reconnects the list again. This search and reclaim space algorithm can encounter three kind of file deletion scenarios. First possibility is that the first file in the list is deleted. In this case, the server would have to find such a node further in the list for which the corresponding file is still existing in the server file database. Search algorithm would take all the data from this valid node including the next node address and its corresponding encryption key. It would encrypt this node's data with the key that server has received from the client for decryption of first node in the list. It would store this newly encrypted node on the address that it has received from the client i.e. address of the first node as stored in the client's state. By doing so, no change is required on the client side. The initial address and the corresponding encryption key already stored at the client side remains valid even if the first node in the list is deleted. Second possible scenario is that the node stored at the initial address sent by the client has an existing file in the file database but some other files from the list are missing. In this case, the search algorithm would delete the nodes for which the corresponding files are deleted and would relink the list. It would update the last node in the list with a valid file identifier and would have to replace its next node address and its corresponding encryption key with the address and key of the next node in the list that contains a valid file identifier. The search algorithm already has the secret key for each node that makes it possible for it to update the node data and re-encrypt it again. Third possibility is a less likely scenario in which case all the files corresponding to a keyword are deleted. In this case, the server would delete all nodes from the list and would send a null response to the client.

```
Search (Client End):
Client gets the encrypted files
if client gets a null response
  client deletes the corresponding data from W array
endif
```

3) Search (Client-side)

Upon receiving a null response from the server, the client would have to update its state. Client would remove the data stored in its W array corresponding to the search keyword.

E. Storage

Storage on the client side is proportionate to the number of keywords. For each keyword, one address and one encryption key are stored. An additional secret key is also stored on the client side. This key is used to encrypt the files. It is also used in the keyed hash function for the address generation for a new keyword & file-identifier combination.

On server side, one encrypted data node is stored for each keyword & file-identifier combination. Each node contains three elements, a file-identifier, an address of the next node and an encryption key of the next node.

The basic version does not remove the extra nodes for the deleted files from the secure index. In space reclaiming version, the extra nodes are removed from the secure index. Removal of extra nodes is not done at the deletion of files. Extra nodes are removed from the secure index in subsequent searches.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on November 14,2020 at 08:20:03 UTC from IEEE Xplore. Restrictions apply.

F. Perfomance

Our proposed scheme is a non-interactive scheme, as all the operations are performed in one round.

Our scheme only uses symmetric key algorithms and a keyed hash function at the client side. On the server side, only symmetric key algorithm is used in the search algorithm. By avoiding the public key cryptosystem especially at the client side, the scheme is easily deployable on low-resourced (both in processing and memory) mobile devices.

Setup is simple and straight-forward, it does not require any cryptographic calculations.

Addition of a new file requires extraction of the keywords at the client side. One keyed hash and one encryption for each keyword is done. On the sever side, the addition algorithm works in O(1). It requires storing of the nodes' data at specific addresses.

Deletion of a file is straight forward. It does not require any processing on the client side. Client just sends the file identifier to the server for deletion. On the server side, the deletion also works in O(1). Server just deletes the corresponding file from the file storage. No change is required in the secure index.

Search is sub-linear, the search algorithm on the server-side only needs to process the nodes equal to the frequency of the search keyword. But there could be an exception to this rule, if a file has been deleted and then a search runs first time after the deletion then the search algorithm would have to process the one extra node along with the valid nodes. But after this search algorithm execution, the extra node for the deleted file is removed from the secure index. Later on, in all subsequent searches, the search algorithm would run in the sub-linear time complexity.

As our search algorithm removes the deleted nodes during the search that's why our scheme does not require to maintain a separate deleted file index. Maintaining a separate deletion index also entails regeneration of the whole index at regular intervals to merge the deletion index into the main index.

G. Security

Our scheme is forward secure and partially backward secure. Forward security is achieved by generation of the insertion nodes in a way that the server is unable to link the newly inserted nodes with any of the already searched keywords. The scheme is partially backward secure as the search of a keyword after a deletion gives out the deleted file identifier but during this search, server deletes the extra nodes and relinks the list again. In all subsequent searches, the search algorithm does not get any deleted file identifiers. Our scheme is indistinguishable against the chosen keyword attack i.e. IND1-CKA & IND2-CKA as outlined in [4]. Our scheme is also indistinguishable against non-adaptive attackers i.e. IND-CKA1 as outlined in [6].

V. Conclusions

FSE fulfills the criteria of the forward privacy. The scheme is space reclaiming in case of file deletions. Space reclaiming feature makes it partially backward secure. Search is sub-linear. Addition and deletion of a file requires a single step execution on the server side. Our scheme does not use any public key cryptosystem algorithms thus making it feasible to be used in the resource restrained environments.

VI. Future work

Security of the scheme needs to be outlined in detail. Detailed analysis is required to check whether the proposed scheme is indistinguishable against an adaptive attacker i.e. IND-CKA2 as proposed in [6]. Implementation of the scheme is also required to check the practicality of the design in different deployment scenarios.

References

[1] E. Stefanov, C. Papamanthou and E. Shi, "Practical Dynamic Searchable Encryption with Small Leakage," in *21st Annual Network and Distributed System Security Symposium*, San Diego, 2014.

[2] R. Bost, "Forward Secure Searchable Encryption," in *Conference on Computer and Communications Security*, Vienna, 2016.

[3] D. X. Song, D. Wagner and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, 2000.

[4] E.-J. Goh, "Secure Indexes," *IACR Cryptology ePrint Archive,* vol. 2003, p. 216, 2003.

[5] Y.-C. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data," in *Applied Cryptography and Network Security*, New York, 2005.

[6] R. Curtmola, J. Garay, S. Kamara and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," in *13th ACM Conference on Computer and Communications Security*, 2006.

[7] P. Van Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel and W. Jonker, "Computationally Efficient Searchable Symmetric Encryption," *Secure data management,* vol. 6358, pp. 87-100, 2010.

[8] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *International Conference on the Theory and Application of Cryptology and Information Security*, Berlin, Heidelberg, 2010.

[9] S. Kamara, C. Papamanthou and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[10] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *International Conference on Financial Cryptography and Data Security*, Berlin, Heidelberg, 2013.

[11] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in cryptology--CRYPTO 2013*, Berlin, Heidelberg, Springer, 2013, pp. 353-37.

[12] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu and M. Steiner, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation," in *21st Annual Network and Distributed System Security Symposium--NDSS 2014*, 2014.

[13] A. A. Yavuz and J. Guajardo, "Dynamic Searchable Symmetric Encryption with Minimal Leakage and Efficient Updates on Commodity Hardware," in *International Conference on Selected Areas in Cryptography*, 2015.

[14] M. S. Islam, M. Kuzu and M. Kantarcioglu, "Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation," in *Ndss*, 2012.

[15] D. Cash, P. Grubbs, J. Perry and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

Authorized licensed use limited to: BEIJING UNIVERSITY OF POST AND TELECOM. Downloaded on November 14,2020 at 08:20:03 UTC from IEEE Xplore. Restrictions apply.