Multi-Client Symmetric Searchable Encryption with Forward Privacy*

Alexandros Bakas and Antonis Michalas

Tampere University,
Tampere, Finland
{alexandros.bakas,antonios.michalas}@tuni.fi

Abstract. Symmetric Searchable encryption (SSE) is an encryption technique that allows users to search directly on their outsourced encrypted data, in a way that the privacy of both the files and the search queries is preserved. Naturally, with every search query, some information is leaked. The leakage becomes even bigger when the scheme is dynamic (i.e. supports file insertions and deletions). To deal with this problem we design a forward private dynamic SSE scheme where file insertions do not leak any information about previous queries. Moreover, our construction supports the multi-client model, in the sense that every user that holds the secret key can perform search queries. Finally, our scheme also focuses on the problem of synchronization by utilizing the functionality offered by Intel SGX.

Keywords: Cloud Security · Forward Privacy · Multi-Client · Symmetric Searchable Encryption

1 Contribution

We extend the work proposed in [5] by constructing a Dynamic Symmetric Searchable Encryption (DSSE) scheme [4] that supports a multi-client model while at the same time, we preserve all the key properties of the original scheme. In particular, our construction:

- Provides **forward privacy** The Cloud Service Provider (CSP) maintains a dictionary Dict that provides a mapping of the extracted keywords to the corresponding file names ($id(f)$). Each entry/address in Dict is calculated with the help of a key $\mathsf{K_w}$ that is revoked after every search for a keyword w. This requires the computation of new addresses for all the affected rows (i.e. addresses that correspond to keyword w). To compute the new addresses, a fresh key $\mathsf{K_w}'$ is needed. This key along with the updated addresses are generated locally by the user who is then send the new addresses to the CSP. This way the CSP cannot know if a file that will be added later, contains

* This work was funded by the ASCLEPIOS: Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare Project No. 826093 EU research project.

a keyword w that a user searched for in the past (i.e. in a previous search). Hence the scheme satisfies the property of forward privacy.
- Is asymptotically optimal. The update cost is $O(m)$ and the search time is $O(\ell)$, where m is the number of unique keywords in a file and ℓ is the number of the resulted files.
- Is parallelizable. The CSP stores the file names ($id(f)$) in a dictionary. The address of each $id(f_i)$ is calculated by the data owner, before inserting the files, and is then hashed. This results to $O(\ell)$ independent hashes for each search. Hence, if the load is distributed to p processors, we achieve optimal search cost $O(\ell/p)$. Similarly, the update cost will be $O(m/p)$.

2 Cryptographic Primitives

In this section, we introduce our notation, and we provide a formal definition of a dynamic SSE along with the security definitions.

2.1 Notation

Given a set $\mathcal{X}$, we use $x \leftarrow \mathcal{X}$ to show that x is sampled uniformly from $\mathcal{X}$ and $x \xleftarrow{\$} \mathcal{X}$ if x is sampled uniformly at random. $|\mathcal{X}|$ denotes the cardinality of $\mathcal{X}$. Given two strings x and y, we use $x||y$ to denote the concatenation of x and y. If $\mathcal{X}, \mathcal{Y}$ are two sets, then we denote by $[\mathcal{X}, \mathcal{Y}]$ all the functions from $\mathcal{X}$ to $\mathcal{Y}$ and by $[\overline{\mathcal{X}, \mathcal{Y}}]$ all the injective functions from $\mathcal{X}$ to $\mathcal{Y}$. We use $R(\cdot)$ for a truly random function and R^{-1} for the inverse function of R. A function $negl(\cdot)$ is called negligible if it grows slower than any inverse polynomial. If $s(n)$ is a string of length n, we denote by $\overline{s}(l)$ its prefix of length l and by $\underline{s}(l)$, its suffix of length l, where $l < n$.

Definition 1 (Pseudorandom Function (PRF)). *Let F be a function such that $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{K}$ is the key-space, $\mathcal{X}$ is the domain of definition and $\mathcal{Y}$ is the range. Moreover, let $\mathsf{F.Gen}(1^\lambda)$ be an algorithm that given the security parameter λ, outputs $k \in \mathcal{K}$. F is a PRF if for all PPT adversaries $\mathcal{A}$:*

$$\begin{aligned} \big| Pr[k \leftarrow \mathsf{F.Gen}(1^\lambda) : \mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1] \\ - Pr[k' \xleftarrow{\$} [\mathcal{X}, \mathcal{Y}] : \mathcal{A}^{R(\cdot)}(1^\lambda) = 1 \big| = negl(\lambda) \end{aligned} \tag{1}$$

Definition 2 (Invertible Pseudorandom Function(IPRF)). *An IRPF with key-space $\mathcal{K}$, domain of definition $\mathcal{X}$ and range $\mathcal{Y}$ consists of two functions $F : (\mathcal{K} \times \mathcal{X}) \rightarrow \mathcal{Y}$ and $F^{-1} : (\mathcal{K} \times \mathcal{Y}) \rightarrow \mathcal{X} \cup \{\perp\}$. Moreover, let $\mathsf{F.Gen}(1^\lambda)$ be an algorithm that given the security parameter λ, outputs $k \in \mathcal{K}$. The functions F and F^{-1} satisfy the following properties:*

1. *$F^{-1}(k, F(k, x)) = x, \forall x \in \mathcal{X}$.*
2. *$F^{-1}(k, y) = \perp$ if y is not an image of F.*
3. *F and F^{-1} can be efficiently computed by deterministic polynomial algorithms.*

4. $F(k,\cdot), F^{-1}(k,\cdot)$ *are injective functions from* $\mathcal{X}$ *to* $\mathcal{Y}$.

An IPRF $F : (\mathcal{K} \times \mathcal{X}) \to \mathcal{Y}$ *is secure if* $\forall$ *PPT adversary* $\mathcal{A}$*:*

$$\begin{aligned}\big|Pr[k \leftarrow \mathsf{F.Gen}(1^\lambda) : \mathcal{A}^{F(k,\cdot),F^{-1}(k,\cdot)}(1^\lambda) = 1] - \\ Pr[k' \xleftarrow{\$} [\overline{\mathcal{X},\mathcal{Y}}] : \mathcal{A}^{R(\cdot),R^{-1}(\cdot)}(1^\lambda) = 1\big| = negl(\lambda)\end{aligned} \tag{2}$$

2.2 Dynamic Symmetric Searchable Encryption

Our construction enables a data owner to share her files with multiple users. However, only the data owner can add and delete files from her collection. We proceed with the definition of a dynamic symmetric searchable encryption scheme (DSSE).

Definition 3 (DSSE). *A dynamic Symemtric Searchable Encryption scheme consists of the following PPT algorithms:*

- $\mathsf{K} \leftarrow \mathsf{KeyGen}(1^\lambda)(1^\lambda)$*: The Data Owner generates a secret key* K *that consists of a key* $\mathsf{K_F}$ *for and IRPF* F *and a key* $\mathsf{K_{SKE}}$ *for a IND-CPA secure symmetric key cryptosystem SKE.*
- $(\mathsf{In_{CSP}}, C)(\mathsf{In_L}) \leftarrow \mathsf{InGen}(\mathsf{K}, F)$*: The data owners runs this algorithm to generate the CSP index* $\mathsf{In_{CSP}}$ *and a collection of ciphertexts* C *that will be sent to the CSP. Moreover, the index* $\mathsf{In_L}$ *is generated, that is stored both locally but is also outsourced to a trusted authority TA.*
- $(\mathsf{In'_{CSP}}, C')(\mathsf{In'_L}) \leftarrow \mathsf{AddFile}(\mathsf{K}, f, \mathsf{In_L})(\mathsf{In_{CSP}}, C)$*: The data owner is running this algorithm to add a file to her collection of ciphertexts. All the indexes and the collection of ciphertexts are updated.*
- $(\mathsf{In'_{CSP}}, C_w)(\mathsf{In'_L}) \leftarrow \mathsf{Search}(\mathsf{K}, w, \mathsf{In_L})(\mathsf{In_{CSP}}, C)$*. This algorithm is executed by a user in order to search for all files* f *containing a specific keyword* w*. The indexes are updated and the CSP also returns to the user the ciphertexts of the files that contain* w*.*
- $(\mathsf{In'_{CSP}}, C')(\mathsf{In'_L}) \leftarrow \mathsf{Delete}(\mathsf{K}, id(f_i), \mathsf{In_L})(\mathsf{In_{CSP}}, C)$*: The data owner runs this algorithm to delete a file from the collection. All the indexes are then updated accordingly.*

The KeyGen and InGen algorithms do not require any interaction. However, the rest of the algorithms require synchronization between the different entities since $\mathsf{In_L}$ is stored both on the owner's side and on TA.

2.3 Security Definitions

Definition 4 (Search Pattern). *The Search Pattern is a mapping between queries and keywords. This mapping is used to tell whether two or more queries were for the same keyword.*

Definition 5 (Access Pattern). *The Access Pattern is defined to be the outcome of each search query.*

Definition 6 (Forward Privacy). *A DSSE scheme satisfies Forward Privacy if for all file insertions after the successful execution of* InGen*, the leakage is limited to the id of the file, its size and the number of unique keywords contained in it.*

Definition 7 (Leakage Functions). *Let* $\mathcal{L}_{\mathsf{InGen}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Delete}}$ *be the leakage functions associated with index creation, file addition, and the search and delete operations.*

- $\mathcal{L}_{\mathsf{InGen}}(F) = (N, n, id(f_i), |f_i|), \forall f_i \in F$*. This function leaks the total size* N *of all the (keyword,* $id(f_i)$*) mappings, as well as the number of files, their id's and their sizes*
- $\mathcal{L}_{\mathsf{Add}}(f) = (id(f), |f|, \#w_i)$*: This function leaks the file id, its size and the number of unique keywords contained in it.*
- $\mathcal{L}_{\mathsf{Search}}(w) = \{$*Access Pattern, Search Pattern*$\}$*: This function leaks the Access and Search Patterns.*
- $\mathcal{L}_{\mathsf{Delete}}(id(f)) = (id(f), \#w_i, l)$*: This function leaks the file id, the number of unique keywords contained in* f *and the number of keywords* j *that will receive and updated address* $\mathrm{addr_w}$ *and value* val_{w_i}*.*

Definition 8 (DSSE **Security).** *Let* $\mathsf{DSSE} = (\mathsf{KeyGen}, \mathsf{InGen}, \mathsf{Add}, \mathsf{Search}, \mathsf{Delete}, \mathsf{Modify})$ *be a dynamic symmetric searchable encryption scheme. Moreover, let* $\mathcal{L}_{\mathsf{InGen}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Delete}}$ *be the leakage functions. We consider the following experiments between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{ADV}$*:*

Real$_{\mathcal{ADV}}(\lambda)$

$\mathcal{ADV}$ outputs a set of files F. $\mathcal{C}$ runs KeyGen *to generate a key* K*, and runs* InGen*. $\mathcal{ADV}$ then makes a polynomial time of adaptive queries $q = \{w, f_1, f_2\}$ such that w is contained in a file $f \in F$, $f_1 \notin F$ and $f_2 \in F$. For each q, she receives back either a search token for w, $\tau_s(w)$, an add token, τ_α, and a ciphertext for f_1 or a delete token τ_d for f_2. Finally, $\mathcal{ADV}$ outputs a bit b.*

Ideal$_{\mathcal{ADV}}(\lambda)$

$\mathcal{ADV}$ outputs a set of files F. $\mathcal{S}$ gets $\mathcal{L}_{\mathsf{InGen}}(F)$ to simulate InGen*. $\mathcal{ADV}$ then makes a polynomial time of adaptive queries $q = \{w, f_1, f_2\}$ such that w is contained in a file $f \in F$, $f_1 \notin F$ and $f_2 \in F$. For each q, $\mathcal{S}$ is given either $\mathcal{L}_{\mathsf{Search}}(w), \mathcal{L}_{\mathsf{Add}}(f_1)$ or $\mathcal{L}_{\mathsf{Delete}}((f_2))$. $\mathcal{S}$ then simulates the tokens and, in the case of addition, a ciphertext. Finally, $\mathcal{ADV}$ outputs a bit b.*

We say that the DSSE scheme is secure in the semi-honest model if for all PPT adversaries $\mathcal{ADV}$ there exists a probabilistic simulator $\mathcal{S}$ such that:

$$|Pr[(Real) = 1] - Pr[(Ideal) = 1]| \leq negl(\lambda)$$

3 Architecture

In this section, we introduce the system model by describing the entities that participate in our construction.

Users: A data owner generates three different indexes. At first, she creates $\mathsf{No.Files[w]}$ which contains a hash of each keyword w, along with the number of files that contain w and $\mathsf{No.Search[w]}$, which contains the number of times a keyword w has been searched by a user. Both of these indexes are of size of $O(m)$, where m is the total number of keywords. Finally, the data owner generates a dictionary Dict, which is a mapping between keywords and filenames The dictionary's size is $O(N) = O(nm)$, where n is the total number of files. To achieve the multi-client model, the data owner outsources $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ to a trusted authority TA on the cloud but also keeps a copy locally. These indexes will allow registered users to create consistent search tokens. Dict is finally sent to the CSP.

Cloud Service Provider (CSP): We consider a cloud computing environment similar to the one described in [10, 11]. The CSP must support SGX since core entities will be running in the trusted execution environment offered by SGX. The CSP storage will consist of the ciphertexts as well as of the dictionary Dict. Each entry of Dict is encrypted under a different key $\mathsf{K_w}$. Thus, given $\mathsf{K_w}$ and the number of files containing a keyword w, the CSP can recover the files containing w.

Trusted Authority (TA): TA is an index storage that stores the $\mathsf{No.Files}$ and $\mathsf{No.Search}$ indexes that have been generated by the data owner. All registered users can contact the TA to access the $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ values for a keyword w. These values are needed to create the search tokens that will allow users to search directly on the encrypted database.

SGX: We provide a brief presentation of the main SGX functionalities. A more detailed description can be found in [6].

- **Isolation:** Enclaves are located in a hardware guarded are of memory they compromise a total of 128MB (only 90MB can be used by software). Intel SGX is based on memory isolation built in the processor along with strong cryptography. The processor tracks which parts of memory belong to which enclave and ensures that only enclaves can access their own memory.
- **Sealing:** Every SGX processor comes with a Root Seal Key with which, data is encrypted when stored in untrusted memory. Sealed data can be recovered even after an enclave is destroyed and rebooted on the same platform.
- **Attestation:** One of the core contributions of SGX is the support for attestation between enclaves of the same (local attestation) and different platforms (remote attestation). In the case of local attestation, an enclave enc_i can verify another enclave enc_j as well as the program/software running in

the latter. This is achieved through a report generated by enc_j containing information about the enclave itself and the program running in it. This report is signed with a secret key $\mathsf{sk}_{\mathsf{rpt}}$ which is the same for all enclaves of the same platform. In remote attestation, enclaves of different platforms can attest each other through a signed quote. This is a report similar to the one used in local attestation. The difference is that instead of using $\mathsf{sk}_{\mathsf{rpt}}$ to sign it, a special private key provided by Intel is used. Thus, verifying these quotes requires contacting Intel's Attestation Server.

4 Construction

In this section we present our construction. We start by describing a multi-client SSE scheme with forward privacy. Our construction constitutes of six different algorithms: $\mathsf{KeyGen}, \mathsf{InGen}, \mathsf{AddFile}, \mathsf{Search}, \mathsf{Delete}$ and Update. At first, the data owner u_i runs KeyGen to generate her secret key. As a next step, she will execute InGen and $\mathsf{AddFile}$ to create the three indexes: $\mathsf{No.Files}[w], \mathsf{No.Search}[w]$ and Dict and encrypt her files. While $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ are outsourced to the TA who is running on the cloud a copy is also stored locally by the user. Dict is sent directly to the CSP, where it will be stored. The CSP will use this dictionary to reply to future search queries. The data owner, can still run the $\mathsf{AddFile}$ algorithm, even after the execution of InGen, if she wishes to add new files to her encrypted file collection. Just like file addition, u_i can also delete files by running Delete. File modification is then simplified to a sequence of Delete and $\mathsf{AddFile}$ for a specified file f. Finally, any user that possesses the data owner's secret key, is allowed to search directly on u_i's encrypted collection, by creating and sending a search token to the CSP.

4.1 MC-SSE

Let $G : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an invertible pseudorandom function (IPRF). Moreover, let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure symmetric key encryption scheme and finally, let $h : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a cryptographic hash function for the security parameter λ.

Key Generation: The data owner generates the secret key $\mathsf{K} = (\mathsf{K}_G, \mathsf{K}_{\mathsf{SKE}})$, where K_G is a key for the IPRF G, while $\mathsf{K}_{\mathsf{SKE}}$ is the key for the CPA-secure symmetric key encryption scheme that will be used for the encryption of user's files. K_G is also sent to the TA. This is a probabilistic algorithm run by the data owner.

Indexing: After the data owner generates the secret key K, she generates the indexes required by the scheme. In particular, she generates the following three indexes:

- $\mathsf{No.Files}$ – stores the total number of files containing a keyword w,

Algorithm 1 KeyGen

Input: Security parameter λ
Output: $\mathsf{K} \leftarrow \mathsf{Gen}(1^\lambda)(1^\lambda)$

1: $\mathsf{K}_G \leftarrow \mathsf{GenIPRF}(1^\lambda)$ ▷ For the IPRF G
2: $\mathsf{K}_{\mathsf{SKE}} \leftarrow \mathsf{SKE.Gen}(1^\lambda)$
3: return $\mathsf{K} = (\mathsf{K}_G, \mathsf{K}_{\mathsf{SKE}})$
4: Send K_G to the TA

- No.Search – stores the number of times each keyword has been searched by a user,
- Dict – a mapping of a keyword and the filename that can be found at.

The data owner outsources No.Files and No.Search to the TA but also stores a copy of each index locally. Dict is sent to the CSP. This protocol is treated like a set of AddFile protocols, thus the data owner is required to internally run AddFile. Note that upon its generation, Dict is directly sent to the CSP. However, this is not the case for No.Search and No.Files. More precisely, before outsourcing the indexes to the TA, the data owner needs to encrypt them using TA's public key. Although this process is not characterized by its efficiency, it will only occur once and it is a necessary trade-off to achieve a multi client scheme. Upon reception, TA decrypts the indexes using its private key and stores them in plaintext.

Algorithm 2 InGen

Input: sk, f_i
Output: $(\mathcal{I}_{CSP}, c_i), \mathcal{I}_{TA} \leftarrow \mathsf{Index}(\mathsf{sk}, f_i)$

1: $C = \{\}$
2: AllMap $= \{\}$
3: **for all** f_i **do**
4: Run AddFile to generate c_i and Map$_i$ ▷ Results are NOT sent to the CSP
5: $C = C \cup c_i$
6: AllMap $= [\{\text{AllMap} \cup \text{Map}_i\}, id(f_i)]$
7: $C_{TA} = \mathsf{PKE.Enc}(\mathsf{pk_{TA}}, (\mathsf{No.Files[w]}, \mathsf{No.Search[w]}))$
8: Send (AllMap, C) to the CSP
9: Send C_{TA} to the TA
10: CSP stores AllMap in Dict
11: $\mathcal{I}_{CSP} = \{\mathsf{Dict}\}$, $\mathcal{I}_{TA} = \{\mathsf{No.Files}, \mathsf{No.Search}\}$

File Insertion: The data owner can add new files to her collection, even after the execution of Algorithm 2. To do so, she retrieves the No.Files$[w]$ and No.Search$[w]$ indexes, that are stored locally on her device. These indexes will allow her to create an add token $\tau_\alpha(f)$ for the file f that she wishes to add. For

each *distinct* keyword $w_i \in f$, she increments $\mathsf{No.Files}[w_i]$ by one and then computes the corresponding address on Dict. Moreover, she computes $c = (\mathsf{K_{SKE}}, f)$ and she sends the results to the CSP (lines 8-11 of Algorithm 3). As a last step, u_i sends an acknowledgement to the TA so that TA will also increment $\mathsf{No.Files}[w]$ by one.

Algorithm 3 AddFile

Input: $\mathsf{sk}, f, \mathcal{I}_{TA}$
Output: $(\mathcal{I'}_{CSP}, C'), \mathcal{I'}_{TA} \leftarrow \mathsf{Add}(\mathsf{sk}, f, \mathcal{I}_{TA})$

1: Map = {}
2: **for all** $w_i \in f$ **do**
3: $\quad \mathsf{No.Files}[w_i] + +$
4: $\quad \mathsf{K}_{w_i} = G(\mathsf{K}_G, h(w_i)||\mathsf{No.Search}[w_i])$
5: $\quad \text{addr}_{w_i} = h(\mathsf{K}_{w_i}, \mathsf{No.Files}[w_i]||0)$ ▷ Address in Dict
6: $\quad \text{val}_{w_i} = \mathsf{Enc}(\mathsf{K_{SKE}}, id(f_i)||\mathsf{No.Files}[w_i])$
7: $\quad$ Map = Map $\cup\{\text{addr}_{w_i}, \text{val}_{w_i}\}$
8: $c \leftarrow \mathsf{SKE.Enc}(\mathsf{K_{SKE}}, f)$
9: $\tau_\alpha(f) = (c, \text{Map})$
10: Send $\tau_\alpha(f)$ to the CSP
11: CSP adds i into C and Map into Dict
12: Send the updated value of $\mathsf{No.Files}$ to TA
13: TA updates $\mathsf{No.Files}$

Search: We now assume that the data owner has successfully shared her secret key K with multiple users so that they can also access her encrypted data. Lets assume that a user u_j who has access to K wishes to search for specific keywords in $u_i's$ encrypted data. To do so, she needs to create a search token $\tau_s(w)$ that will allow her to search for a specific keyword w. To create $\tau_s(w)$ for a specific keyword w, u_j first needs to request the corresponding values $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ from the TA (line 1 of Algorithm 4). After u_j receives these values, she can compute the key K_w for the keyword w as $\mathsf{K_w} = G(\mathsf{K_F}, h(w)||\mathsf{No.Search}[w])$. Apart from that, she increments the $\mathsf{No.Search}$ value by one and computes the updated key for w, K'_w as well as the new addresses addr_w on Dict. She finally stores the new addresses in a list L that will be sent to the CSP (lines 4-11 of Algorithm 4). Upon reception, the CSP forwards $(\mathsf{K}_w, \mathsf{No.Files}[w])$ to the TA to ensure that u_j sent the correct values. At this point, TA retrieves the keywords w and $\mathsf{No.Search}[w]$ by inverting the pseudorandom function G. In particular, TA computes: $G^{-1}(\mathsf{K_G}, \mathsf{K}_w) = G^{-1}(\mathsf{K_G}, G(\mathsf{K_G}, h(w)||\mathsf{No.Search}[w]) = h(w)||\mathsf{No.Search}[w]$. As soon as TA retrieves these values, it can compute $\mathsf{K'}[w]$ by incrementing $\mathsf{No.Search}$ by one. In addition to that, it also computes addr'_w. Finally, it stores addr'$_w$ to a list L_{TA} and sends it to the CSP (lines 12-20 of Algorithm 4). Upon reception, the CSP checks whether $L_u = L_{TA}$ or not. If $L_u \neq L_{TA}$, the CSP outputs $\perp$ and abort the protocol. If $L_u = L_{TA}$, then the

CSP locates the file identifiers by looking at Dict[] and replace the corresponding addresses on Dict and . The files are sent back to the user and the CSP sends an acknowledgement to TA in order to increment the value of $\mathsf{No.Search}[w]$ by one. Finally, this acknowledgement is also forwarded to the data owner, so that she can also update her local indexes.

File Deletion: Apart from adding files to the her collection, the data owner u_i can also delete files. To do so, she first needs to generate a delete token $\tau_d(f)$ and send it to the CSP. The CSP can identify which entries of Dict correspond to the file f that u_i wishes to delete. However, apart from deleting these entries the indexes $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ need to be updated. To this end, the CSP sends f back to u_i before deleting the entries. Upon reception, u_i extracts every keyword w_i contained in f and updates the indexes accordingly (lines 7-10 of Algorithm 5). However, since these indexes are also stored online, u_i will have to send the updated values $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ to the TA as well.

File Modification: Finally, for the data owner u_i to modify a file f (Algorithm 6) that is stored online, she first needs to run the Delete algorithm to make sure that each entry associated with f will be deleted and that all indexes will be updated accordingly. As a next step, u_i modifies f and runs the AddFile algorithm for the updated file.

5 Security Analysis

In this section we prove the security of our construction against the threat model defined in Section 2.3. We construct a simulator $\mathcal{S}$ that simulates the SSE scheme as well as the communications between the enclaves.

Theorem 1. *Let* SKE *be a CPA-secure symmetric key encryption scheme, G an invertible pseudorandom function and h a hash function. Then our construction is secure according to definition 8.*

Proof. We construct a simulator $\mathcal{S}$ that simulates the real algorithms in such a way that any Probabilistic Polynomial Time(PPT) adversary $\mathcal{ADV}$ will not be able to distinguish between the real algorithms and the simulated ones. To prove the security of our construction we are using a hybrid argument where $\mathcal{S}$ is given the leakage functions $\mathcal{L}_{\mathsf{InGen}}, \mathcal{L}_{\mathsf{Add}}, \mathcal{L}_{\mathsf{search}}, \mathcal{L}_{\mathsf{Delete}}$ and simulates the protocol. We assume that upon their initialization, enclaves generate the secret key $\mathsf{sk}_{\mathsf{rpt}}$, used to sign their reports.

Hybrid 0 Everything runs as specified by the protocol.

Hybrid 1: Like Hybrid 0 but instead of InGen, $\mathcal{S}$ is given access to the leakage function $\mathcal{L}_{\mathsf{InGen}} = (N, n, id(f_i), |f_i|), \forall f_i \in F$ and proceeds as follows:

$\mathcal{ADV}$ cannot distinguish between the two hybrids as the simulated dictionary has exactly the same size as the real one. Moreover, the CPA security of the symmetric encryption scheme, ensures that $\mathcal{ADV}$ cannot distinguish between the encryption of actual files and that of a string of zeros.

Algorithm 4 Search

Input: $\mathsf{sk}, f_i, \mathcal{I}_{TA}$
Output: $(\mathcal{I'}_{CSP}, C'), \mathcal{I'}_{TA} \leftarrow \mathsf{Add}(\mathsf{sk}, f_i, \mathcal{I}_{TA})$

User:
Request the values $\mathsf{No.Files}[w]$ and $\mathsf{No.Search}[w]$ for a keyword w, from TA
TA:
Verifies the user and send back the values
User:
$\mathsf{K}_w = G(\mathsf{K}_G, h(w)||\mathsf{No.Search}[w])$
$\mathsf{No.Search}[w] + +$
$\mathsf{K}'_w = G(\mathsf{K}_G, h(w)||\mathsf{No.Search}[w])$
$L_u = \{\}$
for $i = 1$ to $i = \mathsf{No.Files[w]}$ **do**
    $\text{addr}_w = h(\mathsf{K}'_w, i||0)$
    $L_u = L_u \cup \{\text{addr}_w\}$
Send $\tau_s(w) = (\mathsf{K}_w, \mathsf{No.Files}[w], L_u)$ to the CSP
CSP:
Forward $(\mathsf{K}_w, \mathsf{No.Files}[w])$ to TA
TA:
$h(w)||\mathsf{No.Search}[w] = G^{-1}(\mathsf{K}_G, \mathsf{K}_w)$
$\mathsf{K}'_w = G(\mathsf{K_G}, h(w)||\mathsf{No.Search}[w])$
$L_{TA} = \{\}$
for $i = 1$ to $i = \mathsf{No.Files[w]}$ **do**
    $\text{addr}_w = h(\mathsf{K}'_w, i||0)$
    $L_{TA} = L_{TA} \cup \{\text{addr}_w\}$
Send List to the CSP
CSP:
if $L_u = L_{TA}$ **then**
    $C_{F_w} = \{\}$
    for $i = 1$ to $i = \mathsf{No.Files[w]}$ **do**
        $c_{f_i} = \mathsf{Dict}[(h(\mathsf{K}_w, i||0))]$
        $C_{F_w} = F_w \cup \{c_{f_i}\}$
        Delete $\mathsf{Dict}[(h(\mathsf{K}_w, i||0))]$
    Add the new addresses and values as specified in L_u
else
    Output $\perp$
Send C_{F_w} to the user and an acknowledgement to the TA and the Data Owner
TA:
$\mathsf{No.Search}[w] + +$
Data Owner:
$\mathsf{No.Search}[w] + +$

Algorithm 5 Delete

Input: File identifier $id(f)$
Output: File f is deleted

Data owner:
$\mathsf{FileNumber} = \{\}$
for all $w_i \in f_i$ **do**
  if $\mathsf{No.Files}[w_i] > 1$ **then**
    $\text{addr}_{w_i} = h(\mathsf{K}_w, \mathsf{No.files}[w_i||0])$
    $\text{val}_{w_i} = \mathsf{Enc}(\mathsf{K}, \mathsf{No.Files}[w])$
    $\mathsf{No.Files}[w] - -$
    $\text{naddr} = h(\mathsf{K}_w, \mathsf{No.files}[w_i||0])$
    $\text{nval} = \mathsf{Enc}(\mathsf{K}, \mathsf{No.Files}[w])$
  else
    $\text{val}_{w_i} = \mathsf{Enc}(\mathsf{K}, \mathsf{No.Files}[w])$
    $\text{nval} = 0$
    Delete $\mathsf{No.Files}[w_i]$
    Delete $\mathsf{No.Search}[w_i]$
  $\mathsf{FileNumber} = \mathsf{FileNumber} \cup \{h(w), \mathsf{No.Files}[w_i]\}$
$\tau_d(f) = \left[id(f), \{\mathsf{K}_w, (\text{addr}_{w_i}, \text{naddr}_{w_i}), (\text{val}_{w_i}, \text{nval}_{w_i})\}_{i=1}^{\#w \in f}\right]$
Send $\{\mathsf{FileNumber}, \tau_d(f)\}$ to the CSP
CSP:
Forward $\{\mathsf{FileNumber}, \tau_d(f)\}$ to the TA
TA:
for all $h(w) \in \mathsf{FileNumber}$ **do**
  if $\mathsf{No.Files}[w] > 1$ **then**
    Repeat steps (4 - 9)
  else
    Repeat steps (11 - 12)
Send $\{(\text{addr}_{w_i}, \text{naddr}_{w_i}), (\text{val}_{w_i}, \text{nval}_{w_i})\}_{i=1}^{\#w \in f}$ to the CSP
CSP:
$\mathsf{NewVal} = \{\}$
if the values received by the user are Not the same as the ones received by the TA **then**
  Output $\perp$
else
  for $i = 1$ to $i = \#w \in f$ **do**
    if $\text{nval}_{w_i} = 0$ **then**
      Delete val_{w_i}
    else
      $\text{addr}_{w_i} = \text{naddr}_{w_i}$
      $\text{val}_{w_i} = \text{nval}_{w_i}$
  Send acknowledgement to TA
TA:
for all $w_i \in f$ **do**
  if $\mathsf{No.Files}[w_i] > 1$ **then**
    $\mathsf{No.Files}[w_i] - -$
  else
    Delete $\mathsf{No.Files}[w_i]$
    Delete $\mathsf{No.Search}[w_i]$

Algorithm 6 Update file

Input: File identifier $id(f)$
Output: f is updated

Data Owner:
1: Run Delete Algorithm with $h(id(f))$ as input
2: Modify f
3: Run AddFile Algorithm with f as input

Algorithm 7 InGen Simulation

1: $k \leftarrow \mathsf{SKE.Gen}(1^{\lambda})$
2: **for** $i = 1$ to $i = N$ **do**
3: Simulate (a_i, v_i) pairs such that $|a_i| = |v_i|$
4: Store all (a_i, v_i) pairs in a dictionary Dict
5: **for all** $f_i \in F$ **do**
6: $c_i \leftarrow \mathsf{SKE.Enc}(\mathsf{k}, 0^{|\mathsf{f_i}|})$
7: Create a dictionary $\mathsf{KeyStore}$ to store the last K_w of each keyword.
8: Create a dictionary Oracle to reply to the random oracle queries.

Hybrid 2 Like Hybrid 1, but instead of $\mathsf{AddFile}$, $\mathcal{S}$ is given the leakage function $\mathcal{L}_{\mathsf{Add}} = (id(f), |f|, \#w_i)$ and simulates the add token as follows:

Algorithm 8 Add Token Simulation

1: $\mathcal{L}_{add} = \{\}$
2: **for** $i = 1$ to $i = \#w_i$ **do**
3: Simulate $\{a_i, v_i\}$ pairs such that $|a_i| = |v_i|$
4: Add $(id(f), \{a_i, v_i\})$ in Dict
5: $\mathcal{L}_{add} = \mathcal{L}_{add} \cup \{a_i, v_i\}$
6: $c \leftarrow \mathsf{SKE.Enc}(\mathsf{k}, 0^{|\mathsf{f}|})$
7: $\tau_{\alpha}(f) = (id(f), c, \mathcal{L}_{add})$

The simulated add token, allows the $\mathcal{S}$ to keep its dictionary up to date with files provided by $\mathcal{ADV}$ after the execution of InGen. The token provided by the simulator has exactly the same format and size as the real add token. Moreover, we show that the add token can be simulated by only knowing $\mathcal{L}_{\mathsf{Add}}$, and as a result we prove that our scheme preserves forward privacy. Finally, once again the CPA security of the symmetric encryption scheme, ensures that $\mathcal{ADV}$ cannot distinguish between the encryption of actual files and that of a string of zeros.

Hybrid 3 Like Hybrid 1 but instead of running the Search protocol, $\mathcal{S}$ is given the Search and Access patterns and simulates the search tokens, both for the data owner as well as for the rest of the users. Apart from that, $\mathcal{S}$ simulates the

communications between the different entities that participate in the real Search protocol. The search tokens are simulated as shown in Algorithm 9.

Algorithm 9 Search Token Simulation

Generate two random integers and send them to the user ▷ Step 1 is NOT for the data owner
$d = |F_w|$ ▷ Number of files containing w
if $\mathsf{KeyStore}[w] = Null$ **then**
    $\mathsf{KeyStore}[w] \leftarrow \{0,1\}^\lambda$
    $\mathsf{K}_w = \mathsf{KeyStore}[w]$
for $i = 1$ to $i = d$ **do**
    if $\mathsf{Oracle}[\mathsf{K}_w][0][i]$ is Null **then**
        if f_i is added after InGen **then**
            Pick a $(id(f_i), \{a_i, v_i\}$ pair
        else
            Pick an unused $\{a_i, v_i\}$ at random
        $\mathsf{Oracle}[\mathsf{K}_w][0][i] = a_i$
        $\mathsf{Oracle}[\mathsf{K}_w][1][i] = v_i || id(f_i)$
    else
        $a_i = \mathsf{Oracle}[\mathsf{K}_w][0][i]$
        $v_i = \mathsf{Oracle}[\mathsf{K_w}][1][i](|\mathsf{Oracle}[\mathsf{K_w}][1][i]| - |id(f_i)|)$
    Remove a_i from the dictionary
$UpdatedVal = \{\}$
$\mathsf{K}'_w \leftarrow \{0,1\}^\lambda$
$\mathsf{KeyStore}[w] = \mathsf{K}'_w$
for $i = 1$ to $i = d$ **do**
    Generate new a_i and match it with v_i from step 16
    Add $(id(f_i), \{a_i, v_i\})$ to the dictionary
    $UpdatedVal = UpdatedVal \cup \{id(f_i), a_i\}$
    $\mathsf{Oracle}[\mathsf{K}'_w][0][i] = a_i$
    $\mathsf{Oracle}[\mathsf{K}'_w][1][i] = v_i || id(f_i)$
$\tau_s(w) = (\mathsf{K}_w, d, UpdatedVal)$

The $\mathsf{KeyStore}[w]$ dictionary is used to keep track of the last key K_w used for each keyword w. The $\mathsf{Oracle}[\mathsf{K}_w[j][i]$ dictionary is used to reply to ADV's queries. For example, $\mathsf{Oracle}[\mathsf{K}_w[0][i]$ represents the address of a Dict entry assigned to the $i-th$ file in the file collection F. Similarly, $\mathsf{Oracle}[\mathsf{K}_w[1][i]$ represents the masked value needed to recover $id(f)$. It is clear, that the simulated search token has exactly the same size and format as the real one, and as a result no PPT adversary can distinguish between them. Moreover, $\mathcal{ADV}$ cannot tamper with the reports generates by the enclaves during the execution of the local attestation protocols. The reason for this, is that these reports are signed with the secret sk_{rpt} key, which is only know to enclaves that reside in the same platform. As

a result, tampering with the reports implies producing a valid MAC, which can only happen with negligible probability.

Hybrid 3 Like Hybrid 3, but $\mathcal{S}$ is given $\mathcal{L}_{Delete}(f) = (id(f), \#w_i, l)$ and simulates the delete token as described in Algorithm 10. Moreover, $\mathcal{S}$ simulates the communication between TA and CSP.

Algorithm 10 Delete Token Simulation

$\mathcal{L}_{del} = \{\}$
$\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{\mathcal{S}} \cup id(f)$
$c \leftarrow \mathsf{SKE.Enc}(\mathsf{K_{SKE}}, 0^{|id(f)|})$
for $i = 1$ to $i = \ell$ **do**
    Generate a new $\{a'_i, v'_i\}$ pair
    Select an unused $\{a_i, v_i\}$ pair
    $\mathcal{L}_{del} = \mathcal{L}_{del} \cup \{\{a'_i, v'_i\}, \{a_i, v_i\}\}$
    Replace $\{a_i, v_i\}$ with $\{a'_i, v'_i\}$
for $i = \ell + 1$ to $i = \#w$ **do**
    Generate a new v'_i
    Pick an unused $\{a_i, v_i\}$ pair and delete it
    $\mathcal{L}_{del} = \mathcal{L}_{del} \cup \{v'_i, 0\}$
Output $\tau_d(f) = (c, \mathcal{L}_{del})$

The simulated delete token is indistinguishable from the real one since the have the same format and size. Moreover, $\mathcal{ADV}$ could once again try to tamper with the report sent from TA to CSP as part of the protocol. However, as already stated before, this action would imply that $\mathcal{ADV}$ could produce a valid MAC without owning sk_{rpt} which can only happen with negligible probability. Hence, the Hybrids are indistinguishable.

With that hybrid our proof is complete. We constructed a simulator $\mathcal{S}$ that simulates all the real protocols in a way that no PPT adversary $\mathcal{ADV}$ can distinguish between the real and the ideal experiments.

6 Conclusion and Future Work

In this paper, we extended the scheme presented in [5] by presenting a forward private symmetric searchable encryption scheme that supports the multi-client model. In our construction, we deal with the problem of syncrhonization between users, by using the functionality offered by SGX. As a result, our construction can be seen as an important contribution in the field of Searchable Encryption. Furthermore, our scheme can squarely fits constructions like the ones presented in [9, 3, 8, 7].

As future steps, we plan to implement our construction using as datasets either the Wikipedia archive [2] or the Gutenberg ptoject [1]. This will allow us to evaluate the performance of our construction against a variety of realistic

scenarios (i.e. using datasets from which a very large number of keywords can be extracted). Finally, we plan to extend our construction in a way that protection against, the more realistic malicious adversarial model, will be supported.

References

1. Project gutenberg (1971), https://www.gutenberg.org/
2. Wikipedia:database download (2019), https://en.wikipedia.org/wiki/Wikipedia:Database_download
3. Bakas, A., Michalas, A.: Modern family: A revocable hybrid encryption scheme based on attribute-based encryption, symmetric searchable encryption and sgx. Cryptology ePrint Archive, Report 2019/682 (2019), https://eprint.iacr.org/2019/682
4. Dowsley, R., Michalas, A., Nagel, M., Paladi, N.: A survey on design and implementation of protected searchable data in the cloud. Computer Science Review (2017). https://doi.org/https://doi.org/10.1016/j.cosrev.2017.08.001, http://www.sciencedirect.com/science/article/pii/S1574013716302167
5. Etemad, M., Küpçü, A., Papamanthou, C., Evans, D.: Efficient dynamic searchable encryption with forward privacy. Proceedings on Privacy Enhancing Technologies **2018**(1), 5–20 (2018)
6. Fisch, B., Vinayagamurthy, D., Boneh, D., Gorbunov, S.: Iron: functional encryption using intel sgx. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 765–782. ACM (2017)
7. Michalas, A., Dowsley, R.: Towards trusted ehealth services in the cloud. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC). pp. 618–623 (Dec 2015). https://doi.org/10.1109/UCC.2015.108
8. Michalas, A., Weingarten, N.: Healthshare: Using attribute-based encryption for secure data sharing between multiple clouds. In: 2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS). pp. 811–815 (June 2017). https://doi.org/10.1109/CBMS.2017.30
9. Michalas, A.: The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. pp. 146–155. SAC '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3297280.3297297, http://doi.acm.org/10.1145/3297280.3297297
10. Paladi, N., Gehrmann, C., Michalas, A.: Providing user security guarantees in public infrastructure clouds. IEEE Transactions on Cloud Computing **5**(3), 405–419 (July 2017). https://doi.org/10.1109/TCC.2016.2525991
11. Paladi, N., Michalas, A., Gehrmann, C.: Domain based storage protection with secure access control for the cloud. In: Proceedings of the 2014 International Workshop on Security in Cloud Computing. ASIACCS '14, ACM, New York, NY, USA (2014)