Nowhere to Leak: A Multi-client Forward and Backward Private Symmetric Searchable Encryption Scheme

Alexandros Bakas, Antonis Michalas

Nowhere to Leak: A Multi-Client Forward and Backward Private Symmetric Searchable Encryption Scheme*

Alexandros Bakas and Antonis Michalas

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

Abstract. Symmetric Searchable Encryption (SSE) allows users to outsource encrypted data to a possibly untrusted remote location while simultaneously being able to perform keyword search directly through the stored ciphertexts. An ideal SSE scheme should reveal no information about the content of the encrypted information nor about the searched keywords and their mapping to the stored files. However, most of the existing SSE schemes fail to fulfil this property since in every search query, some information potentially valuable to a malicious adversary is leaked. The leakage becomes even bigger if the underlying SSE scheme is dynamic. In this paper, we *minimize the leaked information* by proposing a *forward and backward private* SSE scheme in a multi-client setting. Our construction achieves optimal search and update costs. In contrast to many recent works, each search query only requires one round of interaction between a user and the cloud service provider. In order to guarantee the security and privacy of the scheme and support the multi-client model (i.e. synchronization between users), we exploit the functionality offered by AMD's Secure Encrypted Virtualization (SEV).

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

1 Introduction

Symmetric Searchable Encryption (SSE) is a promising encryption technique that squarely fits the cloud paradigm and can pave the way for the development of cloud services that will respect users' privacy even in the case of a compromised Cloud Service Provider (CSP) [6]. Additionally, SSE schemes can be seen as a first, fundamental step for protecting users' data from both external and *internal* attacks (e.g. a malicious administrator). This is because since in an SSE scheme, users generate all the secret information (encryption key) locally and encrypt all of their data on the client-side (i.e. the encryption key is *never* revealed to the

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

CSP). The service offered by the CSP is only used for storing and retrieving the generated ciphertexts. In contrast to traditional encryption schemes, SSE offers a remarkable functionality – it allows users to search for specific keywords directly through the stored ciphertexts. This is done by asking the CSP to execute search queries in a privacy-preserving way. In other words, the CSP can find all the ciphertexts containing a specific keyword but without knowing the underlying keyword or anything about the content of the corresponding files.

An ideal SSE scheme should reveal no information about the content of the encrypted information nor about the searched keywords and their mapping to the stored files. However, most SSE schemes fail to fulfill this property since in every search or update query, some information potentially valuable to a malicious adversary is leaked. In the early years of SSE, researchers utilized techniques such as oblivious RAM (ORAM). However, according to [20], adopting such a technique is even less efficient than downloading and decrypting the entire database locally. As a result, researchers have come to a silent agreement that *"nothing should be revealed beyond some well defined and "reasonable" leakage"* [3].

Leaked information in SSE schemes has become a problem of paramount importance since it is the main factor in defining the overall level of security. In works such as [11] and [16] it is pointed out that even a small leakage can lead to several privacy attacks. These works were further extended in [24] where the authors assumed that an active adversary can perform file-injection attacks and record the output. This "new" ability allowed the adversary to recover information about past queries only after ten file insertions. This result led researchers to design *forward private* SSE schemes [8,13]. Forward privacy is a notion introduced in [22] and guarantees that that newly added files cannot be related to past search queries. While forward privacy is a very important property, unfortunately it has been shown to also be vulnerable to certain file-injection attacks [24].

While forward privacy secures the content of a past query, its binary property, *backward privacy*, ensures the privacy of future queries. Backward privacy was formalized in [9] where three different flavors were defined. A backward private SSE scheme ensures that queries do not reveal their association with deleted documents. To the best of our knowledge, there are only a handful of backward private schemes per flavor where *none* of them supports the multi-client model.

Our Contribution: We extend the work proposed in [13] by constructing a forward and backward private Dynamic SSE scheme that supports a multi-client model. We deal with the problem of synchronization between multiple clients by utilizing the functionality offered by AMD's SEV [2]. In particular, our construction: ***(1) Provides Forward Privacy. (2) Provides Backward Privacy. (3) Is asymptotically optimal. (4) Is Parallelizable.***

2 Related Work

Our work is based on [13] where the authors presented a Symmetric Searchable Encryption scheme with Forward Privacy, a notion first introduced in [22]. Their construction however is only forward private and limited to a single client

model. In this work, we make use of the functionality offered by AMD's SEV VMs to both extend the original scheme to be backward private and to support the multi-client setting. Another single-client forward private SSE scheme is presented in [8], where the authors designed *Sophos*. While *Sophos* achieves asymptotically optimal search and update costs, $O(\ell)$ and $O(m)$ respectively, a file addition requires $O(m)$ asymmetric cryptographic operations on the user's side. An improvement in the search time of *Sophos* is presented in [17]. Authors of [8] extended their work in [9] by designing a number of SSE schemes that are both forward and backward private. Out of those schemes, $Diana_{del}$ and *Janus* are the most efficient but at the same time they satisfy the weakest notion of backward privacy. In particular, *Janus* achieves its security by using public puncturable encryption. *Fides* is among the first efficient backward private SSE schemes with stronger security guarantees. However, it only satisfies the single-client model. Moreover, the search operation requires two rounds of interaction while our scheme only requires one. An improvement of *Janus* is presented in [23] where authors design *Janus++*. While *Janus++* is more efficient than Janus as it is based on symmetric puncturable encryption, Janus++ can only achieve the same security level as Janus. An SGX-based forward/backward private scheme called *Bunker-B* is presented in [3]. Our construction is similar to that in the sense that we use a trusted execution environment (TEE) to reduce the number of required rounds to one. However, as in the case of *Sophos* and *Fides*, *Bunker-B* only supports the single-client model. Moreover, we believe that SGX in not a suitable TEE for a cloud-based service due to its limitations.

Comparison						
Scheme	MC	FP	BP	Search Time	Update Time	Client Storage
Bunker-B	✗	✓	Type-II	$O(\ell)$	$O(1)$	$O(mlogn)$
Fides	✗	✓	Type-II	$O(\ell)$	$O(1)$	$O(mlogn)$
$Diana_{del}$	✗	✓	Type-III	$O(\ell)$	$O(logN)$	$O(mlogn)$
Janus	✗	✓	Type-III	$O(\ell d)t_{PE.Dec}$	$O(\ell)(t_{PE.Enc} \vee t_{PE.Dec})$	$O(mlogn)$
Janus++	✗	✓	Type-III	$O(\ell d)t_{SPE.Dec}$	$O(\ell)(t_{SPE.Enc} \vee t_{SPE.Dec})$	$O(mlogn)$
Moneta	✗	✓	Type-I	$\widetilde{O}(a_w \log N + \log^3 N)$	$\widetilde{O}(\log^2 N)$	$O(1)$
Orion	✗	✓	Type-I	$O(\ell \log N^2)$	$O(logN^2)$	$O(1)$
Ours	✓	✓	Type-II	$O(\ell/p)$	$O(m/p)$	None

Table 1: N: number of (w, id) pairs, n: total number of files, m: total number of keywords, p: number of processors, k: number of keys, a_w: number of updates matching w, d_w: number of deleted entries matching w, ℓ: result size ($\ell = a_w - d_w$), $(t_{PE.Enc}, t_{PE.Dec})$: encryption and decryption times for a public puncturable encryption scheme, $(t_{SPE.Enc}, t_{SPE.Dec})$: encryption and decryption times for a symmetric puncturable encryption scheme MC: Multi-Client, FP: Forward Privacy, BP: Backward Privacy. $\widetilde{O}$ notation hides polylogarithmic factors.

In [10] authors presented *HardIDX*, a scheme that also supports range queries with the use of SGX [12] based on B^+ trees. *HardIDX* minimizes the leakage by hiding the search pattern but at the same time, their construction is *static*. As a result, it does *not* support file insertions after the generation of the initial index.

Therefore, even though the scheme achieves logarithmic search cost, a direct comparison to our scheme is *not* possible. *ORAM-based approaches:* The first forward private SSE scheme was proposed in [22], where the authors presented an ORAM-based construction. More recently, in [9], authors proposed *Moneta*, an SSE scheme that achieves the strongest level of backward privacy, but at the cost of efficiency. *Moneta* is based on the TWORAM construction presented in [14]. However, as argued in [15], the use of TWORAM renders *Moneta* impractical for realistic scenarios and the scheme can serve mostly as a theoretical result for the feasibility of stronger backward private schemes schemes. Finally, in [15], another ORAM-bsed scheme, *Orion*, is proposed. While *Orion* outperforms *Moneta*, the number of interactions between the user and the CSP depends on the size of the encrypted database. It needs to be noted that in this work, we do not deal with the revocation of the users as for example in [4,5]. In table 1 we see a comparison of the aforementioned schemes to our construction.

3 Background

Notation Let $\mathcal{X}$ be a set. We use $x \leftarrow \mathcal{X}$ if x is sampled uniformly from $\mathcal{X}$ and $x \xleftarrow{\$} \mathcal{X}$, if x is chosen uniformly at random. If $\mathcal{X}$ and $\mathcal{Y}$ are two sets, 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 $\mathcal{Y}$. $R(\cdot)$ is a truly random function and $R^{-1}(\cdot)$ its inverse. A function $negl(\cdot)$ is called negligible, iff $\forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N} : \forall n \geq n_0, negl(n) < n^{-c}$. 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 $l < n$. A file collection is represented as $\mathbf{f} = (f_1, \ldots, f_z)$ while the corresponding collection of ciphertexts is $\mathbf{c} = (c_{f_1}, \ldots, c_{f_z})$. The universe of keywords is $\mathcal{W} = (w_1, \ldots, w_k)$ and the distinct keywords in a file f_i are $w_i = (w_{i_1}, \ldots, w_{i_\ell})$.

Definition 1 (DSSE Scheme). *A Dynamic Symmetric Searchable Encryption (DSSE) scheme consists of the following PPT algorithms:*

- $(\mathsf{In}_{\mathsf{CSP}}, \mathbf{c})(\mathsf{In}_{\mathsf{TA}})(\mathsf{K}) \leftarrow \mathsf{Setup}(\lambda, \mathbf{f})$*: The data owners runs this algorithm to generate the key* K *as well as the CSP index* $\mathsf{In}_{\mathsf{CSP}}$ *and a collection of ciphertexts* $\mathbf{c}$ *that will be sent to the CSP. Additionally, the index* $\mathsf{In}_{\mathsf{TA}}$ *that is stored on a remote location is generated.*
- $(\mathsf{In}'_{\mathsf{CSP}}, R_{w_{i_j}})(\mathsf{In}'_{\mathsf{TA}}) \leftarrow \mathsf{Search}(\mathsf{K}, w_{i_j}, \mathsf{In}_{\mathsf{TA}})(\mathsf{In}_{\mathsf{CSP}}, \mathbf{c})$*. This algorithm is executed by a user in order to search for all files* f_i *containing a specific keyword* w_{i_j}*. The indexes are updated and the CSP also returns to the user the ciphertexts of the files that contain* w_{i_j}*.*
- $(\mathsf{In}'_{\mathsf{CSP}}, \mathbf{c}')(\mathsf{In}'_{\mathsf{TA}}) \leftarrow \mathsf{Update}(\mathsf{K}, f_i, \mathsf{In}_{\mathsf{TA}})(\mathsf{In}_{\mathsf{CSP}}, \mathbf{c}, op)$*, where* $op \in \{\text{add}, \text{delete}\}$*: A user is running this algorithm to update the collection of ciphertexts* $\mathbf{c}$*. Based on the value of op, a new file is either added to the collection or an existing one is deleted.*

Definition 2. *($\mathcal{L}$-Adaptive Security of DSSE) Let DSSE =* $(\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ *be a dynamic symmetric searchable encryption scheme and* $\mathcal{L} = (\mathcal{L}_{stp}, \mathcal{L}_{search}, \mathcal{L}_{update})$

be the leakage function of the DSSE scheme. We consider the following experiments between an adversary $\mathcal{ADV}$ and a challenger $\mathcal{C}$:

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

$\mathcal{ADV}$ outputs a set of files $\mathbf{f}$. $\mathcal{C}$ generates a key K, and runs Setup. $\mathcal{ADV}$ then makes a polynomial number of adaptive queries $q = \{w, f_1, f_2\}$ such that $f_1 \notin \mathbf{f}$ and $f_2 \in \mathbf{f}$. 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 $\{w, f_2\}$. $\mathcal{ADV}$ outputs a bit b.

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

$\mathcal{ADV}$ outputs a set of files $\mathbf{f}$. $\mathcal{S}$ gets $\mathcal{L}_{\mathsf{setup}}(\mathbf{f})$ to simulate Setup. $\mathcal{ADV}$ then makes a polynomial number of adaptive queries $q = \{w, f_1, f_2\}$ such that $f_1 \notin \mathbf{f}$ and $f_2 \in \mathbf{f}$. For each q, $\mathcal{S}$ is given either $\mathcal{L}_{\mathsf{Search}}(w)$ or $\mathcal{L}_{\mathsf{update}}(f_i)$, $i \in \{1, 2\}$. $\mathcal{S}$ 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 if $\forall$ PPT adversary $\mathcal{ADV}$, $\exists$ $\mathcal{S}$ such that:

$$|Pr[(\mathsf{Real}_{\mathcal{ADV}}) = 1] - Pr[(\mathsf{Ideal}_{\mathcal{ADV},\mathcal{S}}) = 1]| \leq negl(\lambda) \quad (1)$$

A DSSE scheme is said to be *forward private* if for all file insertions that take place after the successful execution of the Setup algorithm, the leakage is limited to the size of the file, and the number of unique keywords contained in it. On the other hand, a DSSE scheme is said to be *backward private* if whenever a keyword/document pair $(w, id(f))$ is added into the database and then deleted, subsequent search queries for w do not reveal $id(f)$. More formally:

Definition 3 (Forward Privacy). *An $\mathcal{L}$-adaptively SSE scheme is forward private iff the leakage function $\mathcal{L}_{Update}$ can be written as:*

$$\mathcal{L}_{\mathsf{update}}(op, id(f)) = \mathcal{L}'(op, \#w \in f) \quad (2)$$

Where $\mathcal{L}'$ is a stateless function.

Definition 4 (Backward Privacy). *There are three different flavors of backward privacy (listed in decreasing strength):*

- *Type-I: Backward Privacy with insertion pattern leaks the documents currently matching w and when they were inserted i.e. their timestamps $TimeDB(w)$.*
- *Type-II: Backward Privacy with update pattern leaks the documents currently matching w, $TimeDB(w)$ and a list of timestamps $Updates(w)$ denoting when the updates on w happened.*
- *Type-III: Weak Backward Privacy leaks the documents currently matching they keyword w, $TimeDB(w)$ and $DelHist(w)$, where $DelHist(w)$ reveals the timestamps of the delete updates on w together with the corresponding entries that they remove.*

Our scheme satisfies Type-II backward privacy.

Definition 5. *An $\mathcal{L}$-adaptively SSE scheme is update pattern revealing backward private iff the search and update leakage functions can be written as:*

$$\begin{aligned} \mathcal{L}_{\mathsf{search}}(w) &= \mathcal{L}'(\text{TimeDB(w)}, \text{Updates(w)}) \\ \mathcal{L}_{\mathsf{update}}(op, w, id) &= \mathcal{L}''(op, w_i) \end{aligned} \tag{3}$$

Where the functions $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

Finally, the leakage function $\mathcal{L}_{stp}$ associated with the setup operation is formalised as follows:

$$\mathcal{L}_{stp} = (N, n, c_{id(f_i)}), \forall f_i \in \mathbf{f} \tag{4}$$

Where N is the total size of all the (keyword/filename) pairs, and n is the total number of files in the collection $\mathbf{f}$

3.1 Secure Encrypted Virtualisation

The main advantages of SEV in comparison to its main competitor -Intel SGX- are *(1)* memory size, *(2)* efficiency and *(3)* No SDK or code refactoring are required. In particular, SGX allocates only 128MB of memory for software and applications and thus, making it a good candidate for microtranscations and login services. However, SEV's memory is up to the available RAM and hence, making it a perfect fit for securing complex applications. Moreover, in situations where many calls are required, like in the case of a multi-client cloud service, SEV is known to be much faster and efficient than SGX. The above are summarized in Table 2. More information can be found in [19].

TEE	Memory Size	SDK	Code Refactoring
SEV	Up to Available Ram	Not Required	Not Required
SGX	Up to 128MB	Required	Major Refactoring

Table 2: SEV-SGX Comparison

4 Architecture

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

Users We denote with $\mathcal{U} = \{u_1, \ldots, u_n\}$ the set of all users that have been already registered in a cloud service that allows them to store, retrieve, update, delete and share encrypted files while at the same time being able to search over encrypted data by using our DSSE scheme. The users in our system model are mainly classified into two categories: data owners and simple registered users

that they have not yet upload any data to the CSP. A data owner first needs to locally parse all the data that wishes to upload to the CSP. During this process, she generates three different indexes:

1. $\mathsf{No.Files[w]}$ which contains a hash of each keyword w along with the number of files that w can be found at
2. $\mathsf{No.Search[w]}$, which contains the number of times a keyword w has been searched by a user.
3. Dict a dictionary that maintains a mapping between keywords and encrypted filenames.

Both $\mathsf{No.Files[w]}$ and $\mathsf{No.Search[w]}$ are of size $O(m)$, where m is the total number of keywords while the size of Dict 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). 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 [21]. The CSP must support SEV-enabled since core entities will be running in the trusted execution environment offered by SEV. The CSP storage will consist of the ciphertexts as well as of the dictionary Dict. Each entry of Dict is encrypted under a different symmetric key $\mathsf{K_w}$. Thus, given $\mathsf{K_w}$ and the number of files containing a keyword w, the CSP can locate 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. Similarly to the CSP, the TA is also SEV-enabled.

Deletion Authority (DelAuth) DelAuth is responsible for the deletion of files. Every time a user performs a search operation, the CSP forwards the result R to DelAuth. DelAuth decrypts the result, removes the Dict entries to be deleted and then re-encrypts the remaining filenames and sends them back to the CSP. Like the CSP and TA, DelAuth is also SEV enabled.

TA and DelAuth can be individual entities. For simplicity, we will assume that they are part of the same host.

5 Nowhere to Leak

Formal Construction: Our construction constitutes of three different algorithms, namely $\mathsf{Setup}, \mathsf{Search}$ and Update. Let $G : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$ be an invertible pseudorandom function (IPRF) [7]. Moreover, let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure symmetric key cryptosystem and $h = \{0,1\}^* \to \{0,1\}^\lambda$ be a cryptographic hash function. Due to space constrains, we present the $\mathsf{Setup}, \mathsf{Search}$ and Update Algorithms, in figures 1 and 2 respectively.

$\mathsf{K}_G \leftarrow \mathsf{GenIPRF}(1^\lambda)$
$\mathsf{K}_{\mathsf{SKE}} \leftarrow \mathsf{SKE.Gen}(1^\lambda)$
return $\mathsf{K} = (\mathsf{K}_G, \mathsf{K}_{\mathsf{SKE}})$
$c = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{TA}}, \mathsf{K}_G)$
Send K_G to the TA and K to DelAuth
$\mathbf{c} = \{\}$
AllMap = {}
    For all f_i
    Run the Update algorithm with op = add to generate c_{f_i} and Map_i
$\mathbf{c} = \mathbf{c} \cup c_{f_i}$
AllMap = [{AllMap $\cup$ Map_i}, $c_{id(f_i)}$]
Send (AllMap, $\mathbf{c}$) to the CSP
Send $\mathsf{In}_{\mathsf{TA}} = \{\mathsf{No.Files}, \mathsf{No.Search}\}$ to the TA
CSP stores AllMap in $\mathsf{In}_{\mathsf{CSP}} = \mathsf{Dict}$

Fig. 1: Setup Algorithm

Theorem 1. *Let* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a CPA-secure symmetric key encryption scheme. Moreover, let* G *be an IPRF and* h *a cryptographic hash function. Then our construction is* $\mathcal{L}$*-adaptively secure according to definition 2.*

Proof Sketch To prove the security of our construction against the threat model defined in Section 3, we construct a simulator $\mathcal{S}$ that simulates a perfect view of the real world for a PPT adversary $\mathcal{ADV}$. To do, we make use of a hybrid argument. More precisely, we will design five hybrids $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ and $\mathcal{H}_4$ such that $\mathcal{H}_0$ is the real experiment and $\mathcal{H}_4$ the ideal one. Each hybrid $\mathcal{H}_i$, will be constructed by replacing a real functionality with a simulated one given the corresponding leakage function $\mathcal{L}_i$. Our goal is to prove that no PPT adversary $\mathcal{ADV}$ will be able to distinguish between two consecutive hybrids $\mathcal{H}_i, \mathcal{H}_{i+1}$. The hybrids are illustrated in Table 3. The complete formal proof can be found in the full version of the paper along with a more detailed presentation of our scheme.

Table 3: Hybrid Description

Hybrids	**Description**
$\mathcal{H}_0$	This is the real experiment
$\mathcal{H}_1$	Simulate Setup given $\mathcal{L}_{stp}$
$\mathcal{H}_2$	Simulate Search given $\mathcal{L}_{search}$
$\mathcal{H}_3$	Simulate Update given $\mathcal{L}_{update}$
$\mathcal{H}_4$	This is the ideal experiment

SEARCH:

User sends $h(w_j)$ to the TA

TA:

$\mathsf{K_{w_j}} = G(\mathsf{K_G}, h(w_j)||\mathsf{No.Search[w_j]})$

$L_{up} = \{\}$

$\mathsf{No.Search[w_j]} + +$

$\mathsf{K'_{w_j}} = G(\mathsf{K_G}, h(w_j)||\mathsf{No.Search[w_j]})$

for $i = 1$ to $i = \mathsf{No.Files[w_j]}$

$\text{addr}'_{\mathsf{w_j}} = h(\mathsf{K_w}', i)$

$L_{up} = L_{up} \cup \{\text{addr}_{\mathsf{w_j}}\}$

Send $\tau_s(w_j) = (\mathsf{K_{w_j}}, \mathsf{No.Files[w_j]}, L_{up})$ to the CSP

CSP:

$R_{w_j} = \{\}$

for$i = 1$ to $i = \mathsf{No.Files[w_j]}$

$\text{val}_{\mathsf{w_j}} = \mathsf{Dict}[h(\mathsf{K_{w_j}}), i]$

$R_{w_j} = R_{w_j} \cup \{\text{val}_{\mathsf{w_j}}\}$

Forward R_{w_j} and $\mathsf{K_{w_j}}'$ to the DelAuth

DelAuth:

Send R_{w_j} to the user and an acknowledgement to the CSP.

CSP:

Delete all Dict entries associated with w_j and insert the addresses from L_{up}

UPDATE:

if $op =$ add

Map $= \{\}$

for all $w_{i_j} \in f_i$

$\mathsf{No.Files}[w_{i_j}] + +$

$\mathsf{K_{w_{i_j}}} = G(\mathsf{K_G}, h(w_{i_j})||\mathsf{No.Search[w_{i_j}]})$

$\text{addr}_{\mathsf{w_{i_j}}} = h(\mathsf{K_{w_{i_j}}}, \mathsf{No.Files[w_{i_j}]})$

$\text{val}_{\mathsf{w_{i_j}}} = \mathsf{Enc}(\mathsf{K_{SKE}}, id(f_i)||\mathsf{No.Files[w_{i_j}]})$

Map $= \{\text{addr}_{\mathsf{w_{i_j}}}, \text{val}_{\mathsf{w_{i_j}}}\}$

$c_{f_i} \leftarrow \mathsf{Enc}(\mathsf{K_{SKE}}, f_i)$

Send $\tau_\alpha(f_i) = (c_{f_i}, \text{Map})$ to the CSP

else Initiate the Search protocol for a keyword w_j

After DelAuth forwards R to the user:

for all $c_{id(f_j)} \in R$

$\mathsf{Dec}(\mathsf{K_{SKE}}, c_{id(f_j)}) \rightarrow id(f_j)$

if $\exists \ell : id(f_\ell) \in L_{TBD}$

$L_{del} = \{\}$

Compute $\mathsf{No.Search[w_j]}$ from $\mathsf{K_{w_j}}$

$\mathsf{No.Files[w_j]} - -$

$\mathsf{No.Search[w_j]} + +$

$\mathsf{K_{w_j}} = G(\mathsf{K_G}, h(w_j)||\mathsf{No.Search}[w_j])$

for $i = 1$ to $i = \mathsf{No.Files[w_j]} - 1$ $\quad$ new_addr $= h(K_{w_j}||i)$

new_value $= \mathsf{Enc}(\mathsf{K_{SKE}}, id(f_i)||\mathsf{No.Files}[w_j])$

$L_{del} = \{(\text{new\_addr}, \text{new\_value})\}$

else

Send $\tau_d(f_\ell) = L_{up}$ to the CSP

CSP:

Delete all Dict entries associated with w_j and insert the addresses contained in L_{del}

User:

Update the local indexes in send an acknowledgement to the TA to update its indexes as well

Fig. 2: Search and Update Algorithms

6 Experimental Results

We implemented our scheme in Python 2.7 using the PyCrypto library [1]. To test the overall performance, we used files of different sizes and structures. More precisely, we used a collection of five datasets provided in [18]. Table 4a shows the datasets used in our experiments as well as the total number of unique keywords extracted from each set. Our experiments focused on two main aspects: *(1)* Indexing and *(2)* Searching for a specific keyword. Deletion cannot be realistically measured since to completely delete all entries corresponding to a file, we first need to search for all the keywords contained in the file. Additionally, our dictionaries were implemented as tables in a MySQL database. In contrast to other similar works, we did not rely on the use of data structures such as arrays, maps, sets, lists, trees, graphs, etc. While the use of a database system decreases the overall performance of the scheme it is considered as more durable and close to a production level. Conducting our experiments by solely relying on data structures would give us better results. However, this performance would not give us any valuable insights about how the scheme would perform outside of a lab. Additionally, storing the database in RAM raises several concerns. For example a power loss or system failure could lead to data loss (because RAM is volatile memory). To this end, we ran our experiments in the following two different machines: ***(1)***, AMD Ryzen™ 7 PRO 1700 Processor at 3.0GHz (8 cores), 32GB of RAM running Windows 10 64-bit with AMD SEV Support and ***(2)*** Microsoft Surface Book laptop with a 4.2GHz Intel Core i7 processor (4 cores) and 16GB RAM running Windows 10 64-bit

The reason for measuring the performance on such machines and not only in a powerful desktop – like other similar works – is that in a practical scenario, the most demanding processes of any SSE scheme (e.g. the creation of the dictionary) would take place on a user's machine.

Indexing & Encryption: In our experiments, we measured the total setup time for each one of the datasets shown in table 4a. Each process was run ten times on the commodity laptop and the average time for the completion of the entire process was measured. As can be seen from table 4b, the setup time can be considered as practical and can even run in typical users' devices. We compare the setup times for the commodity laptop and the powerful desktop. Based on the fact that this phase is the most demanding one in an SSE scheme the time needed to index and encrypt such a large number of files is considered as acceptable not only based on the size of the selected dataset but also based on the results of other schemes that do not even offer forward privacy as well as on the fact that we ran our experiments on commodity machines and not on a powerful server. This is an encouraging result and we hope that will motivate researchers to design and implement even better and more efficient SSE schemes but most importantly we hope that will inspire key industrial players in the field of cloud computing to create and launch modern cloud-services based on the promising concept of Symmetric Searchable Encryption.

Search: In this part of the experiments we measured the time needed to complete a search over encrypted data. In our implementation, the search time is

Table 4: Dataset Sizes and Setup Times

No of TXT Files	Dataset Size	Unique Keywords	(w, id) pairs
425	184MB	1,370,023	5,387,216
815	357MB	1,999,520	10,036,252
1,694	670MB	2,688,552	19,258,625
1,883	1GB	7,453,612	28,781,567
2,808	1.7GB	12,124,904	39,747,904

(a) Size of Datasets and Unique Keywords

Dataset \ Testbed	Laptop	Desktop
184MB	22.48m	8.49m
357MB	40.00m	13.51m
670	86.43m	29.51m
1GB	141.60m	48.99m
1.7GB	203.28m	68.44m

(b) Setup time (in minutes)

calculated as the sum of the time needed to generate a search token and the time required to find the corresponding matches at the database. It is worth mentioning that the main part of this process will be running on the CSP (i.e. a machine with a large pool of resources and computational power). To this end, the time to generate the search token was measured on the laptop while the actual search time was measured using the desktop machine described earlier. On average, the time needed to generate the search token on the Surface Book laptop was $9\mu s$. Regarding the actual search time, searching for a specific keyword over a set of 12,124,904 distinct keywords and 39,747,904 addresses required 1.328sec on average while searching for a specific keyword over a set of 1,999,520 distinct keywords and 10,036,252 addresses took 0.131sec.

References

1. PyCrypto – the Python cryptography toolkit (2013), `https://pypi.org/project/pycrypto/`
2. AMD: Secure Encrypted Virtualization API Version 0.22. Tech. rep. (07 2019)
3. Amjad, G., Kamara, S., Moataz, T.: Forward and backward private searchable encryption with sgx. In: Proceedings of the 12th European Workshop on Systems Security. EuroSec '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3301417.3312496, `https://doi.org/10.1145/3301417.3312496`
4. Bakas, A., Dang, H.V., Michalas, A., Zalitko, A.: The cloud we share: Access control on symmetrically encrypted data in untrusted clouds. IEEE Access **8**, 210462–210477 (2020)
5. Bakas, A., Michalas, A.: Modern family: A revocable hybrid encryption scheme based on attribute-based encryption, symmetric searchable encryption and sgx. In: International Conference on Security and Privacy in Communication Systems. pp. 472–486. Springer (2019)
6. Bakas, A., Michalas, A.: Power range: Forward private multi-client symmetric searchable encryption with range queries support. In: 2020 IEEE Symposium on Computers and Communications (ISCC). pp. 1–7. IEEE (2020)
7. Boneh, D., Kim, S., Wu, D.J.: Constrained keys for invertible pseudorandom functions. In: Theory of Cryptography Conference. pp. 237–263. Springer (2017)
8. Bost, R.: $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016 (2016)

9. Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1465–1482. CCS '17, ACM, New York, NY, USA (2017)
10. Brasser, F., Hahn, F., Kerschbaum, F., Sadeghi, A.R., Fuhry, B., Bahmani, R.: Hardidx: Practical and secure index with sgx (2017)
11. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM (2015)
12. Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptology ePrint Archive **2016**(086), 1–118 (2016)
13. 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)
14. Garg, S., Mohassel, P., Papamanthou, C.: Tworam: efficient oblivious ram in two rounds with applications to searchable encryption. In: Annual International Cryptology Conference. pp. 563–592. Springer (2016)
15. Ghareh Chamani, J., Papadopoulos, D., Papamanthou, C., Jalili, R.: New constructions for forward and backward private symmetric searchable encryption. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1038–1055 (2018)
16. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: Ndss. vol. 20, p. 12. Citeseer (2012)
17. Kim, K.S., Kim, M., Lee, D., Park, J.H., Kim, W.H.: Forward secure dynamic searchable symmetric encryption with efficient updates. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1449–1463 (2017)
18. Michalas, A.: Text files from gutenberg database (Aug 2019). https://doi.org/10.5281/zenodo.3360392, `https://doi.org/10.5281/zenodo.3360392`
19. Mofrad, S., Zhang, F., Lu, S., Shi, W.: A comparison study of intel sgx and amd memory encryption technology. In: Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy. pp. 1–8 (2018)
20. Naveed, M.: The fallacy of composition of oblivious ram and searchable encryption. IACR Cryptology ePrint Archive **2015**, 668 (2015)
21. 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
22. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. In: NDSS. vol. 71, pp. 72–75 (2014)
23. Sun, S.F., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S.: Practical backward-secure searchable encryption from symmetric puncturable encryption. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 763–780 (2018)
24. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: 25th USENIX Security Symposium). pp. 707–720 (2016)