Dynamic Verifiable Encrypted Keyword Search Using Bitmap Index and Homomorphic MAC

Rajkumar Ramasamy[1], S.Sree Vivek[1], Praveen George[1], and Bharat S. Rawal Kshatriya[2]

[1] Samsung R&D Institute India, Bangalore, India.
{rajkumar.r, sreevivek.s, praveen.gk}@samsung.com
[2] Penn State Abington, Abington, PA 19001, USA,
bsr17@psu.edu

Abstract. Outsourcing data storage to the cloud securely and retrieving the remote data in an efficient way is a very significant research topic, with high relevance to secure cloud deployment. With the ever growing security and privacy concerns, encrypting the data stored remotely is inevitable but using traditional encryption thwarts performing search operation on the encrypted data. Encrypted keyword search is a cryptographic setting, which offers search functionality and at the same time, ensures security and privacy of the remotely stored data.
Searchable Symmetric Encryption (SSE) is a technique to securely outsource the data, which is encrypted using symmetric key primitives, while maintaining search functionality. While several solutions have been proposed to realize SSE over various data structures, the efficient solution using inverted index is due to Curtmola et.al. Hwang et.al. introduced a SSE scheme based on bitmaps inorder to reduce the index size.
In this paper, we consider Searchable Symmetric Encryption (SSE) in the presence of a Semi-Honest-But-Curious Cloud Service Provider (SHBC-CSP). We have defined a new security notion for SSE in presence of SHBC-CSP, contrived two new SSE schemes and proved their security formally in the proposed security notion. Dynamic Verifiable Encrypted Keyword Search (DVSSE), is the first SSE scheme to the best of our knowledge, which is both dynamic and verifiable. We have implemented our schemes, compared their performance and complexity with existing schemes.
Index terms— Search on protected or encrypted data, Encrypted Keyword Search, Dynamic and Verifiable Encrypted Keyword Search, Homomorphic Tag, Searchable Encryption, Symmetric Searchable Encryption, Random Oracle Model

1 Introduction

Motivation for Searchable Encryption. Cloud storage is a fast growing business model, which offers storage medium in which users can store data remotely. Various advantages provided by the cloud storage model, like omni-presence of data, reduced infrastructure cost, availability and reliability at less cost, and

recent advancements in networking technology promotes users to use such a service. Along with the growth of the cloud storage paradigm, lot of security and privacy concerns have grown widely.

There are several robust and provably secure cryptographic primitives to ensure confidentiality and privacy in a traditional setup. The goal of encryption is to transform the data into ciphertext, such that the ciphertext does not reveal any relation to the plaintext, unless the trapdoor to decryption is made available. When we resort to traditional encryption schemes to address the security and privacy issues in cloud storage, the solution comes with the hindrance to functionality, like searching or performing computation on the data.

The collective situation (of growing reliance on cloud storage and pressing need for security and privacy) demands a new method of encryption which will permit searching and computation on the encrypted data. In the past, there are several constructions proposed to address this problem. This area of research can be broadly categorized into the following:

1. Research to design new cryptographic primitives.
2. Research to design new data structures for Searchable Encryption.
3. Research to improve the security notions.

In this paper, we focus on designing an efficient data structure and improving the security notions of encrypted keyword search problem.

Searchable Encryption. Searchable encryption [7] allows searching on the encrypted data, without sacrificing confidentiality. There are two known ways of searching on encrypted data. In the first approach, keywords can be searched over ciphertext without any special data structures. The first practical searchable encryption scheme of this kind was proposed by Song et.al in [11]. This class of searching leads to search complexity linear in the size of encrypted data stored in the server. Following that, several advancements were proposed for searching on the encrypted data [4][3][9][8][12]. The second approach of searchable encryption is to generate a data structure, which enables indexing keywords of the document in encrypted form and performing search operation on the secure index [5][4].

Index-Based SSE: In the index based SSE approach, client creates a secure index $\mathcal{I}$ from the set of files $\mathbf{F}$ and finite list of keywords $\mathbf{W}$ associated with those files. The secure index is made up of file table T_f and search table T_s. T_f is made up of encrypted files c_i corresponding to files $f_i \in \mathbf{F}$, indexed using file identifiers $id(f_i)$. T_s contains a data structure that stores keyword identifiers $id(w_i)$ of unique keywords $w_i \in \mathbf{W}$. The client shall upload $\mathcal{I}$ to the Cloud Service Provider (CSP) and performs search operation on encrypted data stored in the CSP, to obtain list of files associated with the search keyword.

Verifiable and Dynamic SSE: There are two important attributes of an SSE scheme: a) Verifiability b) Dynamism. A SSE scheme is verifiable if it provides a mechanism to verify the outcome of the search query. This primitive is constructed and studied widely using cryptographic signature techniques and Merkle hash trees. A dynamic SSE scheme provides a method to add or delete files and keywords, to the secure index $\mathcal{I}$ present in the remote server. Verifiability and dynamism are two important for an SSE to be useful in practice.

Scheme	Dynamism	Verifiability	Security	Search Time	Access Pattern
Song et al. [11]	Static	No	CPA	$\mathcal{O}(\|\mathbf{F}\|)$	Hidden
Goh [5]	Dynamic	No	CKA1	$\mathcal{O}(N)$	Not Hidden
Cutrmola et al. [4]	Static	No	CKA2	$\mathcal{O}(M)$	Not Hidden
Kamara et al. [9]	Dynamic	No	CKA2	$\mathcal{O}(k')$	Not Hidden
Hwang et al. [6]	Dynamic	No	CKA2	$\mathcal{O}(M)$	Not Hidden
Chai et al. [2]	Static	Yes	CKA2	$\mathcal{O}(M)$	Not Hidden
This work	Dynamic	Yes	CKA2 + CMA	$\mathcal{O}(M)$	Hidden

Table 1: Comparison among Hwang et.al, VSSE and DVSSE.

(N: Total number of files; M: Total number of Keywords; k': Number of files associated with in any given keyword;)

Semi Honest But Curious Cloud Service Provider (SHBC-CSP): A Honest But Curious Cloud Service Provider (HBC-CSP) is expected to store the encrypted data securely and perform search operations honestly. It tries to learn any useful information of the stored data, from the system without deviating from the protocol. A Semi-Honest-But-Curious Cloud Service Provider (SHBC-CSP) stores the encrypted data honestly, tries to learn the underlying information from the system and may try to deviate from the protocol in a stealthy way. We consider this strong SHBC-CSP adversary who might manipulate search outcome selfishly in order to save its computation or download bandwidth [2].

1.1 Related work

Curtmola et al. [4] defined *IND-CKA2* adversarial model, which is widely used to prove the security of searchable encryption. *IND-CKA2* definition includes the confidentiality of the trapdoor and the data stored in the cloud. Curtmola et al. have also defined two new constructions among which, one is proved to be *IND-CKA1* secure and the other to be *IND-CKA2*. Kamara et al. [9] proposed a dynamic searchable encryption scheme and extended the *IND-CKA2* security model to dynamic *IND-CKA2*. Hwang et. al. has also proposed a dynamic encrypted keyword search scheme using bitmap index [6] and proved the security using dynamic *IND-CKA2* security model as same as [9]. Qi Chai et.al [2] proposed the first verifiable searchable symmetric key encryption scheme. In 2014, Christoph Bösch et.al. has surveyed SSE schemes and presented a comparison of several SSE schemes and their aspects in [1]. Table 1 provides a brief comparison of our scheme with existing schemes.

The scheme defined in [6] uses bitmap data-structure to index the encrypted files. Each encrypted file is assigned a unique number, starting from 0 to m (Number of the files in the cloud). Each unique keyword w has a unique keyword identifier $id(w)$ and a bitmap of size m, which provides a list of files having w. Bitmap is constructed such that, ith bit is set if encrypted file at index i in T_f contains the keyword w. To perform an encrypted keyword search, the cloud

user generates and sends the search trapdoor to the cloud. The cloud looks for the trapdoor entry in the index and returns the corresponding bitmap.

1.2 Our Contribution

In this paper, we analyze the security of encrypted keyword search in the presence of SHBC-CSP and provide a new security model to handle this realistic adversary. The scheme proposed in [6] lacks provision to verify the correctness of search outcome. Due to this, the IND-CKA2 secure scheme by Hwang et al. is insecure in the presence of SHBC-CSP.

We propose two new schemes to perform encrypted keyword search in the presence of SHBC-CSP and prove their security in the newly proposed security model. Our first scheme is an improvement to the existing bitmap based encrypted keyword search proposed in [6]. Though this scheme provides search outcome verifiability, it falls short of dynamic update. Our second scheme offers confidentiality of the bitmap, supports dynamic update and search outcome verifiability, which is required to avoid leakage of access pattern. To achieve dynamic update and search outcome verifiability, we have contrived a new homomorphic MAC using composite residuosity [10].

2 Definitions

Definition 1. *(Dynamic Verifiable SSE:) A dynamic verifiable searchable symmetric encryption (DVSSE) scheme is a tuple of nine algorithms DVSSE = $\langle$KeyGen, BuildIndex, SearchToken, AddToken, DeleteToken, Search, Verify, Add, Delete$\rangle$ such that:*

- $\langle \mathbf{K}, Params \rangle \leftarrow KeyGen(\kappa)$*: is a probabilistic algorithm that takes the security parameters κ as input and outputs the key $\mathbf{K}$ and system parameters.*
- $\mathcal{I} \leftarrow BuildIndex(\mathbf{F}, \mathbf{K})$*: is a probabilistic algorithm executed by the user, that takes the set of plaintext files $\mathbf{F}$ and the key $\mathbf{K}$ as input and outputs the secure index $\mathcal{I}$.*
- $\tau_s \leftarrow SearchToken(w, \mathbf{K})$*: is a deterministic algorithm executed by the user, that takes the key $\mathbf{K}$ and search keyword w as input and outputs the search token τ_s.*
- $\tau_a \leftarrow AddToken(f, \mathbf{K})$*: is a probabilistic algorithm executed by the user, that takes the key $\mathbf{K}$ and file f as input and outputs the add token τ_a.*
- $\tau_d \leftarrow DeleteToken(id(f), \mathbf{K})$*: is a probabilistic algorithm executed by the user, that takes the key $\mathbf{K}$ and file identifier $id(f)$ as input and outputs the delete token τ_d.*
- $\langle x, Tag \rangle \leftarrow Search(\tau_s, \mathcal{I})$*: is a deterministic algorithm run by the CSP, which takes the secure index $\mathcal{I}$ and search token τ_s as input and outputs the obscured bitmap x and verification tag Tag.*
- $\{True, False\} \leftarrow Verify(Tag, x, id(w))$*: is an algorithm executed by the user, that takes verification tag Tag, obscured bitmap x, keyword identifier $id(w)$ and key $\mathbf{K}$ as input and outputs verification result of search outcome.*

- $\mathcal{I}' \leftarrow Add(\tau_a, Params, \mathcal{I})$*: is a probabilistic algorithm run by the CSP, that takes add token* τ_a*, public parameter* $Params$ *and secure index* $\mathcal{I}$ *as input and outputs updated secure index* $\mathcal{I}'$*.*
- $\mathcal{I}' \leftarrow Delete(\tau_d, Params, \mathcal{I})$*: is a probabilistic algorithm run by the CSP, that takes delete token* τ_d*, public parameter* $Params$ *and secure index* $\mathcal{I}$ *as input and outputs updated secure index* $\mathcal{I}'$*.*

Note: A static VSSE scheme will have all above algorithms except $AddToken$, $DeleteToken$, Add and $Delete$.

Definition 2. *Message Authentication Code (MAC) is a triple of three algorithms* $MAC = \langle KeyGen,\ TagGen,\ TagVerify\ \rangle$*.*

- $\{K, Params\} \leftarrow KeyGen(\kappa)$ *: is a probabilistic algorithm that takes the security parameters* κ *as input and outputs the key* K *and system parameters.*
- $Tag \leftarrow TagGen(K, M)$ *: is a probabilistic algorithm which takes the key* K *and a message* M *as input and outputs the MAC* Tag*.*
- $\{Accept, Reject\} \leftarrow Verify(K, M, Tag)$ *: is an algorithm that takes the key* K*, message* M *and MAC tag* (Tag) *as input and outputs* $\{Accept, Reject\}$*.*

Definition 3. *(Dynamic IND-CKA2-security) Let* $DVSSE = \langle KeyGen,\ BuildIndex,\ SearchToken,\ AddToken,\ DeleteToken,\ Search,\ Verify,\ Add,\ Delete\rangle$ *be a dynamic verifiable SSE scheme. The IND-CKA2 security is defined as a game between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{C}$*.*
Setup*:* $\mathcal{C}$ *generates the key* $\boldsymbol{K}$ *of DVSSE scheme and sets the system parameters* $Params$*.* $\mathcal{A}$ *chooses a document collection* $\boldsymbol{F}$*, generates the list of keywords* $\boldsymbol{W}$ *present in* $\boldsymbol{F}$ *and chooses a keyword* $w^* \in \boldsymbol{W}$ *and gives it to* $\mathcal{C}$*.* $\mathcal{C}$ *generates the secure index* $\mathcal{I}$ *for* F *and returns it to* $\mathcal{A}$*.* $\mathcal{C}$ *choose a bit* $b \xleftarrow{R} \{0, 1\}$*. If* $b = 0$*,* $\mathcal{C}$ *sets the actual encrypted bitmap* x^* *(corresponding to* w^**). If* $b = 1$*,* $\mathcal{C}$ *chooses a random* x^* *in the range of encrypted bitmap.*
Training Phase*:* $\mathcal{A}$ *is allowed to query the following oracles, with a restriction that,* $\mathcal{A}$ *should not make any oracle query corresponding to* w^**.*

- $\mathcal{O}_{SearchToken}(w)$*: Returns search token* τ_s*.*
- $\mathcal{O}_{AddToken}(f)$*: Returns add token* τ_a*.*
- $\mathcal{O}_{DeleteToken(f)}$*: Returns delete token* τ_d*.*
- $\mathcal{O}_{Verify(Tag,x,id(w))}$*: Returns Accept or Reject.*
- $\mathcal{O}_{BitmapDecryption(x)}$*: Returns* $B_m(w)$*.*

Challenge Phase*:* $\mathcal{C}$ *gives* $id(w^*)$ *to* $\mathcal{A}$*.* ***Guess****:* $\mathcal{A}$ *outputs a bit* b'*, which is* 0 *if* x^* *encrypts the bitmap corresponding to* w^**, else outputs* 1*.*

The scheme is IND-CKA2 secure if for all probabilistic polynomial time adversaries $\mathcal{A}$*, there exists a negligible function* $negl(.)$ *such that:*

$$Pr[DVSSE_{\mathcal{A}}^{IND\text{-}CKA2}(\kappa) \rightarrow (b = b')] \leq \frac{1}{2} + negl(\kappa)$$

Definition 4. *(UF-CKA-security) Let DVSSE = ⟨ KeyGen, BuildIndex, SearchToken, AddToken, DeleteToken, Search, Verify, Add, Delete⟩ be a dynamic verifiable SSE scheme. The UF-CKA security of DVSSE is defined as a game between an challenger $\mathcal{C}$ and a forger $\mathcal{F}$.*

***Setup**: $\mathcal{C}$ generates the key $\boldsymbol{K}$ of DVSSE scheme and sets the systems parameters params. $\mathcal{F}$ chooses a document collection $\boldsymbol{F}$, generates the list of keywords $\boldsymbol{W}$ present in $\boldsymbol{F}$ and chooses a keyword $w^* \in \boldsymbol{W}$ and gives it to $\mathcal{C}$. $\mathcal{C}$ generates the secure index $\mathcal{I}$ for F and returns $\mathcal{I}$ to $\mathcal{F}$. $\mathcal{I}$ excludes the tag Tag^* for challenge keyword w^*.*

***Training Phase**: $\mathcal{F}$ is allowed to query the oracles defined in the training phase of IND-CKA2 game, with a restriction that, $\mathcal{F}$ should not make any query corresponding to w^*.*

***Forgery**: At the end of training, $\mathcal{F}$ outputs a forged tag Tag^* corresponding to the bitmap $B_m(w^*)$.*

The scheme is Unforgeable against Chosen Keyword Attack (UF-CKA) if for all probabilistic polynomial time forger $\mathcal{F}$, there exists a negligible function $negl(.)$ such that:

$$Pr[DVSSE_{\mathcal{F}}^{UF-CKA}(\kappa) \rightarrow (Tag^*) : Verify(Tag^*, x^*, id(w^*)) = Accept] \leq negl(\kappa)$$

Definition 5. *(Indistinuishability of MAC Tag (IND-CMA)) Let Message Authentication Code $MAC = \langle KeyGen,\ TagGen,\ TagVerify\rangle$. The IND-CMA security is defined as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.*

***Setup**: $\mathcal{C}$ generates the key K of MAC scheme and sets the systems parameters params.*

***Training Phase**: $\mathcal{A}$ can query the following oracles:*

- $\mathcal{O}_{TagGen}(M)$*: This oracle returns the MAC tag Tag.*
- $\mathcal{O}_{TagVerify}(M, Tag)$*: This oracle returns $\{Accept,\ Reject\}$.*

***Challenge**: $\mathcal{A}$ chooses a message M^* and sends it to $\mathcal{C}$. The challenger $\mathcal{C}$ selects a bit $b \xleftarrow{R} \{0,1\}$. If $b = 0$, $\mathcal{C}$ generates the tag $Tag^* = TagGen(K, M^*)$ and if $b = 1$, $\mathcal{C}$ chooses a random tag Tag^* in the range of the tag. $\mathcal{C}$ sends Tag^* to $\mathcal{A}$ as the challenge tag.*

***Guess**: $\mathcal{A}$ outputs a bit b', which is 0 if Tag^* is the tag corresponding to M^*, else outputs 1.*

The MAC is IND-CMA secure if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $negl(.)$ such that:

$$Pr[MAC_{\mathcal{A}}^{IND\text{-}CMA}(\kappa) \rightarrow (b = b')] \leq \frac{1}{2} + negl(\kappa)$$

Definition 6. *(Existential Unforgeability of MAC (EUF-CMA)) Let Message Authentication Code $MAC = \langle KeyGen, TagGen, TagVerify\rangle$. The EUF-CMA security is defined as a game between an adversary $\mathcal{A}$ and a forger $\mathcal{F}$.*

***Setup**: $\mathcal{F}$ generates the key K of MAC scheme and sets the systems parameters $params$.*

***Training Phase**: $\mathcal{F}$ is allowed to query all the oracles as in IND-CMA game.*

***Forgery**: $\mathcal{F}$ outputs a valid message tag pair $\langle M*, Tag* \rangle$, for which $M*$, $\mathcal{A}$ should not have queried to $TagGen$ oracle.*

The MAC is EUF-CMA secure if for all probabilistic polynomial time forger $\mathcal{F}$, there exists a negligible function $negl(.)$ such that:

$$Pr[MAC_{\mathcal{F}}^{EUF\text{-}CMA}(\kappa) \rightarrow \langle M, Tag \rangle : Verify(M, Tag) \rightarrow Accept] \leq negl(\kappa)$$

3 Verifiable SSE Scheme (VSSE)

3.1 Scheme

Our Verifiable Encrypted Keyword Search scheme consists of following five algorithms ($KeyGen, BuildIndex$, $SearchToken, Search, Verify$).
KeyGen(κ) : This algorithm is run by the user to generate the set of keys used in the scheme. Choose three cryptographic MAC's defined as follows:

- $H_1 : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$
- $H_2 : \{0,1\}^\kappa \times \{0,1\}^* \times \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$
- $H_3 : \{0,1\}^\kappa \times \{0,1\}^n \times \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$

Where, the first inputs are cryptographic keys. Output the key $\mathbf{K} = \langle K_1, K_2, K_e, K_h \rangle \xleftarrow{R} \{0,1\}^\kappa$
BuildIndex($\mathbf{F}, \mathbf{K}$) : This algorithm is run by the user to generate and output the secure index $\mathcal{I} = \langle T_f, T_s \rangle$. **Generation of T_f:** For $i = 1$ to m, compute $c_i = SKE.Enc(K_e, f_i)$ and store the tuple $\langle i, c_i \rangle$ as a row in T_f.

Generation of T_s: Extract the keywords in $\mathbf{F}$ and set $\mathbf{W} = \{w_0, \ldots, w_n\}$ and for all w_i in $\mathbf{W}$:

- Generate identifier $id(w_i) = H_1(K_1, w_i)$
- Choose $r_i \xleftarrow{R} \{0,1\}^\kappa$ and obtain mask $h_i = H_2(K_2, w_i, r_i)$
- Create the bitmap $B_m(w_i)[j] = 1$ for all $j \in$ index of encrypted files having the keyword w_i.
- Compute $x_i = B_m(w_i) \oplus h_i$ and $Tag_i = H_3(K_h, B_m(w_i), id(w_i))$
- Store $\langle key, value, Tag \rangle = \langle id(w_i), x_i || r_i, Tag_i \rangle$ as a row in T_s.

SearchToken($\mathbf{w}, \mathbf{K}$) : This algorithm is executed by the user to generate search token.

- Compute and output search token $\tau_s = \langle id(w) \rangle$

Search($\tau_\mathbf{s}, \mathcal{I}$) : This algorithm is executed by the CSP.

- Scan the search index $\mathcal{I}$ to locate $id(w_i)$.
- On success, return the tuple $\langle x_i || r_i, Tag_i \rangle$.

Verify$(\mathbf{x_i}||\mathbf{r_i}, \mathbf{Tag_i})$: This algorithm is run by the user to verify the correctness of the search outcome sent by CSP.

- Parse and extract r_i from $x_i||r_i$.
- Compute the keyword identifier $id(w_i) = H_1(K_1, w_i)$.
- Compute the mask $h_i = H_2(K_2, w_i, r_i)$.
- Extract the bitmap $B_m(w_i) = x_i \oplus h_i$.
- Calculate $Tag'_i = H_3(K_h, B_m(w_i), id(w_i))$.
- If $(Tag'_i = Tag_i)$ output $True$; else output $False$.

3.2 Security Analysis of VSSE

Theorem 1. *If SKE is an IND-CPA secure symmetric encryption scheme and H_2 is an IND-CMA secure cryptographic MAC, then the verifiable SSE scheme (VSSE) is IND-CKA2 secure in the random oracle model.*

Proof: Let $\mathcal{C}$ be the challenger who challenges an adversary $\mathcal{A}$ to break the *IND-CKA2* security of $VSSE$. Let $\mathcal{B}$ be an algorithm which challenges $\mathcal{C}$ to break the *IND-CMA* security of MAC. $\mathcal{B}$ provides oracle access to $TagGen$ and $TagVerify$ algorithms of the MAC scheme to $\mathcal{C}$. $\mathcal{C}$ makes use of the capability of $\mathcal{A}$ and the oracles provided by $\mathcal{B}$ to achieve its goal.

Setup: $\mathcal{C}$ generates three random keys $K_1, K_e, K_h \xleftarrow{R} \{0,1\}^\kappa$. $\mathcal{C}$ designs the MACs H_1, H_2 and H_3 as random oracles and maintains three lists L_1, L_2 and L_3 to answer the oracle queries consistently.

$\mathcal{A}$ chooses a document collection $\mathbf{F}$, generates the list of keywords $\mathbf{W}$ present in $\mathbf{F}$, chooses a keyword $w^* \in \mathbf{W}$ and gives them to $\mathcal{C}$. $\mathcal{C}$ uses $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$ *and* $\mathcal{O}_{H_3}$ oracles (defined in the training phase) and generates the secure index $\mathcal{I}$ for $\mathbf{F}$ as follows:

For all $f_i \in \mathbf{F}$, compute $c_i = SKE.Enc(K_e, f_i)$ and store the tuple $\langle i, c_i \rangle$ as a row in T_f. For all $w_i \in \mathbf{W}$:

- Queries $\mathcal{O}_{H_1}$ with w_i as input to obtains $id(w_i)$.
- If $w_i = w^*$, do the following:
  - $\mathcal{C}$ chooses $r^* \xleftarrow{R} \{0,1\}^\kappa$ and sends $\langle w^*, r^* \rangle$ to $\mathcal{B}$ for the challenge.
  - $\mathcal{B}$ selects a bit $b \xleftarrow{R} \{0,1\}$. If $b = 0$, $\mathcal{B}$ generates the tag $h^* = TagGen(w^*, r^*)$ and if $b = 1$, $\mathcal{C}$ chooses a random tag $h^* \xleftarrow{R} \{0,1\}^\kappa$. $\mathcal{B}$ sends h^* to $\mathcal{C}$.
- If $w_i \neq w^*$, choose $r_i \in_R \{0,1\}^\kappa$ and queries $\mathcal{O}_{H_2}$ with w_i and r_i as input to obtain h_i.
- Create the bitmap $B_m(w_i)[j] = 1$ for all $j \in$ index of encrypted files having the keyword w_i.
- Computes $x_i = B_m(w_i) \oplus h_i$ and query $\mathcal{O}_{H_3}$ with $B_m(w_i)$ and $id(w_i)$ as input to obtain Tag_i.

Stores $\langle key, value, Tag \rangle = \langle id(w_i), x_i||r_i, Tag_i \rangle$ in T_s.

Training Phase: $\mathcal{A}$ is allowed to query the following oracles, with a restriction that, $\mathcal{A}$ should not make any oracle query corresponding to w^*.

$\mathcal{O}_{H_1}$*:* Given keyword w_i as input, $\mathcal{C}$ checks whether a tuple of the form $\langle w_i, id(w_i)\rangle$ appears in the list L_1. If it is present, returns $id(w_i)$ to $\mathcal{A}$. If the tuple is not present, chooses $id(w_i) \xleftarrow{R} \{0,1\}^{\kappa}$, adds the tuple $\langle w_i, id(w_i)\rangle$ to the list L_1 and returns $id(w_i)$ to $\mathcal{A}$.

$\mathcal{O}_{H_2}$*:* When $\mathcal{A}$ makes a query to this oracle with w_i and r_i as input, $\mathcal{C}$ checks whether a tuple of the form $\langle w_i, r_i, h_i\rangle$ appears in the list L_2. If it is present, returns h_i. If the tuple is not present, $\mathcal{C}$, in turn queries the $TagGen$ oracle provided by $\mathcal{B}$ with $x_i||r_i$ as input and obtains the MAC tag h_i. $\mathcal{C}$ inserts the tuple $\langle w_i, r_i, h_i\rangle$ into the list L_2 and returns h_i.

$\mathcal{O}_{H_3}$*:* Given $B_m(w_i)$ and keyword identifier $id(w_i)$ as input, $\mathcal{C}$ checks whether a tuple of the form $\langle B_m(w_i)$,
$id(w_i), Tag_i\rangle$ appears in the list L_3. If it is present, returns Tag_i to $\mathcal{A}$. If the tuple is not present, chooses $Tag_i \xleftarrow{R} \{0,1\}^{\kappa}$, add the tuple $\langle B_m(w_i)$, $id(w_i), Tag_i\rangle$ to the list L_3 and returns Tag_i to $\mathcal{A}$.

$\mathcal{O}_{SearchToken}$*:* Given a keyword w, $\mathcal{C}$ queries $\mathcal{O}_{H_1}$ with w as input and returns the search token $\tau_s = id(w)$.

$\mathcal{O}_{Verify}$*:* Given $\langle Tag,\ x||r\rangle$ and keyword w as input, $\mathcal{C}$ obtains the keyword identifier $id(w)$ by querying $\mathcal{O}_{H_1}$, obtains h by querying $\mathcal{O}_{H_2}$ with w and r as input, and computes $B_m(w) = x \oplus h$. Queries $\mathcal{O}_{H_3}$ with $B_m(w)$ and $id(w)$ as input to obtain Tag'. If $Tag' = Tag$ returns *Accept*; else, returns *Reject*.

$\mathcal{O}_{BitmapDecryption}$*:* Given $x||r$ and keyword w as input, this oracle obtains the keyword identifier $id(w)$ by querying $\mathcal{O}_{H_1}$, obtains h by querying $\mathcal{O}_{H_2}$ with w and r as input. $\mathcal{C}$ computes and returns $B_m(w) = x \oplus h$.

Challenge Phase: $\mathcal{C}$ gives the keyword identifier $id(w^*)$ to $\mathcal{A}$ by querying $\mathcal{O}_{H_1}$.

Guess: $\mathcal{A}$ outputs a bit b' to $\mathcal{C}$. $\mathcal{C}$ sends b' to $\mathcal{B}$. □

Correctness: Note that h^* obtained from $\mathcal{B}$ is used to generate x^* corresponding to w^*. In order to distinguish the encrypted bitmap x^*, $\mathcal{A}$ should have computed h^*. It should be further noted that $\mathcal{A}$ is not allowed to query $\mathcal{O}_{H_2}$oracle corresponding to w^* and hence restricted from obtaining it from $\mathcal{C}$. Let this computed value be $h^{*'}$. If h^*, obtained by $\mathcal{C}$ from $\mathcal{B}$ to generate x^*, is a valid MAC tag corresponding to w^* and r^*, then $h^* = h^{*'}$. Hence, $\mathcal{A}$ would have output a valid guess bit b'. If h^* obtained from $\mathcal{B}$ is not a valid MAC tag, corresponding to w^* and r^*, then $h^* \neq h^{*'}$. Even in this case, $\mathcal{A}$ would have inferred x^* is a random bitmap and would have output a valid guess bit b'. Finally, the indistinguistability of c_is(The individual encrypted files) follow from the *IND-CPA* security of SKE.

Theorem 2. *If H_3 is an EUF-CMA secure cryptographic MAC, then the verifiable SSE scheme (VSSE) is UF-CKA secure in the random oracle model.*

Proof: Let $\mathcal{C}$ be the challenger who challenges a forger $\mathcal{F}$ to break the *UF-CKA* security of $VSSE$. Let $\mathcal{B}$ be an algorithm which challenges $\mathcal{C}$ to break the *EUF-CMA* security of MAC. $\mathcal{B}$ provides oracle access to $TagGen$ and $TagVerify$

algorithms of the MAC scheme to $\mathcal{C}$. $\mathcal{C}$ makes use of the capability of $\mathcal{F}$ and the oracles provided by $\mathcal{B}$ to achieve its goal.

Setup: $\mathcal{F}$ generates three random keys $K_1, K_2, K_e \xleftarrow{R} \{0,1\}^\kappa$. $\mathcal{C}$ designs the MACs H_1, H_2 and H_3 as random oracles and maintains three lists L_1 L_2 and L_3 to answer the oracle queries consistently.

$\mathcal{A}$ chooses a document collection $\mathbf{F}$, generates the list of keywords $\mathbf{W}$ present in $\mathbf{F}$ and chooses a keyword $w^* \in \mathbf{W}$ and gives them to $\mathcal{C}$. $\mathcal{C}$ uses $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$ *and* $\mathcal{O}_{H_3}$ oracles (defined in the training phase) to generate the secure index $\mathcal{I}$ for $\mathbf{F}$ as follows:
For all $f_i \in \mathbf{F}$, compute $c_i = SKE.Enc(K_e, f_i)$ and store the tuple $\langle i, c_i \rangle$ as a row in T_f. For all $w_i \in \mathbf{W}$:

- Queries $\mathcal{O}_{H_1}$ with w_i as input to obtain $id(w_i)$.
- Chooses $r_i \in_R \{0,1\}^\kappa$ and queries $\mathcal{O}_{H_2}$ with w_i and r_i as input to obtain h_i.
- Creates the bitmap $B_m(w_i)[j] = 1$ for all $j \in$ index of encrypted files having the keyword w_i.
- If $w_i \neq w^*$, computes $x_i = B_m(w_i) \oplus h_i$ and queries $\mathcal{O}_{H_3}$ with $B_m(w_i)$ and $id(w_i)$ as input to obtain Tag_i.
- If $w_i = w^*$, computes $x_i = B_m(w_i) \oplus h_i$ and sets $Tag_i = NULL$ ($\mathcal{C}$ does not query $\mathcal{O}_{H_3}$ in this case.)

Stores $\langle key, value, Tag \rangle = \langle id(w_i), x_i || r_i, Tag_i \rangle$ in T_s.

Training Phase: $\mathcal{F}$ is allowed to query the following oracles, with a restriction that, $\mathcal{F}$ should not make any oracle query corresponding to w^*.

Oracles $\mathcal{O}_{H_1}$, $\mathcal{O}_{BitmapDecryption}$, $\mathcal{O}_{SearchToken}$ and $\mathcal{O}_{Verify}$ are similar to the corresponding oracles defined in the *IND-CKA2* proof.

$\mathcal{O}_{H_2}$*:* Given keyword w_i and r_i as input, $\mathcal{C}$ checks whether a tuple of the form $\langle w_i, r_i, h_i \rangle$ appears in the list L_2. If it is present, returns h_i. If the tuple is not present, $\mathcal{C}$ chooses $h_i \xleftarrow{R} \{0,1\}^\kappa$. $\mathcal{C}$ inserts the tuple $\langle w_i, r_i, h_i \rangle$ into the list L_2 and returns h_i.

$\mathcal{O}_{H_3}$*:* Given $B_m(w_i)$ and keyword identifier $id(w_i)$ as input, $\mathcal{C}$ checks whether a tuple of the form $\langle B_m(w_i), id(w_i), Tag_i \rangle$ appears in the list L_3. If it is present, returns Tag_i to $\mathcal{F}$. If the tuple is not present, $\mathcal{C}$, in turn queries the $TagGen$ oracle provided by $\mathcal{B}$ with $B_m(w_i)$, $id(w_i)$ as input and obtains the MAC tag Tag_i, adds the tuple $\langle B_m(w_i), id(w_i), Tag_i \rangle$ to the list L_3 and returns Tag_i to $\mathcal{F}$.

Forgery: At the end of training, $\mathcal{F}$ outputs a forged tag Tag^* corresponding to the bitmap $B_m(w^*)$ and keyword identifier $id(w^*)$. $\mathcal{C}$ sends Tag^* to $\mathcal{B}$ as the forgery for the MAC corresponding to $\langle B_m(w^*)$ and $id(w^*) \rangle$ as message. □

Correctness: In order to output a valid forgery Tag^* corresponding to the bitmap $B_m(w^*)$ and keyword identifier $id(w^*)$, $\mathcal{F}$ should have computed $H_3(B_m(w^*), id(w^*))$. Hence, if Tag^* is a valid forgery, it suffices that the forgery submitted by $\mathcal{C}$ to $\mathcal{B}$ is also valid. □

4 Dynamic Verifiable SSE Scheme (DVSSE)

4.1 Scheme

KeyGen(κ_1, κ_2) : This algorithm generates the keys and sets the system parameters which are used in the system.

- Generate four keys $K_1, K_e, K_h, K_w \xleftarrow{R} \{0,1\}^{\kappa_1}$.
- Choose two κ_2 bit safe primes p, q and compute $n = pq$, compute $\phi(n^2) = pq(p-1)(q-1)$ and $\lambda(n) = lcm(p-1, q-1)$.
- Choose two integers $\alpha, \beta < n$ and compute $g = 1 + \alpha n$ and $h = 1 + \beta n$.
- Choose two cryptographic hash functions defined as $H_1 : \{0,1\}^{\kappa_1} \times \{0,1\}^* \rightarrow \{0,1\}^{\kappa_1}$ and $H_2 : \{0,1\}^{\kappa_1} \times \{0,1\}^* \times \{0,1\}^{\kappa_1} \rightarrow \{0,1\}^{\kappa_1}$

Output the private keys of the user $\mathbf{K} = \langle K_1, K_e, K_h, p, q, \phi(n), \lambda(n), g, h\rangle$ and send $\langle n^2\rangle$ to the CSP in order to perform the add and delete operations.

BuildIndex$(\mathbf{F}, \mathbf{K})$: This algorithm generates the secure index $\mathcal{I}$ and stores in the cloud.

Initialization: Generate the set of distinct keywords $\mathbf{W}$ from $\mathbf{F}$ and for each file f_i in $\mathbf{F}$, assign file identifiers in the range of 1 to m and build secure index, $\mathcal{I} = \langle T_f, T_s\rangle$.

Generation of T_f: For all f_i in $\mathbf{F}$, do as follows.

- Extract the keywords in f_i and set $W_i = \{w_1, \ldots, w_{n'}\}$, where n' is the number of keywords in f_i
- Compute $d_i = SKE.Enc(K_w, W_i)$.
- Compute $c_i = SKE.Enc(K_e, f_i)$.
- Store the tuple $\langle i, c_i, d_i\rangle$ as a row in T_f.

Generation of T_s: Extract the keywords in $\mathbf{F}$ and set $\mathbf{W} = \{w_1, \ldots, w_m\}$ and for all w_i in $\mathbf{W}$, do the following.

- Generate identifier $id(w_i) = H_1(K_1, w_i)$.
- Generate a bitmap index $B_m(w_i)$ and set all bits to 0.
- Generate two random number $r_i, s_i \xleftarrow{R} \mathbb{Z}^*_{n^2}$
- Set $B_m(w_i)[j] = 1 \; \forall \; j : \{f_j$ has the keyword $w_i\}$.
- Compute $x_i = g^{B_m(w_i)} r_i^n \; mod \; n^2$, and
- $y_i = \left(\sum_{j=1:B_m(w_i)[j]=1}^{N} H_2(K_h, j, id(w_i)) \right) \; mod \; \phi(n^2)$
- Compute $Tag_i = h^{y_i} s_i^n \; mod \; n^2$
- Store the tuple $\langle key, value, Tag\rangle = \langle id(w_i), x_i, Tag_i\rangle$ as a row in T_s.

Set the secure index $\mathcal{I} = \langle T_s, T_f\rangle$.

SearchToken$(\mathbf{w_i}, \mathbf{K})$: This algorithm will be run by the user to generate the search token.

– Compute and return the search token $\tau_s = \langle id(w_i) \rangle$.

Search$(\tau_{\mathbf{s}}, \mathcal{I})$: This algorithm will be executed in the cloud by the CSP.

– Scan the search index $\mathcal{I}$ to locate $id(w_i)$.
– On success, extract and return $\langle x_i, Tag_i \rangle$; else $\perp$.

Verify$(\mathbf{Tag_i}, \mathbf{x_i}, \mathbf{id(w_i)})$: This algorithm verifies the correctness of the search outcome.

– Compute $B_m(w_i) = \dfrac{L(x_i^{\lambda} \ mod \ n^2)}{L(g^{\lambda} \ mod \ n^2)}$ and $y_i = \dfrac{L(Tag_i^{\lambda} \ mod \ n^2)}{L(h^{\lambda} \ mod \ n^2)}$, where $L(u) = \left(\dfrac{u-1}{n}\right)$, and
– $y_i' = \left(\sum\limits_{j=1:B_m(w_i)[j]=1}^{N} H_2(K_h, j, id(w_i)) \right) \ mod \ \phi(n^2)$.
– Check whether $y_i \stackrel{?}{=} y_i'$.

If the verification hold, output *Accept*; else output *Reject*.

AddToken$(\mathbf{f}, \mathbf{K})$: This algorithm will be run by the user to generate the addition token. Let the number of files already stored in the database be z.

– Compute $c = SKE.Enc(K_e, f)$ and
– Compute $d = SKE.Enc(K_h, W)$, where W is the set of keywords available in the file f.
– For all keywords w_i in $\mathbf{W}$, perform the following:
  • Generate $id(w_i) = H_1(K_1, w_i)$.
  • Generate two new random numbers $r_i, s_i \stackrel{R}{\leftarrow} \mathbb{Z}_{n^2}^*$.
  • Generate the bitmap addition token $x_i' = g^{2^q} r_i^n \ mod \ n^2$.
  • Generate the tag addition token $Tag_i' = h^{H_2(K_h, 2^z, id(w_i))} s_i^n \ mod \ n^2$.

Return $\tau_a = \langle \{id(w_i), x_i', Tag_i'\}, c, d \rangle$.

Add$(\tau_{\mathbf{a}}, \mathbf{n^2}, \mathcal{I})$: This algorithm will be executed by the CSP to add the new file. Let z be the number of files already stored in the database.

– Insert the tuple $\langle z+1, c, d \rangle$ as a new row in T_f
– For all the tuple $\langle id(w_i), x_i', Tag_i' \rangle$ in τ_a, perform the following:
  • If $id(w_i)$ is not present in T_s, Insert the tuple $\langle id(w_i), x_i, Tag_i \rangle = \langle id(w_i), x_i', Tag_i' \rangle$ as a new row in T_s.
  • Else,
    ∗ Let the position of $id(w_i)$ in T_s be j.
    ∗ Set $x_j = x_j x_i' \ mod \ n^2$
    ∗ Set $Tag_j = Tag_j Tag_i' \ mod \ n^2$

DeleteToken$(\mathbf{id(f)}, \mathbf{K})$: This algorithm is run by the user to generate the token to delete a file $f \in \mathbf{F}$.

– Let the location of f in T_f be z.
– Retrieve the set of encrypted keywords d associated with f and compute $W = SKE.Dec(K_w, d)$.
– For all w_i in W, compute keyword identifier $id(w_i)$.
– For all keyword identifier $id(w_i)$, do as follows:
  • Generate two random numbers $r_i, s_i \xleftarrow{R} \mathbb{Z}^*_{n^2}$.
  • Compute the bitmap modifier $x'_i = g^{-2^{z-1}} r_i^n \ mod\ n^2$.
  • Compute the modification tag $Tag'_i = h^{-H_2(K_h, 2^{z-1}, id(w_i))}\ s_i^n \ mod\ n^2$.
– Send the delete token $\tau_d = \langle id(f), \{id(w_i), x'_i, Tag'_i\}\rangle$ to CSP.

Delete$(\tau_\mathbf{d}, \mathbf{n^2}, \mathcal{I})$: This algorithm is run by the CSP to delete a file f.

– Locate and delete the contents of the row pointed by the file identifier $id(f)$ in T_f.
– Let the position of $id(f)$ in T_s be z.
– For all tuple $\langle id(w_i), x'_i, Tag'_i\rangle$ in τ_d, perform the following:
  • Set $x_i = x_i x'_i \ mod\ n^2$
  • Set $Tag_i = Tag_i Tag'_i \ mod\ n^2$

4.2 Security Analysis of DVSSE

Theorem 3. *If SKE is an IND-CPA secure symmetric encryption scheme and $\mathcal{PE}$ is an IND-CPA secure Paillier encryption scheme, then the dynamic verifiable SSE scheme (DVSSE) is IND-CKA2 secure in the random oracle model.*

Proof: Let $\mathcal{C}$ be the challenger who challenges an adversary $\mathcal{A}$ to break the *IND-CKA2* security of DVSSE. Let $\mathcal{B}$ be an algorithm which challenges $\mathcal{C}$ to break the *IND-CPA* security of $\mathcal{PE}$. $\mathcal{B}$ provides the system parameters of $\mathcal{PE}$ to $\mathcal{C}$. $\mathcal{C}$ makes use of the capability of $\mathcal{A}$ to achieve its goal.

Setup: $\mathcal{C}$ generates three random keys $\langle K_1, K_e, K_h\rangle \xleftarrow{R} \{0,1\}^\kappa$. $\mathcal{C}$ designs the MACs H_1 and H_2 as random oracles and maintains two lists L_1 and L_2 to answer the oracle queries consistently. In addition to these two lists, $\mathcal{C}$ also maintains another list L_3 to store the internal state. $\mathcal{B}$ provides the system public parameters $\langle n, n^2\rangle$ of $\mathcal{PE}$ to $\mathcal{C}$. $\mathcal{C}$ chooses $g, h \xleftarrow{R} \mathbb{Z}^*_{n^2}$. Finally $\mathcal{C}$ sends $\langle n, n^2\rangle$ to $\mathcal{A}$.

$\mathcal{A}$ chooses a document collection $\mathbf{F}$, generates the list of keywords $\mathbf{W}$ present in $\mathbf{F}$, chooses a keyword $w^* \in \mathbf{W}$ and gives them to $\mathcal{C}$. $\mathcal{C}$ sends $B_m(w^*)$ to $\mathcal{B}$ as a challenge message. $\mathcal{B}$ chooses a random bit $b \xleftarrow{R} \{0,1\}$ and sends x^* as the challenge ciphertext. The ciphertext x^* is the encryption of $B_m(w^*)$ if $b = 0$. Otherwise, $\mathcal{B}$ sends a random ciphertext $x^* \xleftarrow{R} \mathbb{Z}^*_{n^2}$.

$\mathcal{C}$ uses $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$ oracles (defined in the training phase) and generates the secure index $\mathcal{I}$ for F as follows: For all $f_i \in \mathbf{F}$:

– Extracts the keywords in f_i and sets $W_i = \{w_1, \ldots, w_{n'}\}$, where n' is the number of keywords in f_i.

– Computes $d_i = SKE.Enc(K_w, W_i)$
– Computes $c_i = SKE.Enc(K_e, f_i)$.
– Stores the tuple $\langle i, c_i, d_i \rangle$ as a row in T_f.

Extract the keywords in $\mathbf{F}$ and set $\mathbf{W} = \{w_1, \ldots, w_n\}$.

For all $w_i \in \mathbf{W}$:

– Query $\mathcal{O}_{H_1}$ with w_i as input to obtain $id(w_i)$.
– If $w_i = w^*$, $\mathcal{C}$ sets $x_i = x^*$.
– If $w_i \neq w^*$,
  • $\mathcal{C}$ chooses $r_i, s_i \xleftarrow{R} \mathbb{Z}^*_{n^2}$. Stores the tuple $\langle id(w_i), r_i, s_i \rangle$ in the list L_3.
  • Creates the bitmap $B_m(w_i)[j] = 1$ for all $j \in$ index of encrypted files having the keyword w_i.
  • Computes $x_i = g^{B_m(w_i)} r_i^n \ mod \ n^2$.
– Computes $y_i = \left(\sum_{j=1:B_m(w_i)[j]=1}^{N} H_2(K_h, j, id(w_i)) \right)$ by querying $\mathcal{O}_{H_2}$ oracle with j, $id(w_i)$ as input.
– Computes $Tag_i = h^{y_i} s_i^n \ mod \ n^2$.

Stores $\langle key, value, Tag \rangle = \langle id(w_i), x_i, Tag_i \rangle$ in T_s.
Training Phase: $\mathcal{A}$ is allowed to query the following oracles, with a restriction that, $\mathcal{A}$ should not make any oracle query corresponding to w^*.

$\mathcal{O}_{H_1}$*:* Given keyword w_i as input, $\mathcal{C}$ checks whether a tuple of the form $\langle w_i, id(w_i) \rangle$ appears in the list L_1. If it is present, returns $id(w_i)$ to $\mathcal{A}$. If the tuple is not present, chooses $id(w_i) \xleftarrow{R} \{0,1\}^\kappa$, adds the tuple $\langle w_i, id(w_i) \rangle$ to the list L_1 and return $id(w_i)$ to $\mathcal{A}$.

$\mathcal{O}_{H_2}$*:* When $\mathcal{A}$ makes a query to this oracle with j and $id(w_i)$ as input, $\mathcal{C}$ checks whether a tuple of the form $\langle j, id(w_i), h_i \rangle$ appears in the list L_2. If it is present, returns h_i. If the tuple is not present, chooses $h_i \xleftarrow{R} \{0,1\}^\kappa$, inserts the tuple $\langle j, id(w_i), h_i \rangle$ into the list L_2 and returns h_i.

$\mathcal{O}_{SearchToken}$*:* Given a keyword w, $\mathcal{C}$ queries $\mathcal{O}_{H_1}$ with w as input and returns the search token $\tau_s = id(w)$.

$\mathcal{O}_{AddToken}$*:* Given a file f as input, $\mathcal{C}$ does the following:

Let the number of files already stored in the database be z.

– Computes $c = SKE.Enc(K_e, f)$.
– Let W be the set of all keywords in the file f.
– For all keywords w_i in W, perform the following:
  • Queries $\mathcal{O}_{H_1}$ with keyword w_i as input to obtains the keyword identifier $id(w_i)$.
  • Generates two new random numbers $r_i, s_i \xleftarrow{R} \mathbb{Z}^*_{n^2}$.
  • Generates the bitmap addition token $x'_i = g^{2^z} r_i^n \ mod \ n^2$
  • Queries $\mathcal{O}_{H_2}$ with j and $id(w_i)$ as input and obtains h_i. Computes the tag addition token $Tag'_i = h^{h_i} s_i^n \ mod \ n^2$

Sets $\tau_a = \langle \{id(w_i), x'_i, Tag'_i\}, c\rangle$ and sends τ_a to $\mathcal{A}$.

$\mathcal{O}_{DeleteToken}$*:* Given a file f, $\mathcal{C}$ performs the following:

- Let the location of f in T_f be z.
- Retrieves the set of encrypted keywords d associated with f and computes $W = SKE.Dec(K_w, d)$
- For all w_i in W, queries $\mathcal{O}_{H_1}$ with keyword w_i as input to obtains keyword identifier $id(w_i)$.
- For all keyword identifier $id(w_i)$, do as follows:
  - Generates two random numbers $r_i, s_i \xleftarrow{R} \mathbb{Z}^*_{n^2}$.
  - Computes the bitmap modifier $x'_i = g^{-2^{z-1}} r_i^n \ mod\ n^2$
  - Queries $\mathcal{O}_{H_2}$ with $j, id(w_i)$ as input to get h_i.
  - Computes the modification tag $Tag'_i = h^{-(h_i)} s_i^n \ mod\ n^2$.

Sends $\tau_d = \langle id(f), \{id(w_i), x'_i, Tag'_i\}\rangle$ to $\mathcal{A}$.

$\mathcal{O}_{Verify}$*:* Given Tag, x and keyword identifier $id(w)$ as input, $\mathcal{C}$ performs the following:

- From L_2, obtains all js and h_j's corresponding to $id(w)$ and constructs the bitmaps $B_m(w)$ from j values.
- Obtain $\langle id(w), r, s\rangle$ from L_3.
- Checks whether $x \stackrel{?}{=} g^{B_m(w)} r^n \ mod\, n^2$ and $Tag \stackrel{?}{=} y^{\sum_{i=1:B_m(w)[i]=1}^{N} (h_i)} s^n \ mod\ n^2$.

If the check holds return *Accept*; else, return *Reject*.

$\mathcal{O}_{BitmapDecryption}$*:* Given x and keyword w as input, $\mathcal{C}$ performs the operation similar to $\mathcal{O}_{Verify}$ and if the verification holds, returns $B_m(w)$. Else return $\perp$.

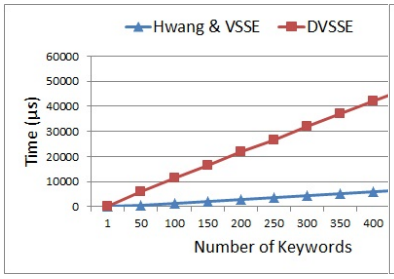
Challenge Phase: $\mathcal{C}$ gives the keyword identifier $id(w^*)$ to $\mathcal{A}$ by querying $\mathcal{O}_{H_1}$.

Guess: $\mathcal{A}$ outputs a bit b' to $\mathcal{C}$. $\mathcal{C}$ sends b' to $\mathcal{B}$. □

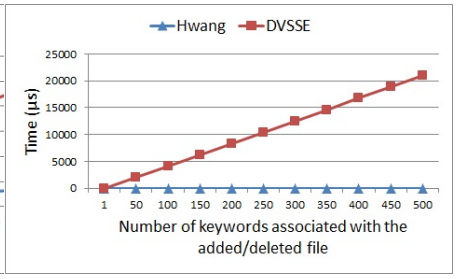
Correctness: Note that x^* obtained from $\mathcal{B}$ is used by $\mathcal{C}$ during the challenge phase. In order to distinguish the encrypted bitmap x^*, $\mathcal{A}$ should have decrypted x^*. If x^*, obtained by $\mathcal{C}$ from $\mathcal{B}$ was a valid encryption corresponding to $B_m(w^*)$, $\mathcal{A}$ decrypts and obtains the correct $B_m(w^*)$. Hence, $\mathcal{A}$ would have output a valid guess bit b'. If x^* obtained from $\mathcal{B}$ was a random element in ciphertext space, then, $\mathcal{B}$ would have identified that x^* is not a valid encryption of $B_m(w^*)$. $\mathcal{A}$ would have output a valid guess bit b', (assuming the capabilities of $\mathcal{A}$). Finally, the indistinguistability of the encrypted files c_is follow from the *IND-CPA* security of SKE.

Theorem 4. *If H_2 is an EUF-CMA secure cryptographic MAC, then the dynamic verifiable SSE scheme (DVSSE) is UF-CKA secure in the random oracle model.*

Proof: Let $\mathcal{C}$ be the challenger who challenges a forger $\mathcal{F}$ to break the *UF-CKA* security of DVSSE. Let $\mathcal{B}$ be an algorithm which challenges $\mathcal{C}$ to break the *EUF-CMA* security of MAC. $\mathcal{B}$ provides oracle access to $TagGen$ and $TagVerify$



(Fig:1) Time to build secure index



(Fig:2) Time to perform add/delete operation

algorithms of the MAC scheme to $\mathcal{C}$. $\mathcal{C}$ makes use of the capability of $\mathcal{F}$ and the oracles provided by $\mathcal{B}$ to achieve its goal.

Setup: $\mathcal{F}$ generates three random keys $\langle K_1, K_e, K_w \rangle \xleftarrow{R} \{0,1\}^{\kappa}$. $\mathcal{C}$ designs the MACs H_1 and H_2 as random oracles and maintains two lists L_1 and L_2 to answer the oracle queries consistently. $\mathcal{C}$ generates two κ_2 bit safe primes p, q and computes $n = pq$. $\mathcal{C}$ computes $\phi(n^2) = pq(p-1)(q-1)$ and $\lambda(n) = lcm(p-1, q-1)$. $\mathcal{C}$ chooses two integers $\alpha, \beta < n$ and computes $g = 1+\alpha n$ and $h = 1+\beta n$. Finally $\mathcal{C}$ sends $\langle n, n^2 \rangle$ to $\mathcal{A}$.

$\mathcal{A}$ chooses a document collection $\mathbf{F}$, generates the list of keywords $\mathbf{W}$ present in $\mathbf{F}$, chooses a keyword $w^* \in \mathbf{W}$ and gives them to $\mathcal{C}$. $\mathcal{A}$ is allowed to make $\mathcal{O}_{H_2}$ oracle queries for any, but one file that has the keyword w^*. $\mathcal{C}$ uses $\mathcal{O}_{H_1}$ and $\mathcal{O}_{H_2}$ oracles (defined in the training phase), generates the secure index $\mathcal{I}$ for F as follows: For all $f_i \in \mathbf{F}$:

- Extracts the keywords in f_i and sets $W_i = \{w_1, \ldots, w_{n'}\}$, where n' is the number of keywords in f_i.
- Computes $d_i = SKE.Enc(K_w, W_i)$.
- Computes $c_i = SKE.Enc(K_e, f_i)$.
- Stores the tuple $\langle i, c_i, d_i \rangle$ as a row in T_f.

Extracts the keywords in $\mathbf{F}$ and set $\mathbf{W} = \{w_1, \ldots, w_N\}$. For all $w_i \in \mathbf{W}$:

- Queries $\mathcal{O}_{H_1}$ with w_i as input to obtain $id(w_i)$
- $\mathcal{C}$ chooses $r_i, s_i \xleftarrow{R} \mathbb{Z}^*_{n^2}$. Stores the tuple $\langle id(w_i), r_i, s_i \rangle$ in the list L_3.
- Creates the bitmap $B_m(w_i)[j] = 1$ for all $j \in$ index of encrypted files having the keyword w_i.
- Computes $x_i = g^{B_m(w_i)} r_i^n \ mod \ n^2$.
- If $w_i = w^*$, $\mathcal{C}$ sets $Tag_i = NULL$
- If $w_i \neq w^*$, compute $y_i = \left(\sum_{j=1:B_m(w_i)[j]=1}^{N} H_2(K_h, j, id(w_i)) \right) mod \ \phi(n^2)$ by querying $\mathcal{O}_{H_2}$ oracle with j, $id(w_i)$ as input.
- Computes $Tag_i = h^{y_i} s_i^n \ mod \ n^2$.
- Stores $\langle key, value, Tag \rangle = \langle id(w_i), x_i, Tag_i \rangle$ in T_s.

Scheme	BuildIndex	SearchToken	Search	AddToken	Add	DeleteToken	Delete	Verify
Hwang et.al.	N SKE.Enc (2 * M) HMAC M Hash M XOR	2 HMAC	1 Hash 1 XOR	1 SKE.Enc (2 * k) HMAC k Hash	k XOR	k HMAC	k XOR	NA
VSSE	N SKE.Enc (3* M) HMAC M XOR	1 HMAC	Neg.	NA	NA	NA	NA	3 HMAC 1 XOR
DVSSE	(2 * N) SKE.Enc M * (k + 1) HMAC (4 * M) Exp. in $\mathbb{Z}^*_{n^2}$	1 HMAC	Neg.	2 * SKE.Enc (2 * k) HMAC (4 * k) Exp. in $\mathbb{Z}^*_{n^2}$	(2 * k) Mult in $\mathbb{Z}^*_{n^2}$	1 SKE.Dec k HMAC (4 * k) Exp. in $\mathbb{Z}^*_{n^2}$	(2 * k) Mult in $\mathbb{Z}^*_{n^2}$	4 Integer Div. k' HMAC 2 Exp. in $\mathbb{Z}^*_{n^2}$

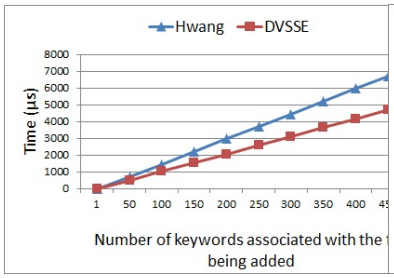
Table 2: Comparison among Hwang et.al, VSSE and DVSSE.

(N: Total number of files; M: Total number of Keywords; k: Number of keywords in any given file; k': Number of files associated with in any given keyword; Neg: Negligible; Exp: Exponentiation; Mult: Multiplication; Div: Division)

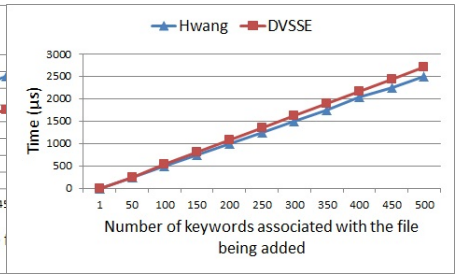
Sends secure index $\mathcal{I} = \langle T_f, T_s \rangle$ to $\mathcal{A}$.

Training Phase: $\mathcal{A}$ is allowed to query the following oracles, with a restriction that, $\mathcal{A}$ should not make any oracle query corresponding to w^*.

$\mathcal{O}_{H_1}$*:* Given keyword w_i as input, $\mathcal{C}$ checks whether a tuple of the form $\langle w_i, id(w_i) \rangle$ appears in the list L_1. If it is present, returns $id(w_i)$ to $\mathcal{A}$. If the tuple is not present, chooses $id(w_i) \xleftarrow{R} \{0,1\}^\kappa$, adds the tuple $\langle w_i, id(w_i) \rangle$ to the list L_1 and returns $id(w_i)$ to $\mathcal{A}$.



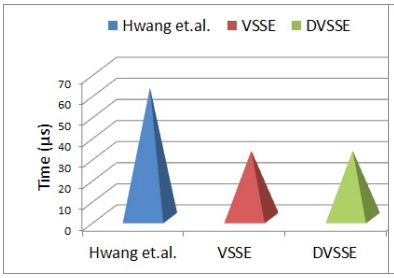
(Fig:3) Time to generate addition token



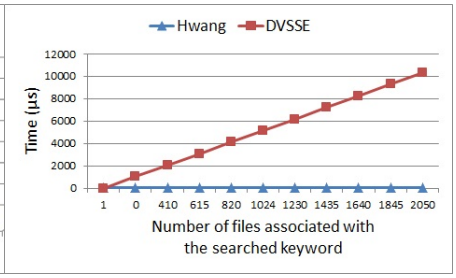
(Fig:4) Time to generate deletion token

$\mathcal{O}_{H_2}$*:* When $\mathcal{A}$ makes a query to this oracle with j and $id(w_i)$ as input, $\mathcal{C}$ checks whether a tuple of the form $\langle j, id(w_i), h_i \rangle$ appears in the list L_2. If it is present, return h_i. If the tuple is not present, $\mathcal{C}$ queries the $TagGen$ oracle provided by $\mathcal{B}$ with j and $id(w_i)$ as input to obtain h_i, inserts the tuple $\langle j, id(w_i), h_i \rangle$ into the list L_2 and returns h_i. (We assume that $\mathcal{A}$ will not query this oracle for any other j other than the js for which $B_m(w_i)[j] = 1$.)

$\mathcal{O}_{SearchToken}$*:* Given a keyword w, $\mathcal{C}$ queries $\mathcal{O}_{H_1}$ with w as input and returns the search token $\tau_s = id(w)$.



(Fig:5) Time to perform one search



(Fig:6) Time to verify search outcome

The oracles $\mathcal{O}_{AddToken}$, and $\mathcal{O}_{DeleteToken}$ are similar to the oracles defined in *IND-CKA2* proof.

$\mathcal{O}_{Verify}$*:* Given Tag, x and keyword identifier $id(w)$ as input, this oracle performs the following:

- Decrypts x by computing $B_m(w) = \dfrac{L(x^{\lambda} \ mod \ n^2)}{L(g^{\lambda} \ mod \ n^2)}$, $y = \dfrac{L(Tag^{\lambda} \ mod \ n^2)}{L(h^{\lambda} \ mod \ n^2)}$.
- For ($j = 1$ to N: $B_m(w)[j] = 1$), queries the $\mathcal{O}_{H_2}$ oracle with j and $id(w)$ as input. Let these values be $\langle h_1, h_2, \ldots, h_k \rangle$, where k is the total number of bits set to 1 in $B_m(w)$.
- Check whether $y \stackrel{?}{=} \sum_{i=1}^{k} h_i \ mod \ \phi(n^2)$.
- If the check holds returns *Accept*; else, returns *Reject*.

$\mathcal{O}_{BitmapDecryption}$*:* Given x and keyword w as input, $\mathcal{C}$ performs the operation similar to $\mathcal{O}_{Verify}$ and if the verification holds, returns $B_m(w)$; else, returns $\perp$.

Forgery: $\mathcal{A}$ outputs Tag^* corresponding to $id(w^*)$. Let Tag^* be $h^{y^*} s^n \ mod \ n^2$, where $y^* = \left(\sum_{j=1:B_m(w^*)[j]=1}^{N} H_2(K_h, j, id(w^*)) \right) \ mod \ \phi(n^2)$. $\mathcal{C}$ performs the following to generate the forgery for $\mathcal{B}$'s challenge.

- Let k be the index of the bit in $B_m(w^*)$, which corresponds to the file identifier for which $\mathcal{A}$ has not queried the output of $\mathcal{O}_{H_2}$ during the training phase.
- $y^* = \left(\sum_{j=1:B_m(w^*)[j]=1/j=k}^{N} H_2(K_h, j, id(w^*)) \right) + H_2(K_h, k, id(w^*))$, which we represent as $y^* = y_1^* + y_2^*$.
- $\mathcal{C}$ can compute y_1^* corresponding to the bitmap $B_m(w^*)$ by querying $\mathcal{O}_{H_2}$ oracle with j and $id(w^*$ as input, for$(j = 1$ to $N : B_m(w^*)[j] = 1/j = k)$.
- Hence, $\mathcal{C}$ can compute $h^{y_2^*} s^n \ mod \ n^2 = Tag^* h^{-y_1^*} \ mod \ n^2$.

– Now, $\mathcal{C}$ decrypts $h^{y_2^*}s^n$ using $\phi(n^2)$ and λ (Paillier decryption) and obtains y_2^* by computing
$y_2^* = \frac{L((h^{y_2^*}s^n)^\lambda \ mod \ n^2)}{L(h^\lambda \ mod \ n^2)}$.

$\mathcal{C}$ submits y_2^* as a forgery to $\mathcal{B}$ □

5 Performance Analysis

We have implemented the new schemes, compared their performance and complexity with Hwang et al.'s scheme. The implementation is done on a Intel Core 2 Quad 2.83GHz machine with 4GB memory, running Ubuntu 14.04. We have implemented the scheme using OpenSSL 1.0.1c, in C language. Table 2 represents the number of cryptographic primitive operations performed in our scheme vs Hwang et.al.'s scheme [6].

For profiling, we have considered a set of 2048 files, with 500 keywords in total. We consider a file can have 20 keywords and any given keyword can have a maximum of 1500 files associated with it.

We have not considered the performance of the symmetric key encryption and the size of the encrypted files as they are not relevant to the keyword search problem that we address. We use HMAC-SHA-256 for the keyed hash function and SHA-256 to implement hash functions. As we use Paillier encryption in DVSSE, the maximum bitmap for a given search table T_s should be limited to 2048 but index of any size can be achieve by cascading multiple tables.

Figure 1 shows the time taken to generate the secure index, in Hawng et.al's scheme, VSSE and DVSSE. As the number of total keywords increases, we notice that the time taken by all the three schemes increase linearly. The performance of VSSE is almost as same as Hwang et.al's scheme. DVSSE takes a slightly more time to generate the secure index, as it involves building homomorphic tags.

Figure 2, 3 and 4, presents the time taken to generate addition token, deletion token and perform addition or deletion operation respectively, against the number of keywords associated with the file being added or deleted. From Figure 2 and 3, we can infer that the performance of addition, deletion token generation algorithms in DVSSE is similar to Hwang et.al's scheme. Figure 4 shows that the addition or deletion operation in DVSSE is costly, compared to Hwang et.al's scheme due to the use of multiplication in $\mathbb{Z}_{n^2}^*$. Figure 5 represents the time taken to verify a search outcome in VSSE and DVSSE, considering a number of files associated with the searched keyword.

6 Conclusion

In this paper, we have proposed a new security model to handle SHBC-CSP, constructed two new schemes to perform encrypted keyword search in the presence of SHBC-CSP and proved the security of these schemes in the newly proposed security model. We have also contrived a new homomorphic MAC and DVSSE

is the first SSE scheme which offers dynamism, verifiability and also security against leakage of access pattern. We have implemented the new schemes and provided their performance details in this paper.

Though the performance of BuildIndex, Add, Delete and Verify algorithms are slightly costlier than Hwang et.al's scheme, it should be noted that DVSSE is the first scheme to provide dynamism, verifiability and also security against leakage of access pattern. However, Hwang et.al's scheme offer only dynamism without verifiability. Improving the performance of DVSSE is considered for future work. We note that there is a possibility for the SHBC-CSP to mount replay attack by returning search result based on outdated database. A trivial solution to avoid this is to use timestamps or cryptographic nonce. However, this might require client to maintain certain local storage. Designing a solution withstanding replay attack without any local storage would be non-trivial and this problem is left open to be solved in this paper.

References

1. C. Bösch, P. H. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, 2014.
2. Q. Chai and G. Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Proceedings of IEEE International Conference on Communications ICC*, pages 917–922, 2012.
3. Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. *IACR Cryptology ePrint Archive*, 2004:51, 2004.
4. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, pages 79–88, 2006.
5. E. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.
6. Y. H. Hwang, J. W. Seo, and I. J. Kim. Encrypted keyword search mechanism based on bitmap index for personal storage services. In *13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014, Beijing, China, September 24-26, 2014*, pages 140–147, 2014.
7. S. Kamara. Encrypted search. *ACM Crossroads*, 21(3):30–34, 2015.
8. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, pages 258–274, 2013.
9. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *the ACM Conference on Computer and Communications Security, CCS'12*, pages 965–976, 2012.
10. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.
11. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy, USA, May 14-17, 2000*, pages 44–55, 2000.

12. Q. Zheng, S. Xu, and G. Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pages 522–530, 2014.

A Definitions

Definition 7. *(Indistinuishability of MAC Tag (IND-CMA)) Let Message Authentication Code* $MAC = \langle KeyGen,\ TagGen,\ TagVerify \rangle$. *The IND-CMA security is defined as a game between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{C}$.

Setup*:* $\mathcal{C}$ *generates the key* K *of* MAC *scheme and sets the systems parameters* $params$.

Training Phase*:* $\mathcal{A}$ *can query the following oracles:*

- $\mathcal{O}_{TagGen}(M)$*: Returns the MAC tag* Tag.
- $\mathcal{O}_{TagVerify}(M, Tag)$*: Returns* $\{Accept,\ Reject\}$.

Challenge*:* $\mathcal{A}$ *chooses a message* M^* *and sends it to* $\mathcal{C}$. *The challenger* $\mathcal{C}$ *selects a bit* $b \xleftarrow{R} \{0,1\}$. *If* $b = 0$, $\mathcal{C}$ *generates the tag* $Tag^* = TagGen(K, M^*)$ *and if* $b = 1$, $\mathcal{C}$ *chooses a random tag* Tag^* *in the range of the tag.* $\mathcal{C}$ *sends* Tag^* *to* $\mathcal{A}$ *as the challenge tag.*

Guess*:* $\mathcal{A}$ *outputs a bit* b', *which is* 0 *if* Tag^* *is the tag corresponding to* M^*, *else outputs* 1.

The MAC is IND-CMA secure if for all probabilistic polynomial time adversaries $\mathcal{A}$, *there exists a negligible function* $negl(.)$ *such that:*

$$Pr[MAC_{\mathcal{A}}^{IND\text{-}CMA}(\kappa) \rightarrow (b = b')] \leq \frac{1}{2} + negl(\kappa)$$

Definition 8. *(Existential Unforgeability of MAC (EUF-CMA)) Let Message Authentication Code* $MAC = \langle KeyGen, TagGen, TagVerify \rangle$. *The EUF-CMA security is defined as a game between an adversary* $\mathcal{A}$ *and a forger* $\mathcal{F}$.

Setup*:* $\mathcal{F}$ *generates the key* K *of* MAC *scheme and sets the systems parameters* $params$.

Training Phase*:* $\mathcal{F}$ *is allowed to query all the oracles as in IND-CMA game.*

Forgery*:* $\mathcal{F}$ *outputs a valid message tag pair* $\langle M*, Tag* \rangle$, *for which* $M*$, $\mathcal{A}$ *should not have queried to* $TagGen$ *oracle.*

The MAC is EUF-CMA secure if for all probabilistic polynomial time forger $\mathcal{F}$, *there exists a negligible function* $negl(.)$ *such that:*

$$Pr[MAC_{\mathcal{F}}^{EUF\text{-}CMA}(\kappa) \rightarrow \langle M, Tag \rangle : Verify(M, Tag) \rightarrow Accept] \leq negl(\kappa)$$