Journal of Basrah Researches ((Sciences)) Vol. (47). No. 1 (2021)

Available online at: https://www.iasj.net/iasj/journal/260/issues

ISSN –1817 –2695

Received date: 13-6-2021 Accepted date: 18-8-2021 Available online date: 31-12-2021

Security proof for backward searchable encryption scheme

Salim S. Bulbul 1,a) and Ayad I. Abdulsada1,b)
Universiyt of Basrah, Education College for pure Science Basrah, Iraq.
1,a) pgs2185@uobasrah.edu.iq
1,b)Corresponding author: ayad.abdulsada@uobasrah.edu.iq

KEYWORDS: Dynamic searchable encryption, Forward privacy, and Backward privacy.

Abstract*:*

Dynamic searchable symmetric encryption schemes (*DSSE*) allows searching and updating the encrypted data without decryption. These schemes introduce new challenges that require new enhanced security properties: *forward* and *backward* privacy. The first property hides the correlation between the newly added files with the previously issued search tokens. The second property makes the deleted files hidden until presenting new search tokens. In this paper, we design a new DSSE scheme with forward and backward privacy and present a rigorous security proof for the proposed scheme. Our proof follows the standard method of real-ideal environment and handles the utilized hash functions as random oracles.

KEYWORDS-SEARCHABLE ENCRYPTION, FORWARD AND BACKWARD PRIVACY, REAL-IDEAL ENVIRONMENT, RANDOM ORACLE.

1. INTRODUCTION

Searchable symmetric encryption (*SSE*) schemes give the ability for user to upload his encrypted files to a remote server and search it without decryption. Efficient schemes [2, 3, 4] allow leaking some information to the server such as *search pattern* (the frequency of appearing certain search keyword) [3, 4] and *access pattern* (search results). Recent studies [16, 22, 23, 7] shown the disaster consequences of such a leakage and thus highlight the importance of existing new security properties in SSE scheme such as forward and backward privacy Dynamic *SSE* schemes extend the functionality of traditional SSE scheme, where update operations can be performed [5, 8, 9, 10]. However, such new schemes introduce new challenges. For example, when new file is added it may be related with previous searches to reveal whether this file was including certain keyword or not. Forward private schemes are designed to resist such a leakage [2, 8, 11, 13]. Additionally, search queries can be linked with deleted files. Such a leakage is thwarted by schemes that support backward privacy. Bost et al. [3] introduces three types of leakage corresponding to backward privacy: Type-I, Type-II, and Type-III.

Contributions. In this paper, we design a DSSE dynamic scheme that searches the encrypted data with single keyword queries and enjoys with forward and Type-II backward privacy. The security of our scheme is proved formally under the model of random oracle.

Paper organization. The paper is organized as follows. Section 2 presents the most relevant works. Section 3 demonstrates basic background. Section 4 reports the proposed scheme. Section 5 shows the security proof. Chapter 6 concludes the whole paper.

2. RELATED WORKS

Kamara et al. [15] introduced the first scheme of DSSE. However, such a scheme is too complex and leaks the search tokens of the keywords contained in an updated document. The security of [15] was enhanced in [26] by using a tree-based index to handle update operations efficiently but at the expense of increased data structure. In this scheme, the leaf nodes of the tree hold pointers to document identifiers. The internal nodes store a binary vector with a size equals the total number of keywords. The bits in the vector of any internal node indicate whether their corresponding keywords are assigned to documents in the leaf nodes branching from that node. This scheme can support parallel search as subtrees can be handled individually. To deal with large scale datasets, Cash et al. [2] proposed an optimized DSSE schemes by using a dictionary-based index. Naveed et al. [17] proposed a DSSE scheme that relies on specific construction called Blind Storage which support update operations. In this scheme documents are split into blocks that are scattered at pseudorandom locations. Their actual scheme does not insert the documents but for each keyword a forward index is constructed and stored into the Blind Storage. However, under such a scheme when multiple documents include a common keyword are added, the server will know this fact.

Forward privacy. File injection attack [23] has inspired the real study of forward privacy in DSSE schemes. Forward privacy was first informally defined by Stefanov et al. [18]. Forward privacy ensures the privacy of newly added documents. Their scheme employs a hierarchical data structure of several levels instead of using an inverted index to store document-keyword pairs. However, under such a structure, all the levels of the hierarchy have to be probed by the server to serve each search request, which hurt search efficiency. The first formal definition of forward privacy is introduced by Bost in [14] along with an optimal communication DSSE scheme, Sophos. Such a scheme is constructed by asymmetric cryptography primitives with the ability of handling only insertion operations. Diana [3] is another forward private DSSE scheme that relies on a constrained pseudorandom function (CPRF) $\hat{F}$ [1, 19], to support parallel search with higher communication overhead. An improved forward private DSSE schemes are presented in [8, 13]. The strategy of [8] to achieve forward privacy was to re-generate the keys of update operation after each search query. However, this method requires re-encrypting the search results by the client and resending them back to the server, which incurs more I/O overhead and one additional round of interaction. FASTIO [21] has improved the locality [2] of DSSE schemes.

Backward privacy. Bost [3] introduced the first formal definition of backward privacy with three different types of information leakage ordered from strong to weak security requirements: Type-I, Type-II, and Type-III. Furthermore, the author introduced four backward private DSSE schemes with variable privacy/performance balance: Fides, Moneta, $\text{Diana}_{\text{del}}$, and Janus. Fides is a Type-II scheme that employs two instances of the forward private DSSE scheme, Sophos [14]. Monita is a Type-I DSSE scheme which is constructed using TWORAM [9], an ORAM-based [12] forward private DSSE scheme that uses heavy communication operations making it suitable for only theoretical instances of Type-I schemes. $\text{Diana}_{\text{del}}$ and Janus are Type-III DSSE schemes that leak more information to the server for better performance. $\text{Diana}_{\text{del}}$ is constructed from constrained pseudo-random functions (CPRF) [1, 11], which incurs high communication and computation costs. Janus employs puncturable encryption [13] that supports the property of incremental update, to release an optimum DSSE scheme with a single-round search and optimal communication cost. However, the search time of Janus is not optimal. Such that, after hundreds of deletion operations, Janus will not be practical, mainly due to the computation and storage costs. Furthermore, $\text{Diana}_{\text{del}}$ and Janus disallow the reinsertion of the previously deleted keyword-document pairs. More advanced DSSE schemes are presented also in [10]. Recently [27] developed two schemes with backward privacy: ΠBP , ΠW BP , where the first one achieves the link pattern security definition, which falls between Type-II and Type-III (see [27]), while the second achieves Type-III of backward privacy.

Attacks on searchable encryption schemes. All searchable encryption schemes methods leak some information such as access pattern, search pattern to the cloud server for improving search efficiency [20]. Here, we review the studies that the target the potential attacks of such information leakages. Islam et al. [22] show how a semi honest server can utilizes the access pattern to determine user's keyword search with high probability. Their model assumes that the server knows the entire document collection to conduct its attacks. Cash et al. [16] improves the attacks of [22], where they assume that some user documents are available to the server. The above-mentioned attacks deal with static SSE scheme. Particularly, they do not consider any interaction between the attacker and the client. Dynamic SSE schemes have introduced new attacks by inserting certain documents in the database. The file injection attack [23] is a popular example for breaking the confidentiality of the client's search queries. In this attack the server selects some documents with specific keyword set (half of the keyword set in $\boldsymbol{W}$) and ask the client to encrypt them and send the results back on the server's index. Then, by using the access pattern leakage, the server will know the set of inserted documents that match the issued search query. When the dynamic SSE scheme leaks the update pattern $TimeDB(w)$, i. e. the times for updating keyword w (will be explained later in Section 3), the server can easily determine the underling search keywords. Their model shows the urgent need of forward privacy in searchable encryption schemes. more precisely, such a property will ensure that update operations are oblivious to the to the keywords, i.e. to the contents of documents. Ning et al. [25] study the effect of passive attacks on SSE schemes. They designed several attacks under variable assumptions.

3. BACKGROUND

A. Cryptographic background

Notations. The security parameter is denoted by $\lambda \in \mathbb{N}$, which is used as input in all algorithms of our scheme. Probabilistic polynomial-time is referred to as PPT. $negl(\lambda)$ stands for a negligible function in λ, where $f: \mathbb{N} \to \mathbb{R}$ is considered a negligible function only iff for all $c > 0, \exists n_0 \in N$ s.t $\forall n \geq n_0, f(n) < n^{-c}$. In the context of the client-server setting, the notation $P(x; y)$ indicates that the protocol P is executed by the input x of the client and the input y of the server. The cardinality of a set X is denoted by $|X|$. $x \xleftarrow{\$} X$ stands for sampling x uniformly from X . Assigning the value y to variable x is denoted by $x \leftarrow y$. The notation || refers to the concatenation operation. The notion $\{0,1\}^{\ell}$ denotes the set of all strings of length ℓ, and $\{0,1\}^{*}$ denotes all strings of arbitrary lengths. A bitwise XOR between x and y is denoted by $x \oplus y$. Consider a collection of D documents that includes textual keywords derived from a defined alphabet Σ, with each document d_i is identified by its identifier $id_i \in \{0,1\}^{\ell}$. The database DB includes a set of pairs of document identifiers and keywords (id, w) such that the keyword w appears in the document with identifier id. The set of all unique keywords that appear in DB is denoted by W of size K $(i.e, K = |W|)$, N stands

for the number of pairs in DB (i.e., $N = |DB|$). The set of documents that included by $DB(w)$.

Pseudorandom functions. Let $GenPRF(1^\lambda)$ be a key generation function, and $F: \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^{\ell'}$ be a pseudorandom function (PRF) family. F is considered a secure PRF family if for all PPT adversaries $\mathcal{A}$,

$$|\Pr[k \leftarrow GenPRF(1^\lambda); \mathcal{A}^{F_k(.)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{R(.)}(1^\lambda) = 1]| \leq negl(\lambda),$$

where R: $\{0,1\}^\ell \to \{0,1\}^{\ell'}$ is a truly random function.

Pseudorandom permutation functions. Let $GenPPR(1^\lambda)$ be a key generation function, and $G: \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ be a pseudorandom permutation function(PPF) family. G is considered a secure PPF family if for all PPT adversaries $\mathcal{A}$, $|\Pr[k \leftarrow GenPPT(1^\lambda); \mathcal{A}^{G_{sk}(.),G^{-1}{}_{sk}(.)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{g(.),g^{-1}(.)}(1^\lambda) = 1]| \leq negl(\lambda)$,

where $g : \{0,1\}^* \to \{0,1\}^\lambda$ is a truly random permutation function.

Hash function. Let $h: \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ be a family of hash functions. The member h_s is identified by indicator $s \in \mathcal{K}$. h is a collision resistant family if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that:

$$\Pr[s \leftarrow \mathcal{K}; (m_1, m_2) \leftarrow \mathcal{A}(h_s) : (m_1 \neq m_2) \wedge (h_s(m_1) = h_s(m_2))] \leq negl(\lambda).$$

Symmetric encryption scheme. A symmetric encryption scheme π includes three algorithms ($KeyGen, Enc, Dec$). The key generation algorithm $KeyGen$ gets the security parameter λ and generates a private key $k \in \{0,1\}^\lambda$. The encryption algorithm Enc uses k to encrypt the message $x \in \{0,1\}^*$ and returns a ciphertext $c \in \{0,1\}^*$. The encryption algorithm would be either probabilistic or deterministic. The Dec algorithm is always deterministic. Notice that, any encryption scheme should get correct results. i.e. for all $k \leftarrow KeyGen(1^\lambda)$ and for all messages $x \in \{0,1\}^*$, it should satisfies $Dec_k(Enc_k(x)) = x$. We consider the indistinguishability against chosen plaintext attacks (IND-CPA) to define the security of the symmetric encryption.

Definition 1(**IND − CPA secuirty**) : Let $\delta = (KeyGen, Enc, Dec)$ be a symmetric encryption scheme. Define $Challenge(x_0, x_1, b) = x_b$, where $b = \{0,1\}$ and $|x_0| = |x_0|$. We say that δ is secure against IND-CPA attack, if for all adversaries $\mathcal{A}$, the advantage $Adv^{cpa}_{\mathcal{A},\delta}(\lambda)$ of $\mathcal{A}$ against δ is negligible in λ for any polynomial-time adversary $\mathcal{A}$, where:

$$Adv^{cpa}_{\mathcal{A},\delta}(\lambda) = \left| P\left[k \xleftarrow{\$} keyGen(1^\lambda): \mathcal{A}^{Enc_k(Challenge(.,.,0))}(1^\lambda)\right] = 1\right| - \left|P\left[k \xleftarrow{\$} keyGen(1^\lambda): \mathcal{A}^{Enc_k(Challenge(.,.,1))}(1^\lambda)\right] = 1\right|$$

Game-based proofs. The notion of real-ideal world is a popular approach to prove the security of cryptographic schemes. In this approach, the real world is the cryptographic construction to be proved, and the ideal word is a simulated construction by using the leakage of the real world. The intuition is that if no adversary can distinguish between the two worlds, then the real world is considered a secure scheme as the leaked information do not provide any useful information to the adversary. The two worlds are described as two games: $\boldsymbol{G_0}$ and $\boldsymbol{G_n}$ and the goal is to see that these two games are indistinguishable. Unfortunately, in many cases, it is difficult to prove this in a direct way. Instead, we need to construct hybrid games: $\boldsymbol{G_1}$-$\boldsymbol{G_{n-1}}$, such that each consecutive game differs slightly. We start from $\boldsymbol{G_0}$ and prove that it is indistinguishable from $\boldsymbol{G_1}$. We proceed until we end with $\boldsymbol{G_n}$. The advantage of adversary to distinguish between $\boldsymbol{G_0}$ and $\boldsymbol{G_n}$ will be the sum of the partial advantages of A to distinguish between ($\boldsymbol{G_0}$, $\boldsymbol{G_1}$), ($\boldsymbol{G_1}$, $\boldsymbol{G_2}$), …., ($\boldsymbol{G_{n-1}}$, $\boldsymbol{G_n}$).

The random oracle model (ROM). Bellare and Rogaway [24] have formally described the Random Oracle Model (or ROM). Under this model there is public random oracle (black box) that can be accessed by all parties. This oracle responds to the query inputs with random outputs. When the it is queried by a repeated input, its output will be the same. Random oracles are usually used as an ideal instance of the secure hash functions in case of applications that need a strong randomness to the output of the hash functions. When ROM are used to prove the security of some schemes, we need to re-program the output of the random oracle with some specific inputs. Under such a treatment the programmed random oracle will not be indistinguishable from a typical random oracle.

B. Dynamic Searchable symmetric encryption (DSSE).

DSSE scheme consists of one algorithm $Setup$ and two protocols $Search$, $Update$ that are executed between a client and a server. The client holds the databased DB and outsources the encrypted database EDB to the server.

- $Setup()$. This algorithm is run by the data owner to initialize the system.
- $Search()$. This protocol is used to search the encrypted database.
- $Update()$. This protocol is responsible for updating the encrypted database.

Document Encryption. Theoretically, the protection of document collection should be included in the searchable encryption scheme as well. However, the confidentiality of the outsourced document collection can be ensured by using any standard encryption algorithm. Hence, it is not a great concern to include the security of document collection into index-based SSE schemes. Thus, information leakage from the encrypted documents (which is solely the number of documents and their sizes) does not have to be mentioned in SSE security definitions.

C. Security

We consider a semi-honest server, where it follows precisely the steps of the protocol but tries to get additional information form the messages (transcripts) it receives

during the execution of the protocol. The security of DSSE must guarantees that a little as possible information is leaked to the server about the database and issued queries. Typically DSSE schemes define a leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$ to control the leaked information to the server when, respectively, the operations $Setup, Search, Update$ are invoked by the client. DSSE scheme with leakage $\mathcal{L}$ is considered secure if no information is revealed to the server beyond the output of $\mathcal{L}$. This is typically formalized by the standard $real/ideal$ experiment of two games: DSSEReal and DSSEIdeal [4, 3]. In DSSEReal game, the server sees the real messages of each operation of DSSE scheme, and output a bit b. In DSSEIdeal game, the server observes simulated messages instead of real ones. The simulator $\mathcal{S}$ is a PPT algorithm that has full access to the components of $\mathcal{L}$ for generating the simulated messages. The server also outputs a bit b. Informally, the server should not be able to distinguish between the above-mentioned games. The two games can be formalized as follows:

• $DSSEReal_{\mathcal{A}}^{\pi} q(\lambda)$: the adversary $\mathcal{A}(1^{\lambda})$ picks a database DB, the game then runs $Setup(\lambda, DB; \perp)$ and gives EDB to $\mathcal{A}$. Then, the adversary issues polynomial number of adaptive queries q_i. If q_i is a search query (Update query) then the game runs the search protocol $((\sigma_i + 1;\ DB_i(w_i));\ EDB_{i+1})) \leftarrow Search(sk, q_i, \sigma_i;\ EDB_i)$ ($Update$ protocol $(\sigma_{i+1},\ EDB_{i+1}) \leftarrow Update(sk, q_i, \sigma_i;\ EDB_i)$),respectively. Eventually, the adversary $\mathcal{A}$ outputs a bit b.

• $DSSEIdeal_{\mathcal{A},\mathcal{S},\mathcal{L}}^{\pi} q(\lambda)$: the adversary $\mathcal{A}(1^{\lambda})$ picks a database DB. The simulator $\mathcal{S}$ of the game uses the leakage function $(\mathcal{L}^{Stp}(DB))$ to constructs an encrypted database EDB and gives it to $\mathcal{A}$. Then, the adversary issues polynomial number of adaptive queries q_i. If q_i is a $search$ query ($Update$ query), the simulator responds by running $\mathcal{S}$ $(\mathcal{L}^{Srch}(q_i))$ $\mathcal{S}$ $((\mathcal{L}^{Updt}(q_i)))$, respectively. Eventually, the adversary $\mathcal{A}$ outputs a bit b.

Difintion 2(**Adaptive secure DSSE**) [18]. An DSSE scheme π with leakage function $\mathcal{L}$ is adaptively secure iff for every PPT adversary $\mathcal{A}$ that issues a polynomial number of queries $q(\lambda)$, there exists a simulator $\mathcal{S}$ that satisfies:

$$|pr[\text{DSSEReal}_{\mathcal{A}}^{\pi} q(\lambda) = 1] - pr[\text{DSSEIdeal}_{\mathcal{A},\mathcal{S},\mathcal{L}}^{\pi} q(\lambda) = 1]| \leq negl(\lambda)$$

Common leakage. We first define common leakage functions, following the formalization of [3]. All leakage functions define an internal state in the form of query set Q, which includes an entry for each issued query. The entries of search queries are in the form (t, w), where t is a timestamp, and w is a search keyword. The entries of update queries are $(t, op, (w, id))$, where op {add, del}. The search pattern $sp(w)$ function, which is related to the search queries, leaks whether two search queries belong to the same keyword. This function is defined as $sp(w) = \{t: (t, w) \in Q\}$.

The function $UpHist(w)$ is used to capture all updates on documents that share the keyword w. It is defined formally as:

$$UpHist(w) = \{(t, add, id)|\ (t, add, id) \in Q\ , (t, del\ id)|\ (t, del, id) \in Q\ \}$$

The function $TimeDB(w)$ for a keyword w captures the timestamp/document-identifier pairs that are added to the database for the keyword w and are not deleted yet.

$$TimeDB(w) = \{(t, id)|(u, add, (w, id) \in Q, \nexists t', (t', del, (w, id) \in Q\}$$

The timestamps for all add and del operations for w in Q are expressed by:

$$Updates(w) = \{t|\ (t, add, (w, id) \in Q \vee (t, del, (w, id) \in Q\}.$$

The $DelHist(w)$ function reveals the deletion history of keyword w to the server. Interestingly, it returns the pairs of timestamps for which specific entries are inserted and then removed.

$$DelHist(w) = \{(t^{add}, t^{del})$$

Difintion 3(**Forward privacy**): an $\mathcal{L}$-adaptively-secure DSSE scheme that supports update operation of a single keyword is forward private iff $\mathcal{L}^{Stup}, \mathcal{L}^{Srch}, and\ \mathcal{L}^{Updt}$ can be expressed as:

$$\mathcal{L}^{Stup} = \emptyset$$
$$\mathcal{L}^{Srch}(w) = \{sp(w), UpHist(w)$$
$$\mathcal{L}^{Updt}(op, w, id) = \mathcal{L}'^{Updt}(op, id)$$

where $\mathcal{L}'$ is a stateless function. This means that update queries do not leak the underlying keyword w. Observe that DB is considered initially empty such that $\mathcal{L}^{Stup}$ leaks no information. Otherwise, leaks the total number of pairs (i.e. $\mathcal{L}^{Stup} = N$).

Definition 4(**backward privacy**) [3]. An $\mathcal{L} = \{\mathcal{L}^{\text{Stup}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}}\}$-adaptively-secure DSSE scheme with backward privacy, iff $\mathcal{L}$ can be written as:

- Type-I:

$\mathcal{L}^{\text{Stup}} = \emptyset, \mathcal{L}^{\text{Updt}}\ (op, w, id) = \mathcal{L}'\ (\text{op})$, and $\mathcal{L}^{\text{Srch}}(w) = \mathcal{L}''\ (TimeDB(w), a_w)$,where a_w is the total number of update operations on w.

- Type-II:

$\mathcal{L}^{\text{Stup}} = \emptyset\ , \mathcal{L}^{\text{Updt}}\ (op, w, id) = \mathcal{L}'\ (op, w)$, and $\mathcal{L}^{\text{Srch}}(w)$ $=\mathcal{L}''\ (TimeDB(w), Updates(w))$.

- Type-III:

$\mathcal{L}^{\text{Stup}} = \emptyset, \mathcal{L}^{\text{Updt}}\ (op, w, id) = \mathcal{L}'\ (op, w)$, and $\mathcal{L}^{\text{Srch}}(w) = \mathcal{L}''\ (TimeDB(w), DelHist(w))$. Where $\mathcal{L}'$ and $\mathcal{L}''$ are two stateless functions.

Note that the $\mathcal{L}^{\text{Srch}}(w)$ of all of the above types should include the search pattern leakage function, $sp(w)$. One could argue that $sp(w)$ could be included implicitly in the other leakage functions of the definition. However, this is not true, as explained by the following scenario. Consider only two search queries on w, that is not available at DB, are issued subsequently at timestamps 10 and 15, respectively.

The search pattern function $sp(w)$ at timestamp 15 is $sp(w) = \{10\}$. However, the leakage information of the other functions is:
$TimeDB(w) = phi, Updates(w) = \emptyset$ and $(w) = \emptyset$.
Observe that $sp(w)$ is not available at the other functions. So, it should be explicitly mentioned in $\mathcal{L}^{\text{Srch}}(w)$ of all backward privacy types.
Furthermore, the $\mathcal{L}^{\text{Srch}}$ (w) of Type-III should include $Updates(w)$.
Finally, we have to mention that Type-III of backward privacy can be achieved only in case of preventing the reinsertion of any entry (document identifier/keyword) after its deletion. Otherwise, the intuition definition of backward privacy will be violated. To see that, consider the following set of query entries related to keyword w: Q { $(1, add, (id, w)), (5, w), (7, del, (id, w)), (9, w)$, $(13, add, (id, w)), (15, del, (id, w)), (17, w)$}.
The $\mathcal{L}^{\text{Srch}}(w)$ at timestamp 5 is
$TimeDB(w) = (1, id), \quad DelHist(w) = \emptyset.$
The $\mathcal{L}^{\text{Srch}}(w)$ at timestamp 9 is
$TimeDB(w) = \emptyset, \quad DelHist(w) = (1, 7).$
The $\mathcal{L}^{\text{Srch}}(w)$ at timestamp 17 is
$TimeDB(w) = \emptyset, DelHist(w) = (1, 7), (1, 15), (13, 7), (13, 15).$
Notice that, the leakage function after timestamp 17 will enable the server from learning that update queries at timestamp 13 and 15 are related to the document identifier id since it has already know that such identifier is correspond to the update queries at timestamp 1 and 7. Thus, the intuition of backward privacy is violated.

4. PROPOSED SCHEME

Figure 1 illustrates our proposed scheme. It includes the following protocols:
Setup. This algorithm uses the security parameter to define three secret keys: K_f, K_g, and K_t. It also defines a local state σ that is stored locally at the client side, and initiates two hash maps: S_e and Sr. S_e stores the encrypted entries, and S_r stores the plaintext results. Notice that σ stores for the keyword w, two counters Cnt [w] (stores the number of update operations on w), and SrchCnt[w] (catches the total number of searches on w).
Update. Client provides a keyword w, a document identifier id, and an update operation op, which is either add or del. This protocol creates a pair of $(Addr, Val)$ for each update operation. Notice that our scheme stores even the delete operation.
Search. Search protocol requires two rounds. Client checks at the beginning the initialization of σ for w. If it is not initialized yet, then w is not exists. Therefore, search terminates early. Otherwise, a tag t_w is generated for w, to help the server for retrieving the plaintext results (if exists) that match w. Next, the client checks the value of Cnt[w] to construct valid values for the search token (t_w, k_w, Cn). Server uses t_w to gather the plaintext results from the recent search query (line 32). The server also gets the new encrypted entries using K_w, and the counter Cn. The server removes the accessed entries from S_e to keep the index up-to-data. In the second round, client decrypts the encrypted entries, and filters the deleted file identifiers.

Assumptions. Let λ be security parameter, $\text{GenPRF}(1^\lambda), \text{GenPPF}(1^\lambda)$, two key generation functions, F: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a pseudorandom function PRF, h: $\{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a Hash function modeled as random oracle, and G: $\{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ be a pseudorandom permutation function PPF.

Setup (λ; ⊥)
Client:
1- $K_t, K_f \xleftarrow{\$} GenPRF(1^\lambda)$
2- $K_g \xleftarrow{\$} GenPPF(1^\lambda)$
3- $\text{SrchCnt}, \text{Cnt} \leftarrow empty\ map$
4- $\sigma \leftarrow \{\text{SrchCnt}, \text{Cnt}\}$
Server:
5- $S_e, S_r \leftarrow empty\ map$
Update $(K_g, K_f, \sigma, id, w, op;\ S_e)$
Client:
6- If $(\sigma[w] = \perp)$ then
7- SrchCnt$[w] \leftarrow 0$,
8- Cnt$[w] \leftarrow 0$
9- End if
10- Cnt$[w] \leftarrow$ Cnt$[w] + 1$
11- $K_w \leftarrow F_{K_f}(w||\text{SrchCnt}[w])$
12- $Addr \leftarrow h(K_w||\text{Cnt}[w]||0)$
13- $mask \leftarrow h(K_w||\text{Cnt}[w]||1)$
14- $sk \leftarrow F_{K_g}(w||\text{SrchCnt}[w])$
15- $if\ op = "add"\ then\ d \leftarrow 1$
16- $else\ d \leftarrow 0$
17- $Val \leftarrow G_{sk}(id||d) \oplus mask$
18- $Send\ (Addr, Val)\ to\ the\ server$
Server:
19- $S_e[Addr] \leftarrow Val$
Search ($K_g, K_f, K_t\ \sigma, w; S_e, S_r$)
Round1 Client:
20- if $(\sigma\ [w] = \perp)$ then
21- Return ∅
22- End if
23- $t_w \leftarrow F_{K_t}(w)$
24- $Cn = cnt[w]$
25- if $(Cn = 0)$ then
26- $K_w \leftarrow \perp$
27- else
28- $K_w \leftarrow F_{K_f}(w||\text{SrchCnt}[w])$
29- End if
30- $send\ (K_w, t_w, Cn)\ to\ the\ server$
Server:
31- $ID_1, ID_2 \leftarrow \{\}$
Retrieve all last search result
32- $ID_1 \leftarrow S_r[t_w]$
33- if $K_w = \perp$then
34- Return ID_1
35- End if

36- for $i = 1$ to Cn do
37- $Addr \leftarrow h(K_w||Cn||0)$
38- $mask \leftarrow h(K_w||Cn||1)]$
39- $ID_2 \leftarrow ID_2 \cup [S_e[Addr] \oplus mask$
40- $Delete\ S_e[Addr]$
41- End for
42- $send\ ID_1, ID_2\ to\ Client$

Round2 Client:

43- if ($|\mathrm{ID}_2| \neq 0$) then
44- $\mathrm{Cnt}[w] \leftarrow 0, \mathrm{SrchCnt}[w] + 1$
45- End if
46- $sk \leftarrow F_{K_g}(w)$
47- for $i = 1$ to $|\mathrm{ID}_2|$ do
48- $(id||d) \leftarrow G_{sk}^{-1}(ID_2[i])$
49- if $(\mathrm{d} = 1)$ then
50- $ID_1 \leftarrow ID_1 \cup \{id\}$
51- else
52- $ID_1 = ID_1 - \{id\}$
53- End if
54- End for
55- Send ID_1 to server

Server:

56- $S_r[t_w] \leftarrow ID_1$

Figure 1: our proposed scheme

SECURITY PROOF

Our scheme is designed to support forward privacy and Type-II of backward privacy. Forward privacy is achieved since the two elements $(Addr, Val)$ are generated using a pseudorandom function F that receives a new input for each update, making it a difficult for the server to distinguish them from random. Furthermore, even the update operation that is evaluated at the server, still hidden which leads to an empty update leakage. For backward privacy, the client sends to the server a search token that enable it to generate a random location which it observes during the previous updates. This allows for the server to know, for each w, the timestamp for each $update$ operation. Except this, the server does not learn anything. Particularly deletions that cancel specific additions are not revealed to the server. This immediately leads to backward privacy of Type-II. For the search queries, only the access pattern and query pattern are leaked, which is standard leakage in the literature. Notice that, even though a fresh key is generated after each search, our scheme still leaks the search pattern since the searched keywords are not re-encrypted again. The fresh key is used to achieve forward privacy, but not protecting the search pattern. However, the actual keywords under search queries are protected; making the server unable to know them.

The security of our scheme can be formalized in the following theorem.

Theorem 1. Let F be a secure PRF, G be a secure PPR, h is hash function modeled as a random oracle, our scheme is an adaptively secure DSSE scheme with forward and backward privacy, such that its leakage is defined as:

$$\mathcal{L}^{\mathrm{stup}} = \emptyset,\ \mathcal{L}^{\mathrm{Updt}}(op, w, id) = \emptyset \text{ and } \mathcal{L}^{\mathrm{Srch}}(w) = (TimeDB(w), Updates(w), sp(w))$$

Proof. Our approach to prove the security of the proposed scheme is to illustrate that real and ideal games are indistinguishable by any PPT distinguisher. To do so, a PPT simulator $\mathcal{S}$ is constructed to simulate the client behavior such that these is no PPT distinguisher can differentiate between the simulated and the real behaviors. $\mathcal{S}$ utilized the available information of leakage functions to simulated the issued tokens (for $search$ and $update$ queries) and the encrypted index. Notice that we ignore the simulation of encrypted collection since it is out the scope of our work. Recall that the actual contents of tokens and the index are not known by the simulator. Despite this, they can be generated as random values. $\mathcal{S}$ utilizes the leakage functions to know the index size, and how one operation is affected by the recent one. Simulator $\mathcal{S}$ simulates the encrypted entries of index S_e by creating random values of proper sizes. Since the original entries of S_e are resulted from a random oracle h, then no PPT distinguisher can differentiate them from the generated entries. $Search$ and $update$ operations affect each other, making their simulation process a more challenging. This is because, $\mathcal{S}$ have to keep track the dependence of each token and relate it with other one to keep them consistent. $\mathcal{S}$ creates a local version of index S_e, and uses the information of leakage functions to update its local index. This local index is employed to ensure tokens consistency.

We start the proof by generating subsequent games: G_0 to G_4, where G_0 is the real game $DSSEReal_A^{\pi}(\lambda)$ and G_4 is the ideal game $DSSEIdeal_{A,S,\mathcal{L}}^{\pi}(\lambda)$. G4 is simulated by using the leakage function $\mathcal{L}$. The intermediate games are hybrids where each one differs slightly from the previous one. We need to prove that successive hybrids are indistinguishable. Since indistinguishability transits from the first game to the last game, we conclude that real game is indistinguishable from ideal game and complete our proof.

Game G_0. This game is exactly matching our constructed DSSE scheme, i.e.

$$P[\mathrm{DSSEReal}_{\mathcal{A}}^{\pi}(\lambda) = 1] - P[G_0 = 1] = 0$$

Game G_1. This hybrid is the same as G0 but instead of generating $\mathrm{K_w}$ and $\mathrm{t_w}$ by using the function F, these values are generated as random values. To do so, G1 maintains two tables KWT and TWT. KWT is used to store the pairs $(w, SrchCnt[w]$, Kw). When w is searched, the game checks whether $(w, SrchCnt[w])$ is present in KWT or not. If so returns K_w; otherwise choose a random K_w and stores the $(w, SrchCnt[w], K_w)$ in KWT. Table TWT works in a similar way to return t_w but it stores (w, t_w) instead. Notice that G_0 and G_1 are indistinguishable; otherwise there is a PPT distinguisher D_1 that can distinguish between each instance of F from a truly random function. Thus,

$$P[G_0 = 1] - P[G_1 = 1] \leq 2.\mathrm{Adv}_{\mathrm{D}_1,\mathrm{F}}^{\mathrm{prf}}(\lambda) \leq negl(\lambda)$$

Game G_2. This hybrid maintains a table GT to simulate the output of function G. This table stores $(w, SrchCnt[w], id, op, output)$, where output is a random

number. If output is repeated in more than one entry in GT the hybrid aborts. However, the probability of such an event is negligible. Again, to distinguish between G_1 and G_2, there should be a PPT distinguisher D_2 that is able to distinguish G from a true random function, i.e,

$$P[G_1 = 1] - P[G_2 = 1] \leq Adv^{prp}_{D_2,G}(\lambda) \leq negl(\lambda)$$

Game G_3. This hybrid generates the two values $(addr, val)$ of update tokens as random strings instead of using hash function h. Game G_3 is formally described in Figure 2. The random string of $addr$ is generated and stored in table HA (lines 8-9). To get consistent results, this random string is used to program the random oracle during search (lines 22-23) by using a table H_1. The random string val is handled in the same way, where it is generated and stored in table HM (lines 8-9) and programed with the random oracle by using a table H_2 (line 24).

Notice that, H_1 is not programmed with HM immediately. For $w, srchCnt[w], Cnt[w]$, the corresponding value $addr$ is generated during update but it is stored into H_1 only when we searched on its corresponding keyword. Now, consider that the adversary server queries H_1 with (K_w,i) before the next search query, it will get $\widehat{addr}$ which differs on $addr$ with high probability. We denote this event by E.

In this case, we have:

$$P[G_3 = 1] - P[G_2 = 1] \leq P[E].$$

Suppose that the adversary makes at most q queries for H_1 and chooses K_w randomly from $\{0,1\}^\lambda$ then $P[E] \leq \frac{q}{2^\lambda}$ which is a negligible and hence G_3 and G_2 are indistinguishable.

Setup $(\lambda; \perp)$
Client:
1- $\text{SrchCnt}, \text{Cnt} \leftarrow empty\ map$
2- $\sigma \leftarrow \{\text{SrchCnt}, \text{Cnt}\}$

Update $(\sigma, id, w, op;\ S_e)$
Client:
3- if $(Cnt[w] = \perp)$ then
4- $\text{SrchCnt}[w] \leftarrow 0,\ \text{Cnt}[w] \leftarrow 0$
5- End if
6- $\text{Cnt}[w] \leftarrow \text{Cnt}[w] + 1$
7- $K_w \leftarrow KWT(w, SrchCnt[w])$
8- $addr \xleftarrow{\$} \{0,1\}^\lambda$
9- $HA[w, srchCnt[w], Cnt[w]) \leftarrow Addr$
10- $mask \xleftarrow{\$} \{0,1\}^\lambda$
11- $HM[w, srchCnt[w], Cnt[w]) \leftarrow mask$
12- $val \leftarrow GT[w, SrchCnt[w], id, op] \oplus mask$
Send $(addr,\ val)$

Search $(K_g, K_f, K_t\ \sigma, w; S_e, S_r)$
Round1 Client:
1- if $(\sigma[w] = \perp)$ then
2- Return $\emptyset$
3- End if
4- $t_w \leftarrow TWT(w)$
5- $K_w \leftarrow KWT(w, SrchCnt[w])$
6- $Cn \leftarrow Cnt[w]$
7- if $(Cn \neq 0)$ then
8- $K_w = KWT(w, SrchCnt[w])$
9- $For\ i = 1\ to\ Cn$
10- $H_1[K_w, i] \leftarrow HA[w, srchCnt[w], i)$
11- $H_2[K_w, i] \leftarrow HM[w, srchCnt[w], i)$
12- else
13- $K_w \leftarrow \perp$
14- End if
15- *Send* (t_w, K_w, cn) *to server*

Figure 2: Game G_3.

Game G_4. This hybrid represents the values $addr$ and $vddr$ of update token in a different way, without altering the transcripts generated by $Search$ and $Update$ operations. Figure 3 illustrates the work of G_4. Notice that in G_4, the values of $SrchCnt[w]$ and $Cnt[w]$ are generated on the fly during search protocol and they are no needed to generate the random numbers of update protocol. To do so, G4 employs two tables: $Updates$ and SP. The first one is used to capture all the update operations since the last search query, while the second one is used to record the timestamps at which search operations are happen. To execute a search for a keyword w, its corresponding search counter $SrchCnt[w]$ and Cnt[w] are generated and programmed with the random oracles H1 (resp. H2) accordingly (line 14 - 24). Notice that, the adversary server does not observe the differences between G_3 and G_4 since both games output two uniformly random strings for the update protocol, and three (t_w, K_w, cn) outputs, that have the same distributions in the two games, for the search protocol Therefore,

$$P[G_3 = 1] - P[G_4 = 1] = 0$$

Setup $(\lambda; \perp)$
Client:
1- $\text{SrchCnt}, \text{Cnt} \leftarrow empty\ map$
2- $\sigma \leftarrow \{\text{SrchCnt}, \text{Cnt}\}$
3- $t = 0$
4- $Updates, SP \leftarrow empty\ table$

Update $(id, w;\ S_e)$
Client:
5- $Append\ (t, id)\ to\ Updates[w]$
6- $HA[t] \xleftarrow{\$} \{0,1\}^\lambda$
7- $HM[t] \xleftarrow{\$} \{0,1\}^\lambda$
8- *Send* $(HA[t], HM[t])$ to the server
9- $t \leftarrow t + 1$

Search $(w;\ S_e, S_r)$
Round1 Client:
10- $Append\ t\ to\ SP[w]$
11- **If** $Updates[w] = \perp$ **then**
12- $Return\ \emptyset$
13- $ScrhCnt = 0\ //\ compute\ SrchCnt[w]$
14- **For** $i = 1$ **to** $|SP[w]| - 1$
15- **If** $\exists t, id\ s.t\ (t, id) \in Updates[w] \wedge t > SP[w, i] \wedge t < SP[w, i+1]$ **then**
16- $SrchCnt = SrchCnt + 1$
17- $K_w = KWT[w, SrchCnt]$

18- $cnt = 0$ // *Compute the no. of updates since the last sear*
19- $TotalSrch = |SP[w]|$
20- **For** $t = SP[w, TotalSrch - 1]$ **to** $SP[w, TotalSrch]$
21- **If** $\exists id\ s.t\ (t, id) \in Updates[w]$ **then**
22- $cnt = cnt + 1$
23- **Endfor**
24- $H_1[kw, cnt, 0] = HA[t]$
25- $H_2[k_W, cnt, 1] = HM[t] \oplus GT[w, SrchCnt, id]$
26- **If** $cnt = 0$ **then**
27- $K_W = \perp$
28- $t_w \leftarrow TWT(w)$

Send (t_W, k_W, cnt) *to the server*

Figure 3: game G_4

The Simulation. Finally, we construct a simulator $\mathcal{S}$ that employs the leakage function $\mathcal{L}$ to generate the ideal game $DSSEIdeal^{\pi}_{A,S,\mathcal{L}}(\lambda)$ which is identical to game G_4. The only difference is that $\mathcal{S}$ does not know w. Instead it uses the w' $= min\ sp(w)$, which is available in the leakage function. The simulator uses the $Update(w)\ and\ TimeDB(w)$ to simulate game G_4.

Hence: $P[G_4 = 1] - P[DSSEIdeal^{\pi}_{A,S,\mathcal{L}}(\lambda)] = 0$

By combining all the results from all the above-mentioned games, we get that:

$$P[DSSEReal^{\pi}_{A}(\lambda) = 1] - P[DSSEIdeal^{\pi}_{A,S,\mathcal{L}}(\lambda) = 1] < = 2.Adv^{prf}_{D_1,F}(\lambda) + Adv^{ppf}_{D_2,G}(\lambda) \leq negl(\lambda)$$

SYSTEM IMPLEMENTATION

Figure 1 compares our scheme with other schemes in terms of search token generation time at the client-side. Notice that our scheme outperforms the competent schemes. MITRA and MITRA* require more time to generate the search tokens when the matched documents grow. Our results were obtained on a synthetic database DB of size $|DB| = 3 * 10^5$ entries, where entries are randomly generated.

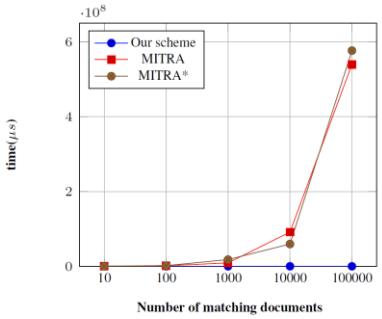

Figure 1. Times for building trapdoor

5. CONCLUSION

Dynamic searchable symmetric encryption schemes with forward and backward privacy are used to search, update the encrypted data without violating the privacy of underling files and search keywords. Proving the security of such schemes is a challenging task. In this paper we design a secure DSSE scheme along with its rigorous security proof. Our proof is designed under the random oracle environment. We notice the importance of backward privacy within the dynamic searchable encryption scheme, where it can thwart many potential attacks.

REFERENCES

[1] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In International conference on the theory and application of cryptology and information security, pages 280–300. Springer, 2013.

[2] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In NDSS, volume 14, pages 23–26. Citeseer, 2014.

[3] Raphël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1465–1482, 2017.

[4] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Annual cryptology conference, pages 353–373. Springer, 2013.

[5] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In International conference on applied cryptography and network security, pages 442–455. Springer, 2005.

[6] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In ACM CCS 2016. 79–88.

[7] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," Information Sciences, vol. 265, 2014.

[8] Mohammad Etemad, Alptekin Küpcü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. Proceedings on Privacy Enhancing Technologies, 2018(1):5–20, 2018.

[9] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: efficient oblivious ram in two rounds with applications to searchable encryption. In Annual International Cryptology Conference, pages 563–592. Springer, 2016.

[10] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 1038–1055, 2018.

[11] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct randolli functions. In 25th Annual Symposium on Foundations of Computer Science, pages 464–479. IEEE, 1984.

[12] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. Journal of the ACM (JACM), 43(3):431–473, 1996.

[13] Matthew D Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In 2015 IEEE Symposium on Security and Privacy, pages 305–320. IEEE, 2015.

[14] Raphael Bost. Σοφος : Forward secure searchable encryption. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 1143–1154, 2016.

[15] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 965–976, 2012.

[16] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pages 668–679, 2015.

[17] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. Dynamic searchable encryption via blind storage. In 2014 IEEE Symposium on Security and Privacy, pages 639–654. IEEE, 2014.

[18] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In NDSS, volume 71, pages 72-75, 2014.

[19] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In: ACM CCS 13. Ed. by AhmadReza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 669–684 (cit. on pp. 30, 87).

[20] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property preserving encrypted databases. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pages 644–655, 2015.

[21] Xiangfu Song, Changyu Dong, Dandan Yuan, Qiuliang Xu, and Minghao Zhao. Forward private searchable symmetric encryption with optimized i/o efficiency. IEEE Transactions on Dependable and Secure Computing, 2018.

[22] M. Islam, M. Kuzu, and M. Kantarcioglu, Access pattern disclosure on searchable encryption: Ramification, attack and mitigation, in Network and Distributed System Security (NDSS), 2012.

[23] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In 25th {USENIX} Security Symposium ({USENIX} Security 16), pages 707–720, 2016.

[24] Bellare, Mihir, and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols." Proceedings of the 1st ACM Conference on Computer and Communications Security. 1993.

[25] Jianting Ning, Jia Xu, Kaitai Liang, Fan Zhang, and Ee-Chien Chang. Passive attacks against searchable encryption. IEEE Transactions on Information Forensics and Security, 14(3):789{802, 2019.

[26] Kamara, Seny, and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. International conference on financial cryptography and data security. Springer, Berlin, Heidelberg, 2013.

[27] S. Chatterjee, S. K. Parshuram Puria, and A. Shah, “Efficient backward private searchable encryption,” Journal of Computer Security, vol. 28, no. 2, pp. 229–267, 2020.