RSPP: A Reliable, Searchable and Privacy-Preserving e-Healthcare System for Cloud-Assisted Body Area Networks

Lei Yang
Department of Electrical
Engineering and Computer Science
The University of Kansas, KS, 66045

Qingji Zheng
Bosch Research and Technology Center
Robert Bosch LLC
Pittsburgh, PA, 15222

Xinxin Fan
Bosch Research and Technology Center
Robert Bosch LLC
Pittsburgh, PA, 15222

***Abstract*—The integration of cloud computing and Internet of Things (IoT) is quickly becoming the key enabler for the digital transformation of the healthcare industry by offering comprehensive improvements in patient engagements, productivity and risk mitigation. This paradigm shift, while bringing numerous benefits and new opportunities to healthcare organizations, has raised a lot of security and privacy concerns. In this paper, we present a reliable, searchable and privacy-preserving e-healthcare system, which takes advantage of emerging cloud storage and IoT infrastructure and enables healthcare service providers (HSPs) to realize remote patient monitoring in a secure and regulatory compliant manner. Our system is built upon a novel dynamic searchable symmetric encryption scheme with forward privacy and delegated verifiability for periodically generated healthcare data. While the forward privacy is achieved by maintaining an increasing counter for each keyword at an IoT gateway, the data owner delegated verifiability comes from the combination of the Bloom filter and aggregate message authentication code. Moreover, our system is able to support multiple HSPs through either data owner assistance or delegation. The detailed security analysis as well as the extensive simulations on a large data set with millions of records demonstrate the practical efficiency of the proposed system for real world healthcare applications.**

I. Introduction

In recent years, with the fast development of cloud computing and Internet of Things (IoT), the conventional healthcare industry is being reshaped to a more flexible and efficient paradigm of e-healthcare. In a typical e-healthcare setting, a group of wearable and/or implantable devices (e.g., smart watches, bracelets, pacemakers, etc.), which forms a wireless body area network (BAN), gathers key vital signs (e.g., heart rate, blood pressure, temperature, pulse oxygen, etc.) from patients at home periodically. Those information is aggregated into a single file called personal health information (PHI) at an IoT gateway and then forwarded to a cloud server for storage. Third-party healthcare service providers (HSPs) can monitor patients' PHI and provide timely diagnosis and reactions by submitting on-demand queries to cloud storage. Although the increasing adoption of cloud computing and IoT services in healthcare industry helps reduce IT cost and improves patient outcomes, security and privacy of PHI are still major concerns as highlighted by the numerous reported data breaches due to malicious attacks, software bugs or accidental errors [1]. In particular, the healthcare regulations such as the Health Insurance Portability and Accountability Act (HIPAA) explicitly require that PHI be secured even as it migrates to the cloud.

While simply encrypting PHI before outsourcing it to the cloud can ensure the regulatory compliance of a healthcare system, it makes PHI utilization (e.g., query by third party HSPs) particularly challenging. Searchable encryption technology (see [2]–[4] for pioneering work), which allows encrypted documents to be searched as is by augmenting them with an encrypted search index, provides a promising solution to addressing the aforementioned dilemma. An important line of research on searchable encryption is searchable symmetric encryption (SSE), which is considered more practical in terms of search efficiency for large datasets when compared to its public key-based counterpart. During the past decade, many provably secure SSE schemes [3], [5]–[12] have been proposed, which make trade-offs among security, search performance and storage overhead by exploring static- [3], [5], [6] and dynamic datasets [7]–[12] as well as various data structures such as an inverted index [3], [7], a document-term matrix [13], a dictionary [10], etc.

We note that previous SSE schemes mainly focus on general search applications on encrypted database. The static SSE schemes that process static datasets and do not support subsequent updates are clearly not suitable for our e-healthcare applications. Moreover, most previous dynamic SSE schemes (except for [11]) work on a setting where a large static dataset is first processed and outsourced to the cloud storage, followed by a number of (infrequent) update operations, which is quite different from the e-healthcare applications where PHI files are created and uploaded to the cloud periodically at a fixed frequency (e.g., every 10 minutes). To prevent the cloud server from inferring sensitive information related to a patient (e.g., activity pattern, diet habit, etc.) based solely on observation of the stored encrypted indices, a dynamic SSE scheme with forward privacy[1] is highly desirable. In addition,

[1]For a dynamic SSE scheme, forward privacy means that when a new keyword and file identifier pair is added, the cloud server does not know anything about this pair [12].

for remotely monitoring patients' health status, HSPs should be able to perform search on PHIs encrypted by patients. Hence, our system should support a multi-user setting where the data owner and data user might be different. Last but not least, the reliability of an e-healthcare system is also critical and any incorrect or incomplete search results could lead to significant consequences, thereby highlighting the requirement for a verification mechanism to be deployed into the system.

Motivated by the above observations, we present a reliable, searchable and privacy-preserving e-healthcare system for cloud assisted BAN in this paper. The proposed system is built upon a novel dynamic SSE scheme with forward privacy and delegated verifiability, which enables both patients and HSPs to conduct privacy-preserving search on the encrypted PHIs stored in the cloud and verify the correctness and completeness of retrieved search results simultaneously. Our contributions can be summarized as follows:

1) We proposed a dedicated and efficient dynamic SSE scheme for e-healthcare applications where PHIs are generated and stored in the cloud periodically. Our scheme is able to achieve a sub-linear search efficiency and forward privacy by maintaining an increasing counter for each keyword at an IoT gateway.
2) We presented an efficient mechanism that provides patient-controlled search capability for HSPs, thereby extending our system to a multi-user setting. This desired property is realized through a novel application of Bloom filter [14] on the data owner (i.e., a patient) side.
3) We designed a lightweight delegated verification scheme based on a combination of Bloom filter, Message Authentication Codes (MACs) and aggregate MACs [15], which enables patients to delegate the capability of verifying the search results to HSPs.

The rest of this paper is organized as follows. We state system model, and design goals in Section II and introduce the notations and preliminaries in Section III. To make the exposition clear, we present the basic construction of our novel dynamic SSE with forward privacy in Section IV, and extend it to the full construction supporting the multi-user setting and verifiability in Section V. We show security analysis and performance evaluation in Section VI and Section VII, respectively. Finally, we discuss the related work in Section VIII and conclude this work in Section IX.

II. Problem Formulation

A. System Model

The system model of our proposed reliable, searchable and privacy-preserving e-healthcare system involves four entities as shown in Fig. 1: a patient, an IoT gateway, a cloud server and several HSPs. The patient is the data owner whose health status is monitored by a group of wearable devices forming a BAN. The IoT gateway is the data aggregator which aggregates the periodically collected data into a single PHI file, extracts keywords, builds an encrypted index, and encrypts the PHI files. The encrypted index and PHI files are then sent to the

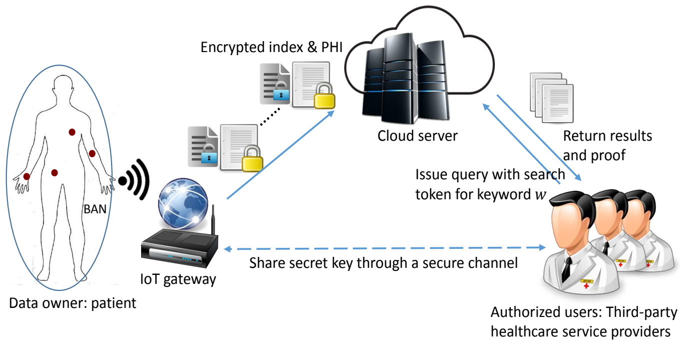
Encrypted index & PHI
Cloud server
Return results and proof
Issue query with search token for keyword w
BAN
IoT gateway
Share secret key through a secure channel
Data owner: patient
Authorized users: Third-party healthcare service providers

Fig. 1: The system model of a reliable, searchable and privacy-preserving e-healthcare system.

cloud server for storage. Multiple HSPs act as the data users that provide healthcare services for the patient by querying and retrieving his/her encrypted PHIs from the cloud. We note that the e-healthcare system described above has the following unique properties with respect to data processing:

- The PHI files are created by the IoT gateway and stored in the cloud *periodically* (e.g., every 10 minutes).
- The PHI files are *always added* into the cloud storage and file deletion or modification is not needed.
- The total number of unique keywords extracted from all the PHI files is not very large, due to the limited range of values for vital signs.

B. Threat Model

As assumed in most previous work on SSE [2], [3], [7], [11], [12], the cloud server is generally "honest-but-curious", thereby faithfully performing the protocol but making inferences about the stored encrypted documents and data owner's private information. More specifically, in our e-healthcare system, the cloud server might try to infer whether a newly uploaded PHI file contains certain keyword or two PHI files contain the same keyword. Furthermore, the cloud server may also observe the queries submitted by HSPs (so-called search pattern) or the search results (so-called access pattern) to determine whether the same keyword is being searched. Additionally, considering the possibility of accidental system errors on the cloud server as well as the potential attacks from external adversaries, the cloud server might return incorrect or incomplete search results to data user. Finally, we assume that there is no collusion between data users and cloud server, or between users.

C. Design Goals

In this work, we aim to design a reliable, searchable and privacy-preserving e-healthcare system which enables third-party HSPs to provide healthcare services for patients by searching on their encrypted PHIs incrementally stored on the cloud in a privacy-preserving and verifiable manner. The design goals of our system are as follows:

1) **Search efficiency**. The search complexity on the cloud should be optimally sub-linear $O(k)$, where k is the number of PHIs containing the queried keyword.

2) **Forward privacy**. The cloud should not learn whether the newly stored PHIs contain some specific keywords.
3) **Multi-user support**. HSPs should be able to perform patient-controlled search on behalf of a patient.
4) **Verifiability**. HSPs should be able to verify the correctness and completeness of the search results.

Note that hiding search and access patterns in a general SSE setting can be achieved using the oblivious RAM (ORAM) [16]. However, ORAM-based schemes, while providing strong protection for privacy, incur significant computational and communication overhead for search. To ensure the practicality of our system, we did not consider employing the ORAM based approach to protect those patterns in this work.

III. Notations, Preliminaries and Definition

A. Notations and Preliminaries

Let $e \leftarrow S$ denote selecting an element e from a set S uniformly at random, $\{0,1\}^n$ be the set of binary strings of length n, $\{0,1\}^*$ be the set of all finite length binary strings, and $||$ denote the concatenation of two strings. The data file f is uniquely identified by the identity $\mathsf{ID}(f)$ and contains a set of distinct keywords $W(f) = \{w_1,...,w_l\}$. Let TBL be a hash table storing key-value pairs (key,val) such that TBL[key] = val, TBL[key] := val denote assigning val to key, and key $\in$ TBL denote that key is an element of the key set in TBL.

Let $\mathcal{F}_1 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, $\mathcal{F}_2 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$, $\mathcal{F}_3 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{3\lambda}$ be three secure pseudorandom functions and $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a secure hash function. Let SE = (SE.GenKey,SE.Enc,SE.Dec) be a semantic secure symmetric encryption where SE.GenKey is the key generation algorithm, SE.Enc is the encryption algorithm and SE.Dec is the decryption algorithm. Let Mac = (Mac.GenKey,Mac.GenMac) be a secure message authentication code scheme, where Mac.GenKey is the key generation algorithm, and Mac.GenMac is the message authentication code generation algorithm.

Bloom filter is a space efficient data structure to represent a set S and allow efficient membership query. A Bloom filter BF is an array of m-bit, which are set to 0 initially, and associated with k independent universal hash functions $\mathcal{H}_1, \ldots, \mathcal{H}_k$, such that $\mathcal{H}_i : \{0,1\}^* \rightarrow \{0, \ldots, m-1\}$. Given $e \in S$, the bits with respect to $\mathcal{H}_i(e), 1 \leq i \leq k$, are set to 1. To query whether e is an element of S or not, one can check whether all bits with respect to $\mathcal{H}_i(e), 1 \leq i \leq k$, are equal to 1. If not, $e \notin S$ for sure. Otherwise, $e \in S$ in a high probability due to the false positive rate. Suppose the outputs of all hash functions are in uniform random distribution and n elements are hashed into the BF, the false positive rate is $(1-e^{-kn/m})^k$. A BF usually associates with two algorithms:

- $\mathsf{BF} \leftarrow \mathsf{BFAdd}(\mathsf{BF}, e)$: This algorithm hashes an element e into the Bloom filter BF.
- $\{0,1\} \leftarrow \mathsf{BFVerify}(\mathsf{BF}, e)$: This algorithm outputs 1 if e is an element of S where all elements were hashed into BF (with certain false positive rate); and 0 otherwise.

B. Definition for Dynamic Symmetric Searchable Encryption

Similar to the notation [12], let $((c_{out}), (s_{out})) \leftarrow protocol((c_{in}), (s_{in}))$ denote the protocol running between the data owner and the server, where the data owner takes as input c_{in} and outputs c_{out}, and the server takes as input s_{in} and outputs s_{out}.

Definition 1: The verifiable DSSE scheme that supports streaming data consists of the following algorithms/protocols:

- $\mathsf{K} \leftarrow \mathsf{GenKey}(1^\lambda)$: Given a security parameter λ, the data owner runs the algorithm to generate the secret key K.
- $((\mathsf{state}'_c),(\mathsf{state}'_s,C)) \leftarrow \mathsf{AddFile}((\mathsf{K},\mathsf{state}_c,f), (\mathsf{state}_s))$: The data owner takes as inputs the secret key K, current state information state_c and the file f containing a set of keywords $W(f)$, and the server takes as input its current state information state_s. The data owner runs this protocol to outsource C (the encryption form of the file f) to the server and updates its own state to state'_c. The server also updates its own state to state'_s. Initially, both state_c and state_s are empty.
- $\mathsf{token} \leftarrow \mathsf{GenToken}(\mathsf{K},\mathsf{state}_c,w)$: The data owner runs this algorithm to generate search token token, by taking as input K, state_c and w.
- $(\mathsf{rst},\mathsf{prof}) \leftarrow \mathsf{Search}(\mathsf{state}_s,\mathsf{token})$: Given the search token token, the server runs this algorithm to output the search result rst consisting of a set of file identifiers. Moreover, the server generates the proof prof showing the correctness of the search result.
- $\{0,1\} \leftarrow \mathsf{SSEVerify}(\mathsf{K},\mathsf{state}_c,w,\mathsf{rst},\mathsf{prof})$: The data owner (or authorized user) runs this algorithm to verify the correctness of the search result rst, given K, state_c, w, and prof.

Basically, the verifiable DSSE scheme supporting streaming data aims to achieve the following security goals: forward privacy, verifiability and confidentiality of outsourced data and queried keyword.

IV. Dynamic SSE Achieving Forward Privacy

For the sake of simplicity, we first present the DSSE construction achieving forward privacy, and leave the full-fledged DSSE design to the next section.

A. Design Rational

Informally, forward privacy in DSSE demands that when adding a new file, the server should not learn whether the newly added file contains certain keyword that has been queried before or not, unless the keyword is queried again. Therefore, it is sufficient to achieve forward privacy if any keyword in the newly added file will not be linked to any encrypted keywords stored in the server.

Instead of using computationally heavy cryptographic primitives (e.g., ORAM), in this paper we exploit the combination of locally stored state information and chaining technique in a subtle way, and utilize the lightweight cryptographic primitives to achieve forward privacy, which is explained as follows.

The data owner associates to each keyword a counter, indicating the number of outsourced encrypted files having

- $\underline{\mathsf{K} \leftarrow \mathsf{GenKey}(1^\lambda)}$: Let $\mathcal{F}_1 : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda, \mathcal{F}_2 : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ be two pseudorandom functions, $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a secure hash function and SE be a secure symmetric key encryption. Given the security parameter λ, the data owner selects $\mathcal{K} \leftarrow \{0,1\}^\lambda$, runs $\mathsf{SE.GenKey}$ to get $\mathcal{K}_{\mathsf{SE}}$, and sets $\mathsf{K} = (\mathcal{K}_{\mathsf{SE}}, \mathcal{K})$.
- $\underline{((\mathsf{state}'_c), (\mathsf{state}'_s, C)) \leftarrow \mathsf{AddFile}((\mathsf{K}, \mathsf{state}_c, f), (\mathsf{state}_s))}$: Suppose that the identifier of file f is $\mathsf{ID}(f)$ and the set of keywords extracted from f is $W(f) = \{w_1, \dots, w_l\}$. Note that when the system was initialized, $\mathsf{state}_c = \mathsf{TBL}_c = \emptyset$ and $\mathsf{state}_s = \mathsf{TBL}_s = \emptyset$ where TBL_c and TBL_c are hash tables. The protocol proceeds as follows:

  The data owner:

```
Let $\mathsf{Ind}$ be an empty set, and run $C \leftarrow \mathsf{SE.Enc}(\mathcal{K}_{\mathsf{SE}}, f)$ for file $f$
for each keyword $w \in W(f)$ do
    Let $\mathcal{K}_{\mathsf{prev}} = 0^\lambda$, $\mathsf{cnt} = 1$ and $\mathsf{cnt}_{\mathsf{prev}} = 0$
    if $w \in \mathsf{TBL}_c$ then
        Retrieve $\mathsf{cnt}$ from $\mathsf{TBL}_c$ with respect to $w$
        Let $\mathcal{K}_{\mathsf{prev}} = \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$, $\mathsf{cnt}_{\mathsf{prev}} = \mathsf{cnt}$ and $\mathsf{cnt} = \mathsf{cnt} + 1$
    end
    Compute $\mathcal{K}_{\mathsf{cnt}} \leftarrow \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$
    Compute $\tau_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$ and $\mu_{\mathsf{cnt}} = \langle \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}_{\mathsf{prev}})||\mathcal{K}_{\mathsf{prev}} \rangle \bigoplus \mathcal{F}_2(\mathcal{K}_{\mathsf{cnt}}, \tau_{\mathsf{cnt}})$
    Let $\mathsf{TBL}_c[w] := \mathsf{cnt}$ and $\mathsf{Ind} = \mathsf{Ind} \bigcup \{(\tau_{\mathsf{cnt}}, \mu_{\mathsf{cnt}})\}$
end
Send $(C, \mathsf{ID}(f), \mathsf{Ind})$ to the server and let $\mathsf{state}'_c = \mathsf{TBL}_c$
```

  The server:

  Upon receiving $(C, \mathsf{ID}(f), \mathsf{Ind})$ from the data owner, the server proceeds as follows:

```
for each $(\tau, \mu) \in \mathsf{Ind}$ do
    Let $\mathsf{TBL}_s[\tau] := \mu||\mathsf{ID}(f)$
end
Store $C$ locally and set $\mathsf{state}'_s = \mathsf{TBL}_s$
```

- $\underline{\mathsf{token} \leftarrow \mathsf{GenToken}(\mathsf{K}, \mathsf{state}_c, w)}$: Given the keyword w to be queried, the data owner generates the search token as follows: (i) Retrieve cnt from state_c with respect to w, (ii) Compute $\mathcal{K}_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$ and (iii) Let $\mathsf{token} = (\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}), \mathcal{K}_{\mathsf{cnt}})$, which will be sent to the server.
- $\underline{\mathsf{rst} \leftarrow \mathsf{Search}(\mathsf{state}_s, \mathsf{token})}$: Given $\mathsf{token} = (\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}), \mathcal{K}_{\mathsf{cnt}})$, the server conducts the search by letting rst be an empty set, $\tau' = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$, $\mathcal{K}' = \mathcal{K}_{\mathsf{cnt}}$, and running the following algorithm:

```
while $\mathcal{K}' \neq 0^\lambda$ do
    Retrieve $\mu||\mathsf{ID}(f)$ from $\mathsf{TBL}_s$ with respect to $\tau'$ and let $\mathsf{rst} = \mathsf{rst} \bigcup \{\mathsf{ID}(f)\}$
    Let $\tau'||\mathcal{K}' = \mu \bigoplus \mathcal{F}_2(\mathcal{K}', \tau')$ (which results in $\mathcal{F}_1(\mathcal{K}, w||(i-1))||\mathcal{K}_{i-1}$ if the current counter is $i$)
end
Return $\mathsf{rst}$ as the search result
```

Fig. 2: The DSSE construction achieving forward privacy. Note that downloaded encrypted files can be decrypted with $\mathcal{K}_{\mathsf{SE}}$.

the keyword so far. That is, the data owner locally maintains the state information (i.e., pairs of keyword and counter). Suppose the counter associated to keyword w is cnt, the index with respect to w, stored in the server, is a collection of tuples $\{(\tau_1, \mathsf{ID}(f_1)), \dots, (\tau_{\mathsf{cnt}}, \mathsf{ID}(f_{\mathsf{cnt}}))\}$ where $\tau_i = \mathcal{F}_1(\mathcal{K}, w||i), 1 \leq i \leq \mathsf{cnt}, \mathcal{F}_1$ is a secure pseudorandom function, $\mathcal{K}$ is a private key and $f_1, \dots, f_{\mathsf{cnt}}$ are files having keyword w. When adding a new file f containing the keyword w, the data owner sends to the server the following tuple

$$(\tau_{\mathsf{cnt}+1},\ \mathsf{ID}(f))$$

where $\tau_{\mathsf{cnt}+1} = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}+1)$. Thanks to $\mathcal{F}_1$, without knowing $\mathcal{K}$ the server cannot know whether $\tau_{\mathsf{cnt}+1}$ is generated from the same keyword as that of $\tau_i, 1 \leq i \leq \mathsf{cnt}$. Note that the data owner does not need to maintain all previous states for each keyword because file deletion is not needed in healthcare.

While binding counter to a keyword can break the correlation of two identical keywords, it raises another challenge: given one search token generated from the keyword and the counter, the server can only retrieve one single file identifier. That is, to retrieve all file identifiers having the specific keyword, the data owner has to enumerate all previous counters and generate search tokens, which is rather costly in term of bandwidth for search.

To mitigate this disadvantage, we use the following chaining technique, which implicitly links the tuples corresponding to the same keyword together (let $\tau_i = \mathcal{F}_1(\mathcal{K},w||i)$,$0 \leq i \leq \mathsf{cnt}$):

$$\begin{array}{lll} \tau_1, & \langle \tau_0||0^\lambda \rangle \bigoplus \mathcal{F}_2(\mathcal{K}_1, \tau_1), & \mathsf{ID}(f_1) \\ \tau_2, & \langle \tau_1||\mathcal{K}_1 \rangle \bigoplus \mathcal{F}_2(\mathcal{K}_2, \tau_2), & \mathsf{ID}(f_2) \\ & \dots & \\ \tau_{\mathsf{cnt}}, & \langle \tau_{\mathsf{cnt}-1}||\mathcal{K}_{\mathsf{cnt}-1} \rangle \bigoplus \mathcal{F}_2(\mathcal{K}_{\mathsf{cnt}}, \tau_{\mathsf{cnt}}), & \mathsf{ID}(f_{\mathsf{cnt}}) \end{array}$$

where $\mathcal{F}_2$ is another secure pseudorandom function and

$\mathcal{K}_i, 1 \leq i \leq \mathsf{cnt}$, is a random key derived from the counter i. Obviously, without knowing $\mathcal{K}_i, i \geq \mathsf{cnt}$, the server cannot correlate τ_{cnt} with $\tau_j, j < \mathsf{cnt}$, even though they might be generated from the same keyword (but different counter). On the other hand, given τ_{cnt} and $\mathcal{K}_{\mathsf{cnt}}$, the server is able to obtain $\mathsf{ID}(f_{\mathsf{cnt}})$ and recover $\tau_{\mathsf{cnt}-1}$ and $\mathcal{K}_{\mathsf{cnt}-1}$ by computing

$$\langle \tau_{\mathsf{cnt}-1} || \mathcal{K}_{\mathsf{cnt}-1} \rangle \bigoplus \mathcal{F}_2(\mathcal{K}_{\mathsf{cnt}}, \tau_{\mathsf{cnt}}) \bigoplus \mathcal{F}_2(\mathcal{K}_{\mathsf{cnt}}, \tau_{\mathsf{cnt}}).$$

The server then obtains all file identifiers by iterating such process until that the key is λ-bit of zero.

B. Construction

We show the construction in Fig. 2. Here the random key $\mathcal{K}_{\mathsf{cnt}}$ for keyword w is generated by applying the pseudorandom function such that $\mathcal{K}_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$. In addition, the data owner stores the state information (i.e., pairs of (w, cnt)) in the hash table TBL_c, which maps keyword w to the counter cnt. On the other hand, the server also stores the state information (i.e., the encrypted index) in the hash table TBL_s. We can see that given the keyword w, the search complexity is linear to the number of files containing w, which is sublinear to the number of outsourced encrypted files.

Optimization I: Speed up search operation. Note that the server might be able to speed up the search further: given $\mathsf{token} = (\tau_{\mathsf{cnt}}, \mathcal{K}_{\mathsf{cnt}})$ where $\tau_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$, the server can update its state information by setting $\mathsf{TBL}_s[\tau_{\mathsf{cnt}}] = \perp ||\mathsf{rst}$, where $\perp$ is a stop sign and rst is the search result with respect to token. By doing this, the server not only accelerates the search without repeating the iterations, but also saves the storage by storing file identifiers only.

V. Full-fledged DSSE Construction

In this section, we present the full-fledged DSSE. In contrast to the DSSE presented above, the full-fledged DSSE not only achieves forward privacy, but also supports search capability enforcement and delegated verifiability, where the former allows the data owner (i.e., patients) to enforce controlled search capability, and the latter enables authorized data users (i.e., HSPs) to verify the correctness of the search result.

A. High Level Idea

Search Capability Enforcement. In order to enforce search capability, we need to resolve two questions: (i) how to grant authorized data users with search capability; (ii) how to revoke authorized data user's privilege if necessary. Furthermore, we require that the approach should be efficient without extensive interaction between the data owner and authorized data users.

Granting search capability requires the data owner to distribute the secret key (i.e., $\mathcal{K}_{\mathsf{SE}}$ and $\mathcal{K}$) and state information (i.e., the counter for each keyword) to authorized data users efficiently and securely. While secret key distribution can be done efficiently through a one-time off-line setup, state information distribution might be costly because authentication (between the data owner and the authorized user) is needed when authorized data users fetch the fresh state information, which is frequently updated. Note that making the data owner's state information public (even if encrypted) will harm the forward privacy because the server can infer which keyword (or encrypted keyword) was contained in the newly added file.

To address the above issue, we adopt the "document-and-guess" approach: The server maintains a Bloom filter BF_s, and puts each received encrypted keyword $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$ into the Bloom filter BF_s, and the authorized user, having the secret key already and fetching BF_s from the server, can guess the latest counter value by enumerating $(1, \ldots, \mathsf{cnt}, \mathsf{cnt}+1)$ such that $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$ is an element hashed to BF_s but $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}+1)$ not (suppose the false positive rate of BF_s is extremely low, e.g., 2^{-30} in our experiments).

On the other hand, in order to allow the data owner to revoke authorized users' search capability, we use the group key idea: The data owner generates a symmetric key r, which is securely shared with the server and all authorized users, such that the search token of keyword w generated by authorized users should be $\mathsf{SE.Enc}(r, \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})||\mathcal{K}_{\mathsf{cnt}})$ and the server can recover $(\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}), \mathcal{K}_{\mathsf{cnt}})$ with the stored r via $\mathsf{SE.Dec}$, where SE is a secure symmetric encryption. When an authorized data user was revoked, the data owner only needs to update the group key r to r' and the revoked user cannot generate valid search token without knowing r'.

Delegated Verifiability. The purpose of delegated verifiability is to allow authorized users (including the data owner) to verify that (i) correctness and completeness of search result, meaning the search result correctly consists of all file identifiers; and (ii) the integrity of the retrieved data files.

First, authorized users can leverage the counter value (if existing) to check whether the server returned the correct number of file identifiers because the counter value indicates the number of files having the specific keyword. Hence, in order to assure that authorized users get correct counter value (which is guessed from BF_s), we need to enable the data user to verify that the cloud faithfully inserts the keywords into Bloom filer. To do so, the data owner also maintains a Bloom filter BF_c, which is built from $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$, and generates a MAC on BF_c (together with a time stamp). If the server operates correctly, $\mathsf{BF}_c = \mathsf{BF}_s$ holds. Thus, only the MAC is uploaded to the server, which is then used by authorized users to check the integrity of the received BF_s to assure the correctness of the guessing counter value.

However, only assuring correct number of file identifiers is not enough, authorized users need to verify the correctness of the retrieved files with respect to the keyword w. To achieve this, each keyword is associated to an aggregate MAC, which is the result of aggregating MACs of all outsourced encrypted files containing w.

B. Main Construction

Based on the above ideas, we present the full-fledged DSSE construction as shown in Fig. 3, which highlights the difference from the basic construction in red color. The data owner maintains the state information (i.e., tuples of $(w, \mathsf{cnt}, \gamma_{\mathsf{cnt}})$) with a hash table TBL_c mapping w to $\mathsf{cnt}, \gamma_{\mathsf{cnt}}$, where γ_{cnt} is the aggregation of the MAC for the concatenation of the file

- $\underline{\mathsf{K} \leftarrow \mathsf{GenKey}(1^\lambda)}$: Let $\mathcal{F}_1 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, $\mathcal{F}_3 : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{3\lambda}$ be two pseudorandom functions, $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$ be a secure hash function, SE be a secure symmetric key encryption, Mac be a secure message authentication code. Given the security parameter λ, the data owner selects $\mathcal{K} \leftarrow \{0,1\}^\lambda$, runs SE.GenKey to get $\mathcal{K}_{\mathsf{SE}}$, runs Mac.GenKey to get $\mathcal{K}_{\mathsf{Mac}}$, and sets $\mathsf{K} = (\mathcal{K}, \mathcal{K}_{\mathsf{SE}}, \mathcal{K}_{\mathsf{Mac}})$.
- $\underline{((\mathsf{state}'_c), (\mathsf{state}'_s, C)) \leftarrow \mathsf{AddFile}((\mathsf{K}, \mathsf{state}_c, f), (\mathsf{state}_s))}$: Suppose that the identifier of file f is $\mathsf{ID}(f)$ and the set of keywords extracted from f is $W(f) = \{w_1, \ldots, w_l\}$. Note that when the system was initialized, $\mathsf{state}_c = (\mathsf{TBL}_c = \emptyset, \mathsf{BF}_c = \emptyset)$ and $\mathsf{state}_s = (\mathsf{TBL}_s = \emptyset, \mathsf{BF}_s = \emptyset)$ where TBL_c and TBL_s are two hash tables, and BF_c and BF_s are two Bloom filters. The protocol proceeds as follows:

  The data owner:

  ```
  Let Ind be an empty set, and run $C \leftarrow \mathsf{SE.Enc}(\mathcal{K}_{\mathsf{SE}}, f)$ for file $f$
  for each keyword $w \in W(f)$ do
      Let $\mathcal{K}_{\mathsf{prev}} = 0^\lambda$, $\mathsf{cnt}_{\mathsf{prev}} = 0$, $\mathsf{cnt} = 1$, $\gamma_{\mathsf{prev}} = 0^\lambda$        ($\gamma_{\mathsf{prev}}$ is an aggregate MAC)
      if $w \in \mathsf{TBL}_c$ then
          Retrieve $(\mathsf{cnt}, \gamma_{\mathsf{cnt}})$ from $\mathsf{TBL}_c$ with respect to $w$
          Let $\mathcal{K}_{\mathsf{prev}} = \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$, $\mathsf{cnt}_{\mathsf{prev}} = \mathsf{cnt}$, $\gamma_{\mathsf{prev}} = \gamma_{\mathsf{cnt}}$, and $\mathsf{cnt} = \mathsf{cnt} + 1$
      end
      Compute $\mathcal{K}_{\mathsf{cnt}} \leftarrow \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$, $\gamma_{\mathsf{cnt}} = \gamma_{\mathsf{prev}} \bigoplus \mathsf{Mac.GenMac}(\mathcal{K}_{\mathsf{Mac}}, C||w)$ (The output of Mac.GenMac is
      $\lambda$-bit length)
      Compute $\tau_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$, $\mu_{\mathsf{cnt}} = \langle \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}_{\mathsf{prev}})||\mathcal{K}_{\mathsf{prev}}||\gamma_{\mathsf{cnt}} \rangle \bigoplus \mathcal{F}_3(\mathcal{K}_{\mathsf{cnt}}, \tau_{\mathsf{cnt}})$
      Compute $\mathsf{BF}_c \leftarrow \mathsf{BFAdd}(\mathsf{BF}_c, \tau_{\mathsf{cnt}})$
      Let $\mathsf{TBL}_c[w] := (\mathsf{cnt}, \gamma_{\mathsf{cnt}})$, $\mathsf{Ind} = \mathsf{Ind} \bigcup \{(\tau_{\mathsf{cnt}}, \mu_{\mathsf{cnt}})\}$
  end
  Generate the MAC $\sigma \leftarrow \mathsf{Mac.GenMac}(\mathcal{K}_{\mathsf{Mac}}, \mathsf{BF}_c||T)$ where $T$ is the current time stamp
  Send $(C, \mathsf{ID}(f), \mathsf{Ind}, \sigma, T)$ to the server and let $\mathsf{state}'_c = (\mathsf{TBL}_c, \mathsf{BF}_c)$
  ```

  The server:

  Upon receiving $(C, \mathsf{ID}(f), \mathsf{Ind}, \sigma, T)$ from the data owner, the server proceeds as follows:

  ```
  for each $(\tau, \mu) \in \mathsf{Ind}$ do
      Let $\mathsf{TBL}_s[\tau] := \mu||\mathsf{ID}(f)$ and $\mathsf{BF}_s \leftarrow \mathsf{BFAdd}(\mathsf{BF}_s, \tau)$
  end
  Store $C$ locally and set $\mathsf{state}'_s = (\mathsf{TBL}_s, \mathsf{BF}_s, \sigma, T)$
  ```

Suppose the data owner generated $r \leftarrow \mathsf{SE.GenKey}$ and securely shared r with authorized users and the server.

- $\underline{\mathsf{token} \leftarrow \mathsf{GenToken}(\mathsf{K}, \mathsf{BF}_c, r, w)}$: The data owner generates the search token as follows: (i) Retrieve $(\mathsf{cnt}, \gamma_{\mathsf{cnt}})$ from TBL_c with respect to w; (ii) Compute $\mathcal{K}_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$; and (iii) Let $\mathsf{token} = \mathsf{SE.Enc}(r, \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})||\mathcal{K}_{\mathsf{cnt}})$, which will be sent to the server.
- $\underline{(\mathsf{rst}, \mathsf{prof}) \leftarrow \mathsf{Search}(\mathsf{state}_s, r, \mathsf{token})}$: The server runs $\mathsf{SE.Dec}(r, \mathsf{token})$ to get $(\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})||\mathcal{K}_{\mathsf{cnt}}$, and conducts the search by retrieving $\mu_{\mathsf{cnt}}||\mathsf{ID}(f)$ from TBL_s with respect to $\tau' = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$, computing $\mu_{\mathsf{cnt}} \bigoplus \mathcal{F}_3(\mathcal{K}_{\mathsf{cnt}}, \tau')$ to get γ_{cnt}, letting $\mathcal{K}' = \mathcal{K}_{\mathsf{cnt}}$, $\mathsf{prof} = (\sigma, T, \mathsf{BF}_s, \gamma_{\mathsf{cnt}})$, $\mathsf{rst} = \emptyset$, and

  ```
  while $\mathcal{K}' \neq 0^\lambda$ do
      Retrieve $\mu||\mathsf{ID}(f)$ from $\mathsf{TBL}_s$ with respect to $\tau'$, and let $\mathsf{rst} = \mathsf{rst} \bigcup \{\mathsf{ID}(f)\}$
      Let $\tau'||\mathcal{K}'||\gamma' = \mu \bigoplus \mathcal{F}_3(\mathcal{K}', \tau')$ (which results in $\mathcal{F}_1(\mathcal{K}, w||(i-1))||\mathcal{K}_{i-1}||\gamma_{i-1}$ if the current counter is $i$)
  end
  Return rst as the search result and prof as the proof
  ```

- $\underline{\mathsf{SSEVerify}(K, w, \mathsf{cnt}, \mathsf{rst}, \mathsf{prof})}$: Given $\mathsf{prof} = (\sigma, T, \mathsf{BF}_s, \gamma_{\mathsf{cnt}})$, the data owner check whether the size of rst is equal to the counter cnt or not. If not, then return 0 and abort. Otherwise, the verification proceeds as follows:
  - Given $\mathsf{ID}(f_i) \in \mathsf{rst}, 1 \leq i \leq \mathsf{cnt}$, fetch encrypted data files $C_1, \ldots, C_{\mathsf{cnt}}$ from the server.
  - If both equations hold, then output 1; otherwise output 0 (The data owner might not check Eq.(2) because of knowing correct cnt):

$$\bigoplus_{i=1}^{\mathsf{cnt}} \mathsf{Mac.GenMac}(\mathcal{K}_{\mathsf{Mac}}, C_i||w) \overset{?}{=} \gamma_{\mathsf{cnt}} \;\; (1) \qquad\qquad \mathsf{Mac.GenMac}(\mathcal{K}_{\mathsf{Mac}}, \mathsf{BF}_s||T) \overset{?}{=} \sigma \;\; (2)$$

Fig. 3: The full-fledged DSSE construction achieving forward privacy, search capability enforcement and delegated verifiability. Note that the downloaded encrypted files can be decrypted with $\mathcal{K}_{\mathsf{SE}}$.

and w so far. The reason of concatenating the file and w as input, rather using the file itself, is to prevent the replacement attack: given keyword w_1, the server might intentionally return the search result for another keyword w_2, an aggregate MAC and the set of file identifiers, which has the same number of file identifiers as that for keyword w_1.

Also, the data owner uses the timestamp T (together with the Bloom filter BF_c) to generate the MAC for preventing the replaying attack that the server might possibly return stale search result. We implicitly leverage the fact that the new file is periodically uploaded (e.g., every 10 minutes), so that authorized users can use the timestamp T to assure the aggregate MAC is newly generated by the data owner.

Due to the lack of knowledge about cnt, authorized users (other than the data owner) generate the search token as shown in Fig. 4, where the WHILE loop is to guess the counter value. Note that with the guessing counter value and the shared key from the data owner, authorized users are able to run $\mathsf{SSEVerify}$ to verify the correctness of the research result.

```
token ← GenToken(K, BF_s, r, w): After fetching the
Bloom filter BF_s from the server, the authorized data
user generates the search token as follows:
  Let cnt = 1;
  while TRUE do
    $\tau_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$
    if $\mathsf{BFVerify}(\mathsf{BF}_s, \tau_{\mathsf{cnt}})$ outputs 1 then
      $\mathsf{cnt} = \mathsf{cnt} + 1$
    else
      $\mathsf{cnt} = \mathsf{cnt} - 1$
      break;
    end
  end
  Compute $\mathcal{K}_{\mathsf{cnt}} = \mathcal{F}_1(\mathcal{K}, \mathcal{H}(w||\mathsf{cnt}))$;
  Let token = $\mathsf{SE.Enc}(r, \mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})||\mathcal{K}_{\mathsf{cnt}})$,
  which will be sent to the server.
```

Fig. 4: The algorithm for the authorized user generating search token. The data owner has already distributed $K = (\mathcal{K}, \mathcal{K}_{\mathsf{SE}}, \mathcal{K}_{\mathsf{Mac}})$ and r to the authorized user.

Optimization II: Speed up guessing the latest counter with binary search. Instead of guessing the counter value linearly, authorized users can use the binary search to accelerate the guessing: The authorized user sets a large enough upper bound Max, and conducts the binary search for the latest counter cnt within $[1, Max]$ such that $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$ is an element hashed to BF_s while $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt}+1)$ not.

Optimization III: Reduce the number of elements hashed to BF_s. Note that the number of elements hashed into BF_s might become huge due to the increasing counter value cnt when generating $\mathcal{F}_1(\mathcal{K}, w||\mathsf{cnt})$ for keyword w. This results into a drawback: In order to keep low false positive rate, the size of BF_s becomes very large, which incurs costly bandwidth when authorized users retrieve it from the server. To get rid of it, the "regular update" strategy can be used:

- Given the state information TBL_c, the data owner regularly (e.g., annually) generates a new Bloom filter BF_c, which implicitly stores the current counter cnt_L for each keyword w, generates the MAC and sends BF_c and the MAC to the server.
- The server lets $\mathsf{BF}_s = \mathsf{BF}_c$ and proceeds as in Fig. 3.
- After receiving BF_s, the authorized user extracts the counter cnt_L first, and then guesses the latest counter starting from cnt_L.

By doing this, BF_s only contains elements with counters beginning with cnt_L (rather than from 1) for keyword w, and therefore its size can be reduced when keeping the same false positive rate. In addition, implicitly storing cnt_L for keyword w in BF_c can be done as follows: Given cnt_L, the data owner hashes $\mathcal{F}_1(\mathcal{K}, w||\mathsf{pos}||\mathsf{digit}_{\mathsf{pos}})$ to BF_c where $\mathsf{digit}_{\mathsf{pos}}$ is the least significant digit of cnt_L when $\mathsf{pos} = 1$, and the authorized user can guess cnt_L by enumerating the combination of $\mathsf{pos} = 1, \ldots$ and $\mathsf{digit}_{\mathsf{pos}} = 0, \ldots, 9$. For example, given $\mathsf{cnt}_L = 456$ for keyword w, $\mathcal{F}_1(\mathcal{K}, w||1||6), \mathcal{F}_1(\mathcal{K}, w||2||5)$ and $\mathcal{F}_1(\mathcal{K}, w||3||4)$ were hashed to BF_c. Authorized users can guess cnt_L by enumerating pos and $\mathsf{digit}_{\mathsf{pos}}$ and checking BF_s (since $\mathsf{BF}_s = \mathsf{BF}_c$) to determine whether $\mathcal{F}_1(\mathcal{K}, w||\mathsf{pos}||\mathsf{digit}_{\mathsf{pos}})$ has been hashed into BF_s or not, until that there exists some pos such that none of elements $\mathcal{F}_1(\mathcal{K}, w||\mathsf{pos}||\mathsf{digit}_{\mathsf{pos}}), \mathsf{digit}_{\mathsf{pos}} = 0, \ldots, 9$, was hashed into BF_s.

VI. Security Analysis

We evaluate the security of our full-fledged construction to show that it achieves the security goals described in Section II-C. We skip the formal proof here (which occurs in the full version of this work) due to the space limit.

Data confidentiality. The outsourced files are encrypted with the secure symmetric encryption together with secret key $\mathcal{K}_{\mathsf{SE}}$. Without leaking $\mathcal{K}_{\mathsf{SE}}$ to the server, data confidentiality is naturally assured by the secure symmetric encryption.

Index confidentiality. Since each keyword in the index (i.e., $state_s$) is encrypted by the secure pseudorandom function $\mathcal{F}_1$, without knowing the secret key, the server cannot learn the keyword from the index.

Forward privacy. As discussed in the Section V, our construction encrypts the combination of the increasing counter and the keyword together, which makes the server unable to link the keyword in the newly added file to any stored encrypted keyword, without knowing the secret key $\mathcal{K}$. In addition, a secure pseudorandom function is used to mask the connection of tuples generated from the same keyword but with consecutive counter values, without knowing the corresponding secret key, the server cannot correlate these tuples together. That is, the server cannot know whether the newly added file contains any stored encrypted keyword, without knowing the secret keys for the pseudorandom functions.

Search token privacy. The keyword associated with the search token is protected with a secure pseudorandom function. Without knowing the key $\mathcal{K}$, the server cannot learn the keyword.

Search capability enforcement. Our construction implicitly shares the state information using the Bloom filter, and uses the group key to assure that only authorized users can generate valid search tokens. Therefore, the data owner can enforce the search capability securely (note that the cloud and users are not allowed to collude in our assumption).

Verifiability. Our construction uses the timestamp and the MAC to assure the freshness and correctness of Bloom filter BF_s, which further assures the correctness of the counter value for any keyword, forcing the server honestly returning correct number of the encrypted files. Moreover, the construction uses the aggregate MAC to assure the integrity of the returned files with respect to the keyword. Therefore, given the secure message authentication scheme, our construction assures that the authorized users and data owner can correctly verify the returned search result with an overwhelming probability.

VII. Performance Evaluation

In this section, we present the empirical performance result by simulating the e-healthcare system with the full-fledged DSSE implementation.

Implementation: We implemented the full-fledged DSSE in JAVA, and instantiated $\mathcal{F}_1, \mathsf{Mac}$ with HMAC-SHA-1, $\mathcal{F}_3$ with HMAC-SHA-512, SE with AES and $\mathcal{H}$ with SHA-1. In addition, we implemented all optimizations as mentioned above. We simulated the e-healthcare system by developing three separate processes for the data owner, the server and the authorized user respectively. The three processes communicate with each other via RESTful API, and were running in a laptop with 2.5GHz Intel i5 CPU, 8GB RAM and MAC OS.

Dataset: In the experiment each PHI file consists of 15 pairs of attribute[2] in the format of attribute:value, which is treated as one single keyword (e.g., $w = heartbeat : 75$). To simulate the scenario that the IoT gateway assembles and uploads a new PHI file in every 10 minutes and lasts for 20-year, 1,051,200 synthesized PHI files were uploaded.

Performance on the Data Owner. The average time for the data owner running $\mathsf{AddFile}$ is 190 milliseconds, and the size of hash table (i.e, TBL_c) is around 1.3MB after uploading one million PHI files. The data owner also maintains a Bloom filter (i.e., BF_c) of around 5MB by setting the false positive rate as 2^{-30}, and updates it every year (i.e., after adding $144 \times 365 = 52,560$ new files as in Optimization III). We note that if the Bloom filter can be updated more frequently (e.g., less than every year), the size of the Bloom filter can be further reduced.

Performance on the Server. We note that given a search token for keyword w at time T_{i+1}, the complexity of running Search is linear to the number of encrypted files that contain w and were uploaded within the interval of T_i and T_{i+1} (sublinear to the number of encrypted files), where T_i is the last time when w was searched for (The time of initializing the system can be regarded as T_0, at which the search result for any keyword is null). The reason is that, with Optimization I, the server stores in a consecutive manner all identifiers of encrypted files having w at T_i, and can access them in a constant time thereafter. Therefore, we evaluated the search performance in the two scenarios: (i) new keyword search, which simulates that keyword w has never been queried before, and (ii) recurring keyword search, which simulates that keyword w has been queried before. Fig. 5 shows the performance. We can see that the search performance is closely related to the number of newly added files within the interval between two consecutive queries for the same keyword. In addition, we can see that the search performance is quite practical since returning 100 files identifiers for new keyword search (resp. recurring keyword search) only costs around 2 seconds (resp. 1 second) (note that one million files and the corresponding index were stored in the server).

[2]The attributes include heartbeat, blood sugar, blood pressure, temperature and so on as in http://www.clouddx.com/downloads/Heart-Friendly-Report-2015-12-24-092313.pdf

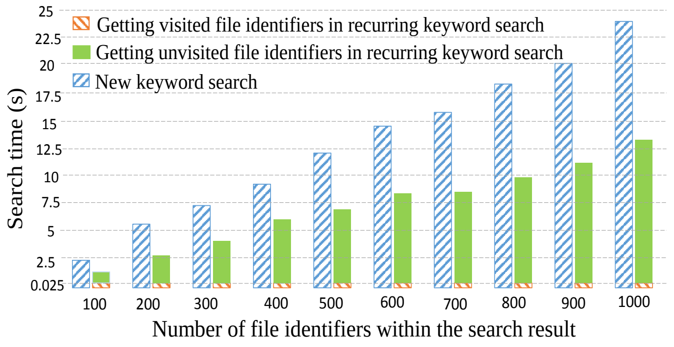

Fig. 5: Performance for search operation running by the server storing one million files and the corresponding index (i.e., state_s). Note that for recurring keyword search, half the number of file identifiers in the search result were newly added since the last time of the same keyword query (called unvisited identifiers), and the other half has been added to the server before the last time of the same keyword query (called visited identifiers).

Performance on the Authorized User. The time for the authorized user generating search token can be neglected (approximately 10 ms) due to the binary search (Optimization II). Therefore, we concentrated on the execution time for the authorized user verifying the correctness of the search result. The performance result is shown in Fig. 6, where we divided the verification time into two parts: one is for verifying the correctness of the Bloom filter (i.e., BF_s retrieved from the server) and the other one is for verifying the aggregate MAC over all returned files. We can see that the time for verifying the correctness of the Bloom filter is quite similar (e.g., around 55 ms in our experiments) no matter how many files are within the search result, and the time of verifying the aggregate MAC over all returned files is linear to the number of files. We can see that verification is practical because, even when dealing with the search result having 1,000 files, the verification time is only around 135 ms.

VIII. Related Work

Cloud-assisted IoT system has become a popular design paradigm in many applications [17]–[20], since the powerful

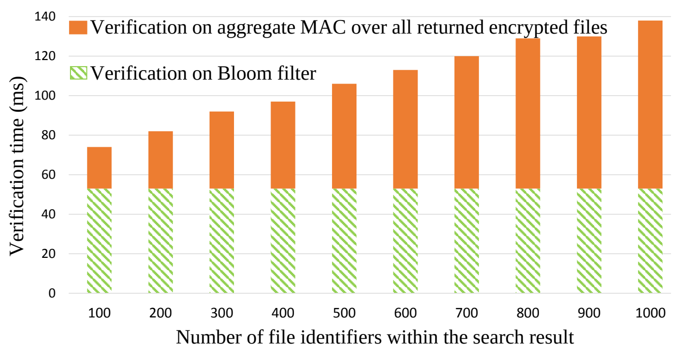

Fig. 6: Performance for the authorized user verifying the correctness of the search result, i.e., verifying the correctness of the Bloom filter and aggregate MAC over all returned files.

computation and storage capabilities of cloud can overcome the constrains of IoT devices. This paper particularly relates to searchable encryption in e-healthcare:

Searchable Encryption. Song *et al.* [2] first explored the problem of searchable symmetric encryption and presented a scheme with linear search time. Curtmola *et al.* [3] gave the first inverted index based scheme to achieve sub-linear search time. Although this scheme greatly boosts search efficiency, it does not support dynamic dataset. Since then, several schemes [7]–[12] about dynamic SSE have been proposed, among which [7]–[10] fail to provide forward privacy. Moreover, the previous work in [12] offers forward privacy using a complicated hierarchical data structure, whereas the contribution in [11] only achieves limited forward privacy (i.e., leaks the keywords contained in a new file if they have been searched for in the past). Besides dynamic SSE, verifiable SSE have been studied by [13], [21]–[23], which enables users to verify search results by using some verifiable structure such as the Merkle tree or an accumulator. However, previous work did not pay special attention to dataset with sequentially added files, which might leak additional information during the process of updating verifiable structure.

Secure data storage for e-healthcare. Several searchable encryption schemes [17]–[19] have been proposed for e-healthcare applications. Tan *et al.* [17] proposed a lightweight IBE scheme to encrypt the sensing data and store it on a cloud. However, their public-key based scheme makes search over encrypted data very inefficient. Li *et al.* [18] presented an authorized search scheme over encrypted health data, which aims to realize search in a multi-user setting by enforcing fine-grained authorization before performing search operations. However, their search scheme is based on the predicate encryption, which is less efficient than SSE. Tong *et al.* [19] proposed a SSE-based healthcare system, which achieves high search efficiency and partially hides the search and access patterns by using the redundancy. However, their scheme depends on a trusted private cloud and is not able to support dynamic data.

IX. Conclusion

In this paper, we proposed a reliable, searchable and privacy-preserving e-healthcare system. The core of our system is a novel and full-fledged dynamic SSE scheme with forward privacy and delegated verifiability, which is dedicatedly designed to protect sensitive PHI files on cloud storage and enable HSPs to search on the encrypted PHI under the control of patients. The salient features such as forward privacy and delegated verifiability are achieved by a unique combination of the increasing counter, Bloom filter and aggregate MAC. Our experimental results and security analysis demonstrate that the proposed system provides a promising solution for meeting the stringent security and performance requirements of the healthcare industry in practice.

References

[1] S. Kuranda, "The 10 biggest data breaches of 2015 (so far)," 2015. [Online]. Available: http://www.crn.com/slide-shows/security/300077563/the-10-biggest-data-breaches-of-2015-so-far.htm

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical technizheng ues for searches on encrypted data," in *Proceedings of S&P'00*. IEEE, 2000.

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of CCS'06*. ACM, 2006, pp. 79–88.

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYPT'04*, 2004.

[5] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proceedings of ICDCS'10*, 2010.

[6] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proceedings of CRYPTO'13*. Springer, 2013.

[7] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of CCS'12*. ACM, 2012.

[8] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proceedings of FC'13*. Springer, 2013.

[9] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proceedings of S&P'14*. IEEE, 2014.

[10] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proceedings of NDSS'14*, 2014.

[11] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proceedings of CCS'14*. ACM, 2014.

[12] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage." in *Proceedings of NDSS'14*, 2014.

[13] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Proceedings of CANS'13*. Springer, 2013, pp. 309–328.

[14] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[15] J. Katz and A. Y. Lindell, "Aggregate message authentication codes," in *Proceedings of CT-RSA'08*. Springer, 2008, pp. 155–169.

[16] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[17] C. C. Tan, H. Wang, S. Zhong, and Q. Li, "Ibe-lite: a lightweight identity-based cryptography for body sensor networks," *IEEE Trans. Inf Technol Biomed*, vol. 13, no. 6, pp. 926–932, 2009.

[18] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted personal health records in cloud computing," in *Proceedings of ICDCS'11*. IEEE, 2011, pp. 383–392.

[19] Y. Tong, J. Sun, S. S. Chow, and P. Li, "Cloud-assisted mobile-access of health data with privacy and auditability," *IEEE J. Biomed Health Inform*, vol. 18, no. 2, pp. 419–429, 2014.

[20] L. Yang, A. Humayed, and F. Li, "A multi-cloud based privacy-preserving data publishing scheme for the internet of things," in *Proceedings of ACSAC' 2016*. ACM, 2016.

[21] Q. Zheng, S. Xu, and G. Ateniese, "Vabks: verifiable attribute-based keyword search over outsourced encrypted data," in *INFOCOM'14*.

[22] R. Cheng, J. Yan, C. Guan, F. Zhang, and K. Ren, "Verifiable searchable symmetric encryption from indistinguishability obfuscation," in *Proceedings of CCS'15*. ACM, 2015, pp. 621–626.

[23] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proceedings of INFOCOM'15*. IEEE, 2015.