Privacy Preserving Visual Log Service with Temporal Interval Query using Interval Tree-based Searchable Symmetric Encryption

Viet-An Pham, Dinh-Hieu Hoang, Huy-Hoang Chung-Nguyen, Mai-Khiem Tran and Minh-Triet Tran
Faculty of Information Technology - Software Engineering Lab - University of Science, VNU-HCM
E-mail: pvan@apcs.vn, hdhieu@apcs.vn, cnhhoang@apcs.vn, tmkhiem@selab.hcmus.edu.vn, tmtriet@fit.hcmus.edu.vn

Keywords: searchable symmetric encryption, temporal interval query, visual concept extraction

Received: March 15, 2020

Visual logs become widely available via personal cameras, visual sensors in smart environments, or surveillance systems. Storing such data in public services is a common convenient solution, but it is essential to devise a mechanism to encrypt such data to protect sensitive information while enabling the capability to query visual content even in encrypted format at the services. More precisely, we need smart systems that their security and practicality must be balanced against each other. As far as we know, in spite of their importance in preserving personal privacy, such reliable systems have not gained sufficient attention from researchers. This motivates our proposal to develop a smart secure service for visual logs with a temporal interval query. In our system, visual log data are analyzed to generate high-level contents, including entities, scenes, and activities happening in visual data. Then our system supports data owners to query these high-level contents from their visual logs at the server-side in a temporal interval while the data are still encrypted. Our searchable symmetric encryption scheme TIQSSE utilizes interval tree structure and we prove that our scheme achieves efficient search and update time while also maintaining all important security properties such as forward privacy, backward privacy, and it does not leak information outside the desired temporal range.

Povzetek: Problem uravnoteženja proizvodne poti je predstavljen odprto, brez omejitev npr. števila delavcev, zato je izviren. Avtorji testirajo več algoritmov in predlagajo najboljšega.

1 Introduction

In daily activities, people usually take photos and record video clips to capture moments and events in their lives. Besides, with the booming trend of developing smart interactive environments, such as smart homes, offices, or even cities, visual sensors are densely integrated to our habitats to record then analyse external contexts, such as monitoring users, objects, activities, etc. Consequently, visual lifelogs become increasingly available and are usually uploaded to store in online storage services.

In this paper, we target two challenging problems to better develop an online storage service for private visual data: (i) to search photos or video clips based on their content, and (ii) to protect private data leakage at server-side accidentally or intentionally.

First, we aim to bridge the gap between visual data and their semantics by allowing data owners to search with keywords. Each photo or frame in a video clip is processed to extract high-level concepts, including entities, scene attributes, activities, etc. Different types of high-level concept extractors can be plugged into our framework to meet specific requirements in real applications. Consequently, a photo or video frame can be considered as a document or a set of concepts, which are ready to be retrieved by keywords. We also demonstrate a prototype smart edge camera which can be re-configured remotely to generate visual data with associated extracted concepts.

Second, a typical solution to protect data secrecy is to encrypt before uploading data to an online storage server. However, after encryption, data are no longer suitable to be searched normally. Symmetric Searchable Encryption (SSE), first proposed by Song et al. [23], can be used as a promising solution to privately save data while maintaining the ability to search in a collection of encrypted records. We adopt the approach of SSE in our proposed solution, and carefully design it to ensure the property of a dynamic SSE [13], i.e. to add, update, and delete data efficiently without re-encrypting the whole database.

Besides, we also consider the forward and backward privacy criteria for SSE. Informally, the former means that an update query does not leak information if a newly added document contains keywords that were searched in the past, while the latter is to make sure that it is impossible to retrieve data from deleted files. Forward privacy has been receiving a lot of attention, while backward privacy is only studied in recent years. Most of the existing schemes suffer from key-size overgrowing after deletion queries [2, 4], thus limits the practicality of these schemes.

Moreover, in particular cases, there are new security properties that must be satisfied: search only in a temporal interval, and do not leak any information outside of the requested range. For example, a police wants to check the private security camera of a company from a range of time for a criminal event. The company wants to provide the information exactly from the requested range and not leak any information from other temporal intervals. A similar problem is when we want to search for some disease in a medical database in a temporal interval, it is best to prevent leaking information of patients in other time. This motivates us to define Temporal Interval Query SSE (TIQSSE), a new SSE problem to search by keywords for documents in a particular temporal interval.

This work is the extension paper of previous TIQSSE work [20], with more in-depth explanations and analysis. This paper is also a significantly enhanced version of [7]. Our previous work only guarantees a one-sided access pattern. For more clarity, the one-sided access pattern means that it can only preclude adversaries from extracting information about the documents that were added after the queried interval, while still leaking information of documents that were added before the requested range. In this paper, there is a great improvement on security since our SSE scheme now guarantees two-sided access pattern, which means it also prevents adversaries from gaining information of added documents.

Our newly defined problem is different from the existing range query SSE schemes [1, 14]. In a range query SSE scheme, a server returns every document whose key/identifier is in a queried range. In our temporal interval SSE problem, the server only examines documents whose identifiers are inside the temporal interval to select the documents containing a query keyword w.

Our secure SSE scheme does not suffer from key-size overgrowing after sufficient deleting queries like previous schemes. Our idea is based on $\Sigma o\varphi o\varsigma$ from Raphael Bost et al. [2] in 2016 and modifies it to match our problem. Although there are many improved constructions later [4, 24], these ideas are not suitable for our problems that the use cases we target require efficient deletion operations which (1) have an acceptable time complexity and (2) do not increase server-side usage.

Our main contributions in this paper are as follows.

- We propose a solution for a public visual data storage service to assist data owners to search their photos and video clips with keywords, i.e. concepts extracted from visual content, and preserve data privacy in query and data manipulation (insert, update, delete). We also develop a prototype smart edge camera to handle concept extraction for recorded photos or video clips.

- We also define TIQSSE as a new SSE problem to search with encrypted documents in a temporal interval while preventing data leakage outside the requested range. We then propose an efficient solution to search for a keyword in documents within a determined time range and achieves both forward and backward privacy.

In Section 2, we briefly review approaches and methods related to the two main aspects of our work, visual retrieval with concepts, and searchable symmetric encryption. We propose a smart secure framework for visual data storage service and smart edge camera in Section 3. in Section 4, We review the necessary preliminaries of cryptography, then define the novel TIQSSE problem. Our scheme which tackles this problem is introduced in Section 5. The security analysis of our proposed scheme is presented in Section 6. In Section 7, we draw our conclusion and discuss some fascinating directions for future works.

2 Related work

2.1 Visual retrieval with semantic concepts

Visual log retrieval is one of the important problems to analyse and understand visual content. Different approaches have been proposed to provide users with various modalities to input queries and get retrieved results [17, 18, 26]. Visual semantic concepts from images are usually used as tags or keywords for interactive retrieval systems[26, 25]. The concepts can be detected using available APIs, such as Google Cloud Vision API, or pre-trained object detectors, such as Yolo [21], FasterRCNN [22], etc. Besides, scene attributes and categories [30] can be extracted from images to augment further environmental information of visual data[25]. Some works also utilize captioning [27] or activity recognition to capture the dynamic nature of an image or video clip[16, 15].

In this work, we propose to integrate different concept extractors to create the associated metadata for each photo or video clip stored in the smart visual service. We also develop a prototype smart edge camera that can locally extract concepts in certain tasks before uploading visual data to online storage service (see Section 2.1).

2.2 Searchable symmetric encryption

Song et al. [23] first proposed a solution to Searchable Symmetric Encryption in 2000. Although the first SSE scheme was not efficient, it provided a solid foundation for the problem. Many works were proposed [10, 6] to improve search time and security. However, leakage problems in SSE were not formally defined. Curtmola et al. [8] were the first to explicitly define the general acceptable leakage criteria for SSE problems, including search patterns and access patterns that are frequently used several years later.

Although the previous schemes were optimal in search time, there was no way to update a database without re-encrypting the whole database. To remove this limitation, in 2012, dynamic SSE was proposed by Kamara et al. [13]. Their scheme can efficiently add or remove files with the trade-off by leaking some information when those queries are executed. In particular, forward privacy and backward privacy are not fully satisfied.

SSE problem is continuously studied and improved. Raphael Bost achieved forward privacy in 2016 [2], and also achieved backward privacy one year later [4]. In 2018, Sun et al. [24] proposed Puncturable Symmetric Encryption to construct and improve backward secure. Unfortunately, all schemes mentioned above not only suffer from key-size overgrowing after many deleting queries, but also do not support range query property that we need.

Other than proposing new SSE constructions, many efforts were made to attack the proposed security models. Some notable works are inference attacks on deterministic encryption (DTE) and order-preserving encryption (OPE) [19, 11], leakage-abuse attacks [5, 3, 11, 12] and File-Injection attacks [29, 12].

Before us, there are many works about range queries. However, they all are different from ours. Their solution is used for indexing in relational databases and return entities that have acquired attributes within some range, while in our scheme, we need to return all the files containing the searched keyword in a period.

3 Smart secure framework for visual data storage service

In this section, we present our proposal for a smart secure framework for a visual data storage service. We are inspired by the idea of edge computing to shift the concept extraction task toward the smart camera. There has been an ongoing interest on this shift, particularly from privacy-aware users due to recent breaches in data centers, where sensitive user data is processed and may be used for malicious purposes. If the process is on users' premise, they will have more control over the data that is generated.

3.1 Smart edge camera with concept extraction

Figure 1 illustrates the process for concept extraction from photos/clips in a smart edge camera before uploading visual data with their associated metadata to the secure visual service. Different modules for various concept types can be deployed in the smart edge camera, such as object detection, person recognition, action recognition, scene attribute, category classification, and image captioning.

In our prototype, we utilize NVIDIA Jetson Nano embedded computers with dedicated 128 Maxwell CUDA cores to handle various machine learning tasks. Our smart edge camera prototype can be specialized for various specific tasks with different models to be deployed and updated (see Figure 2). In our model repository, not only there are existing pre-trained models, such as ResNet-50, MobileNet-v2, SSD ResNet-18, SSD Mobilenet-V2, Tiny YOLO V3, but we also prepare our own custom models for other tasks, such as custom object detectors for contexts

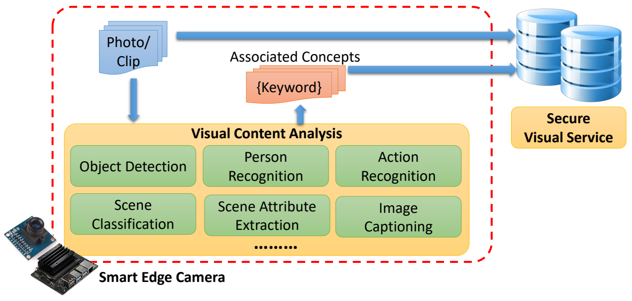

Figure 1: Concept extraction from photos/clips in a smart edge camera.

originated from Vietnam or image captioning with concept augmentation [27].

Future custom models can also be created and further optimized with various techniques such as quantization, fusion, and scheduling available in NVIDIA TensorRT SDK, then deployed to the smart camera. Due to its cloud nature, the devices' software can be remotely updated, and additional machine learning models can be added in a secure manner.

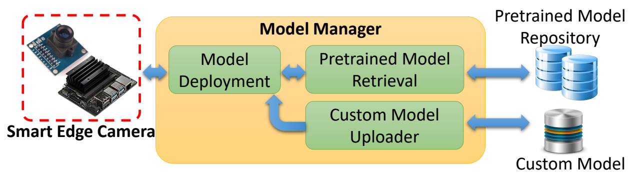

Figure 2: Model update for an edge camera.

3.2 Components in a smart secure visual system

We propose a scenario in which a system collects, processes, and synchronizes the data from various cameras, including the proposed smart edge ones, to a visual data server that utilizes our proposed secure scheme for SSE. Figure 3 illustrates the three key components of the system: a storage and query processing server, camera nodes, and query nodes.

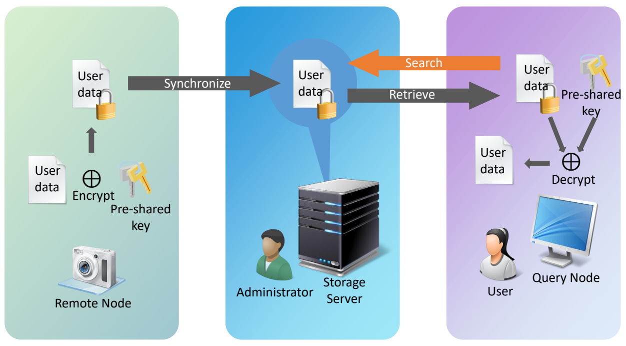

Figure 3: Main components in smart secure visual system.

In our system, the storage and query processing server supports multiple users, and the server owner can be different from the data owners. The owners of the server can fully examine the stored data, but are expected not to understand or to exploit useful information from stored data. Thus, to ensure this crucial property of our visual system, i.e. preserving data privacy for data owners, we define a new problem of Temporal Interval Query SSE (in Section 4) and propose an efficient solution for this problem (in Section 5).

A user, after signing up, is provided a means to submit and retrieve data over commonly utilized protocols, such as HTTP SSL, SMB, or SFTP. Querying is done over an API with a common contract protocol implemented in gRPC, a protocol buffer library that utilizes HTTP2 over an SSL Channel. With gRPC's wide adoption status across numerous languages and libraries, the implementation is relatively easy and open for everyone. Connections to the server are secured with the server's certificate by default. We assume this certificate is self-signed and pre-installed on every query node via personal trusted channels beforehand. A user usually plays both roles as a generator party at upload time from a camera node and a querying party at retrieve time from a query node, which can be his or her mobile device. Thanks to the loosely coupled architecture, our proposed system allows new users to dynamically join in without any interruptions on the server-side using a streamlined user interface.

4 Temporal interval query searchable symmetric encryption

In this section, we first provide background knowledge that includes several cryptographic primitives and the dynamic SSE problem. Then we introduce the definition and security properties for TIQSSE.

4.1 Preliminaries

For consistency in presentation, we denotes:

- $x \overset{\$}{\leftarrow} \{0,1\}^n$ as randomizing n bits then store the result to x.
- n as the number of added files. $\mathbf{F_n}$ as the n-th file. $\mathbf{EF_n}$ as the encrypted file corresponding to $\mathbf{F_n}$.
- $\perp$ as *null* or *empty*. λ as the security parameter. Security parameter means that unless specified explicitly, the keys used in SSE scheme is λ-bit in length, and the probability for an adversary to break the scheme is $2^{-\lambda}$.

We use several cryptographic primitives from Dan Boneh and Victor Shoup [9] which includes: negligible function, pseudo random generator (PRG), pseudo random function (PRF), simulator, and symmetric encryption. For the symmetric encryption, we denote the encryption of plaintext m with secret key sk as $\mathrm{SE.enc}(sk, m)$, and the decryption of ciphertext c with secret key sk as $\mathrm{SE.dec}(sk, c)$.

We also inherit the idea of trapdoor permutation from Bost et al. [2] and denote the function as π. Formally: One can compute π of p_1 with the secret key K_s: $p_2 \leftarrow \pi(K_s, p_1)$. Given p_2 in π's proper, one can derive the original p_1 with the public key: $p_1 \leftarrow \pi^{-1}(K_p, p_2)$. Finally, for all p we have $p = \pi(K_s, \pi^{-1}(K_p, p)) = \pi^{-1}(K_p, \pi(K_s, p))$.

4.2 Dynamic symmetric searchable encryption

In SSE, we view the database as an array of files $\mathbf{F} = (\mathbf{f_1}, \mathbf{f_2}, ..., \mathbf{f_n})$ where $\mathbf{f_i}$ consists of multiple words $(w_1, w_2, ..., w_{m_i})$. Later when the client request a search on keyword w, the client obfuscate or encrypt w into trapdoor $\mathbf{T}$ and give it to server. The server when receiving $\mathbf{T}$ must return a list of result identifiers $\mathbf{R} = (id_1, id_2, ..., id_r)$ such that when returned to the client, for every i we have $w \in \mathbf{F_{id_i}}$. It is notable that the act of obfuscating w into $\mathbf{T}$ is essential because it hides the original keyword from the server, in this paper we call this as trapdoor generation procedure.

In other words, dynamic SSE consists of one algorithm $\mathbf{Setup}$ and two protocols $\mathbf{Search}$ and $\mathbf{Update}$.

- In $\mathbf{Setup}$ phase, the client creates some keys and keypairs that will be used in the other 2 protocols.
- The $\mathbf{Search}$ protocol consists of multiple interactions between client and server when the client request a search. For each client's request, the server should receive the trapdoor $\mathbf{T}$ and return a list of files as we mentioned in the above paragraph.
- The $\mathbf{Update}$ protocol is comprised of 2 types of updates which is add a new file and delete an existed file. Depending on which update protocol, the encrypted database on the untrusted server will be modified based on the SSE scheme.

$\mathbf{Correctness}$. An SSE is *correct* if the probability that the search protocol returns the false results to client is negligible.

$\mathbf{Security}$. The SSE scheme Σ is said to be adaptively secure, if for any adversary $\mathcal{A}$ who issues a polynomial number of queries q(λ), there exist a polynomial-time simulator $\mathcal{S}$ such that:

$$|P[\mathrm{SSEReal}^{\Sigma}_{\mathcal{A}}(\lambda, q) = 1] - P[\mathrm{SSEIdeal}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, q) = 1]| < negl(\lambda)$$

Informally, the simulator $\mathcal{S}$ can be thought of as an efficient probabilistic algorithm such that its output distribution is identical to the real scheme's output distribution. Then, the theorem above can be semantically understood as: if we can prove there exists a simulator $\mathcal{S}$ of SSE scheme Σ, then it is very hard for the adversary to distinguish between the real case with a simulation case. Hence, we achieve adaptive security for SSE.

4.3 Definition of temporal interval query SSE

Temporal Interval Query SSE continues to use the model of the original dynamic Symmetric Searchable Encryption but modifies the **Search** protocol. When the client issues a search request, firstly he chooses a range of interest $[L; R]$, then he chooses a keyword w he wants to search on, then he generates the trapdoor vector $\mathbf{T}$ that represents the keyword w for that range $[L; R]$, finally he gives $(\mathbf{T}, L, R)$ to server. The server when receiving $(\mathbf{T}, L, R)$ must return a list of result identifiers $\mathbf{R_{w,L,R}} = (id_1, id_2, ..., id_r)$ such that when returned to the client, for every i we have $w \in F_{id_i}$ and $L \leq id_i \leq R$ (see Figure 4).

4.4 Security

Forward privacy: Informally, a SSE scheme achieves forward privacy if its Update query does not leak any information about the newly added file even if it contains keywords that are previously searched keywords. For example, the client searched for a keyword w. Later, when the client add a file $\mathbf{F}$ that contains w, the server should not know that w exists inside $\mathbf{F}$. Many researches [5] showed that if a scheme does not attain forward privacy, the client's queries, or even the plaintext, can be revealed even with small leakage. There also exist attacks [29] that can effectively exploit the vulnerability of those schemes to break query privacy. In addition, forward privacy can also improves time and space performances [2].

Backward privacy: To have backward privacy in dynamic SSE, we must prevent the adversary from gaining knowledge of deleted files from new queries. For example, if there exists a deleted file $\mathbf{F}$ that contains a word w and has never been queried, in the future when client search for word w, it is expected to prevent the server from knowing $w \in \mathbf{F}$.

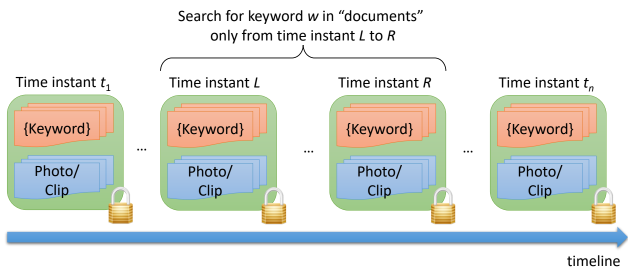

Figure 4: An example of temporal interval query.

In order to have searchable property over encrypted data, there must be some leaking information throughout the process. We follow many previous works [2, 4, 24] and call this as leakage function $\mathcal{L} = (\mathcal{L}^{\text{Stp}}, \mathcal{L}^{\text{Srch}}, \mathcal{L}^{\text{Updt}})$. The leakage function $\mathcal{L}$ is used to express the information learned by the untrusted server from 3 protocols **Setup**, **Search**, **Update**.

Setup leakage: In the setup algorithm, the client generates some keys and keypairs for later usage in **Search** and **Update** protocol. Because of that, the leakage of setup phase is the public keys (if there is any) that the client wants to share with the server $\mathcal{L}^{\text{Stp}} = \mathbf{PK}$.

Search leakage: Firstly, let $\mathbf{Q}$ as the search requests of the client where $\mathbf{Q_i} = (\mathbf{T_i}, L_i, R_i)$; $\mathbf{R}$ as the results returned by the untrusted server; $\mathbf{R_i}$ as the result of $\mathbf{Q_i}$ where its content is $(id_{i,1}, id_{i,2}, ..., id_{i,r_i})$; $\mathbf{H}$ as the history of previous searches from the client that $\mathbf{H} = (\mathbf{Q}, \mathbf{R})$. We define the allowed leakage of search protocol is comprised of search pattern $\sigma(\mathbf{H})$ and access pattern $\alpha(\mathbf{H})$.

The access pattern $\alpha(\mathbf{H})$ indicates the leakage of the returned values of the queries. That is for each query we want to only leak the existence of keyword w within the existed files within the interval $[L; R]$ and non elsewhere.

The search pattern $\sigma(\mathbf{H})$ represents the leakage of the query parameters from the client. The search pattern consists of 2 levels of security:

- Perfect security: when analyzing 2 different queries i and j with common keyword w, it is very hard for the server to deduce $\mathbf{T_i}$ and $\mathbf{T_j}$ to be the same keyword w. Because of that in this setup, the client perfectly hide the search pattern and can secure against many inference attack types [19, 11].

- Weak security: when analyzing 2 different queries i and j with common keyword w, the server can easily deduce $\mathbf{Q_i}$ and $\mathbf{Q_j}$ has the same search keyword w if and only if the queried range $[L_i; R_i]$ intersects with $[L_j; R_j]$ at some point.

Update leakage: The update leakage consists of the leakage of add new file protocol and delete file protocol. The add new file protocol leaks n as the number of added files and the size of all the files. The delete file protocol leaks the identifier of deleted files.

5 Proposed scheme for TIQSSE

In this section, the parts are arranged as the followings. Firstly, we outline our scheme at the first part, the remaining parts describe how our scheme works in setup protocol, add new file protocol, search protocol and delete file protocol.

5.1 Scheme outline

There are 7 polynomial-time algorithms in total. The first 6 algorithms are executed by the client, while the last algorithm is done by the server.

- $(sk, \mathbf{K}) \leftarrow \text{KeyGen}(1^\lambda)$: is a probabilistic algorithm that uses the security parameter λ to setup the secret key sk for encryption/decryption and a vector of key-pairs $\mathbf{K}$ for trapdoor generation procedure.
- $\mathbf{EF_n} \leftarrow \text{Enc}(sk, \mathbf{F_n})$: is a probabilistic algorithm that encrypt $\mathbf{F_n}$ into $\mathbf{EF_n}$.
- $\mathbf{F_n} \leftarrow \text{Dec}(sk, \mathbf{EF_n})$: is the reverse algorithm of Enc.
- $t_n \leftarrow \text{Trpdr}(sk, \mathbf{K}, n, w)$: is a deterministic algorithm that illustrates the process of generating trapdoor value of keyword w in file $\mathbf{F_n}$ into trapdoor value t_n.
- $\mathbf{I_n} \leftarrow \text{CreateIndex}(sk, n, \mathbf{F_n})$: is a deterministic algorithm that illustrates the process of creating an index file $\mathbf{I_n}$. This index file acts as a look up table for the untrusted server to search on in the search protocol.
- $\mathbf{T} \leftarrow \text{SearchToken}(sk, \mathbf{K}, w, L, R)$: is a deterministic algorithm that illustrates the process when the client prepare the search request. Using the keys and (w, L, R) it outputs trapdoor $\mathbf{T}$ and give it to the server.
- $\mathbf{R_{w,L,R}} \leftarrow \text{Search}(\mathbf{T}, \mathbf{I_{L,\dots,R}}, L, R)$: be a deterministic algorithm that illustrates how the untrusted server uses the trapdoors $\mathbf{T}$ and the range of interest $[L, R]$ to return the appropriate files back to the client.

5.2 Setup protocol

The client runs KeyGen algorithm: $(sk, \mathbf{K}) \leftarrow \text{KeyGen}(1^\lambda)$ to generate sk and $\mathbf{K}$. For sk, the client can randomize λ bits and store it in sk as: $sk \xleftarrow{\$} \{0,1\}^\lambda$. For $\mathbf{K}$ which consists of 2 trapdoor permutation keypairs $(\mathbf{K_{ul}}, \mathbf{K_{ur}})$, the client can generate these keypairs by generating trapdoor permutation keypair on λ bits.

A side note here is the client must assure that the id of each files given to the server are incremental. Thus, id starts from -1 and n starts from 0 when no file has been added.

5.3 Add new file protocol

In the following algorithms, let n be the number of added files, $\mathbf{F_n}$ be the new file that client wants to add, $\mathbf{EF_n}$ be the encrypted file of $\mathbf{F_n}$, $\mathbf{I_n}$ be the encrypted index of $\mathbf{F_n}$.

The main idea for the add new file protocol is for each file $\mathbf{F_i}$, the client encrypts $\mathbf{F_i}$ into $\mathbf{EF_i}$ and generates the encrypted index $\mathbf{I_i}$. Finally, the client gives $\mathbf{EF_i}$ and $\mathbf{I_i}$ to the server. The usage of encrypted index $\mathbf{I_i}$ is for the server to indicate whether word w is contained inside the corresponding encrypted file $\mathbf{EF_i}$ or not.

5.3.1 Encrypt file

With the above idea, because the searching step at the server-side only requires the encrypted index, it is trivial to encrypt the file $\mathbf{F_n}$ using popular symmetric encryption algorithm like AES. Plus, the decryption algorithm is the reverse function of encryption. Let SE be the symmetric encryption algorithm:

- Enc : $\mathbf{EF_n} \leftarrow \text{SE.enc}(sk, \mathbf{F_n})$
- Dec : $\mathbf{F_n} \leftarrow \text{SE.dec}(sk, \mathbf{EF_n})$

5.3.2 Create encrypted index

In this section we use a data structure called $\mathbf{map}$ for the encrypted index. Firstly, we initialize $\mathbf{I_n}$ as an empty map. Then for each word w in $\mathbf{F_n}$ we create the trapdoor and store it inside $\mathbf{I_n}$ as following procedure:

1. $t_n \leftarrow \text{Trpdr}(sk, \mathbf{K}, n, w)$.
2. Check whether t_n already exists in $\mathbf{I_n}$ as: $t_n \in \mathbf{I_n}$, if yes, then repeatedly randomizing t_n as: $t_n \xleftarrow{\$} \{0,1\}^\lambda$ until $t_n \notin \mathbf{I_n}$. After this step, we call t_n *garbage data* if $t_n \neq \text{Trpdr}(sk, \mathbf{K}, n, w)$.
3. Store t_n into $\mathbf{I_n}$.

With the algorithm above, there will not have any word duplication within an index file **I** because whenever a word already exists inside, it will be substituted as a *garbage word* by randomization. Furthermore, when provided a trapdoor t, the server can easily check whether t exists inside an encrypted index **I**, which enables searchability.

5.3.3 Trapdoor generation

The interval tree model is mainly for this step and is quite complicated. Briefly, Interval tree (or Segment tree) is a binary tree where each node contains information about a specific range $[L, R]$. Every none-leaf node also has two child node to manages $[L, M]$ and $[M+1, R]$ where $M = \lfloor \frac{L+R}{2} \rfloor$. The complexity of search and update query is $\mathcal{O}(\log_2 n)$.

Firstly, we visualize how interval tree trapdoor generation works, then we give an example of it in practice. We also provide graphical Figure 5 that is easier for readers to visualize the appearance of interval tree in our model.

- We view the files as nodes at level 0 and is arranged from left to right with incremental id starting from 0 to $n-1$.

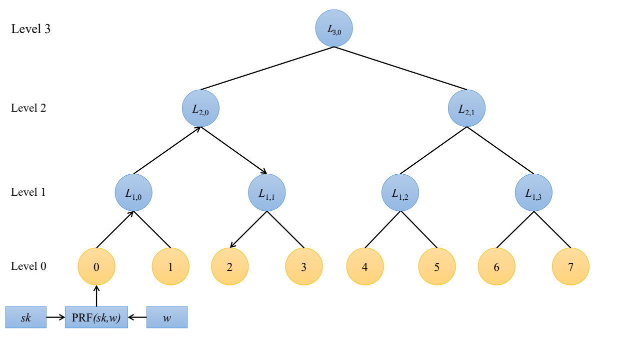

Figure 5: Interval Tree for Trapdoor generation visualization and trapdoor transformation in practice.

- The nodes at higher levels are treated as interval nodes, that is it will cover an interval of continuous files. For example in Figure 5, node $L_{2,1}$ covers files from 4 to 7 and node $L_{1,1}$ covers files 2 and 3.
- From any word w, client can generate trapdoor for file 0 by: $t_0 \leftarrow \mathrm{PRF}(sk, w)$. And from any node, the client can "move" the trapdoor at that node onto the **Up-Right** node by using trapdoor permutation on secret component of $\mathbf{K_{ur}}$ as: $t_{ur} \leftarrow \pi(K_{ur_s}, t)$. And from any node, we can "move" trapdoor to the **Down-Left** node by applying reverse trapdoor permutation using public component of $\mathbf{K_{ur}}$ as: $t_{dl} \leftarrow \pi^{-1}(K_{ur_p}, t)$. In case of moving **Up-Left** and **Down-Right**, we just need to apply π and π^{-1} on $\mathbf{K_{ul}}$ like above steps.

The important point here is the public key of $\mathbf{K_{ur}}$ and $\mathbf{K_{ul}}$ is available for both client and server but only the client holds the secret component of $\mathbf{K_{ur}}$ and $\mathbf{K_{ul}}$. Hence, for any node $L_{i,j}$, the server can only move the trapdoor value to nodes in sub-tree of $L_{i,j}$. However, the client can move anywhere he wants because he holds the secret key.

We denote $t_{i,j}$ to be the trapdoor value at node $L_{i,j}$; t_i and $t_{0,i}$ to be the trapdoor value at F_i. Below we show an example of generating t_2 from keyword w. Figure 5 also demonstrates the process.

1. $t_{0,0} \leftarrow \mathrm{PRF}(sk, w)$
2. $t_{1,0} \leftarrow \pi(K_{ur_s}, t_{0,0})$
3. $t_{2,0} \leftarrow \pi(K_{ur_s}, t_{1,0})$
4. $t_{1,1} \leftarrow \pi^{-1}(K_{ul_p}, t_{2,0})$
5. $t_{0,2} \leftarrow \pi^{-1}(K_{ur_p}, t_{1,1})$

5.4 Search protocol

To search for the existence of keyword w within range $[L; R]$, the client runs SearchToken algorithm to create trapdoor vector $\mathbf{T}$ then give it to the server. The server will use the range $[L; R]$ and $\mathbf{T}$ to run Search algorithm and return the appropriate encrypted files to the client.

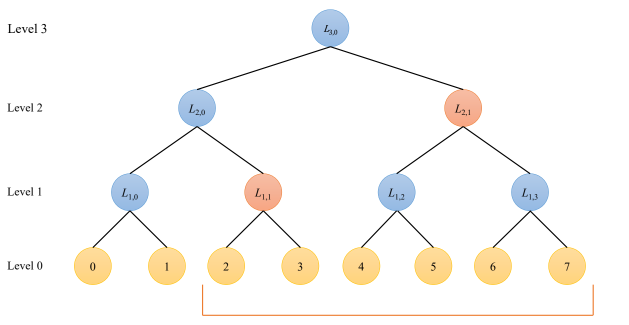

Figure 6: Client issue SearchToken from file 2 to file 7, sending $t_{1,1}$ and $t_{2,1}$ to query the server.

5.4.1 SearchToken algorithm

Previously we mentioned that given a trapdoor $t_{i,j}$ at some node $L_{i,j}$, the server can easily compute all trapdoor values at nodes inside sub-tree of $L_{i,j}$. With this special characteristic, the client only needs to compute trapdoor value at nodes such that it covers only in range $[L; R]$. To optimize the computational complexity, the client needs to find as a minimal number of interval nodes as possible.

The algorithm is simple. Iterates from L to R, let $L_{0,i}$ be our current node, while there exists an up-right parent node and the parent node still covers within the range $[L; R]$, traverse to the parent node and repeat the process. After finding the appropriate parent of $L_{0,i}$, let k be the level of that parent node, we can skip the next 2^k nodes and repeat the process until we cover all nodes from $[L; R]$.

In Figure 6, we demonstrate SearchToken when issuing query from file 2 to file 7, the red nodes $L_{1,1}$ and $L_{2,1}$ is sufficient to cover the range $[2, 7]$. After finding the interval nodes that cover $[L; R]$, the client can use trapdoor generation procedure mentioned earlier to calculate $\mathbf{T} = (t_{1,1}, t_{2,1})$ and send it to server.

5.4.2 Search algorithm

For each value $t_{i,j}$ in the received trapdoor vector $\mathbf{T}$, the server can use π^{-1} with the public key of $\mathbf{K_{ur}}$ and $\mathbf{K_{ul}}$ to traverse down to any nodes in sub-tree of $t_{i,j}$. After traversing down to the level 0 nodes, the server can easily check whether trapdoor $t_{0,i}$ exists inside the encrypted index $\mathbf{I_i}$. Finally, the server returns all the encrypted file $\mathbf{EF_i}$ that satisfies above conditions.

5.5 Delete protocol

To delete a file, the client sends a single variable k to the server to indicates that he wants to delete $\mathbf{EF_k}$ and $\mathbf{I_k}$. After that, the server deletes $\mathbf{EF_k}$ and $\mathbf{I_k}$ in its database/hard drive. Later when Search algorithm occurs, without the data of $\mathbf{I_k}$, the server cannot check trapdoor t exists in $\mathbf{I_k}$ or not because I_k has already been deleted. However, only deleting $\mathbf{EF_k}$ and $\mathbf{I_k}$ is not efficient because the server must iterate through every file at level 0, even the deleted ones, which would result in $\mathcal{O}(R-L)$.

We can optimize the runtime by storing an additional boolean isDeleted in each node $L_{i,j}$. The isDeleted boolean indicates whether the entire sub-tree of $L_{i,j}$ has been deleted or not. Then we fix the deletion algorithm as following: when deleting the k-th file, mark $L_{0,k}$.isDeleted as True. Then iterate to the parent of $L_{0,k}$, if the 2 children of that current parent are also deleted, then mark that parent as deleted and continues to move onto its parent and repeat the process.

With the above optimization, the Search algorithm by the server will be modified a little bit. At server-side while traversing to the nodes of sub-tree of $L_{i,j}$, if server encounters a deleted node, the server can ignore all the nodes in that sub-tree, which can optimize the computation.

6 Scheme analysis

6.1 Correctness

We will prove the *correctness* of our scheme based on the *correctness* that we introduced in section 4.2. Our proof has two parts:

The search result contains all documents having the searched keyword w. Obviously, for each document having the searched keyword w, the tree associated with w must mark it containing this keyword. Thus, on executing the searching protocol, server will see that the document is included in the tree, which means that the document will be listed in the result.

For any document which does not have the searched keyword w, the probability that it is listed in the search result is extremely small. We consider some arbitrary document. Let m be the size of the output of trapdoor function which is used to encode the words, $size$ the size of the document, $amtw$ the amount of valid distinct keywords of that document, and $dictsize$ the amount of valid distinct keywords of the whole dataset.

Obviously, the probability that there does not exist any garbage data that collides with a valid keyword is:

$$p = \binom{2^m - dictsize - amtw}{size - amtw} \bigg/ \binom{2^m - amtw}{size - amtw}$$

which is reduced as,

$$p = \prod_{i=1}^{size-amtw} \frac{2^m - dictsize - size + amtw + i}{2^m - size + i}$$

By using a suitable trapdoor permutation whose output size m is big enough, we make the probability that the server falsely determines a searched keyword w exists in a document, which equals $(1-p)$, very small. The bigger m is, the more precise the returned results are.

6.2 Formal adaptive security proof

We formally describe adaptive security proof of our scheme based on section 4.2. We retrieve the following games from the TIQSSE scheme:

Game G_0. The first game G_0 is completely identical to the real world game $\mathrm{SSEReal}_{\mathcal{A}}^{\Sigma}(\lambda, q)$. Thus,

$$P[SSEReal_{\mathcal{A}}^{\Sigma}(\lambda, q) = 1] = P[G_0 = 1]$$

Game G_1. In this game, we replace the function PRF by a truly random key generator. More precisely, G_1 will get a random element in the domain of PRF whenever it comes to a new word w, and stores this element in a table Key containing all key associated with each queried word w. So in order to exist some adversary who can distinguish between G_0 and G_1, he must break the security of PRF. Therefore we have:

$$P(G_0 = 1) - P(G_1 = 1) \leq \mathrm{Avd}_{\mathcal{B}_1}^{PRF}(\lambda)$$

Game G_2. G_2 does not use trapdoor permutation anymore. Instead, it uses random oracles for π and programs π^{-1} such that $\pi_{key}(\pi_{key}^{-1})(i) = i$ for any arbitrary key and i. Obviously, the problem of distinguishing between G_1 and G_2 can be reduced to the problem of cracking the onewayness of π. Since our scheme uses two pair of public-private keys, there exists an efficient adversary $\mathcal{B}_2$ such that:

$$P(G_1 = 1) - P(G_2 = 1) \leq \mathrm{Avd}_{\mathcal{B}_2}^{OW}(\lambda)$$

The simulator $\mathcal{S}$. We construct our simulator $\mathcal{S}$ identical to game G_2 which changes PRF and Trapdoor Permutation as random oracles:

$$P(G_2 = 1) = P(SSEIdeal_{\mathcal{A},\mathcal{S},\mathcal{L}_\Sigma}^{\Sigma} = 1)$$

Combining all above results, we conclude that we can simulate the original scheme and achieve adaptive security mentioned in Section 4.2:

$$\begin{aligned} P(SSEReal_{\mathcal{A}}^{\Sigma} = 1) - P(SSEIdeal_{\mathcal{A},\mathcal{S},\mathcal{L}_\Sigma}^{\Sigma} = 1) \\ \leq \mathrm{Avd}_{\mathcal{B}_1}^{PRF}(\lambda) + \mathrm{Avd}_{\mathcal{B}_2}^{OW}(\lambda) \end{aligned}$$

6.3 Informal adaptive security proof

To make it more understandable, we also provide an informal proof for the adaptive security of SSE. From our scheme we can derive several consequences:

Truly random encrypted file. The encrypted file **EF** is obtained by encrypting the plain file with a secure symmetric encryption algorithm. So to break the randomness of **EF** the adversary must break the symmetric encryption algorithm.

Truly random encrypted index. Let us recall how we create an encrypted index. Firstly from keyword w we apply PRF to generate t_0. So if an adversary able to break the randomness of t_0, he must break PRF. After that, we apply several trapdoor permutation π and π^{-1} to generate other t_i values. Again, to break the randomness of t_i, the adversary must find a way to crack trapdoor permutation. Hence, we claim that we generate truly random encrypted indexes.

Conclusion. From the 2 above proofs, we say that our scheme can use a random oracle to simulate the process of generating encrypted file EF and encrypted index I and therefore we achieve adaptive secure SSE mentioned in section 4.4.

6.4 Access pattern security

Without knowing secret component of $\mathbf{K_{ul}}$ and $\mathbf{K_{ur}}$, when receiving some trapdoor $t_{i,j}$ of node $L_{i,j}$, in order for the adversary to figure out trapdoor value at parent node of $L_{i,j}$, he must find a way to break the trapdoor permutation function. Furthermore in our scheme, the client only gives server values $t_{i,j}$ that cover the interval $[L; R]$. With this, our scheme achieves access pattern security.

6.5 Search pattern security

Assume that the client has issued 2 queries $\mathbf{Q_1} = (\mathbf{T_1}, L_1, R_1)$ and $\mathbf{Q_2} = (\mathbf{T_2}, L_2, R_2)$ where $\mathbf{T_1}$ and $\mathbf{T_2}$ refers to the same keyword w. There are 2 cases:

1. If $[L_1; R_1]$ intersects with $[L_2, R_2]$: let k be a number where $k \in [L_1; R_1] \cap [L_2; R_2]$, the server obviously can check T_1 and T_2 be the same search keyword because the trapdoor t_k of the 2 search queries will be the same.

2. If $[L_1; R_1]$ does not intersect with $[L_2, R_2]$: it is impossible for the untrusted server to check T_1 and T_2 to be the same search keyword unless he can calculate π and achieve trapdoor value at parent nodes of $\mathbf{T}$, which is very hard and the probability is negligible.

By analyzing the 2 above cases, we claim that our scheme achieves weak search pattern security of TIQSSE.

6.6 Forward privacy

We already stated that our scheme achieves access pattern security which prevents the server from gaining knowledge of outside the interval query. Furthermore, it is obvious that newly added files are outside of past search queries. To sum up, we claim that our scheme obtains forward privacy.

6.7 Backward privacy

When receiving a deleting query, the server deletes the encrypted file along with the encrypted index on the database/hardware which prevents the server from gaining more knowledge from it in future searches.

However, if the attacker can clone the encrypted index to elsewhere without the client's knowledge, then our scheme does not achieve backward privacy. Because of that, our scheme can only guarantee backward privacy if we assume that the system is honest-but-curious. The honest-but-curious property implies that the system follows explicitly how the scheme is supposed to do, but still listens to the client's queries and try to exploit for vulnerabilities. This is an important property that has been used widely in many constructions [8, 13, 2, 4, 24].

6.8 Complexity analysis

Search complexity. Let $n_{add} = R - L + 1$ be the number of historically added documents, n be the number of remaining documents, and m be the number of document-deleted segments in the searching range $[L, R]$, respectively. For each value $t_{i,j}$ received from client, server must traverse all nodes of the sub-tree associated with $t_{i,j}$. The size of that sub-tree is equal to $2 \cdot n_{leaf} - 1$ where n_{leaf} is the number of leaves of that sub-tree. However, we don't have to consider nodes which is assigned as deleted. That means we do not traverse any sub-tree whose root is deleted. In conclusion, the search complexity is $\mathcal{O}(n+m)$.

Add document complexity. Obviously, the server's time complexity of Add operation is $\mathcal{O}(1)$. For the client's time complexity of Add operation: The first step (*encrypt file*) implementation time depends on which symmetric encryption is used. Let $size$ be the number of words in file. Because the complexity of Trpdr is $\mathcal{O}(\log(n))$ where n is the identifier of the file, the complexity of creating encrypted index is $\mathcal{O}(\log(n) \cdot size)$.

Delete document complexity. Trivially, the client's time complexity of deleting document is $\mathcal{O}(1)$. On server, the complexity of deleting document is proportional to the number of iterations which is not greater than $log(n)$ where n is the total number of historically added documents.

Storage complexity. The client's storage is $\mathcal{O}(1)$ because the user only needs to store the keys. About the server's storage, we can see that the size of encrypted indexes is the same as that of encrypted data. Plus, each node of the interval tree holds an isDeleted attribute that totally costs the complexity storage as the number of existing nodes in the tree. Therefore, we can conclude that the storage complexity of server is $\mathcal{O}(esize)$ where $esize$ is the size of encrypted data.

7 Discussion and conclusion

In this paper, we first introduce a smart secured multimedia service to provide visual content analysis with smart edge cameras while preserving the privacy of data owners. We also define a new problem for range queries with searchable symmetric encryption to prevent sensitive information leakage to service provider. Finally, we propose a secured scheme for the new problem.

It is impractical that our scheme only supports single keyword search although there has been many previous works toward multi keywords search [28]. To make our scheme works for multi keywords search, we can combine multiple keywords that is near each other into one single word. For example, we can combine k consecutive words into one and mark it as existed in the encrypted index. For further versatile, we can even permute these k words into $k!$. With this approach, we have multi keywords search property.

In order to achieve perfect secrecy in search pattern, we can simply not searching for keyword w of the same range $[L; R]$ that has been issued before. Because the client has already searched for word w within range $[L; R]$, we can store the result in our memory. Later when we want to query the keyword w within other range $[L_2; R_2]$, we first eliminate all the shared intersection on word w that has been searched before, let the interval after elimination be $\mathbf{S}$. Then, we only query on $\mathbf{S}$ and merge the returned results with the solutions in our memory. With this approach, we can achieve perfect secrecy of the search pattern because we never search for keyword w that intersects with the previous search interval. However, the downside of this is the client must have some mechanism to store previously search queries, which makes it impractical.

Currently, we are improving our system to optimize its performance, scalability, and reliability. We also analyse other data structures to enhance the solution for SSE and consider potential leakage via side-channel information.

8 Acknowledgement

This research is partially supported by research funding from Advanced Program in Computer Science and Honors Program, University of Science, Vietnam National University - Ho Chi Minh City. We also would like to express our appreciation to AIOZ Pte Ltd and VINIF for supporting our research team.

References

[1] Boelter, T., Poddar, R., Popa, R.A.: A secure one-roundtrip index for range queries. IACR Cryptology ePrint Archive **2016**, 568 (2016)

[2] Bost, R.: $\sigma o\varphi o\varsigma$: Forward secure searchable encryption. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1143–1154. ACM (2016). https://doi.org/10.1145/2976749.2978303

[3] Bost, R., Fouque, P.A.: Thwarting leakage abuse attacks against searchable encryption-a formal approach and applications to database padding. IACR Cryptology ePrint Archive **2017**, 1060 (2017)

[4] Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1465–1482. ACM (2017). https://doi.org/10.1145/3133956.3133980

[5] Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 668–679. ACM (2015). https://doi.org/10.1145/2810103.2813700

[6] Chang, Y.C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: IACR Cryptology ePrint Archive (2004)

[7] Chung-Nguyen, H.H., Pham, V.A., Hoang, D.H., Tran, M.T.: Keyword-search interval-query dynamic symmetric searchable encryption. In: International Conference on Future Data and Security Engineering. pp. 673–680. Springer (2019). https://doi.org/10.1007/978-3-030-35653-8_46

[8] Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011). https://doi.org/10.3233/jcs-2011-0426

[9] Dan Boneh, V.S.: A Graduate Course in Applied Cryptography (2017)

[10] Goh, E.J., et al.: Secure indexes. IACR Cryptology ePrint Archive **2003**, 216 (2003)

[11] Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M., Ristenpart, T.: Leakage-abuse attacks against order-revealing encryption. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 655–672. IEEE (2017). https://doi.org/10.1109/sp.2017.44

[12] Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: Ndss. vol. 20, p. 12. Citeseer (2012)

[13] Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 965–976. ACM (2012). https://doi.org/10.1145/2382196.2382298

[14] Kerschbaum, F., Tueno, A.: An efficiently searchable encrypted data structure for range queries. In: Lecture Notes in Computer Science, pp. 344–364. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-29962-0_17

[15] Le, N., Nguyen, D., Hoang, T., Nguyen, T., Truong, T., Duy, T.D., Luong, Q., Vo-Ho, V., Nguyen, V., Tran, M.: Smart lifelog retrieval system with habit-based concepts and moment visualization. In: Proceedings of the ACM Workshop on Lifelog Search Challenge, LSC@ICMR 2019, Ottawa, ON, Canada, 10 June 2019. pp. 1–6 (2019). https://doi.org/10.1145/3326460.3329155

[16] Le, N., Nguyen, D., Nguyen, V., Tran, M.: Lifelog moment retrieval with advanced semantic extraction and flexible moment visualization for exploration. In: Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum, Lugano, Switzerland, September 9-12, 2019. (2019)

[17] Le, T.K., Ninh, V.T., Dang-Nguyen, D.T., Tran, M.T., Zhou, L., Redondo, P., Smyth, S., Gurrin, C.: Lifeseeker - interactive lifelog search engine at lsc 2019. In: Proceedings of the 2019 ACM Workshop on The Lifelog Search Challenge. ACM (2019). https://doi.org/10.1145/3326460.3329162

[18] Münzer, B., Leibetseder, A., Kletz, S., Primus, M.J., Schoeffmann, K.: lifexplore at the lifelog search challenge 2018. In: Proceedings of the 2018 ACM Workshop on The Lifelog Search Challenge. pp. 3–8 (2018). https://doi.org/10.1145/3210539.3210541

[19] Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 644–655. ACM (2015). https://doi.org/10.1145/2810103.2813651

[20] Pham, V.A., Hoang, D.H., Chung-Nguyen, H.H., Tran, M.K., Tran, M.T.: Privacy preserving visual log service with temporal interval query using interval tree-based searchable symmetric encryption. In: Proceedings of the Tenth International Symposium on Information and Communication Technology. pp. 425–432 (2019). https://doi.org/10.1145/3368926.3369701

[21] Redmon, J., Farhadi, A.: Yolov3: An incremental improvement (2018), `http://arxiv.org/abs/1804.02767`

[22] Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1. pp. 91–99. NIPS'15, MIT Press, Cambridge, MA, USA (2015). https://doi.org/10.1109/tpami.2016.2577031

[23] Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. pp. 44–55. IEEE (2000). https://doi.org/10.1109/secpri.2000.848445

[24] Sun, S.F., Yuan, X., Liu, J.K., Steinfeld, R., Sakzad, A., Vo, V., Nepal, S.: Practical backward-secure searchable encryption from symmetric puncturable encryption. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 763–780. ACM (2018). https://doi.org/10.1145/3243734.3243782

[25] Tran, M., Truong, T., Duy, T.D., Vo-Ho, V., Luong, Q., Nguyen, V.: Lifelog moment retrieval with visual concept fusion and text-based query expansion. In: Working Notes of CLEF 2018 - Conference and Labs of the Evaluation Forum, France, September 2018. (2018)

[26] Truong, T.D., Dinh-Duy, T., Nguyen, V.T., Tran, M.T.: Lifelogging retrieval based on semantic concepts fusion. In: Proceedings of the 2018 ACM Workshop on The Lifelog Search Challenge. pp. 24–29. ACM (2018). https://doi.org/10.1145/3210539.3210545

[27] Vo-Ho, V.K., Luong, Q.A., Nguyen, D.T., Tran, M.K., Tran, M.T.: Personal diary generation from wearable cameras with concept augmented image captioning and wide trail strategy. In: Proceedings of the Ninth International Symposium on Information and Communication Technology. pp. 367–374. SoICT 2018, ACM (2018). https://doi.org/10.1145/3287921.3287955

[28] Xia, Z., Wang, X., Sun, X., Wang, Q.: A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE transactions on parallel and distributed systems **27**(2), 340–352 (2015). https://doi.org/10.1109/tpds.2015.2401003

[29] Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 707–720 (2016)

[30] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A.: Places: A 10 million image database for scene recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (2017). https://doi.org/10.1109/tpami.2017.2723009