Assignment: 3-Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data

Author

Dalian University of Technology
Assignment of System Security

March 28, 2015

Overview

Introduction of the Paper

C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *INFOCOM, 2012 Proceedings IEEE*, pp. 451–459, IEEE, 2012.

Introduction of the Paper

Purpose

Solve the problem of secure and efficient fuzzy search over encrypted outsourced cloud data

Measures

- Suppressing technique
- Building a private trie-traverse searching index

Performance

Correctly achieves the defined similarity search functionality with **constant** searching time!

System and Threat Model

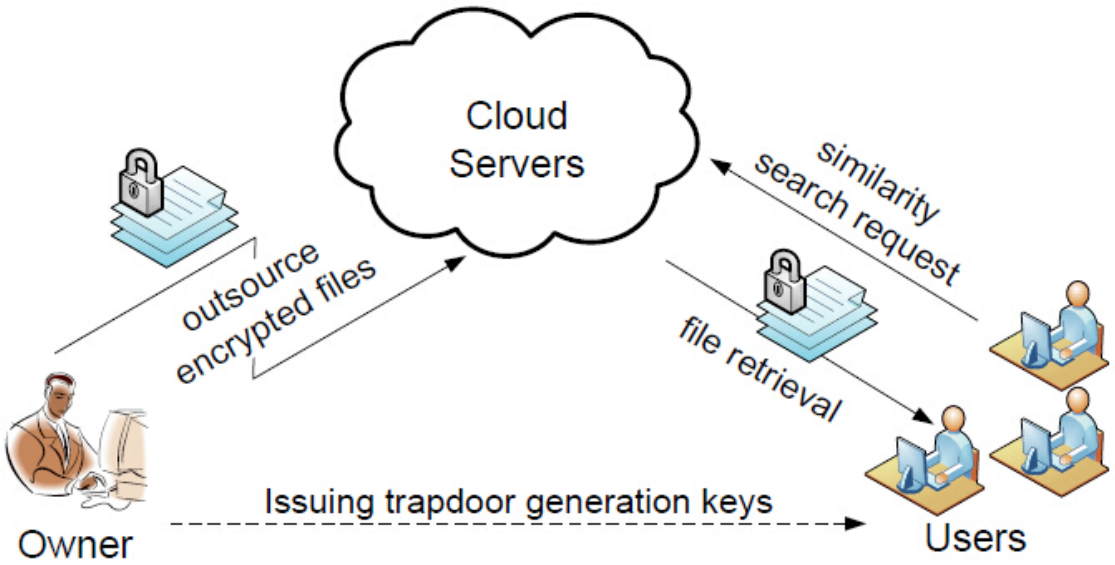


Figure: Architecture of similarity keyword search over outsourced cloud data

- data owner: the individual/enterprise customer,who has a collection of n data files $C = (F_1, F_2, \ldots, F_n)$ to be stored in the cloud server.
- $W = \{w_i, w_w, \ldots, w_p\}$ is denoted as a predefined set of distinct keywords in C

System and Threat Model

- Files are encrypted before outsourced
- The data owner will distribute search request(trapdoor) generation keys *sk* to authorized users.(Assume that the authorization will be done appropriately)
- An authorized user uses trapdoor generation key to generate a search request via some one-way function to search word w, and submit it to the cloud.
- The cloud then performs the search over the data collection C without decryption and sends back all encrypted files containing the specific keyword w, denoted as FID_w.
- The similarity keyword search scheme returns the closest possible results based on aforementioned measures.
- At last, the user decrypts files they received from the cloud.

Assumption: Honest-but-curious clound server

To ensure the securely similarity searching schema:

Honest

Correctly follows the designated protocol specification

Curious

Infer and analyze the message flow received during the protocol so as to learn additional information

We follow the security definition deployed in the traditional searchable symmetric encryption(SSE)

Notations

C the file collection to be outsourced, denoted as a set of n data files $C = (F_1, F_2, \ldots, F_n)$.

W the distinct keywords extracted from file collection C, denoted as a set of m words $W = \{w_i, w_w, \ldots, w_p\}$.

$\mathcal{I}$ the index built for privacy-assured similarity search.

T_w the trapdoor generated by a user as a search request of input keyword w via some one-way transformation.

$S_{w,d}$ similarity keyword set of w, where d is the similarity threshold according to a certain similarity metrics.

FID_{w_i} the set of identifiers of files in C that contain keyword w_i.

$f(key, \cdot), g(key, \cdot)$ pseudorandom function (PRF), defined as: $\{0,1\}^* \times key \rightarrow \{0,1\}^{\ell}$.

$Enc(key, \cdot), Dec(key, \cdot)$ symmetric key based semantic secure encryption/decryption function.

Edit Distance

Quantitative measurement

The edit distance $ed(w_1, w_2)$ between two words w_1 and w_2 is the mininum number of primitive operations, including character insertion, deletion and substitution, necessary to transform one of them into the other.

Similarity keyword set

Given a keyword w, we let $S_{w,d}$ denote its similarity set of words, such that any $w' \in S_{w,d}$ satisfies $ed(w, w') \leq d$ for a certain integer d.

Example

Consider the keyword $w_0 = CENSOR$
a words set $W = \{CESOR, CENSER, CEANSOR\}$
for any $w' \in W, ed(w_0, w') \leq 1$ holds,
i.e. $w' \in S_{w_0,1}$ and $W \subseteq S_{w_0,1}$

Building Similarity Keyword Sets

Straightforward approach

Simply enumerating all possible words w_i' satisfying the similarity criteria $ed(w_i, w_i') \leq d$

For the keyword $w_0 = CENSOR$, consider just one substitution operation with charaters on first character.
There are 26 items
$\{AENSOR, BENSOR, \ldots, YENSOR, ZENSOR\}$
So $S_{w_0,1}$ will be
$[6 + (6 + 1)] \times 26 + 1$

Suppression technique

Consider only the positions of the primitive edit operations.Specifically, we use a wildcard * to denote all three operations of character insertion, deletion and substitution at any position.

Now,
$S_{SENSOR,1} = \{SENSOR, *SENSOR, *ENSOR, S*ENSOR, S*NSOR, \ldots, SENSO*R, SENSO*, SENSOR*\}$.
Size can be reduced to $S_{w_0,1}$ will be
$[6 + (6 + 1)] \times 1 + 1$

Building Similarity Keyword Sets

Algorithm 1: CreateSimilaritySet(w_i, d)

Data: keyword w_i and threshold distance d
Result: similarity keyword set $S_{w_i,d}$
begin
  if $d > 1$ **then**
1   CreateSimilaritySet(w_i, $d-1$);
  if $d = 0$ **then**
2   set $S_{w_i,d} = \{w_i\}$;
  else
    for $k \leftarrow 1$ *to* $|S_{w_i,d-1}|$ **do**
      for $j \leftarrow 1$ *to* $2 \times |S_{w_i,d-1}[k]| + 1$ **do**
        if j *is odd* **then**
3   Set *variant* as $S_{w_i,d-1}[k]$;
4   Insert $\star$ at position $\lfloor (j+1)/2 \rfloor$;
        else
5   Set *variant* as $S_{w_i,d-1}[k]$;
6   Replace $\lfloor j/2 \rfloor$-th character with $\star$;
        if *variant is not in* $S_{w_i,d-1}$ **then**
7   Set $S_{w_i,d} = S_{w_i,d} \cup \{variant\}$;

The size of $S_{w_i,d}$ will be $\mathcal{O}(\ell^d)$,opposing to $\mathcal{O}(\ell^d \times 26^d)$ obtained in the straightforward approach.

Generating Searching Request

Theorem

The intersection of the similarity sets $S_{w_i,d}$ and $S_{w,d}$ for keyword w_i and search input w is not empty if and only if $ed(w, w_i) \leq d$.

Proof.

- Completeness(i.e. $ed(w, w_i) \leq d \rightarrow S_{w_i,d} \cap S_{w,d} \neq \emptyset$):
  - $w \rightarrow w_i$ need at most d primitive operations.
    the result after these operations is marked as w^*
  - w^* is naturally in $S_{w,d}$
  - w^* can be transformed into w_i, so it must be in $S_{w_i,d}$
  - $w' \in S_{w_i,d} \cap S_{w,d}$

Generating Searching Request

Proof.

- Soundness(i.e. $S_{w_i,d} \cap S_{w,d} \neq \emptyset \rightarrow ed(w, w_i) \leq d$)

  w^* the common element in $S_{w_i,d} \cap S_{w,d}$

  1. w^* does not contain any wildcard *,
     then $w^* = w = w'$,and $ed(w, w') = 0 \leq d$
  2. w^* does contain some wildcard *(at most d *'s),
     change * in w^* back to the character in w and w_i,
     denote the result as w'^* and $w_i'^*$ with both sharing $d-1$ different *'s.
     $w'^* \rightarrow w_i'^*$ need at most one primitive operation.
     So, $ed(w'^*, w_i'^*) \leq 1$
     $\Rightarrow ed(w, w_i) \leq d$

The Basic Scheme

τ the maximum size of the similarity keyword set $S_{w_i,d}$ for $w_i \in W$, i.e.,$\tau = \max\{|S_{w_i,d}|\}_{w_i \in W}$ where $|W| = p$.

Prepocessing phase(the owner)

1. picks random key x,y,and builds index $\mathcal{I} = \left\{ f(x, w_i'), Enc(sk_{w_i'}, FID_{w_i}) \right\}_{w_i' \in S_{w_i,d}, 1 \le i \le p}$, where secret key $sk_{w_i'} = g(y, w_i')$
2. insert extra $\tau|W| - |\mathcal{I}|$ dummy entries(using random values) in $\mathcal{I}$ for padding.
3. randomly shuffles $\mathcal{I}$, outsources $\mathcal{I}$, encrypted C to cloud.

The Basic Scheme

Searching phase(the user)

1. generates $S_{w,d}$ from input w via Algorithm 1 and derives $T_{w'} = (f(x, w'), g(y, w'))$ for each $w' \in S_{w,d}$
2. generates $\tau - |S_{w,d}|$ dummy trapdoors by randomly choosing j from $\{f(x,j), g(y,j)\}_{1 \le j \le \tau|W|-|\mathcal{I}|}$
3. Cloud server compares all received trapdoors $\{f(x, w')\}$(and $\{f(x,j)\}$) with $\mathcal{I}$ uses the corresponding $\{g(y, w')\}$ to decrypt the matched entries, and returns the union of file identifiers, $\{FID_{w_i}\}_{ed(w,w_i) \le d}$.
4. The user retrieves and decrypts the files of interest.

Improvement

Storage and search cost is $\mathcal{O}(\tau\,|W|)$.
Bloom Filters can be introduced in to reduce the searching cost to $\mathcal{O}(|W|)$

The Symbol-based Trie-Traverse Searching Schema

All similar words in the trie can be found by a depth-first search

Construct a multi-way tree for storing the similarity keyword elements over a finite symbol set.All trapdoors sharing a common prefix have a common node. The root is associated with an empty set.

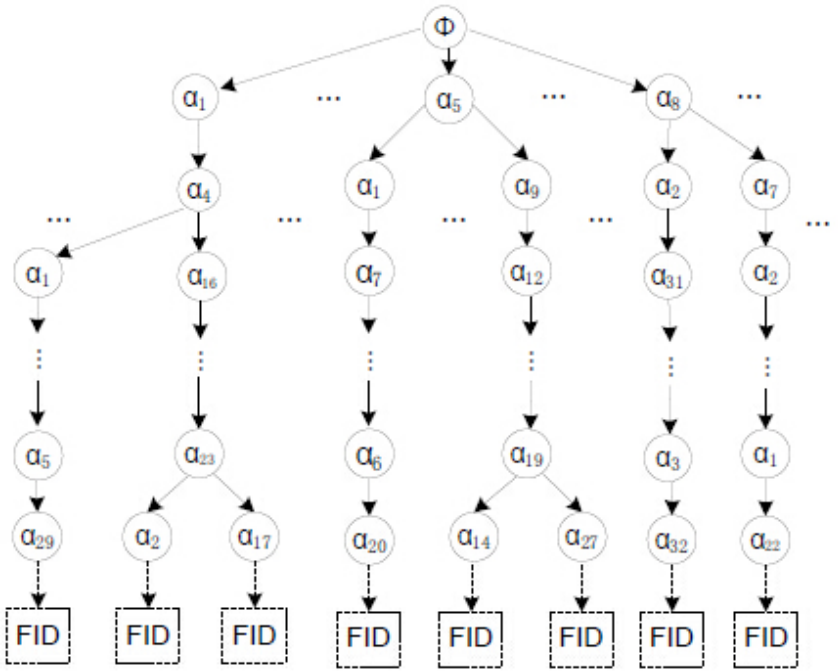


Figure: The depth of the tree is l/θ

- Assume $\Delta = \{\alpha_i\}$ is a predefined symbol set.
- $|\Delta| = 2^\theta$
- each symbol $\alpha_i \in \Delta$ is denoted by a θ-bit binary vector.
- l is the output length of one-way function $f(key, \cdot)$.

The Symbol-based Trie-Traverse Searching Schema

Preprocessing phase(the owner)

1. computes $f(x, w'_i)$ for each $w'_i \in S_{w_i,d}, 1 \leq i \leq p$ together with dummy entries.
2. divides them into symbols as $\alpha_{i_1} \cdots \alpha_{i_{l/\theta}}$ from Δ.
3. builds up a trie G_W covering all $w_i \in W$.
4. attaches $\left\{Enc(sk_{w'_i}, FID_{w_i})\right\}_{w'_i \in S_{w_i,d}, 1 \leq i \leq p}$ to G_W, outsourced it with encrypted collection C to the cloud.

Searching phase(the user)

1. sends a set of τ trapdoors(research request): $\{T_{w'}\}_{w' \in S_{w,d}}$ and $\tau - |w' \in S_{w,d}|$ dummy trapdoors.
2. the cloud server divides each $f(x, w')$ into a sequence of symbols from Δ. Perform the search over the trie G_W.
3. decrypts matches entries via $g(y, w')$ and return $\{FID_{w_i}\}_{ed(w,w_i) \leq d}$ to the user.

The Symbol-based Trie-Traverse Searching Schema

Algorithm 2: SearchingTree($\{T'_w\}, G_w$)

Data: Searching Trapdoor set $\{T'_w\}$ and G_w
Result: Result ID set
begin
  for $i \leftarrow 1$ *to* $|\{T'_w\}|$ **do**
1 set $currentnode$ as $root$ of G_w;
    for $j \leftarrow 1$ *to* l/θ **do**
2 Set α as α_j in $f(x, w')$ within the i-th T'_w;
      if *no child of* $currentnode$ *contains* α **then**
3 **break**;
4 Set $currentnode$ as $child$ containing α;
    if $currentnode$ *is* $leafnode$ **then**
5 Append $currentnode.FIDs$ to $resultIDset$;
      if $i = 1$ **then**
6 return $resultIDset$;
7 return $resultIDset$;

The search cost at the server side is only $\mathcal{O}(1)$(a constant related to l/θ)

Scheme Evaluation

	SSE	**Basic**	**Trie-traverse**
Preprocessing	$\mathcal{O}(\|W\|)$	$\mathcal{O}(\tau\,\|W\|)$	$\mathcal{O}(\tau\,\|W\|)$
Index size	$\mathcal{O}(\|W\|)$	$\mathcal{O}(\tau\,\|W\|)$	$\mathcal{O}(\tau\,\|W\|)$
Search cost	$\mathcal{O}(1)$	$\mathcal{O}(\tau\,\|W\|)$	$\mathcal{O}(1)$
Similarity search	No	Yes	No

Table: Comparison of SSE schemes

Experiment

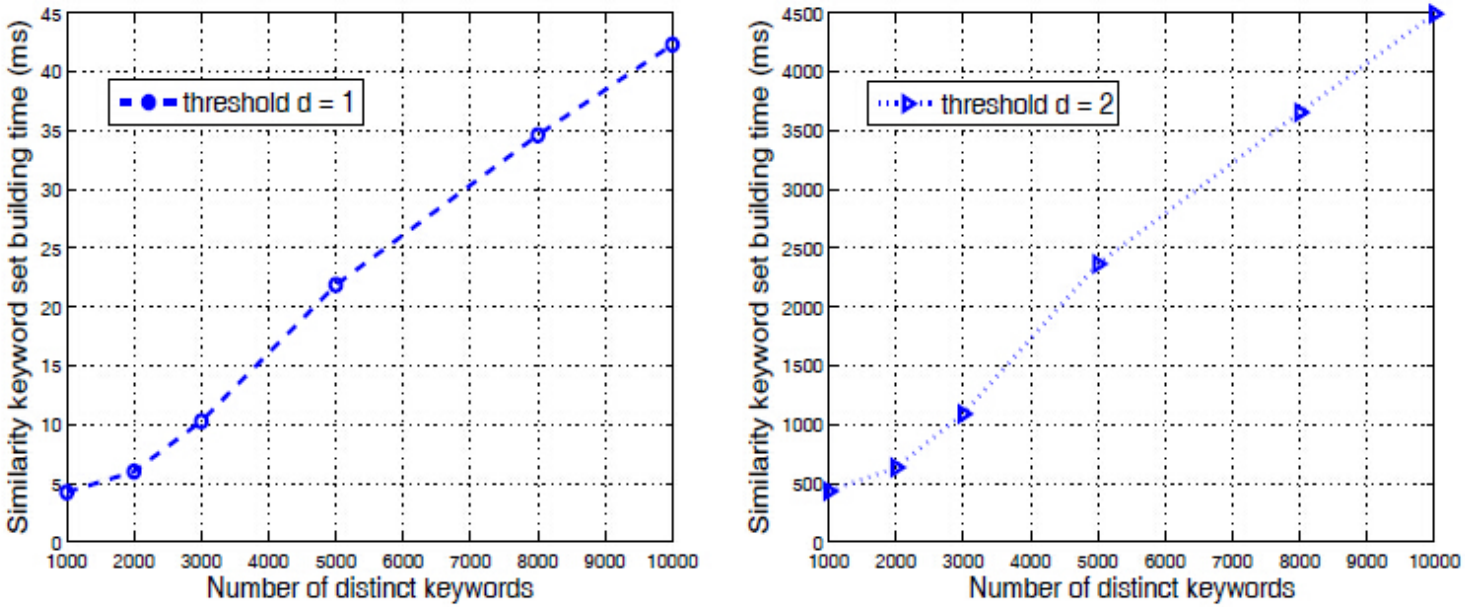
Experiment Design

- A real data set: RFC, 5,731 plaintext files, 277MB
- C programming language
- Local workstation
- Cloud side:Amazon Elastic Computing Cloud(EC2)

Note

In our experiment the dominant factor affecting the performance is th number of unique keywords to be indexed, not the file collection size.

Cost for Generating Similarity Keyword Set

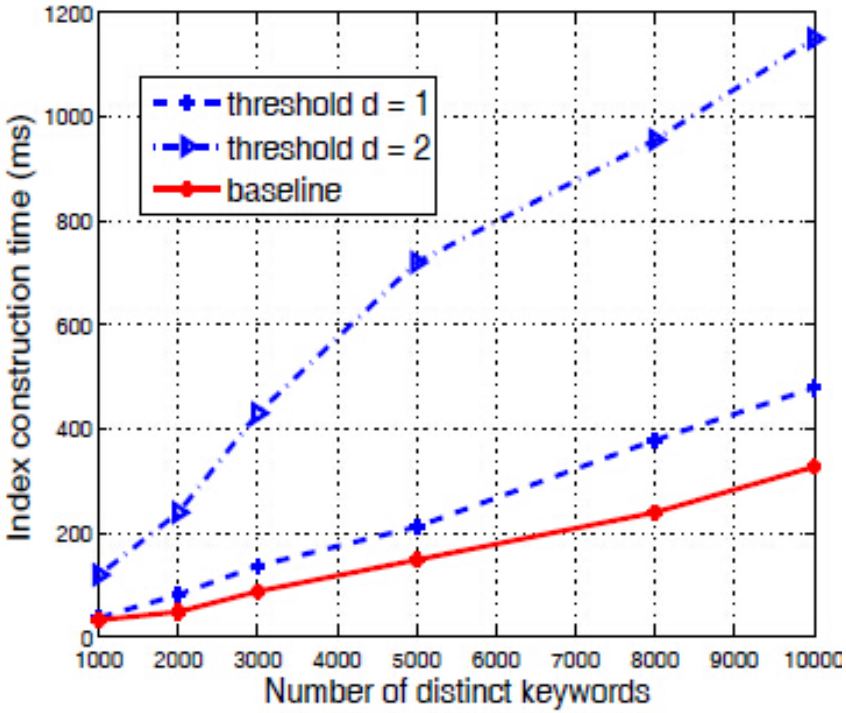


(a) $d = 1$. (b) $d = 2$.

Figure: Similarity set construction time using wildcard-based approach with different choices of edit distance d

The construction time increases linearly with the number of keywords.

Cost For Building Searchable Index



(a) Index construction time.

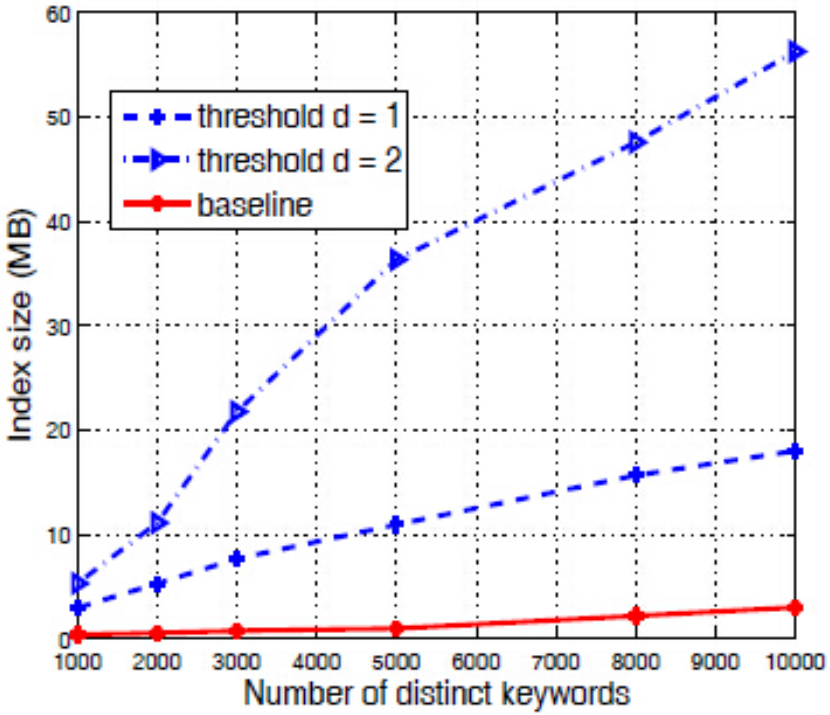
Figure: Time cost for searchable index construction with different choices of edit distance d

For completeness, we also include the index building time of existing SSE as a baseline for comparison here.

The whole index construction is just a one-time cost and can be conducted off-line

Similar to the similarity keyword set construction, the index construction time increases linearly with the number of distinct keywords.

Cost For Building Searchable Index



(b) Index storage size.

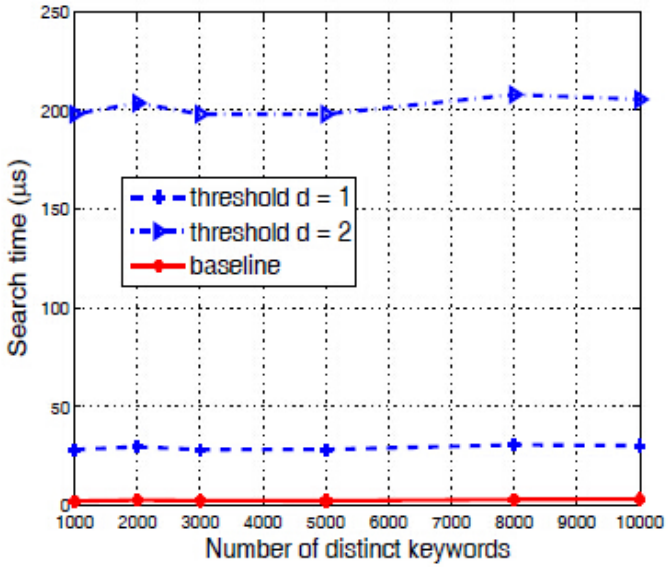
Figure: Storage cost for searchable index construction with different choices of edit distance d

Again, our approach consumes more storage space than the baseline due to the multi-way tree structure and the additional entries in the index corresponding to the similarity keywords

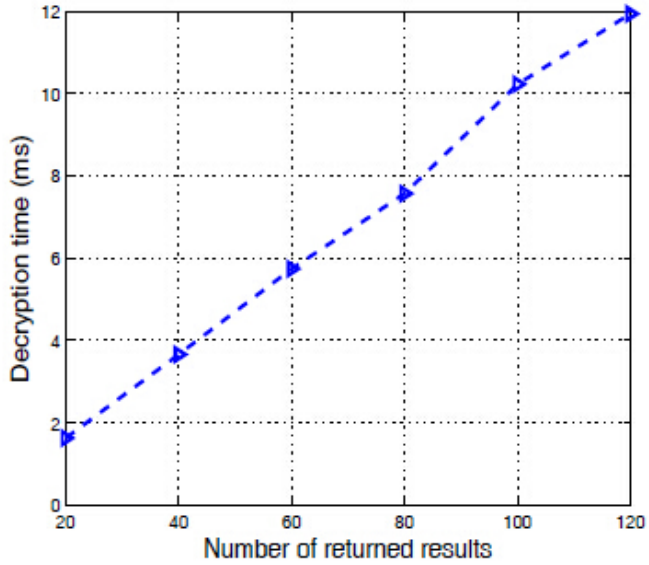
But can be deemed as reasonable cost of supporting similarity search

"average keyword length" also sightly influence the time and space cost of building searchable index.

Cost For Searching the Index



(a) Cloud side search time.



(b) User side decryption cost.

The proposed mechanism cost constant search time.

Cost for results retrieval and decryption is plainly determined by the number of retrieved results

Thanks