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

Dehai Wang
Siyu Huang
Xinyi Li

Dalian University of Technology
Assignment of System Security

April 1, 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

Introduction of the Paper

Purpose

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

Measures

Introduction of the Paper

Purpose

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

Measures

- Suppressing technique

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

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

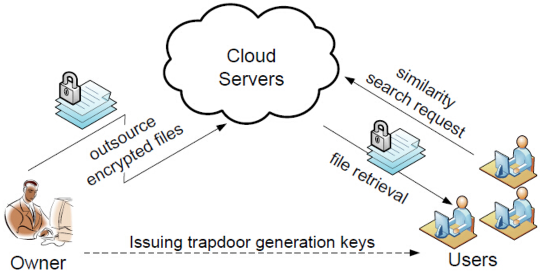


Figure: Architecture of similarity keyword search over outsourced cloud data

System and Threat Model

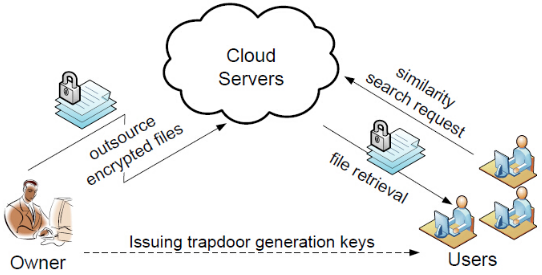


Figure: Architecture of similarity keyword search over outsourced cloud data

- 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.

System and Threat Model

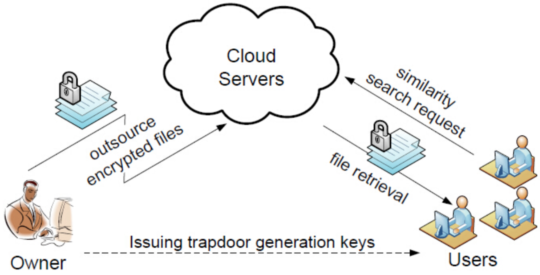


Figure: Architecture of similarity keyword search over outsourced cloud data

- 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

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)

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.

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.

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.

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 scheme:

Assumption: Honest-but-curious clound server

To ensure the securely similarity searching scheme:

Honest

Correctly follows the designated protocol specification

Assumption: Honest-but-curious clound server

To ensure the securely similarity searching scheme:

Honest

Correctly follows the designated protocol specification

Curious

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

Assumption: Honest-but-curious clound server

To ensure the securely similarity searching scheme:

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

Notations

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

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 $W = \{w_i, w_w, \ldots, w_p\}$.

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.

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.

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.

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.

Notations

- C the file collection to be outsourced, denoted as a set of n data files $C = (F_1, F_2, \dots, F_n)$.
- W the distinct keywords extracted from file collection C, denoted as a set of m words $W = \{w_i, w_w, \dots, w_p\}$.
- $\mathcal{I}$ 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: $\{0,1\}^* \times key \rightarrow \{0,1\}^\ell$.

Notations

- C the file collection to be outsourced, denoted as a set of n data files $C = (F_1, F_2, \dots, F_n)$.
- W the distinct keywords extracted from file collection C, denoted as a set of m words $W = \{w_i, w_w, \dots, w_p\}$.
- $\mathcal{I}$ 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: $\{0,1\}^* \times key \to \{0,1\}^\ell$.
- $Enc(key, \cdot), Dec(key, \cdot)$ 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 minimum number of primitive operations, including character insertion, deletion and substitution, necessary to transform one of them into the other.

Edit Distance

Quantitative measurement

The edit distance $ed(w_1, w_2)$ between two words w_1 and w_2 is the minimum 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.

Edit Distance

Quantitative measurement

The edit distance $ed(w_1, w_2)$ between two words w_1 and w_2 is the minimum 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

Building Similarity Keyword Sets

Straightforward approach

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

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
$[6 + (6 + 1)] \times 26 + 1$

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.

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**
        CreateSimilaritySet(w_i, $d-1$);
    if $d = 0$ **then**
        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**
                    Set *variant* as $S_{w_i,d-1}[k]$;
                    Insert $\star$ at position $\lfloor (j+1)/2 \rfloor$;
                else
                    Set *variant* as $S_{w_i,d-1}[k]$;
                    Replace $\lfloor j/2 \rfloor$-th character with $\star$;
                if *variant is not in* $S_{w_i,d-1}$ **then**
                    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

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* $ed(w, w_i) \leq d$.

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. $ed(w, w_i) \leq d \rightarrow S_{w_i,d} \cap S_{w,d} \neq \emptyset$):

$\square$

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 $S_{w_i,d}$
  - $w^* \in S_{w_i,d} \cap S_{w,d}$

$\square$

Generating Searching Request

Proof.

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

$\square$

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 $S_{w_i,d} \cap S_{w,d}$

$\square$

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 $ed(w, w') = 0 \leq d$

$\square$

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, $ed(w'^*, w_i'^*) \leq 1$
     $\Rightarrow ed(w, w_i) \leq d$

$\square$

The Basic Scheme

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 $|W| = p$.

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 $|W| = p$.

Preprocessing phase(the owner)

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 $|W| = p$.

Preprocessing phase(the owner)

1. picks random key x,y,and builds index $\mathcal{I} = \left\{ f(x, w_i'), Enc(sk_{w_i'}, FID_{w_i}) \right\}_{w_i' \in S_{w_i,d}, 1 \leq i \leq p}$, where secret key $sk_{w_i'} = g(y, w_i')$

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 $|W| = p$.

Preprocessing 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.

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 $|W| = p$.

Preprocessing 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)

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 $w' \in S_{w,d}$

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 $\{f(x, j), g(y, j)\}_{1 \leq j \leq \tau |W| - |\mathcal{I}|}$

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 $\{f(x, j), g(y, j)\}_{1 \leq j \leq \tau|W| - |\mathcal{I}|}$
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, $\{FID_{w_i}\}_{ed(w, w_i) \leq d}$.

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 $\{f(x,j), g(y,j)\}_{1 \leq j \leq \tau|W| - |\mathcal{I}|}$
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, $\{FID_{w_i}\}_{ed(w,w_i) \leq d}$.
4. The user retrieves and decrypts the files of interest.

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 $\{f(x,j), g(y,j)\}_{1 \leq j \leq \tau |W| - |\mathcal{I}|}$
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, $\{FID_{w_i}\}_{ed(w,w_i) \leq d}$.
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 $\mathcal{O}(|W|)$

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.

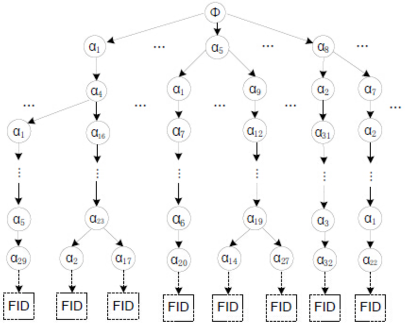


Figure: The depth of the tree is l/θ

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.

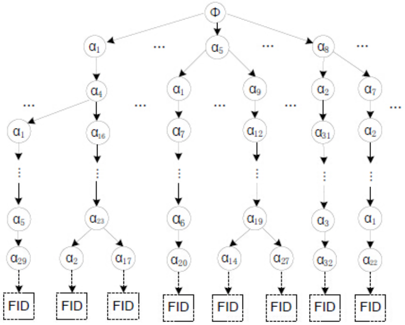


Figure: The depth of the tree is l/θ

- 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

The Symbol-based Trie-Traverse Searching Schema

Preprocessing phase(the owner)

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.

Preprocessing phase(the owner)

1. computes $f(x, w'_i)$ for each $w'_i \in S_{w_i,d}$,$1 \le i \le p$ together with dummy entries.
2. divides them into symbols as $\alpha_{i_1} \cdots \alpha_{i_{l/\theta}}$ from Δ.

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 $w_i \in W$.

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.

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)

The Symbol-based Trie-Traverse Searching Schema

Preprocessing phase(the owner)

1. computes $f(x, w_i')$ for each $w_i' \in S_{w_i,d}, 1 \le i \le p$ 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 $\left\{Enc(sk_{w_i'}, FID_{w_i})\right\}_{w_i' \in S_{w_i,d}, 1 \le i \le p}$ 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.

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.

The Symbol-based Trie-Traverse Searching Schema

Preprocessing phase(the owner)

1. computes $f(x, w'_i)$ for each $w'_i \in S_{w_i,d}, 1 \le i \le p$ 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 $\left\{ Enc(sk_{w'_i}, FID_{w_i}) \right\}_{w'_i \in S_{w_i,d}, 1 \le i \le p}$ 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 returns $\{FID_{w_i}\}_{ed(w,w_i) \le d}$ 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**
    set $currentnode$ as $root$ of G_w;
    for $j \leftarrow 1$ *to* l/θ **do**
      Set α as α_j in $f(x, w')$ within the i-th T'_w;
      if *no child of* $currentnode$ *contains* α **then**
        break;
      Set $currentnode$ as $child$ containing α;
    if $currentnode$ is $leafnode$ **then**
      Append $currentnode.FIDs$ to $resultIDset$;
      if $i = 1$ **then**
        return $resultIDset$;
  return $resultIDset$;

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

Security Gurantee

Security Gurantee

- Our searching mechanism always returns the same search results for the same search requests.
- The cloud server can build up access patterns and search patterns along the interactions with users.

Security Gurantee

- Our searching mechanism always returns the same search results for the same search requests.
- The cloud server can build up access patterns and search patterns along the interactions with users.

Thus, the security guarantee should ensure that nothing beyond the pattern and the outcome of a series of search requests be leaked.

Security Gurantee

- Our searching mechanism always returns the same search results for the same search requests.
- The cloud server can build up access patterns and search patterns along the interactions with users.

Thus, the security guarantee should ensure that nothing beyond the pattern and the outcome of a series of search requests be leaked.

The non-adaptive semantic security guarantee

The non-adaptive attack model only considers adversaries (i.e., the cloud server) that cannot choose search requests based on the trapdoors and search outcomes of previous searches. (Since only users with authorized secret keys can generate search trapdoors.)

Notations

Notations

History an interaction between the user and the cloud server, determined by a file collection C and a set of keywords searched by the user, denoted as $H_q = (C, w^1, \ldots, w^q)$.

Notations

History an interaction between the user and the cloud server, determined by a file collection C and a set of keywords searched by the user, denoted as $H_q = (C, w^1, \ldots, w^q)$.

View given a history H_q under some secret key K, the cloud server can only see an encrypted version of the history, i.e., the view $V_K(H_q)$, including: the index $\mathcal{I}$ of C; the trapdoors of the queried keywords $\{T_{w'}\}_{w' \in \{S_{w^1,d}, \ldots, S_{w^q,d}\}}$; and the encrypted file collection of C, denoted as $\{e_1, \ldots, e_n\}$.

Notations

History an interaction between the user and the cloud server, determined by a file collection C and a set of keywords searched by the user, denoted as $H_q = (C, w^1, \ldots, w^q)$.

View given a history H_q under some secret key K, the cloud server can only see an encrypted version of the history, i.e., the view $V_K(H_q)$, including: the index $\mathcal{I}$ of C; the trapdoors of the queried keywords $\{T_{w'}\}_{w' \in \{S_{w^1,d}, \ldots, S_{w^q,d}\}}$; and the encrypted file collection of C, denoted as $\{e_1, \ldots, e_n\}$.

Trace given a history H_q and an encrypted file collection C, the trace of $Tr(H_q)$ captures the precise information to be learned by cloud server, including: the size of the encrypted files $\{|F_1|, \ldots, |F_n|\}$; the outcome of each search, $\{FID_{w_i}\}_{ed(w_i, w^j) \leq d}$ for $1 \leq j \leq q$; and the pattern $\prod_q$ for each search. Here $\prod_q$ is a symmetric matrix where the entry $\prod_q[i,j]$ stores the intersection $\{T_{w'}\}_{w' \in S_{w^i,d} \cap S_{w^j,d}}$.

Security Analysis

Security Analysis

Security Strength

Given two histories with the identical trace, the cloud server is not able to distinguish the views of the two histories.
In other words, the cloud server cannot extract additional knowledge beyond the information we are willing to leak (i.e., the trace) and thus our mechanism is secure.

Security Analysis

Security Strength

Given two histories with the identical trace, the cloud server is not able to distinguish the views of the two histories.
In other words, the cloud server cannot extract additional knowledge beyond the information we are willing to leak (i.e., the trace) and thus our mechanism is secure.

Theorem

Our similarity keyword search schemes meet the non-adaptive semantic security.

Security Analysis

Security Strength

Given two histories with the identical trace, the cloud server is not able to distinguish the views of the two histories.
In other words, the cloud server cannot extract additional knowledge beyond the information we are willing to leak (i.e., the trace) and thus our mechanism is secure.

Theorem

Our similarity keyword search schemes meet the non-adaptive semantic security.

Due to space limitation, we only give the proof for the basic approach. The proof of other schemes follow similarly.

Proof

Definition (Simulator $\mathcal{S}$)

Given $Tr(H_q)$, it can simulate a view V_q^* indistinguishable from cloud server's view $V_K(H_q)$ with probability negligibly close to 1, for any $q \in \mathbb{N}$, any H_q and randomly chosen K.

Proof

Definition (Simulator $\mathcal{S}$)

Given $Tr(H_q)$, it can simulate a view V_q^* indistinguishable from cloud server's view $V_K(H_q)$ with probability negligibly close to 1, for any $q \in \mathbb{N}$, any H_q and randomly chosen K.

- l: the security parameter of the RBF $f(key, \cdot)$ (output length)

Proof

Definition (Simulator $\mathcal{S}$)

Given $Tr(H_q)$, it can simulate a view V_q^* indistinguishable from cloud server's view $V_K(H_q)$ with probability negligibly close to 1, for any $q \in \mathbb{N}$, any H_q and randomly chosen K.

- l: the security parameter of the RBF $f(key, \cdot)$ (output length)
- $\tau = \max \{|S_{w_i,d}|\}_{w_i \in W}$

Proof

Definition (Simulator $\mathcal{S}$)

Given $Tr(H_q)$, it can simulate a view V_q^* indistinguishable from cloud server's view $V_K(H_q)$ with probability negligibly close to 1, for any $q \in \mathbb{N}$, any H_q and randomly chosen K.

- l: the security parameter of the RBF $f(key, \cdot)$ (output length)
- $\tau = \max\{|S_{w_i,d}|\}_{w_i \in W}$
- $|W|$, and size of padded FID_{w_i} are known to $\mathcal{S}$

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
  $T^*[i]$ and $C^*[i]$: the i-th row entry in T^* and C^*.

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
  $T^*[i]$ and $C^*[i]$: the i-th row entry in T^* and C^*.
  - To generate T^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $t_i^* \in \{0,1\}^l$, and sets $T^*[i] = t_i^*$.

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
  $T^*[i]$ and $C^*[i]$: the i-th row entry in T^* and C^*.
  - To generate T^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $t_i^* \in \{0,1\}^l$, and sets $T^*[i] = t_i^*$.
  - To generate C^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $c_i^* \in \{0,1\}^{|FID_{w_i}|}$, and sets $C^*[i] = c_i^*$.

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
  $T^*[i]$ and $C^*[i]$: the i-th row entry in T^* and C^*.
  - To generate T^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $t_i^* \in \{0,1\}^l$, and sets $T^*[i] = t_i^*$.
  - To generate C^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $c_i^* \in \{0,1\}^{|FID_{w_i}|}$, and sets $C^*[i] = c_i^*$.
- Due to the semantic security of the symmetric encryption, no probabilistic polynomial-time (P.P.T.) adversary can distinguish between e_i and e_i^*, or between $Enc(sk_{w_i'}, FID_{w_i})$ and c_i.

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
  $T^*[i]$ and $C^*[i]$: the i-th row entry in T^* and C^*.
  - To generate T^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $t_i^* \in \{0,1\}^l$, and sets $T^*[i] = t_i^*$.
  - To generate C^*,for $1 \leq i \leq \tau\,|W|$, $\mathcal{S}$ selects a random $c_i^* \in \{0,1\}^{|FID_{w_i}|}$, and sets $C^*[i] = c_i^*$.
- Due to the semantic security of the symmetric encryption, no probabilistic polynomial-time (P.P.T.) adversary can distinguish between e_i and e_i^*, or between $Enc(sk_{w_i'}, FID_{w_i})$ and c_i.
- Also due to the pseudo-randomness of the trapdoor function, no P.P.T. adversary can distinguish between $f(x, w_i')$ and a random string t_i^*.

Proof

- For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \le i \le n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
  $T^*[i]$ and $C^*[i]$: the i-th row entry in T^* and C^*.
  - To generate T^*,for $1 \le i \le \tau\,|W|$, $\mathcal{S}$ selects a random $t_i^* \in \{0,1\}^l$, and sets $T^*[i] = t_i^*$.
  - To generate C^*,for $1 \le i \le \tau\,|W|$, $\mathcal{S}$ selects a random $c_i^* \in \{0,1\}^{|FID_{w_i}|}$, and sets $C^*[i] = c_i^*$.
- Due to the semantic security of the symmetric encryption, no probabilistic polynomial-time (P.P.T.) adversary can distinguish between e_i and e_i^*, or between $Enc(sk_{w_i'}, FID_{w_i})$ and c_i.
- Also due to the pseudo-randomness of the trapdoor function, no P.P.T. adversary can distinguish between $f(x, w_i')$ and a random string t_i^*.
- Thus, $V_K(H_0)$and V_0^* are indistinguishable.

Proof

- For $q \geq 1$, $\mathcal{S}$ builds
  $V_q^* = \left\{ e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*, \left\{ T_{w'}^* \right\}_{w' \in \left\{ S_{w^1,d}, \ldots, S_{w^q,d} \right\}} \right\}$
  - e_i^* is still randomly drawn from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.

Proof

- For $q \geq 1$, $\mathcal{S}$ builds
  $V_q^* = \left\{ e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*, \left\{ T_{w'}^* \right\}_{w' \in \left\{ S_{w^1,d}, \ldots, S_{w^q,d} \right\}} \right\}$
  - e_i^* is still randomly drawn from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.

Proof

- For $q \geq 1$, $\mathcal{S}$ builds
  $V_q^* = \left\{ e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*, \left\{ T_{w'}^* \right\}_{w' \in \left\{ S_{w^1,d}, \ldots, S_{w^q,d} \right\}} \right\}$
  - e_i^* is still randomly drawn from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
- the result $\{FID_{w_i}\}_{ed(w_i, w^j) \leq d}$ for the j-th search request is the union of file identifiers in the matched entries of the index

Proof

- For $q \geq 1$, $\mathcal{S}$ builds $V_q^* = \left\{ e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*, \{T_{w'}^*\}_{w' \in \{S_{w^1,d}, \ldots, S_{w^q,d}\}} \right\}$
  - e_i^* is still randomly drawn from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
- the result $\{FID_{w_i}\}_{ed(w_i,w^j) \leq d}$ for the j-th search request is the union of file identifiers in the matched entries of the index
  - $\{FID_{w_i}\}_{ed(w_i,w^j) \leq d}$ can be rewritten as $\bigcup_{k=1}^{\alpha_j} FID_{w_{j,k}}$

Proof

- For $q \geq 1$, $\mathcal{S}$ builds
  $V_q^* = \left\{ e_1^*, e_2^*, \ldots, e_n^*, \mathcal{I}^*, \left\{ T_{w'}^* \right\}_{w' \in \left\{ S_{w^1,d}, \ldots, S_{w^q,d} \right\}} \right\}$
  - e_i^* is still randomly drawn from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$.
- Let $\mathcal{I}^* = (T^*, C^*)$.
- the result $\{FID_{w_i}\}_{ed(w_i,w^j) \leq d}$ for the j-th search request is the union of file identifiers in the matched entries of the index
  - $\{FID_{w_i}\}_{ed(w_i,w^j) \leq d}$ can be rewritten as $\bigcup_{k=1}^{\alpha_j} FID_{w_{j,k}}$
  - α_j denotes the number of matches for the j-th search

Proof

- To build the first trapdoor set $\left\{T^*_{w'}\right\}_{w' \in \left\{S_{w^1,d}\right\}}$ for search input w^1,the simulator $\mathcal{S}$ does the following:

Proof

- To build the first trapdoor set $\left\{T^*_{w'}\right\}_{w' \in \left\{S_{w^1,d}\right\}}$ for search input w^1,the simulator $\mathcal{S}$ does the following:
  - Select α_1 random strings $t^*_{1,1}, \ldots, t^*_{1,\alpha_1} \in \{0,1\}^l$ and set them to α_1 non-assigned entries $T^*[i_{1,1}], \ldots, T^*[i_{1,\alpha_1}]$.

Proof

- To build the first trapdoor set $\left\{T^*_{w'}\right\}_{w' \in \left\{S_{w^1,d}\right\}}$ for search input w^1,the simulator $\mathcal{S}$ does the following:
  - Select α_1 random strings $t^*_{1,1}, \ldots, t^*_{1,\alpha_1} \in \{0,1\}^l$ and set them to α_1 non-assigned entries $T^*[i_{1,1}], \ldots, T^*[i_{1,\alpha_1}]$.
  - Select α_1 random strings $\rho^*_{1,1}, \ldots \rho^*_{1,\alpha_i} \in \{0,1\}^l$ and set $C^*[i_{1,k}] = Enc(\rho^*_{1,k}, FID_{w_i,k})$ for $1 \leq k \leq \alpha_1$

Proof

- To build the first trapdoor set $\left\{T^*_{w'}\right\}_{w' \in \left\{S_{w^1,d}\right\}}$ for search input w^1,the simulator $\mathcal{S}$ does the following:
  - Select α_1 random strings $t^*_{1,1}, \ldots, t^*_{1,\alpha_1} \in \{0,1\}^l$ and set them to α_1 non-assigned entries $T^*[i_{1,1}], \ldots, T^*[i_{1,\alpha_1}]$.
  - Select α_1 random strings $\rho^*_{1,1}, \ldots \rho^*_{1,\alpha_i} \in \{0,1\}^l$ and set $C^*[i_{1,k}] = Enc(\rho^*_{1,k}, FID_{w_i,k})$ for $1 \leq k \leq \alpha_1$
  - Set remaining $\tau - \alpha_1$ trapdoors as random value pairs $(t^*_{1,k}, \rho^*_{1,k}) \in \{0,1\}^l \times \{0,1\}^l$ for $\alpha_1 \leq k \leq \tau$.

Proof

- To build the first trapdoor set $\left\{T^*_{w'}\right\}_{w' \in \left\{S_{w^1,d}\right\}}$ for search input w^1,the simulator $\mathcal{S}$ does the following:
  - Select α_1 random strings $t^*_{1,1}, \ldots, t^*_{1,\alpha_1} \in \{0,1\}^l$ and set them to α_1 non-assigned entries $T^*[i_{1,1}], \ldots, T^*[i_{1,\alpha_1}]$.
  - Select α_1 random strings $\rho^*_{1,1}, \ldots \rho^*_{1,\alpha_i} \in \{0,1\}^l$ and set $C^*[i_{1,k}] = Enc(\rho^*_{1,k}, FID_{w_i,k})$ for $1 \leq k \leq \alpha_1$
  - Set remaining $\tau - \alpha_1$ trapdoors as random value pairs $(t^*_{1,k}, \rho^*_{1,k}) \in \{0,1\}^l \times \{0,1\}^l$ for $\alpha_1 \leq k \leq \tau$.
- For trapdoor simulation of w^j for $2 \leq j \leq q$, if $\left|\prod_q[i,j]\right| = 0$ for all $i < j$, the simulator repeats the same process as simulating trapdoors for w^1.

Proof

- To build the first trapdoor set $\left\{T^*_{w'}\right\}_{w' \in \left\{S_{w^1,d}\right\}}$ for search input w^1,the simulator $\mathcal{S}$ does the following:
  - Select α_1 random strings $t^*_{1,1}, \ldots, t^*_{1,\alpha_1} \in \{0,1\}^l$ and set them to α_1 non-assigned entries $T^*[i_{1,1}], \ldots, T^*[i_{1,\alpha_1}]$.
  - Select α_1 random strings $\rho^*_{1,1}, \ldots \rho^*_{1,\alpha_i} \in \{0,1\}^l$ and set $C^*[i_{1,k}] = Enc(\rho^*_{1,k}, FID_{w_i,k})$ for $1 \leq k \leq \alpha_1$
  - Set remaining $\tau - \alpha_1$ trapdoors as random value pairs $(t^*_{1,k}, \rho^*_{1,k}) \in \{0,1\}^l \times \{0,1\}^l$ for $\alpha_1 \leq k \leq \tau$.
- For trapdoor simulation of w^j for $2 \leq j \leq q$, if $\left|\prod_q[i,j]\right| = 0$ for all $i < j$, the simulator repeats the same process as simulating trapdoors for w^1.
- Otherwise, let β_j denotes the number of file identifier list in $\left\{FID_{w_j,k}\right\}_{1 \leq k \leq \alpha_j}$, which have been assigned already in $\left\{FID_{w_i,k}\right\}_{1 \leq k \leq \alpha_i, i<j}$.

Proof

Next, $\mathcal{S}$ does:

Proof

Next, $\mathcal{S}$ does:

- Choose β_j trapdoors from existing $\left\{ T^*_{w'} \right\}_{w' \in \left\{ s_{w^i,d} \right\}, i<j}$ that match to the common β_j file identifier list of $\left\{ FID_{w_j,k} \right\}_{1 \leq k \leq \alpha_j} \cap (\bigcup_{i=1}^{j-1} \{FID_{w_i,k}\}_{1 \leq k \leq \alpha_i})$, and assign them to trapdoor simulation of w^j.

Proof

Next, $\mathcal{S}$ does:

- Choose β_j trapdoors from existing $\left\{T^*_{w'}\right\}_{w' \in \left\{s_{w^i,d}\right\}, i<j}$ that match to the common β_j file identifier list of $\left\{FID_{w_j,k}\right\}_{1 \le k \le \alpha_j} \cap \left(\bigcup_{i=1}^{j-1} \left\{FID_{w_i,k}\right\}_{1 \le k \le \alpha_i}\right)$, and assign them to trapdoor simulation of w^j.
- If $\alpha_j > \beta_j$, the simulator $\mathcal{S}$ builds $\alpha_j - \beta_j$ entries in $\mathcal{I}^*$ via the same process as simulating trapdoors for w^1.

Proof

Next, $\mathcal{S}$ does:

- Choose β_j trapdoors from existing $\left\{T^*_{w'}\right\}_{w' \in \left\{s_{w^i,d}\right\}, i<j}$ that match to the common β_j file identifier list of $\left\{FID_{w_j,k}\right\}_{1 \le k \le \alpha_j} \cap \left(\bigcup_{i=1}^{j-1} \left\{FID_{w_i,k}\right\}_{1 \le k \le \alpha_i}\right)$, and assign them to trapdoor simulation of w^j.
- If $\alpha_j > \beta_j$, the simulator $\mathcal{S}$ builds $\alpha_j - \beta_j$ entries in $\mathcal{I}^*$ via the same process as simulating trapdoors for w^1.
- $\mathcal{S}$ further checks $\prod_q[i,j]$ for $i < j$, finds from already generated $\left\{T^*_{w'}\right\}_{w' \in \left\{s_{w^i,d}\right\}, i<j}$ the common γ_j trapdoors that do not have matched entries in index $\mathcal{I}^*$, and assigns them to the current trapdoor simulation of w^j.

Proof

Next, $\mathcal{S}$ does:

- Choose β_j trapdoors from existing $\left\{T^*_{w'}\right\}_{w' \in \left\{s_{w^i,d}\right\}, i<j}$ that match to the common β_j file identifier list of $\left\{FID_{w_j,k}\right\}_{1 \le k \le \alpha_j} \cap (\bigcup_{i=1}^{j-1} \left\{FID_{w_i,k}\right\}_{1 \le k \le \alpha_i})$, and assign them to trapdoor simulation of w^j.
- If $\alpha_j > \beta_j$, the simulator $\mathcal{S}$ builds $\alpha_j - \beta_j$ entries in $\mathcal{I}^*$ via the same process as simulating trapdoors for w^1.
- $\mathcal{S}$ further checks $\prod_q[i,j]$ for $i < j$, finds from already generated $\left\{T^*_{w'}\right\}_{w' \in \left\{s_{w^i,d}\right\}, i<j}$ the common γ_j trapdoors that do not have matched entries in index $\mathcal{I}^*$, and assigns them to the current trapdoor simulation of w^j.
- Set remaining $\tau - \alpha_j - \gamma_j$ trapdoors as random value pairs from $\{0,1\}^l \times \{0,1\}^l$.

proof

Thus

proof

Thus

Proof.

$\square$

Thus

Proof.

- The correctness of the constructed view is easy to demonstrate by searching on $\mathcal{I}^*$ via $\left\{T^*_{w'}\right\}_{w' \in s_{w^i,d}}$ for each i.

$\square$

proof

Thus

Proof.

- The correctness of the constructed view is easy to demonstrate by searching on $\mathcal{I}^*$ via $\left\{T^*_{w'}\right\}_{w' \in s_{w^i,d}}$ for each i.
- There is no P.P.T. adversary can distinguish between V^*_q and $V_K(H_q)$

□

proof

Thus

Proof.

- The correctness of the constructed view is easy to demonstrate by searching on $\mathcal{I}^*$ via $\left\{T^*_{w'}\right\}_{w' \in s_{w^i,d}}$ for each i.
- There is no P.P.T. adversary can distinguish between V^*_q and $V_K(H_q)$
- In particular, the simulated encrypted ciphertext is indistinguishable due to the semantic security of the symmetric encryption.

$\square$

proof

Thus

Proof.

- The correctness of the constructed view is easy to demonstrate by searching on $\mathcal{I}^*$ via $\left\{T^*_{w'}\right\}_{w' \in s_{w^i,d}}$ for each i.
- There is no P.P.T. adversary can distinguish between V^*_q and $V_K(H_q)$
- In particular, the simulated encrypted ciphertext is indistinguishable due to the semantic security of the symmetric encryption.
- The indistinguishability of index and trapdoors is based on the indistinguishability of the pseudorandom function output and a random string.

□

	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

Experiment Design

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

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

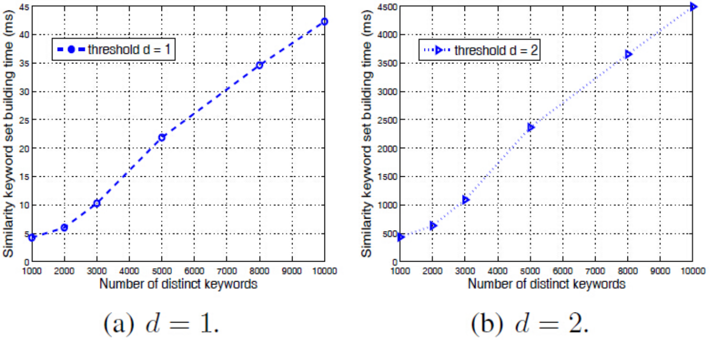


Figure: Similarity set construction time using wildcard-based approach with different choices of edit distance *d*

Cost for Generating Similarity Keyword Set

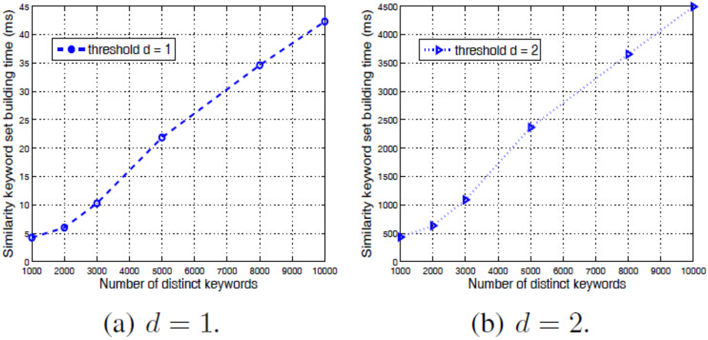
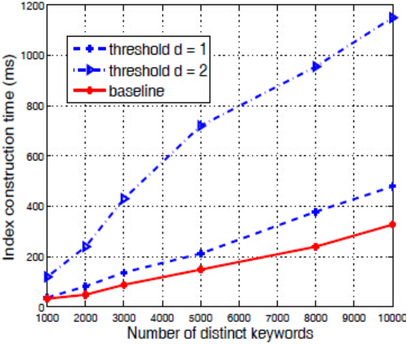


Figure: Similarity set construction time using wildcard-based approach with different choices of edit distance d

The construction time increases linearly with the number of keywords.

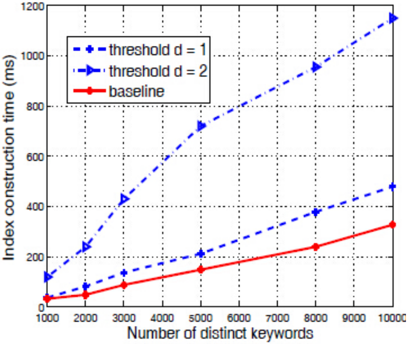
Cost For Building Searchable Index



(a) Index construction time.

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

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.

Cost For Building Searchable Index

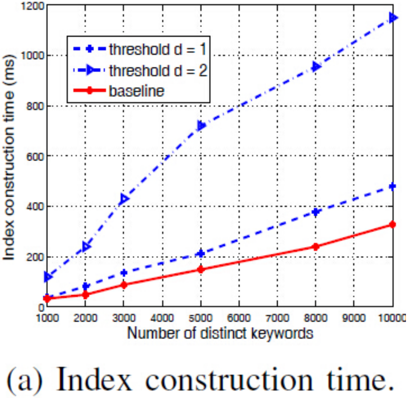
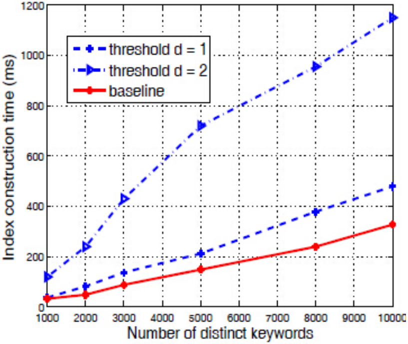


Figure: Time cost for searchable index construction with different choices of edit distance *d*

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

Cost For Building Searchable Index



(a) Index construction time.

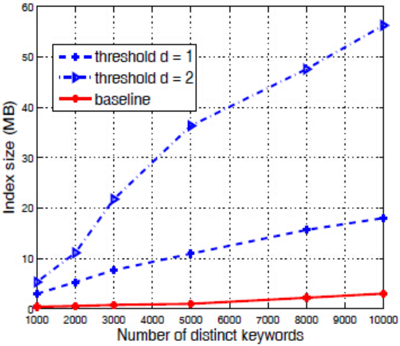
Figure: Time cost for searchable index construction with different choices of edit distance *d*

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.

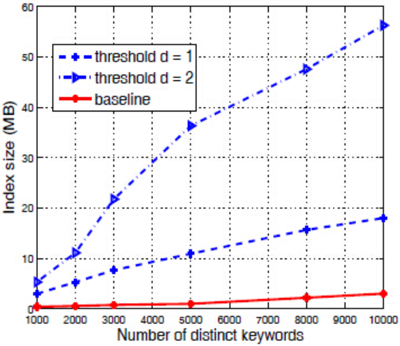
Cost For Building Searchable Index



(b) Index storage size.

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

Cost For Building Searchable Index

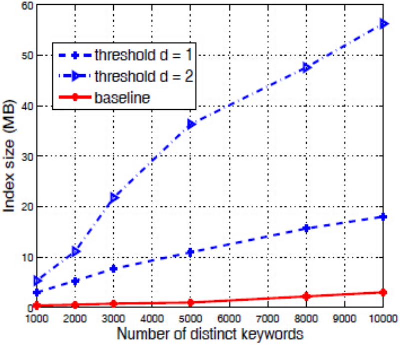


(b) Index storage size.

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

Cost For Building Searchable Index



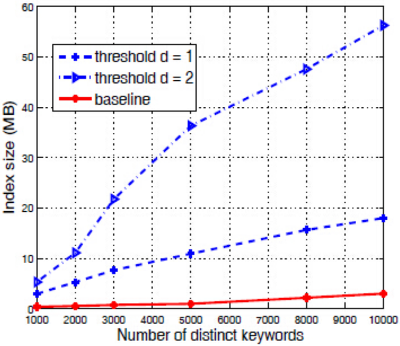
(b) Index storage size.

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

Cost For Building Searchable Index



(b) Index storage size.

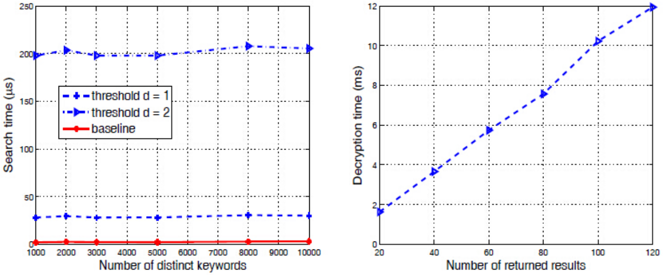
Figure: Storage cost for searchable index construction with different choices of edit distance *d*

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

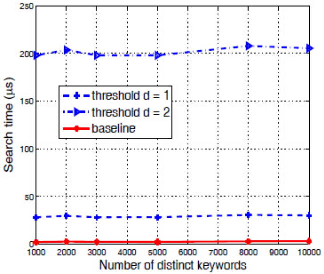
"Average keyword length" also sightly influence the time and space cost of building searchable index.

Cost For Searching the Index

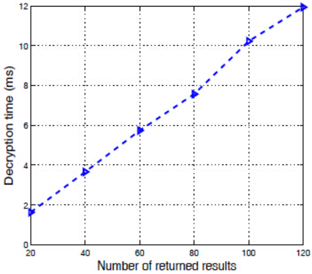


(a) Cloud side search time. (b) User side decryption cost.

Cost For Searching the Index



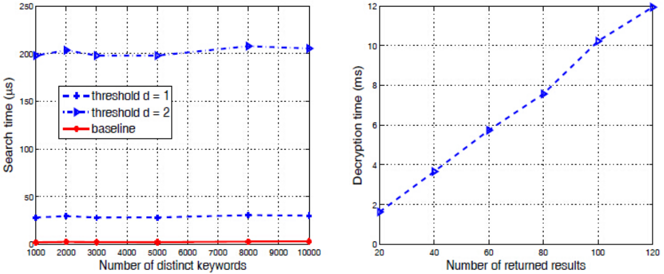
(a) Cloud side search time.



(b) User side decryption cost.

The proposed mechanism cost constant search time.

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