Backward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates

Hyung Tae Lee

Chonbuk National University, Republic of Korea

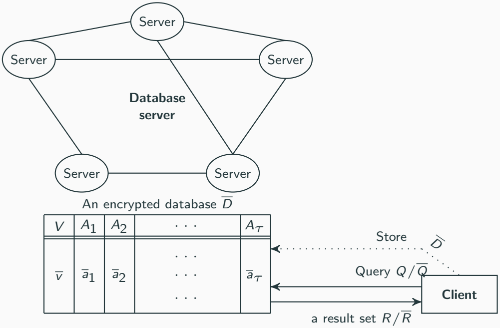
June 14, 2019@ Workshop on Modern Trends in Cryptography

Table of contents

Introduction

Motivations (from the Viewpoint of My Research Direction)

- Private keyword search on encrypted data is one of key factors for secure cloud computing.



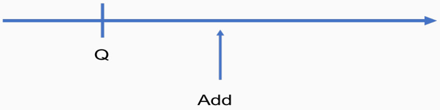
- (Fully) Homomorphic encryption may resolve this problem, but it is not practical yet:
  - 1.5 ms for equality test of two encrypted 64-bit integers in amortized time
  - 29 ms for comparison of two encrypted 64-bit integers in amortized time
- Searchable encryption may be another candidate for providing a practical solution to this problem. ($1 - 10\mu s/1$ keword search on 1 data)

Dynamic Searchable Symmetric Encryption

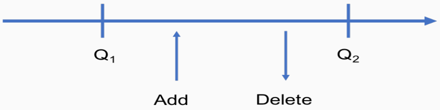
- A dynamic searchable symmetric encryption (DSSE) is an encryption scheme that
  - allows data updates (addition/deletion),
  - supports comparisons between an encrypted keyword and a tag.
- Consist of the following algorithms
  - $\mathsf{Setup}(1^\lambda) \to (\mathsf{ST}, \mathsf{msk}; \mathsf{EDB})$
  - $\mathsf{Addition}(\mathsf{msk}, \mathsf{ST}, f; \mathsf{EDB}) \to (\mathsf{ST}'; \mathsf{EDB}')$
  - $\mathsf{Deletion}(\mathsf{msk}, \mathsf{ST}, f; \mathsf{EDB}) \to (\mathsf{ST}'; \mathsf{EDB}')$
  - $\mathsf{Search}(\mathsf{msk}, \mathsf{ST}, w; \mathsf{EDB}) \to (\mathsf{ST}'; I)$

Additional Security Requirements in the Dynamic Setting

- Forward security: No leak any information about newly added data against an adversary who has information for previous queries



- Backward security: No leak any information about the data added and deleted between two successive queries on the same keyword

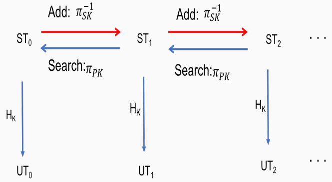


Background: Previous Results for Forward/Backward DSSE

Previous Results I: Sophos ($\Sigma o\phi o\sigma$)

- Bost presented a DSSE achieving forward security. (ACM CCS 2016)
- Exploited a trapdoor permutation π to update an encryption key after search queries

1. Compute and use $\mathsf{ST}_{i+1} = \pi_{SK}(\mathsf{ST}_i)$ to add a new data
2. Give ST_c as a token for keyword search



- Realized a trapdoor permutation using RSA, but it is too expensive.

Previous Results II: Janus

- Bost, Minaud and Ohrimenko formalized a backward secure DSSE and presented a generic construction of a backward secure DSSE using a forward secure DSSE. (ACM CCS 2017)
- Design strategy
  - Assume there is a forward secure DSSE and a secure puncturable encryption (PE).
  - Operate two databases $\mathrm{EDB}_{\mathrm{add}}$ and $\mathrm{EDB}_{\mathrm{del}}$ for addition and deletion, respectively
  - Tag: an output of a function evaluating at index and keyword
  - Store a ciphertext $(ct, t) = \mathrm{Enc}(pk, ind, t)$ for PE at $\mathrm{EDB}_{\mathrm{add}}$
  - Store a punctured key at the tag t at $\mathrm{EDB}_{\mathrm{del}}$
- Forward security: Achieved by updating an encryption key for PE
- Backward security: Achieved by using PE
- They realized PE by employing the Green and Miers scheme, but it is quite expensive.

Previous Results III: Janus++

- Sun et al. proposed a new symmetric PE (SPE) from using puncturable PRF.
- They proposed Janus++, the improved version of Janus, using the proposed SPE. (ACM CCS 2018)
- The proposed SPE PE $= (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Punc})$
  - $\mathsf{KG}(1^\lambda, d)$: Choose a random key sk_0 and set $\mathsf{msk} = (sk_0, d)$.
  - $\mathsf{Enc}(\mathsf{msk}, m, t)$:
    1. Compute $sk_i = H(sk_{i-1}, i)$ for all $1 \le i \le d$.
    2. Compute $k = \oplus_{i=0}^{d} F(sk_i, t)$ and $ct = \mathsf{SE.Enc}(k, m)$.
    3. Output (ct, t).
  - $\mathsf{Punc}(SK_{i-1}, t_i')$:
    1. Compute $psk_i = \mathsf{F.Punc}(sk_{i-1}, t_i')$ and $sk_i = H(sk_{i-1}, i)$.
    2. Set $msk_i = (sk_i, d)$.
    3. Output $SK_i = (msk_i, psk_1, \cdots, psk_i)$.
  - $\mathsf{Dec}(SK_i, ct, t)$:
    1. If $i < d$, compute $sk_\ell = H(sk_{\ell-1}, \ell)$ for $i \le \ell \le d$.
    2. Evaluate $k' = \oplus_{s=1}^{i} \mathsf{F.Eval}(psk_s, t) \oplus_{\ell=i}^{d} F(sk_\ell, t)$.
    3. Recover $m' = \mathsf{SE.Dec}(k', ct)$.

Previous Results IV: Dual Dictionary and Forward Secure DSSE

- Kim et al. proposed a new database structure, dual dictionary, which can be seen as a combination of forward and inverted indexes. (ACM CCS 2017).

id_1	w_1	w_3	w_5	$\cdots$
id_2	w_1	w_2	w_3	$\cdots$
id_3	w_3	w_7	w_8	$\cdots$
$\vdots$		$\vdots$		

⟨ Forward index ⟩

- Pros: Addition/Deletion
- Cons: Keyword search

w_1	id_2	id_2		$\cdots$
w_2	id_1		$\cdots$	
w_3	id_1	id_2	id_3	$\cdots$
$\vdots$		$\vdots$		

⟨ Inverted index ⟩

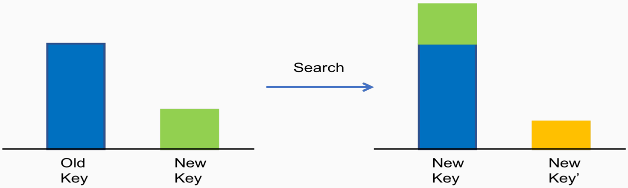
- Pros: Keyword search
- Cons: Addition/Deletion

Label 1	Label 2	Value
$\mathrm{H}(T_{id_1}, 1)$	$\mathrm{H}(T_{w_1}, 1)$	$\mathrm{Enc}(T_{w_1}, id_1)$
	$\vdots$	
$\mathrm{H}(T_{id_3}, 1)$	$\mathrm{H}(T_{w_7}, 1)$	$\mathrm{Enc}(T_{w_7}, id_3)$
	$\vdots$	

⟨ Dual dictionary⟩

Previous Results IV: Dual Dictionary and Forward Secure DSSE (Cont.)

- On top of the dual dictionary structure, they presented a new forward secure DSSE by updating an encryption key after search queries.



- There is no known result about backward secure DSSE using the dual dictionary.
- A naive application of Janus/Janus++ does not take an advantage of the dual dictionary.

Our Construction

Overview of Our Construction

- Goal: Design a new backward secure DSSE using the dual dictionary structure
- Conclusion: Achieve the goal by using small modifications of Janus/Janus++
- Building blocks
  - Dual dictionary structure
  - Puncturable encryption (PE)
- How to achieve forward/backward security
  - Forward security: Update an encryption key for PE after search queries
  - Backward security: Use a puncturable encryption

Concrete Description I: Setup

Algorithm 1 Setup(1^λ)$\rightarrow$ (ST; EDB)

Notation. F, G, H: PRFs with key k_F, k'_F, k_G, k_H

PE := (KG, Enc, Dec, Punc): puncturable encryption

$\mathsf{Idx} : \{0,1\}^* \rightarrow \mathcal{I}$ for an index set $\mathcal{I}$

1: EDB $\leftarrow \emptyset$ /* encrypted DB */

2: ST $\leftarrow \emptyset$ /* private state */

3: msk $\leftarrow (k_F, k_G, k_H)$ /* master key */

- F: To generate keys for computing label function H
- G: To generate a tag of index and keyword for PE
- H: To compute labels for index and keyword, respectively

Concrete Description II-1: Addition @ Client

Algorithm 2 Addition$(\mathsf{msk}, \mathsf{ST}, f; \mathsf{EDB}) \rightarrow (\mathsf{ST}'; \mathsf{EDB}')$ @ Client

Require: $\mathsf{msk}, f, \mathsf{ST}$

Ensure: $\mathsf{ST}', \mathbf{D}$

$id \leftarrow \mathsf{Idx}(f)$ and $k_{id} \leftarrow F(k_F, id)$

$cnt_{id} \leftarrow 0, \mathbf{D} \leftarrow \emptyset$

$W \leftarrow \{w | w \in f\} = \{w_1, \ldots, w_n\}$

while $W \neq \emptyset$ **do**

  $w \xleftarrow{\$} W$ and $k_w \leftarrow F(k'_F, w)$

  $W \leftarrow W \backslash \{w\}$

  if $\mathsf{ST}[w].\mathsf{Kwd} = \perp$ **then**

    $(\mathrm{EK}_w, \mathrm{SK}_w^{(0)}) \leftarrow \mathsf{PE.KG}(1^\lambda, d)$

    $ucnt_w \leftarrow 0$

    $\mathsf{ST} \leftarrow (w, \mathrm{EK}_w, ucnt_w, (\mathrm{SK}_w^{(0)}, t_0))$

  else

    $(\mathrm{EK}_w, ucnt_w) \leftarrow \mathsf{ST}[w].(\mathsf{Ekey}, \mathsf{Ucnt})$

  $cnt_{id} \leftarrow cnt_{id} + 1, ucnt_w \leftarrow ucnt_w + 1$

  $\mathsf{label}_{id} \leftarrow H(k_{id}, cnt_{id}), \mathsf{label}_w \leftarrow H(k_w, ucnt_w)$

  $t_{w,id} \leftarrow G(k_G, w \parallel id), \mathsf{pval}_w^{(id)} \leftarrow \mathsf{PE.Enc}(\mathrm{EK}_w, id, t_{w,id})$

  $\mathbf{D} \leftarrow \mathbf{D} \cup \{(\mathsf{label}_{id}, \mathsf{label}_w, \mathsf{pval}_w^{(id)})\}$

  $\mathsf{ST}[w].\mathsf{Ucnt} \leftarrow ucnt_w$

Send $\mathbf{D}$ to the server

Dual dictionary:

$\mathsf{Label}_{\mathsf{id}}$	Label_w	pval

Concrete Description II-2: Addition @ Server

Algorithm 3 Addition($\mathsf{msk}, \mathsf{ST}, f; \mathsf{EDB}$) $\rightarrow$ ($\mathsf{ST}'; \mathsf{EDB}'$)@Server

Require: $\mathbf{D}, \mathsf{EDB}$

Ensure: EDB'

1: $\mathsf{EDB}' \leftarrow \mathsf{EDB}$

2: **for** each $(\mathsf{label}_{id}, \mathsf{label}_w, \mathsf{pval}_w^{(id)}) \in \mathbf{D}$ **do**

3: $\quad \mathsf{EDB}' \leftarrow (\mathsf{label}_{id}, \mathsf{label}_w, \mathsf{pval}_w^{(id)})$

Concrete Description III-1: Deletion @Client

Algorithm 4 Deletion(msk, ST, f; EDB) $\rightarrow$ (ST$'$; EDB$'$) @Client

Require: msk, f, ST

Ensure: k_{id}, ST$'$

$id \leftarrow \mathsf{Idx}(f)$ and $k_{id} \leftarrow F(k_F, id)$

Send k_{id} to the server

$W \leftarrow \{w | w \in f\} = \{w_1, \ldots, w_n\}$

for each $w \in W$ **do**

  $t_{w,id} \leftarrow G(k_G, w \parallel id)$

  $\mathrm{SK}_w^{(i-1)} \leftarrow \mathsf{ST}[w].\mathsf{Psk}$, $\mathrm{SK}_w^{(i)} \leftarrow \mathsf{PE.Punc}(\mathrm{SK}_w^{(i-1)}, t_{w,id})$

  $\mathsf{ST}[w].\mathsf{Psk} \leftarrow \mathrm{SK}_w^{(i)}$, $\mathsf{ST}[w].\mathsf{Tag} \leftarrow \mathsf{ST}[w].\mathsf{Tag} \cup \{t_{w,id}\}$

Concrete Description III-2: Deletion @Server

Algorithm 5 Deletion(msk, ST, f; EDB) $\rightarrow$ (ST$'$; EDB$'$) @Server

Require: k_{id}, EDB

Ensure: EDB$'$

$cnt_{id} \leftarrow 1$
while true **do**
    label$_{id} \leftarrow H(k_{id}, cnt_{id})$
    if EDB[label$_{id}$] $\neq \perp$ **then**
        EDB $\leftarrow$ EDB$\setminus\{$EDB[label$_{id}$]$\}$
        $cnt_{id} \leftarrow cnt_{id} + 1$
    else return EDB
EDB$' \leftarrow$ EDB

Label$_{\text{id}}$	Label$_w$	Pval
	$\vdots$	
$\cancel{\text{H}(k_{id_3}, 1)}$	$\cancel{\text{H}(k_{w_7}, 1)}$	$\cancel{\text{PE.Enc}(\cdots)}$
	$\vdots$	

$\langle$ When deleting id_3 $\rangle$

Concrete Description IV-1: Search @Client

Algorithm 6 Search(msk, ST, w; EDB) $\rightarrow$ (ST$'$; I) @ Client

Require: msk, w, ST

Ensure: ST$'$, τ_w

if ST[w].Kwd $= \perp$ **then return** $\emptyset$

$k_w \leftarrow F(k'_F, w)$

$(ucnt_w, (\mathrm{SK}_w, T_w)) \leftarrow$ ST[w].(Ucnt, Psk, Tag)

$\tau_w \leftarrow (\mathrm{SK}_w, T_w, k_w, ucnt_w)$

Send τ_w to the server

$(\mathrm{EK}'_w, (\mathrm{SK}_w{}', t'_0)) \leftarrow$ PE.KG$(1^\lambda, d)$

$ucnt_w \leftarrow 0$

ST $\leftarrow (w, \mathrm{EK}'_w, ucnt_w, (\mathrm{SK}'_w, t'_0))$

Concrete Description IV-2: Search @Server

Algorithm 7 Search(msk, ST, w; EDB) $\rightarrow$ (ST$'$; I) @Server

Require: τ_w, EDB
Ensure: $I \subset \mathcal{I}$, EDB$'$

```
I ← ∅, i ← 1
while i ≤ ucnt_w do
    label_w ← H(k_w, i)
    if EDB[label_w] ≠ ⊥ then
        pval_w = (ct_w, t_w) ← EDB[label_w].Pval
        id ← PE.Dec(SK_w, ct_w, t_w)
        NewR ← NewR ∪ {(id, t_w)}
    i ← i + 1
OldR ← EDB_cache[k_w]
OldR ← OldR \ {(id, t_w) | (id, t_w) ∈ OldR ∧ t_w ∈ T_w}
Res ← NewR ∪ OldR
EDB_cache[k_w] ← Res
Send Res to the client
```

Label$_{\text{id}}$	Label$_w$	Pval
H(k_{id_1}, 1)	H(k_{w_1}, 1)	PE.Enc($\cdots$)
	$\vdots$	

Comparison

- Computational complexity

	Janus++		Ours	
Search	$O(a_w)$	PE.Dec	$O(a_w - d_w)$	PE.Dec
Add	$O(N)$	PE.Enc	$O(N)$	PE.Enc
Delete	$O(d_{id})$	PE.Punc	$O(d_{id})$	PE.Punc

a_w: the number of added documents that include keyword w

d_w: the number of deleted documents that include keyword w

N: the total number of document/keyword pairs

d_{id}: the number of keywords in the document to be deleted

- Storage

	Janus++	Ours
Client	$O(W)$	$O(W)$
Server	$O(\Sigma_w(a_w + d_w))$	$O(\Sigma_w(a_w - d_w))$

W: the total number of keywords in DB

Conclusion & Future Works

Conclusion & Future Works

- We provided a new backward secure DSSE under the dual dictionary structure by adjusting Janus/Janus++.
- In terms of theoretical efficiency analysis, the search time/the storage of ours are faster/smaller than those of the original Janus++, respectively, but the advantage is incremental.
- Need implementations for our proposed construction to compare more accurate efficiency in practice
- The maximum number of punctured points is restricted in Sun et al.'s SPE. Can we remove this limitation?

Thanks for your attention!

&

Question?