汇报人: 刘美涵、温晴

Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates

Kee Sung Kim, Minkyu Kim, Dongsoo Lee,
Je Hong Park and Woo-Hwan Kim.

CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, October 2017 Pages 1449–1463

Content

- Introduction
- Scheme
- Realization
- Security Proof
- Experiment
- Discussion

Introduction

- Dynamic
  - variable number of document/keyword pairs
  - superior scalability
- Forward Security
  - prevent file-injection attacks
- Effective Updates

Introduction

Previous Work

- inefficient
- no actual deletion
- high complexity
- ......

This Work

- Dual Dictionary
  - optimal search(the inverted index for searching, the forward index for updating)
  - actual deletion
- Fresh Tokens
  - no related with the previous tokens
  - Forward security

Scheme —— Dual Dictionary

- A new data structure.
- Linked dictionaries to represent both inverted and forward indexes.

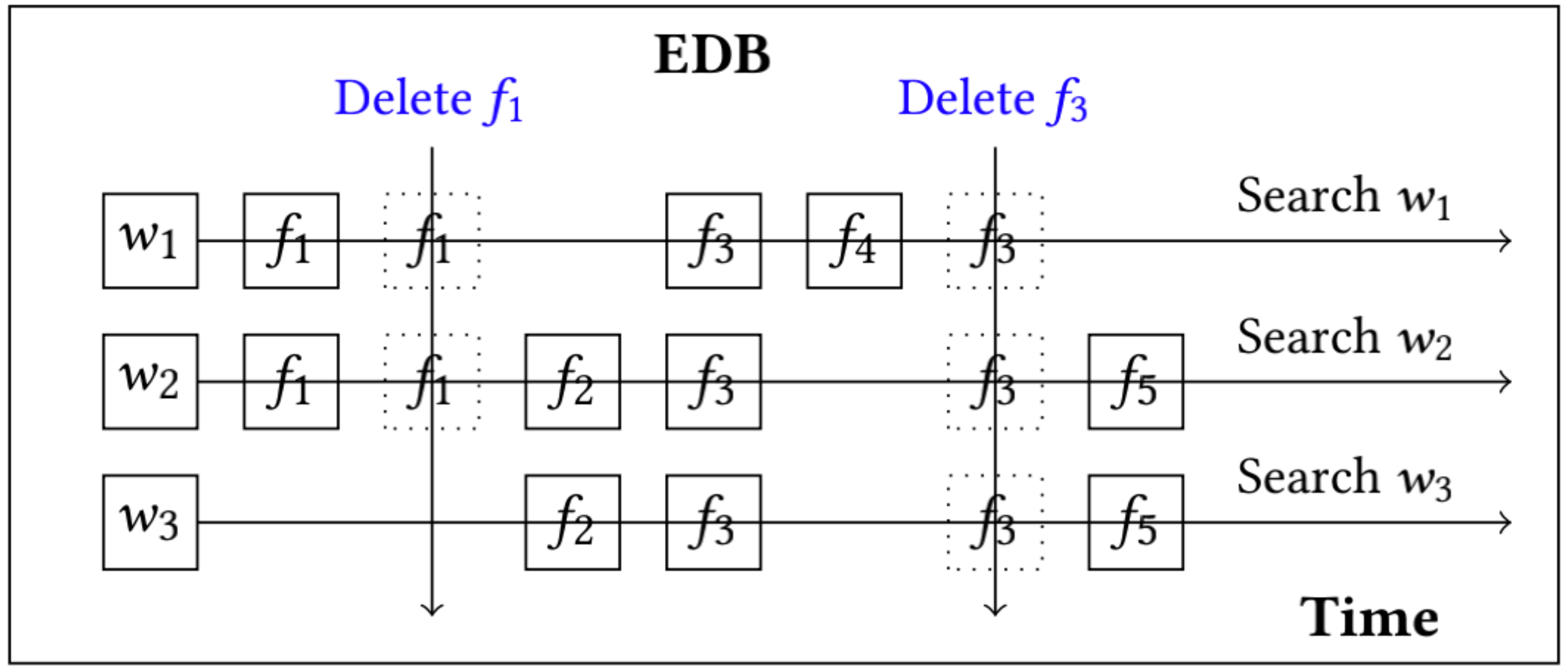


Figure Basic idea of EDB

Scheme —— Fresh Tokens

- After a search query is processed, a fresh key is used to encrypt ind of newly added documents.
- Old search tokens become unusable.
- Decrease the leakage from updates.

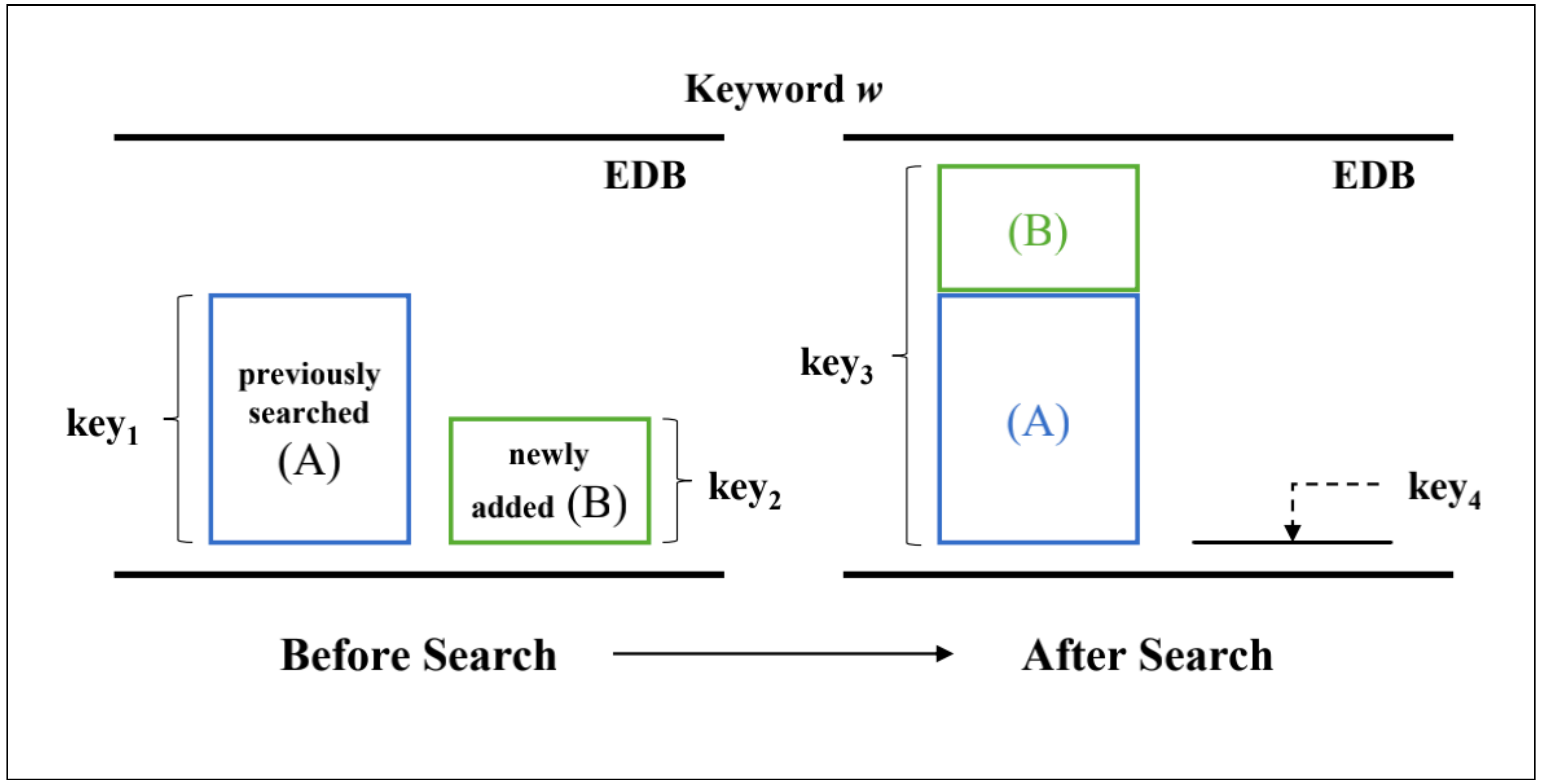
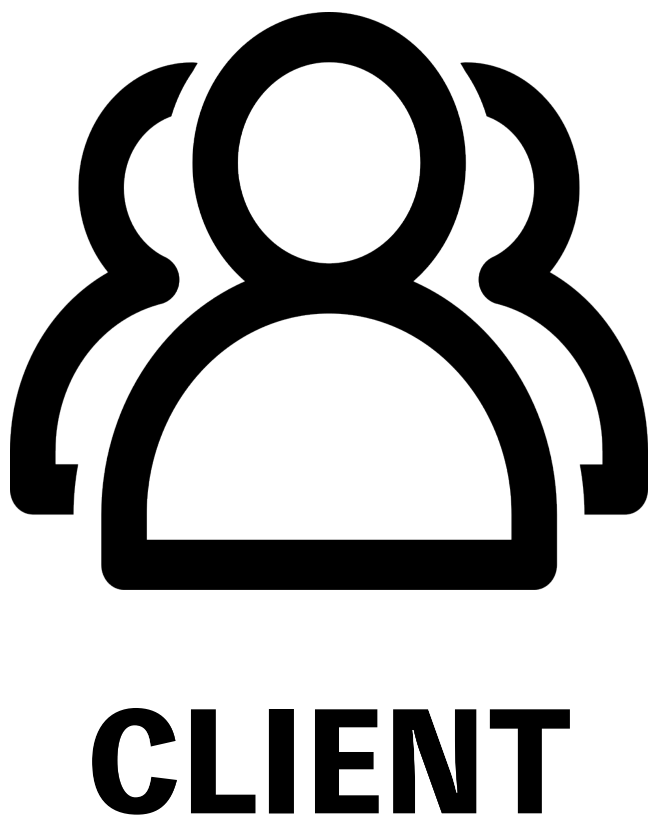


Figure Keyword usage in the search operation

Realization —— SE

SE = (Setup, Search, Addition, Deletion)

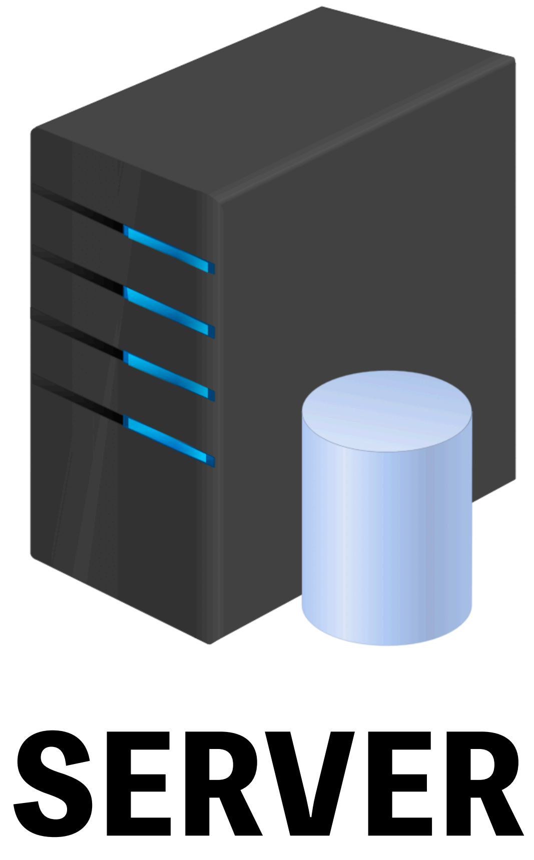


$$\mathsf{SE.Setup}(\lambda\,;\,\perp) \rightarrow (\sigma\,;\,\mathsf{EDB})$$

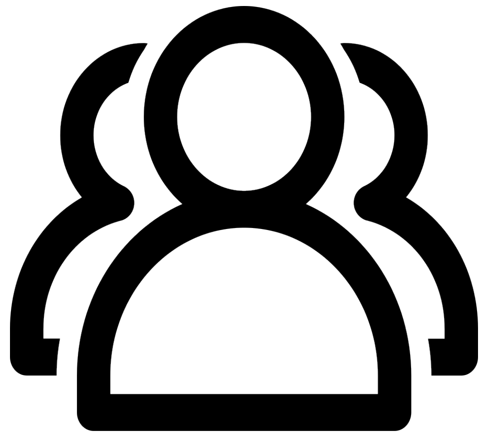
$$\mathsf{SE.Search}((\sigma, w)\,;\,\mathsf{EDB}) \rightarrow ((\sigma', \mathsf{DB}(w))\,;\,\mathsf{EDB}')$$

$$\mathsf{SE.Addition}((\sigma, f)\,;\,\mathsf{EDB}) \rightarrow (\sigma'\,;\,\mathsf{EDB}')$$

$$\mathsf{SE.Deletion}((\sigma, \mathsf{ind})\,;\,\mathsf{EDB}) \rightarrow (\emptyset\,;\,\mathsf{EDB}')$$



Realization —— SE.Setup



Dict_{kwd}

w	$\text{key}^{(w)}$	$\text{cnt}^{(w)}$	$\text{ukey}^{(w)}$	$\text{ucnt}^{(w)}$

key_{prf}

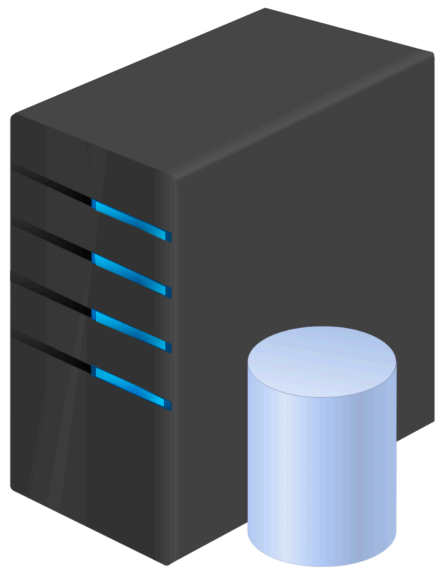
EDB

Dict1

$\text{label}^{(\text{ind})}$	$\text{label}^{(w)}$

Dict2

$\text{label}^{(w)}$	$\text{label}^{(\text{ind})}$	data



Algorithm 1 SE.Setup$(\lambda\ ;\ \perp) \to (\sigma\ ;\ \text{EDB})$

Client$(\lambda) \to (\sigma, \text{EDB})$

$\text{Dict}_{\text{kwd}} \leftarrow \text{Dict.Create}(\emptyset)\,;\quad \text{key}_{\text{prf}} \xleftarrow{\$} \{0,1\}^{\lambda}$

$\sigma \leftarrow (\text{key}_{\text{prf}}, \text{Dict}_{\text{kwd}})$

$\text{EDB} \leftarrow \text{DLDict.Create}(\emptyset)$

Send EDB **to** Server

Realization —— SE.Addition(1)

Algorithm 2 SE.Addition$\big((\sigma, f)\ ;\ \mathsf{EDB}\big) \rightarrow (\sigma'\ ;\ \mathsf{EDB}')$

Client$(\sigma, f = (\mathsf{ind}, \mathsf{DB}(\mathsf{ind}))) \rightarrow (\sigma', \mathsf{AddSet})$

$\mathsf{key}^{(\mathsf{ind})} \leftarrow F(\mathsf{key}_{\mathsf{prf}}, \mathsf{ind})\,; \quad \mathsf{cnt}^{(\mathsf{ind})} \leftarrow 0\,; \quad \mathsf{AddSet} \leftarrow \emptyset$
$\mathsf{RefSet} \leftarrow \mathsf{DB}(\mathsf{ind})$
while $|\mathsf{RefSet}| \neq 0$ **do**
  $w \xleftarrow{\$} \mathsf{RefSet}\,; \quad \mathsf{RefSet} \leftarrow \mathsf{RefSet} \setminus \{w\}$
  if $\mathsf{Dict}_{\mathsf{kwd}}.\mathsf{Get}(w) = \perp$ **then**
    $\mathsf{key}^{(w)} \leftarrow \emptyset\,; \quad \mathsf{cnt}^{(w)} \leftarrow 0\,;$
    $\mathsf{ukey}^{(w)} \xleftarrow{\$} \{0,1\}^{\lambda}\,; \quad \mathsf{ucnt}^{(w)} \leftarrow 0$
    $\mathsf{Dict}_{\mathsf{kwd}} \leftarrow \mathsf{Dict}_{\mathsf{kwd}}.\mathsf{Insert}(w, (\mathsf{key}^{(w)}, \mathsf{cnt}^{(w)}, \mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)}))$
  else
    $(\mathsf{key}^{(w)}, \mathsf{cnt}^{(w)}, \mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)}) \leftarrow \mathsf{Dict}_{\mathsf{kwd}}.\mathsf{Get}(w)$
  end if

- AddSet: The set of records to be added into EDB.
- RefSet: a copy of DB(ind).
- For every keyword in DB(ind), find the related record in $\mathsf{Dict}_{\mathsf{kwd}}$.
- If there is no related record, create a record, and insert it into $\mathsf{Dict}_{\mathsf{kwd}}$.

Realization —— SE.Addition(2)

$\quad \mathsf{cnt}^{(\mathsf{ind})} \leftarrow \mathsf{cnt}^{(\mathsf{ind})} + 1\,; \quad \mathsf{label}^{(\mathsf{ind})} \leftarrow H_1(\mathsf{key}^{(\mathsf{ind})}, \mathsf{cnt}^{(\mathsf{ind})})$

$\quad \mathsf{ucnt}^{(w)} \leftarrow \mathsf{ucnt}^{(w)} + 1\,; \quad \mathsf{label}^{(w)} \leftarrow H_2(\mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)})$

$\quad \mathsf{data} \leftarrow \mathsf{ind} \oplus H_3(\mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)})$

$\quad \mathsf{AddSet} \leftarrow \mathsf{AddSet} \cup \{(\mathsf{label}^{(\mathsf{ind})}, \mathsf{label}^{(w)}, \mathsf{data})\}$

$\quad \mathsf{Dict}_{\mathsf{kwd}}$

$\quad \leftarrow \mathsf{Dict}_{\mathsf{kwd}}.\mathsf{Insert}(w, (\mathsf{key}^{(w)}, \mathsf{cnt}^{(w)}, \mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)}))$

end while

$\sigma' \leftarrow (\mathsf{key}_{\mathsf{prf}}, \mathsf{Dict}_{\mathsf{kwd}})$

send AddSet **to** Server

Server(EDB, AddSet) $\rightarrow$ EDB$'$

for each $(\mathsf{label}^{(\mathsf{ind})}, \mathsf{label}^{(w)}, \mathsf{data}) \in \mathsf{AddSet}$ **do**

$\quad \mathsf{EDB} \leftarrow \mathsf{DLDict}.\mathsf{Insert}(\mathsf{EDB}, (\mathsf{label}^{(\mathsf{ind})}, \mathsf{label}^{(w)}, \mathsf{data}))$

end for

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

- $\mathsf{key}^{(\mathsf{ind})} = F(\mathsf{key}_{\mathsf{prf}}, \mathsf{ind})$
- $\mathsf{label}^{(\mathsf{ind})} = H_1(\mathsf{key}^{(\mathsf{ind})}, \mathsf{cnt}^{(\mathsf{ind})})$
- $\mathsf{label}^{(w)} = H_2(\mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)})$
- $\mathsf{data} = \mathsf{ind} \oplus H_3(\mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)})$
- Client updates Dictkwd and send AddSet to Server.
- Server insert all records in AddSet into EDB.

Realization —— SE.Delete

Algorithm 3 SE.Deletion$((\sigma, \mathsf{ind})\ ;\ \mathsf{EDB}) \rightarrow (\emptyset\ ;\ \mathsf{EDB}')$

Client$(\sigma, \mathsf{ind}) \rightarrow \mathsf{dtoken}^{(\mathsf{ind})}$

$\mathsf{key}^{(\mathsf{ind})} \leftarrow F(\mathsf{key}_{\mathsf{prf}}, \mathsf{ind})$
$\mathsf{dtoken}^{(\mathsf{ind})} \leftarrow \mathsf{key}^{(\mathsf{ind})}$
send $\mathsf{dtoken}^{(\mathsf{ind})}$ **to** Server

Server$(\mathsf{dtoken}^{(\mathsf{ind})}, \mathsf{EDB}) \rightarrow \mathsf{EDB}'$

$\mathsf{cnt}^{(\mathsf{ind})} \leftarrow 1$
while (1) **do**
  $\mathsf{label}^{(\mathsf{ind})} \leftarrow H_1(\mathsf{dtoken}^{(\mathsf{ind})}, \mathsf{cnt}^{(\mathsf{ind})})$
  if $\mathsf{DLDict.Remove}(\mathsf{EDB}, \mathsf{label}^{(\mathsf{ind})}) = \perp$ **then**
    break
  else
    $\mathsf{EDB} \leftarrow \mathsf{DLDict.Remove}(\mathsf{EDB}, \mathsf{label}^{(\mathsf{ind})})$
    $\mathsf{cnt}^{(\mathsf{ind})} \leftarrow \mathsf{cnt}^{(\mathsf{ind})} + 1$
  end if
end while
$\mathsf{EDB}' \leftarrow \mathsf{EDB}$

- $\mathsf{key}^{(w)}$ is transmitted to the server as a deletion token $\mathsf{dtoken}^{(\mathsf{ind})}$.
- Server repeat the process of deleting ind corresponding with $\mathsf{label}^{(\mathsf{ind})}$ by incrementing the counter $\mathsf{cnt}^{(\mathsf{ind})}$.

Realization —— SE.Search(1)

Algorithm 4 SE.Search$((\sigma, w)\ ;\ \mathsf{EDB}) \rightarrow ((\sigma', \mathsf{DB}(w))\ ;\ \mathsf{EDB}')$

Client$(\sigma, w) \rightarrow \mathsf{token}^{(w)}$

```
nkey^(w) <-$ {0,1}^λ
if Dict.Get(Dict_kwd, w) = ⊥ then
    return ∅
end if
(key^(w), cnt^(w), ukey^(w), ucnt^(w)) ← Dict.Get(Dict_kwd, w)
token^(w) ← (key^(w), cnt^(w), ukey^(w), ucnt^(w), nkey^(w))
send token^(w) to Server
```

Server$(\mathsf{EDB}, \mathsf{token}^{(w)}) \rightarrow (\mathsf{EDB}', \mathsf{ncnt}^{(w)}, \mathsf{ResultSet})$

```
parse token^(w) as (key^(w), cnt^(w), ukey^(w), ucnt^(w), nkey^(w))
ResultSet ← ∅ ;   j ← 0
SUEdb(EDB, key^(w), cnt^(w), nkey^(w), j, ResultSet) →
(EDB, j, ResultSet)
SUEdb(EDB, ukey^(w), ucnt^(w), nkey^(w), j, ResultSet) →
(EDB, j, ResultSet)
EDB' ← EDB ;   ncnt^(w) ← j
send (ncnt^(w), ResultSet) to Client
```

Client$(\mathsf{nkey}^{(w)}, \mathsf{ncnt}^{(w)}) \rightarrow \sigma'$

```
nukey^(w) <-$ {0,1}^λ
Dict'_kwd
← Dict.Insert(Dict_kwd, (w, (nkey^(w), ncnt^(w), nukey^(w), 0)))
σ' ← (key_prf, Dict'_kwd)
```

- The client creates a new secret key $\mathsf{nkey}^{(w)}$.
- $\mathsf{token}^{(w)} = (\mathsf{key}^{(w)}, \mathsf{cnt}^{(w)}, \mathsf{ukey}^{(w)}, \mathsf{ucnt}^{(w)}, \mathsf{nkey}^{(w)})$.
- The server uses $\mathsf{key}^{(w)}$ to search ind by increasing a counter from 1 to $\mathsf{cnt}^{(w)}$.
- $\mathsf{ncnt}^{(w)}$ reflecting the current number of document identifiers stored in the EDB.

Realization —— SE.Search(2)

Subroutine: **SUEdb**(EDB, key, cnt, nkey, j, ResultSet)

```
for i = 1 to cnt do
    label^(w) ← H_2(key, i)
    if DLDict.Get(EDB, label^(w)) = ⊥ then
        continue
    end if
    j ← j + 1
    (label^(ind), data) ← DLDict.Get(EDB, label^(w))
    ind ← data ⊕ H_3(key, i)
    ResultSet ← ResultSet ∪ {ind}
    nlabel^(w) ← H_2(nkey, j);    ndata ← ind ⊕ H_3(nkey, j)
    EDB ← DLDict.Remove(EDB, label^(w))
    EDB ← DLDict.Insert(EDB, (label^(ind), nlabel^(w), ndata))
end for
return (EDB, j, ResultSet)
```

- If there are deleted documents, the retrieval proceeds up to $\mathsf{cnt}^{(w)} + \mathsf{ucnt}^{(w)}$.
- Whenever a ind is extracted, the server creates a label $\mathsf{label}^{(w)}$ and a data ndata using the new key $\mathsf{nkey}^{(w)}$.

Security Proof —— Definition

A DSSE scheme is *$\mathcal{L}$-adaptively-secure* if

$$\left|\Pr[\mathrm{Game}_{R,\mathcal{A}}(\lambda) = 1] - \Pr[\mathrm{Game}_{\mathcal{S},\mathcal{A}}(\lambda) = 1]\right| \leq \mathrm{negl}(\lambda).$$

A DSSE scheme is *forward secure* if there exists a leakage function $\bar{\mathcal{L}}$ such that its $\mathcal{L}_{\mathsf{Addition}}$ can be written as

$$\mathcal{L}_{\mathsf{Addition}}(\mathsf{ind}, \mathsf{W}) = \bar{\mathcal{L}}(\mathsf{ind}, |\mathsf{W}|).$$

Security Proof —— Conclusion

THEOREM 4.1. *Let F be a secure PRF. Then our scheme is* $\mathcal{L}$-adaptively-secure *in the (programmable) random oracle model, where the leakage function collection $\mathcal{L}$ is defined as follows:*

- $\mathcal{L}_{\mathsf{Setup}}(\lambda) = \emptyset$,
- $\mathcal{L}_{\mathsf{Search}}(w) = (\mathsf{sp}(w), \mathsf{HistDB}(w))$,
- $\mathcal{L}_{\mathsf{Addition}}(\mathsf{ind}, \mathsf{DB}(\mathsf{ind})) = (\mathsf{ind}, |\mathsf{DB}(\mathsf{ind})|)$,
- $\mathcal{L}_{\mathsf{Deletion}}(\mathsf{ind}) = \mathsf{ind}$.

for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a prf-adversary $\mathcal{B}$ such that

$$\left|\Pr[\mathrm{Game}_{R,\mathcal{A}}(\lambda) = 1] - \Pr[\mathrm{Game}_{\mathcal{S},\mathcal{A}}(\lambda) = 1]\right| \le \mathrm{Adv}^{\mathrm{prf}}_{F,\mathcal{B}}(\lambda) + \mathrm{poly}(\lambda)/2^{\lambda}.$$

We thus conclude the resulting probability is $\mathsf{negl}(\lambda)$ by assuming that the PRF F is secure.

Experiment

- Provide optimal complexity in every point of view

Table 1: Comparison with DSSE schemes supporting forward security

Scheme	Data		Communication		Computation	
	Client	Server	Search	Update	Search	Update
[31]	$O(N^{\alpha})_{(0<\alpha<1)}$	$O(N^+)$	$O(n_w + \log N^+)$	$O(\log N^+)$	$O(\min\{a_w + \log N^+, n_w \log^3 N^+\})$	$O(\log^2 N^+)$
[32]	$O(m+n)$	$O(m \times n)$	$O(n_w)$	$O(m)$	$O(n)$	$O(m)$
[19]	$O(1)$	$O(m+N)$	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(k \log^3 N)$	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(k \log^2 N)$
[4]	$O(m)$	$O(N^+)$	$O(n_w)$	$O(k)$	$O(a_w + d_w)$	$O(k)$
Ours	$O(m)$	$O(N)$	$O(n_w)$	$O(k)$	$O(a_w)$	$O(k)$

The complexities are based on retrieving documents containing a keyword w or updating documents containing k unique keywords. The following notations are used throughout the paper. N is the total number of document/keyword pairs in the database, while m (resp. n) is the number of keywords (resp. documents) in the database. n_w is the size of search result set for keyword w, and a_w (resp. d_w) is the number of times the queried keyword w was historically *added to* (resp. *deleted from*) the database. N^+ is the total number of document/keyword pairs historically stored in the database, i.e., $N^+ = \sum_w (a_w + d_w)$. The notation $\tilde{O}$ hides the $\log \log N$ factors.

Experiment

Table 2: Comparison of the number of major internal functions in Sophos and our scheme for a single pair (ind, w)

Scheme	C/S	Search			Add		
		T	H	F	T^{-1}	H	F
Sophos	Client	-	-	1	1	2	1
	Server	1	2	-	-	-	-
Ours	Client	-	-	-	-	3	1
	Server	-	4	-	-	-	-

T: trapdoor permutation, T^{-1}: inverse of trapdoor permutation,
H: hash function, F: PRF

- Handle most operations with hash function only
- Maximize efficiency
- The same level of security

Experiment

- less CPU load and less client storage space
- superior in speed for the case of a small number of matched documents, but the efficiency is relatively low on other cases

Table 3: Comparison with EDB creation using Enron email dataset

Implementation	Time (ms)	Pairs per sec.	Storage (KiB)	
			Client	Server
Our scheme (with LSH-256)	451,824	137,263.4	86,093	5,243,625
Our scheme (with SHA-256)	469,039	132,225.3	86,089	5,245,237
Sophos (with RSA-2048, Document-level)	9,494,200	6,532.3	272,360	2,242,700
Sophos (with RSA-2048, Keyword-level)	9,628,816	6,441.0	272,364	2,241,436
Sophos (with RSA-512, Document-level)	1,085,146	57,152.6	46,453	2,242,712

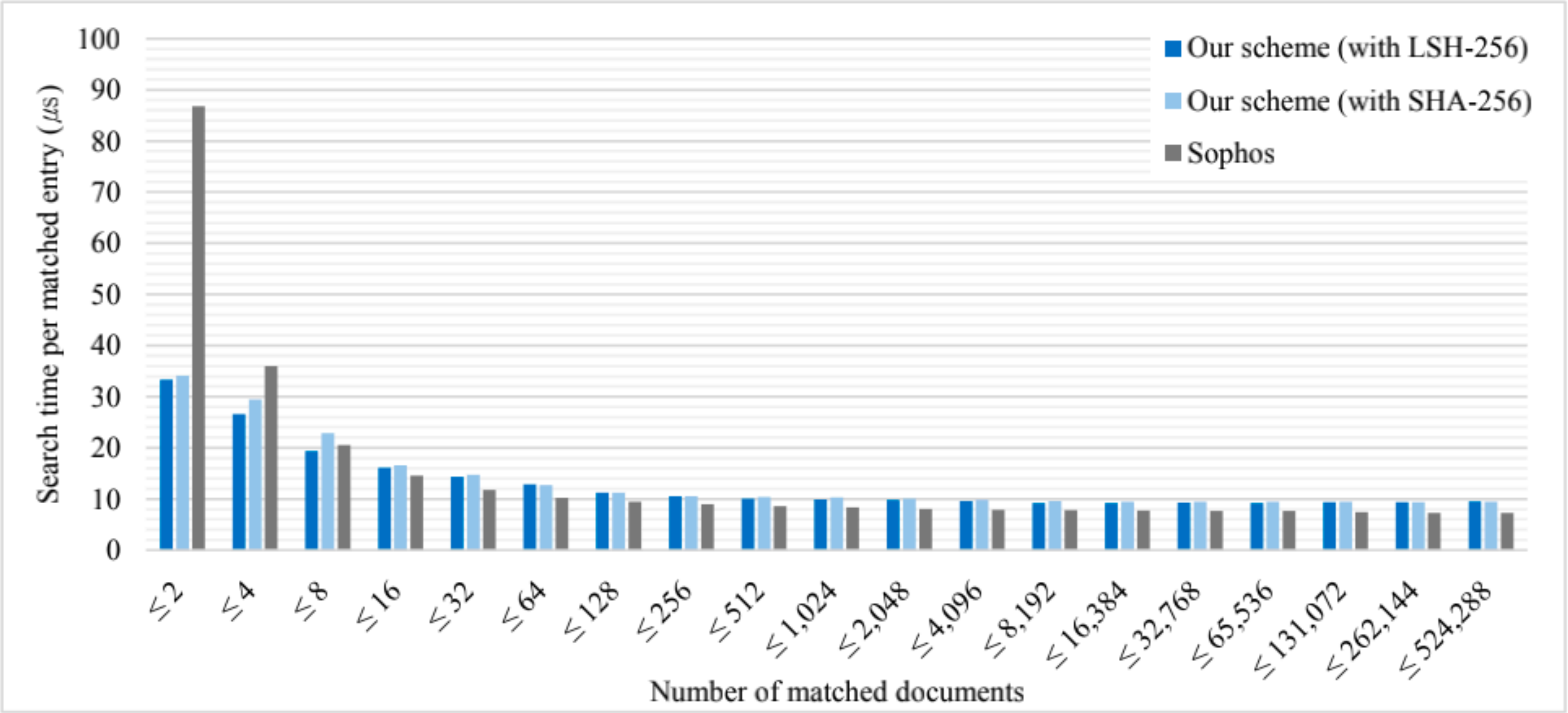


Figure 6: Comparison with search time per matched document

Experiment

- our scheme keeps the amount of data unchanged in this evaluation
- feasible without degrading capacity and performance in environments where updates are frequent.

Table 4: Comparison of operation performance during add-delete-search iterations (unit: pairs per sec.)

Iteration	Our scheme			Sophos		
	Add	Delete	Search	Add	Delete	Search
Init.	132,870	-	91,265	6,898	-	98,315
200k	124,941	117,406	71,140	6,903	6,968	37,825
400k	127,804	118,227	72,910	6,903	6,915	22,461
600k	130,321	117,977	71,620	6,924	6,934	17,153
800k	127,631	118,150	72,683	6,953	6,960	13,479

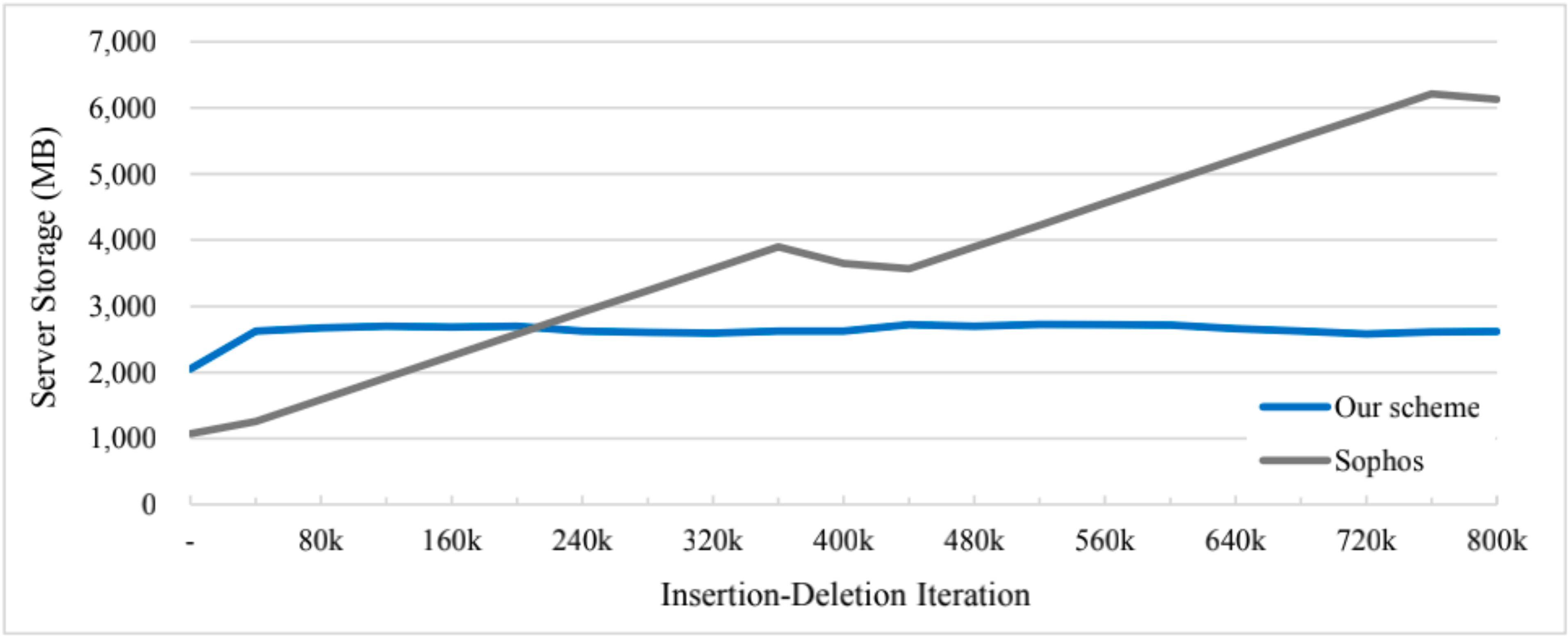


Figure 7: Server storage usage by repeated addition-deletion operations

Discussion

- Leakage Comparison with Previous Schemes
  - the only possible difference compared to our scheme is that the update function of Sophos reveals nothing
- Security against Malicious Adversaries
  - considers only passive adversaries
  - by applying [1], we can easily get the verifiable version of our scheme
- Easy Deletion
  - low complexity, feasible, no meaningless data

[1] Dwaine Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G.Edward Suh. 2003. Incremental multiset hash functions and their application to memory integrity checking. In Advances in Cryptology - ASIACRYPT 2003: 9th International Conference on the eory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 – December 4, 2003. Proceedings, Chi-Sung Laih (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 188–207.

Thanks

报告人：刘美涵、温晴

2021.3.30