Dynamic Searchable Symmetric Encryption

Seny Kamara, Charalampos Papamanthou, Tom Roeder

Presentated by Hui Zhang
Phd@ECE,UMD wayne.huizhang@gmail.com

Introduction

- Searchable symmetric encryption (SSE) allows a client to encrypt its data in such a way that this data can still be searched.
- Given a token, the server can search over the encrypted data and return the appropriate encrypted files.
- The most immediate application of SSE is to the design of searchable cryptographic cloud storage systems. Dropbox, Microsoft Skydrive, Apple iCloud, and some public cloud storage infrastructure like Amazon S3 need such encryption schemes.

Informal index-based SSE scheme

- **Encrypt**: takes as input an index δ and a sequence of n files **f**= (f1, f2, …, fn) and outputs an encrpted index **γ** and a sequence of n ciphertexts **c** = (c1, c2, …, cn)

- **Search**: the client generates a search token T_w and given T_w , **γ** and **c** ,the server can find the identifiers I_w of files that contain *w*

- * here, the index means the inverted index table/ index information of word/file pairs

Previous work

Formal security notions for SSE have evolved

- CKA1(security against chosen-keyword attacks)
  1. Γ and c don't reveal any info about f other than the number of files n and their length
  2. Γ and a token reveal at most the output of the search (the file identifier)
- CKA2:
  - CKA1 only secure if the search queries are independent of (Γ,c) and of previous search results(not Adaptive)
  - CKA2 provides stronger security

Previous work

- Schemes [8,6,21] that are CKA2-secure have limitations:
  - 1) high time complexity
  - 2) not explicitly dynamic
    - One can't add or remove files without either re-indexing the entire data collection or making use of generic and relatively expensive dynamization techs.
- The only DSSE and CKA2-secure is [23]. The main limitations are the size of the encrypted index is relatively large

Comparison of SSE schemes

Scheme	Dynamism	Security	Search time	Index size
SWP00 [22]	static	CPA	$O(\|\mathbf{f}\|)$	N/A
Z-IDX [15]	dynamic	CKA1	$O(\#\mathbf{f})$	$O(\#\mathbf{f})$
CM05 [5]	static	CKA1	$O(\#\mathbf{f})$	$O(\#\mathbf{f} \cdot \#W)$
SSE-1 [8]	static	CKA1	$O(\#\mathbf{f}_w)$	$O(\sum_w \#\mathbf{f}_w + \#W)$
SSE-2 [8]	static	CKA2	$O(\#\mathbf{f})$	$O(\#\mathbf{f} \cdot \#W)$
vLSDHJ10 [23]	dynamic	CKA2	$O(\log \#W)$	$O(\#W \cdot m_f)$
CK10 [6]	static	CKA2	$O(\#\mathbf{f}_w)$	$O(\#W \cdot m_f)$
KO12 [21]	static	UC	$O(\#\mathbf{f})$	$O(\#W \cdot \#\mathbf{f})$
this paper	dynamic	CKA2	$O(\#\mathbf{f}_w)$	$O(\sum_w \#\mathbf{f}_w + \#W)$

Table 1: Comparison of several SSE schemes. Search time is per keyword w and update time is per file f. $\mathbf{f}$ is the file collection, $|\mathbf{f}|$ is its bit length, $\#\mathbf{f}$ is the number of files in $\mathbf{f}$, $\#\mathbf{f}_w$ is the number of files that contain the keyword w, $\#W$ is the size of the keyword space and m_f is the maximum (over keywords) number of files in which a keyword appears.

Contributions

1. Present a formal security definition for dynamic SSE.
2. Construct the first SSE scheme that is dynamic, CKA2-secure and achieves *optimal* search time.
3. Describe first implementation and evaluation of an SSE scheme based on *inverted index*
4. Conduct a performance evaluation of the scheme.

DSSE definition

- A dynamic index-based SSE scheme is a tuple of ***nine*** polynomial-time algorithms:

1. $K \leftarrow \mathsf{Gen}(1^k)$
2. $(\gamma, \mathbf{c}) \leftarrow \mathsf{Enc}(K, \mathbf{f})$
3. $\tau_s \leftarrow \mathsf{SrchToken}(K, w)$
4. $(\tau_a, c_f) \leftarrow \mathsf{AddToken}(K, f)$
5. $\tau_d \leftarrow \mathsf{DelToken}(K, f)$
6. $\mathbf{I}_w := \mathsf{Search}(\gamma, \mathbf{c}, \tau_s)$
7. $(\gamma', \mathbf{c}') := \mathsf{Add}(\gamma, \mathbf{c}, \tau_a, c)$
8. $(\gamma', \mathbf{c}') := \mathsf{Del}(\gamma, \mathbf{c}, \tau_d)$
9. $f := \mathsf{Dec}(K, c)$

SEE-1 construction

- Components:
  - Index
  - Search Table T_s
  - Deletion Table T_d
  - Search Array A_s
  - Deletion Array A_d
- Making SSE-1 dynamic
  1. Use non-committing encryption scheme to make SSE-1 CKA2-secure
  2. Hard to allow run-time addition and deletion of files. Addressed with (1), (2), (3)

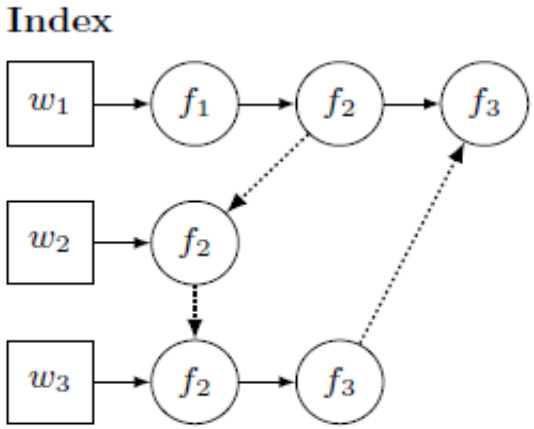
High-level methods to be dynamic

- (file deletion)Add an extra data structure called deletion array that the server can query
- (pointer modification)Encrypt the pointers stored in a node with a homomorphic encryption scheme
- (memory management) Maintain a free list that the server uses to add new nodes

An Illustrative Example

- Searching
- Adding a document
- Deleting a document

An Illustrative Example



Search Table $\mathtt{T}_s$

$F_{K_1}(w_1) \longrightarrow (4 \mid\mid 1) \oplus G_{K_2}(w_1)$

$F_{K_1}(w_2) \longrightarrow (0 \mid\mid 2) \oplus G_{K_2}(w_2)$

$F_{K_1}(w_3) \longrightarrow (5 \mid\mid 0) \oplus G_{K_2}(w_3)$

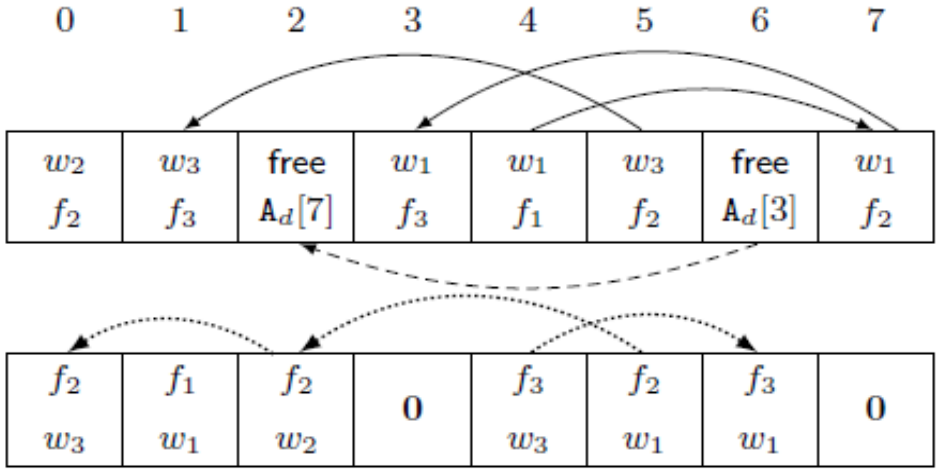
free $\longrightarrow 6$

Deletion Table $\mathtt{T}_d$

$F_{K_1}(f_1) \longrightarrow 1 \oplus G_{K_2}(f_1)$

$F_{K_1}(f_2) \longrightarrow 5 \oplus G_{K_2}(f_2)$

$F_{K_1}(f_3) \longrightarrow 4 \oplus G_{K_2}(f_3)$



Search Array $\mathtt{A}_s$

Deletion Array $\mathtt{A}_d$

Figure 1: A small example of a dynamic encrypted index.

Security

- **All practical SSE schemes leak some information.** Unfortunately, the extent to which the practical security of SSE is affected by this leakage is not well understood and depends greatly on the setting in which SSE is used. **We are aware of only one concrete attack [18] that exploits this leakage and it depends strongly on knowledge of previous queries and statistics about the le collection**. We note, however, that our scheme leaks more than most previously-known constructions since it is dynamic and there are correlations between the information leaked by its various operations. In the following, we provide a framework for describing and comparing the leakage of SSE schemes

Performance

- The unit of measurement in all of the micro-benchmarks is the file/word pair:

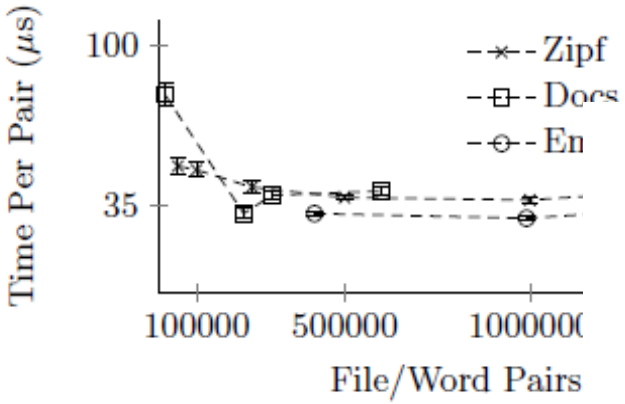


Figure 5: SSE.Enc.

operation	time	stddev
SSE.Search	7.3	0.6
SSE.AddToken	37	2
SSE.DelToken	3.0	0.2
SSE.Add	1.6	0.4
SSE.Del	24	1

Table 2: Execution time (in μs) per unit (word or file) for SSE operations.

Performance

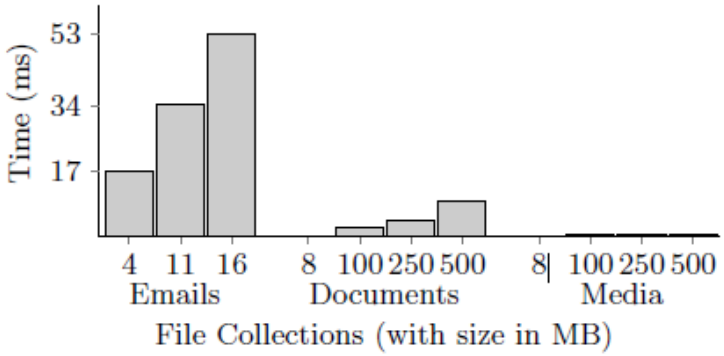


Figure 7: Execution time for SSE.Search.

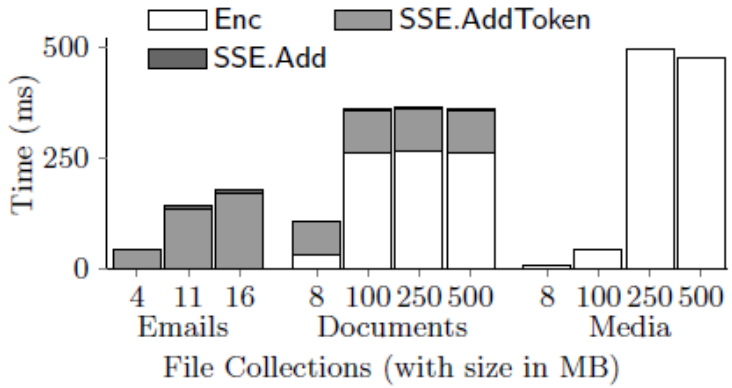


Figure 8: Execution time for adding a file.

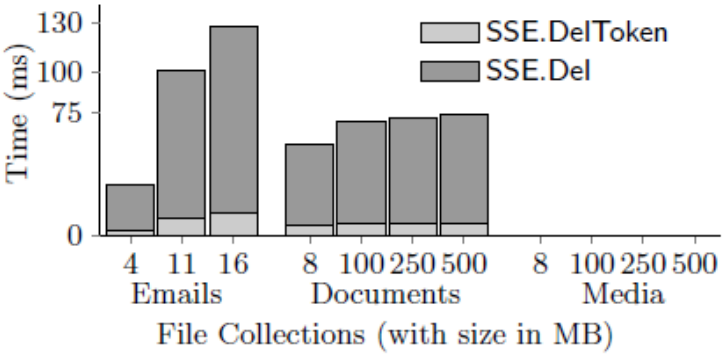


Figure 9: Execution time for deleting a file.

Reference

- [1] Kamara, Seny, Charalampos Papamanthou, and Tom Roeder. "Dynamic searchable symmetric encryption." *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

Thank you !