様々な処理を実現する検索可能暗号

NOZOE, Shunta / 野副, 俊太

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
12
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2017-03-31
(URL)
https://doi.org/10.15002/00014411

様々な処理を実現する検索可能暗号
Searchable Symmetric Encryption Supporting Various Operations

野副 俊太
Shunta Nozoe
法政大学大学院 情報科学研究科 情報科学専攻
E-mail: shunta.nozoe.7z@stu.hosei.ac.jp

Abstract

Cloud service enables users not only to save their data to an external server but also to process the data at the server. However, the data stored at could server may be stolen and tampered with by malicious party or administrator of the server. If user encrypts data before storing his data to the server to solve this problem, the data won't be leaked to the others. However, no operation can be performed if data are encrypted. To overcome such problem, a cryptographic protocol called SSE-1 was proposed where SSE-1 allows user to search with keyword over encrypted data. However, it is difficult to put SSE-1 to practical use since SSE-1 does not support addition and deletion operations. In this paper, we propose new protocol named ASSE (Array SSE). ASSE is a protocol for searchable encryption supporting addition and deletion operations. In ASSE, index is represented by simple bit sequence. We implement ASSE and DSSE which is the existing scheme to support addition and deletion operations. We run them on Tomcat server. Implementation result shows that search and deletion operation are faster than DSSE' s operations and additions operation is more secure than DSSE's. The result show that the ASSE is highly practical.

1. 研究背景

近年，計算機の計算能力の向上や仮想化技術の発展，ネットワークの大容量化や通信技術の進歩により，クラウドコンピューティングは，より多くの人にとって身近なものとなっている．クラウドはユーザに高い利便性を提供する一方で，サーバ上に置いた情報が漏洩するなど，従来のシステムでは起こらなかった新しいセキュリティ上のリスクが発生する．その一つとしてクラウド上のデータやそのデータに対する情報処理の内容がサーバ管理者や通信を傍受している第三者に漏洩する恐れがある．また，ユーザ側はクラウド管理者側の管理体制を知ることができないため，セキュリティ上の事故が起きた際に，ユーザの側では対策を講じることができない．このような脅威への対策として，データを安全に保管するために，あらかじめデータを暗号化した上でサーバ上で保管する運用法が考えられる．しかし，単に通常の暗号でデータを暗号化した場合，サーバが平文の内容を知らないため，暗号化した状態のままでは検索処理を行うことができなくなってしまい，利便性が低下する．また，サーバ上でデータを一時的に復号することで，復号したデータに対して検索を行うことが可能だが，その場合復号されたデータがサーバ管理者や第三者に漏洩する危険性がある．この問題を解決するためには，データを復号することなく情報処理ができるように暗号化する必要があり，これを実現する方法として秘匿演算と呼ばれる技術が知られている．秘匿演算とは，暗号化された状態のまま情報処理を行うプロトコルであり，その一つに検索可能暗号が存在する．検索可能暗号とは，暗号化されたデータに対して，復号することなく検索をすることが可能な技術である．2000 年に Song らによって提案され，その後は共通鍵/公開鍵ベースの両方が研究されている．共通鍵ベースの検索可能暗号 (Searchable Symmetric Encryption，以下 SSE) は，ドキュメント集合を暗号化する際にドキュメントとキーワードの対応関係を元にインデックスを生成する．このインデックスに対してキーワードに対応する検索トークンを用いて検索を行うことで，ドキュメントを復号することなくキーワードに対応するドキュメントを得ることができる．2006 年には，Curtmola らによって安全性の定義と，それを満たす効率的な方式である SSE-1[1] が提案された．以降は，SSE-1 をベースにした方式が多く提案されている．しかし，SSE-1 では，ドキュメントの追加・削除を考慮していない．ここでのドキュメントの追加・削除とは，インデックスを更新することを意味する．追加・削除を実現する自明な方法として，追加・削除のたびに新しいインデックスを生成することが考えられるが，この方法では安全性や効率に問題がある．検索可能暗号の実用化にあたって，ユーザが使用できるデータ容量が有限であることを考慮すれば，追加と削除の両方が可能な方式が望ましいといえる．そこで，ドキュメントの追加・削除が可能な SSE の研究が進められ，2012 年には Kamara らによって SSE-1 をベースにした Dynamic SSE[2] が提案された．しかし，この方式では追加の際にインデックスの部分情報が漏洩してしまう問題やインデックスサイズが大きくなってしまう問題を抱えている．本研究は SSE-1 や Dynamic SSE を参考に追加・削除の際の情報漏洩をより少なく，インデックスサイズをより小さくした方式を提案し，既存の方式との比較をするものである．

2. 準備

X を有限集合とする．このとき，$x \xleftarrow{\$} X$ とは, x が X から一様に選ばれたことを意味する．PPT とは確率的多項式時間 (*probabilistic polynomial time*) のことである．

Superviser: Prof. Satoshi Obana

ビット列 b の i 番目のビットを，$b[i]$ と表記する．w はキーワード，D はドキュメント，c は暗号化されたドキュメントを表す．入力 x と秘密鍵 K を用いる鍵付き関数 F を $F_K(x)$ と表記する．また，入力 x と秘密鍵 K を用いる鍵付き置換 P を $P_K(x)$ と表記する． 秘密鍵 K を用いる暗号 E で平文 m を暗号化する場合 $\mathrm{E.Enc}_K(m)$，暗号文 c を復号する場合 $\mathrm{E.Dec}_K(c)$ と表記する．

2.1. 擬似ランダム関数

鍵付き対関数 $f : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^m$ が擬似ランダム関数であるとき，f は以下の条件を満たす．

- 入力 $x \in \{0,1\}^n$ と鍵 $K \in \{0,1\}^s$ が与えられたとき，$F_K(x)$ を計算する確率的多項式時間アルゴリズム (PPT) が存在する．
- $\mathcal{F}$ を $\{0,1\}^n$ から $\{0,1\}^m$ へ写すすべての関数の集合としたとき，任意の PPT アルゴリズム A と無視できるほど小さい値 ϵ に対して以下の式が成立する．
  $|Pr_{K \leftarrow \{0,1\}^s}[A^{F_K}(1^t) = 1] - Pr_{f \in \mathcal{F}}[A^f(1^t) = 1]| < \epsilon$

2.2. 擬似ランダム置換

鍵付き対置換 $f : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$ が擬似ランダム関数であるとき，f は以下の条件を満たす．

- 入力 $x \in \{0,1\}^n$ と鍵 $K \in \{0,1\}^s$ が与えられたとき，$P_K(x)$ を計算する確率的多項式時間アルゴリズム (PPT) が存在する．
- $\mathcal{F}$ を $\{0,1\}^n$ から $\{0,1\}^n$ へ写すすべての置換の集合としたとき，任意の PPT アルゴリズム A と無視できるほど小さい値 ϵ に対して以下の式が成立する．
  $|Pr_{K \leftarrow \{0,1\}^s}[A^{P_K}(1^t) = 1] - Pr_{p \in \mathcal{P}}[A^p(1^t) = 1]| < \epsilon$

2.3. Searchable Symmetric Encryption

Searchable Symmetric Encryption (SSE) による検索システムの構築に必要なものは，ドキュメントの集合およびそのドキュメントを暗号化した暗号化ドキュメント集合，検索に使用するキーワードの集合（辞書）である．以後，$\mathcal{D}$ をドキュメントの集合とする．また，$\mathcal{D}$ に含まれるドキュメントの数を m とする．各ドキュメントには 1 から m までの id が重複なく割り振られている．このとき，id 番号 j のドキュメントを D_j と表記する．$\mathbf{c}$ を暗号化されたドキュメントの集合とする．また，$\mathbf{c}$ に含まれる暗号化ドキュメントの数を m とする．各ドキュメントには 1 から m までの id が重複なく割り振られている．id 番号 j の暗号化ドキュメントを c_j と表記する．このとき，D_j 暗号化したものが c_j であり， c_j を復号したものが D_j である．キーワードとは次の 3 種類に分類されるビット列である．一つ目は $\mathcal{D}$ に属するいずれかのドキュメントに含まれる単語である．二つ目は $\mathcal{D}$ に属するどのドキュメントにも含まれない単語である．三つ目は 意味を持たないランダムなビット列である．このビット列を ”ダミーワード” と呼称する．キーワードの最大長を max ビットとする．Δ をキーワードの集合とする．また，Δ を ”辞書” と呼称する．辞書に含まれるキーワードの数を n とする．各キーワードには 1 から n までの id が重複なく割り振られている．id 番号 i のキーワードを w_i と表記する．

SSE のシステムモデルは以下の通りである．

ユーザはドキュメント集合 $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ と辞書 $\Delta = \{w_1, \ldots, w_d\}$ を持つ．辞書はあらかじめ定まっているものとする．キーワード $w \in \Delta$ による検索結果を，$\mathcal{D}(w) = \{id_j | w \in \mathcal{D}_j\}$ と定義する．ユーザは $\mathcal{D}$ の各要素を暗号化し，暗号化ドキュメント集合 $\mathbf{c} = \{c_1, \ldots, c_n\}$ および，検索の際に利用するインデックス $\mathcal{L}$ を生成し，サーバに保存しておく．インデックス $\mathcal{L}$ は $\mathcal{D}$ の各ドキュメントに対する Δ 内の全キーワードによる検索結果を元に生成される．検索を行う際は，検索したいキーワード w から検索トークン τ を生成しサーバへ送信する．サーバはドキュメントを復号することなく τ を用いて $\mathcal{L}$ に対して検索処理を行い，検索結果 $\mathcal{D}(w)$ を得る．この結果に該当する暗号化ドキュメント集合 $\mathbf{c}' = \{c_j | c_j \in \mathbf{c}, id_j \in \mathcal{D}(w)\}$ がサーバからユーザへ送信される．ユーザは受信した $\forall c \in \mathbf{c}'$ を復号することでキーワードに該当するドキュメントを得ることができる．この検索過程において，検索トークン τ は暗号化されており，ユーザとサーバ間を行き来したものは τ と暗号化ドキュメント c のみである．また，復号はユーザ側でしか行われていないため，サーバ管理者や通信を傍受している第三者に情報を漏らすことなく検索処理を行うことができる．なお検索トークン τ をトラップドアと呼ぶ．

SSE は次の 5 つの関数からなる．

1. $K \leftarrow \mathbf{Gen}(1^k)$:
   秘密鍵を生成する関数．セキュリティパラメタ k を入力とし，秘密鍵 K を出力する．
2. $(\mathcal{L}, \mathbf{c}) \leftarrow \mathbf{Enc}_K(\mathcal{D}, \Delta)$:
   ドキュメントの暗号化とインデックスの生成を行う関数．鍵 K，ドキュメント集合 $\mathcal{D}$，キーワード辞書 Δ を入力とし，インデックス $\mathcal{L}$ と暗号化ドキュメント集合 $\mathbf{c}$ を生成する．
3. $\tau_w \leftarrow \mathbf{SrchToken}_K(w)$:
   キーワードをもとに検索に必要な値（トラップドア）を計算するための関数．検索キーワード w を入力とし，トラップドア τ_w を出力する．τ_w は w を暗号化することで生成される．
4. $\mathcal{D}(w) \leftarrow \mathbf{Search}(\mathcal{L}, \tau_w)$:
   検索処理を行う関数．インデックス $\mathcal{L}$ とトラップドア τ_w を入力とし，検索結果 $\mathcal{D}(w)$ を出力する．
5. $\mathcal{D} \leftarrow \mathbf{Dec}_K(c)$:
   暗号化されたドキュメントを復号する関数．鍵 K，暗号化ドキュメント c を入力とし，復号化ドキュメント $\mathcal{D}$ を出力する．

2.4. SSE-1

SSE-1 では，キーワード毎にそのキーワードに該当するドキュメント id の線形リストを作成し，インデックスとする．リストの各ノードは共通鍵暗号で暗号化され，配列 A のランダムな箇所に格納される．ノードにはドキュメント id と次のノードのアドレス（線形リスト上における次のノードの格納箇所を示す配列 A のポインタ），次のノードを復号するための復号鍵を格納している．つまり，ひとつのノードを復号することで次のノードのアドレスと復号鍵を得られる仕組みであり，検索したいキーワード w に対応するリストの先頭ノードを復号すれば，それ以降のノードが順次復号でき，検索結果を求めることができる．逆に，先頭ノードのアドレスと復号鍵がなければ，2 番目以降のノードのアドレスが分からないため，線形リストの長さ = キーワードに該当するドキュメントの数が漏れることがない．各線形リストの先頭ノードのアドレスと復号鍵は配列 T に格納されている．キーワード w に対応する線形リストの先頭ノード情報は配列 T の $\pi(w)$ 番目に格納される（π は擬似ランダム置換）

SSE-1 では，安全かつ効率的にインデックスを更新する方法が存在しない．インデックスを更新できないため，ドキュメントの削除や追加を行った際，その処理内容が検索結果に反映されない．

2.5. Dynamic SSE

Dynamic SSE（以下 DSSE）は，SSE-1 をベースに追加と削除処理を可能にした方式である．SSE-1 における線形リストを検索用と削除用のそれぞれ 2 つずつ所持することで削除処理と追加処理を実現している．

DSSE の問題は，サーバ側が，検索処理の際にトラップドアを保管しておくことでドキュメントを追加した際にサーバ側が所持するトラップドアを用いて新しいドキュメントに対する検索結果を得ることができる点や，追加処理の際にインデックスの情報がいくつか漏洩する点，インデックスを生成した時点であらかじめ追加可能領域を決めるため追加できるドキュメントの数に限りがある点である．

3. 提案方式

本章では追加・削除処理に対応した提案方式の ASSE について説明する．

ASSE はインデックスの更新が可能な方式である．SSE-1 をもとに，追加と削除処理を実現した．SSE-1 や DSSE では線形リストを用いてインデックスを構成していたが，ASSE ではビット列を用いてインデックスを構成する．

3.1. モデル

ASSE のモデルを以下に示す:

ユーザ側が所持

- 擬似ランダム置換 : P_1, P_2
- 擬似ランダム関数 : F, G, E
- PCPA 安全な対象暗号 : SKE1, SKE2
- 秘密鍵 : $K_1, K_2, K_3, K_4, K_5, K_6, K_7$

サーバ側が所持

- 配列 : A_1, A_2
- テーブル : T_1, T_2
- ビット列 e
- 暗号化ドキュメント集合 : $\mathbf{c}$
- 擬似ランダム関数 : F
- 秘密鍵 : K_3

A_1, A_2, T_1, T_2, e をまとめてインデックスとする．

3.2. 構成

ASSE の方式は，SSE と共通の **Gen**, **Enc**, **SrchToken**, **Search**, **Dec** および，以下の 5 つの関数によって構成される．

6. $(I', \mathbf{c}') \leftarrow \mathbf{Delete}_K(id, I, \mathbf{c})$:
   削除処理を実行する関数．ドキュメントの削除とインデックスの更新を行う．インデックス I と暗号化ドキュメント集合 $\mathbf{c}$ を入力とし，更新後のインデックス I' と暗号化ドキュメント集合 $\mathbf{c}'$ を出力する．
7. $at1 \leftarrow \mathbf{AddToken1}_K(\boldsymbol{w})$:
   追加処理の準備をする関数．追加するドキュメントに含まれるキーワードの集合 $\boldsymbol{w}$ を入力とし，追加処理トークン $at1$ を出力する．$at1$ は $\boldsymbol{w}$ の各キーワードを暗号化することで生成される．
8. $at2 \leftarrow \mathbf{AddToken2}_K(at1, I)$:
   追加処理の準備をする関数．追加処理トークン $at1$ を入力とし，追加処理トークン $at2$ を出力する．$at2$ は $at1$ の各暗号化キーワードとペアにしてテーブル T_2 に格納されていた値．
9. $at3 \leftarrow \mathbf{AddToken3}_K(at2, D)$:
   追加処理の準備をする関数．追加処理トークン $at2$ を入力とし，追加処理トークン $at3$ を出力する．
10. $(I', \mathbf{c}') \leftarrow \mathbf{Add}_K(at3, I, \mathbf{c})$:
    追加処理を実行する関数．ドキュメントの追加とインデックスの更新を行う．インデックス I と追加処理トークン $at3$ を入力とし，更新後のインデックス I' と暗号化ドキュメント集合 $\mathbf{c}'$ を出力する．

10 の関数のうち，**Gen**, **Enc**, **SrchToken**, **Dec**, **Addtoken1**, **Addtoken3** はユーザ側で実行する．残りの **Search**, **Delete**, **Addtoken2**, **Add** はサーバ側で実行する．

3.3. インデックスの詳細と処理

インデックスを構成するビット列はキーワード毎に 1 つ生成され，そのキーワードの検索結果を格納する．ビット列の長さはドキュメントの数と同じである．ビット列の各ビットはドキュメントの id と対応していて，そのドキュメントがキーワードの検索結果に含まれるか否かの

情報を持つ．ビット列の j ビット目が 1 ならキーワードの検索結果に id 番号が j のドキュメントが含まれるが，0 ならば含まれない．

図はある 3 つのドキュメントと 4 つのキーワードの対応関係を表したものである．これらのドキュメントとキーワードからインデックスを生成する流れをしめす．

$$\boldsymbol{D} = (D_1, D_2, D_3)$$
$$\Delta = (w_1, w_2, w_3, w_4)$$

$$\begin{matrix} w_1, w_3 \in D_1 \\ w_2 \in D_2 \\ w_1, w_2, w_4 \in D_3 \end{matrix}$$

図 1　ドキュメントとキーワードの対応関係

インデックスは配列 A_1, A_2，テーブル T_1, T_2 および，ビット列 e によって構成される．A_1 はキーワード毎に生成されるビット列を格納する配列である．また，T_1 は A_1 に格納されたビット列のアドレスを記録する．A_1, T_1 の長さはキーワードの数に等しい．前述したとおりビット列の各ビットはドキュメントの id 番号に対応しており，キーワード w_i の検索結果を格納したビット列の j 番目のビットが 1 ならば，キーワード w_i で検索した際の検索結果にドキュメント D_j が含まれる．（ドキュメント D_j はキーワード w_i を含む．）これらのビット列は見るだけで検索結果が分かるようになっているため，配列 A_1 に格納する前に暗号化する必要がある．キーワード w_i の検索結果を記録したビット列は，擬似ランダム置換 G および擬似ランダム関数 F で w_i を変換した値 $F_{K_3}(G_{K_2}(w_i))$ との排他的論理和をとることで暗号化され，A_1 の空き領域のなかからランダムに選ばれた場所に格納される．このとき，ビット列の格納場所のアドレスは，擬似ランダム置換 P で w_i を変換した値 $P_{1K_1}(w_i)$ と組にして，テーブル T_1 に格納される．

図の例では，キーワード w_1 がドキュメント D_1 と D_3 に含まれるため，w_1 のビット列は 101 となる．このビット列は $F_{K_3}(G_{K_2}(w_1))$ との排他的論理和をとることで暗号化され，配列 A_1 に格納される. このとき，配列 A_1 の 1 番目に格納されたので，テーブル T_1 の $P_{1K_1}(w_1)$ に 3 が格納される．

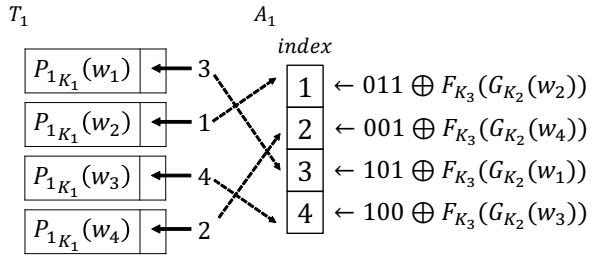

図 2　配列 A_1 とテーブル T_1 の対応関係

ビット列 e はドキュメントの有無を記録する．e の各ビットはドキュメントの id 番号に対応しており，e の j ビット目が 1 ならば id 番号が j のドキュメントがサーバ上に存在し，逆に 0 ならば存在しない．e の長さはドキュメントの最大数と等しい．e の長さからインデックス生成時のドキュメント数を引いた値が，後述する追加処理によってサーバ上に後から追加することのできるドキュメント数の上限となる．

図の例では，id 番号が 1,2,3 の 3 つのドキュメントが存在しているため e の 1 ビット目から 3 ビット目は 1 になる．e の長さは 5 ビットなので，後 2 つのドキュメントを追加できる．現在，id 番号が 4,5 のドキュメントは存在していないため e の 4 ビット目と 5 ビット目は 0 になる．

e : 00111

図 3　ドキュメントの有無を管理するビット列 e

A_2 は追加処理に必要な情報を保持したビット列を格納するための配列である． A_2 の長さ m' は追加処理によって後から追加することのできるドキュメントの最大数に等しい．ビット列の長さはキーワードの数に等しく，各ビットはそれぞれのキーワードの検索結果を記録したビット列の配列 A_1 における格納箇所に対応している．A_2 の k 番目に格納されるビット列は，全てのキーワード $(w_1, w_2, ..., w_n)$ を擬似ランダム置換 G および擬似ランダム関数 F で変換した値 $(F_{K_3}(G_{K_2}(w_1)), F_{K_3}(G_{K_2}(w_2)), ..., F_{K_3}(G_{K_2}(w_n)))$ の $m+k$ 番目のビットを集めたものである．(m はインデックス生成時のドキュメント数) つまり，キーワード w の検索結果を保持したビット列が A_1 の l 番目に格納されている場合，A_2 の k 番目に格納されるビット列の l 番目のビットは $F_{K_3}(G_{K_2}(w))$ の $m+k$ 番目のビットとなる．ただし，ビット列は格納される前に，格納箇所のアドレスとドキュメント数を足した数 $m+k$ を擬似ランダム関数 E で変換した値 $E_{K_5}(m+k)$ と排他的論理和をとることで暗号化されている．

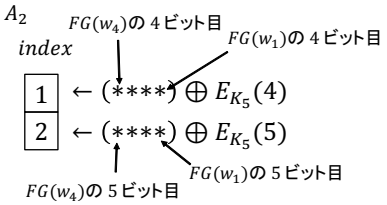

図 4　配列 A_2

T_2 は，配列 A_1 に格納されているビット列と各キーワードの対応関係の情報を保持している．テーブル T_1 との違いは，A_1 上のアドレスを暗号化してから格納している点である．キーワード w を擬似ランダム置換 P_2 で暗号化した値と A_1 の格納要素のなかで w に対応しているビット列が格納されている箇所のアドレスを暗号

SKE2 で暗号化した数値をペアにしたものを保持する．このとき，SKE2 は PCPA 安全を満たす．

図 5 の例では，テーブル T_2 に キーワード $T_2[P_{2K_4}(w_2)]$ と SKE2.Enc$_{K_7}(1)$ がペアとなって格納されているため，キーワード w_2 の検索結果を記録したビット列は配列 A_1 の 1 番目に格納されている．（図 2 参照）

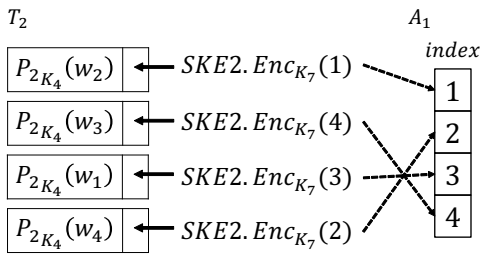

図 5　配列 A_1 とテーブル T_2 の対応関係

検索処理の際，ユーザは検索キーワード w から $P_{1K_1}(w)$ と $G_{K_2}(w)$ を計算しサーバへ送る (3. Srch-Token での処理)．$P_{1K_1}(w)$ は w の検索結果を記録したビット列を格納している A_1 上のアドレスを T_1 から得るため，$G_{K_2}(w)$ は A_1 に格納されているビット列を復号するために必要な値である．サーバはビット列 $b_s = (A_1[T_1[P_{1K_1}(w)]] \oplus F_{K_3}(G_{K_2}(w))) \wedge e$ を生成し，b_s の各ビットを調べる (4. Search での処理)．e との積をとるのは，存在しないドキュメントを検索結果から除くためである．b_s の i 番目が 1 ならばドキュメント D_i は w の検索結果に含まれる．

削除処理の際は，ユーザがサーバに削除するドキュメントの id を送る．サーバはドキュメントの有無を管理するビット列 e の id 番目のビットを 0 にする (6. Delete での処理)．この操作により，検索時に e との席をとる際に b_s の id 番目のビットが必ず 0 になるため，削除したドキュメント D_{id} が検索結果に表れなくなる．なお，キーワード w で検索する際に，削除したドキュメントが w の検索結果に含まれるかどうかの情報はサーバに漏れてしまう．情報漏えいを防ぐためには，検索時に $G_{K_2}(w)$ の代わりに $F_{K_3}(G_{K_2}(w)) \wedge e$ を送信し，この値をビット列を復号する際に用いる必要がある．この検索方法を Secure Search と呼称する．

追加処理の際，ユーザはまず追加するドキュメント D_{m+1} を検索結果に持つキーワードの一覧 $w_{add} = (w1, w2, ..., wa)$ を用意する．(m はドキュメント数．ただし，今までに削除したドキュメントの数も含む．) 次にそれらのキーワードをすべて擬似ランダム置換 P_2 で暗号化した後サーバへ送信する (7. AddToken1 での処理)．サーバ側は受け取った値 $(P_{2K_1}(w1), P_{2K_1}(w2), ..., P_{2K_1}(wa))$ とペアになる値を T_2 から取り出してユーザへ送る (8. AddToken2 での処理)．これらの値は A_1 における w_{add} の各要素の検索結果を記録したビット列の格納箇所のアドレスを暗号化したものである．ユーザは受け取ったアドレス $(addrs_{w1}, addrs_{w2}, ..., addrs_{wa})$ を復号した後，擬似ランダム関数 E で追加するドキュメントの id 番号 $m+1$ を暗号化する．次に，$E_{K_5}(m+1)$ の $addrs_{w1}$ ビット目, $addrs_{w2}$ ビット目, ... , $addrs_{wa}$ ビット目 をそのビットの値と 1 との排他的論理和に置き換え $E_{K_5}(m+1)'$ とする．その後，$E_{K_5}(m+1)'$ をサーバへ送信する (9. AddToken3 での処理)．サーバは，A_2 の先頭に格納されているビット列 $A_2[1]$ を取り出した後，A_1 に格納されているビット列を更新する．更新は，$A_2[1]$ の i 番目のビットと $E_{K_5}(m+1)'$ の i 番目のビットの排他的論理和を A_1 の i 番目に格納されているビット列の最後に付け足すことで完了する (10. Add での処理)．$A_2[1]$ の i 番目のビットは $F_{K_3}(G_{K_2}(w_i))$ の $m+1$ 番目のビットと $E_{K_5}(m+1)$ の i 番目のビットの排他的論理和，$E_{K_5}(m+1)'$ の i 番目のビットは $E_{K_5}(m+1)$ の i 番目のビットと 0 または 1 （キーワード w_i が D_{m+1} を含むか否か) との排他的論理和である．二つの値の排他的論理和は $F_{K_3}(G_{K_2}(w_i))$ の $m+1$ 番目のビットと 0 または 1（キーワード w_i が D_{m+1} を含むか否か) との排他的論理和なので，この排他的論理和をビット列の最後に付け加えることによって，元のビット列 (長さ m. 各ビットは $F_{K_3}(G_{K_2}(w_i))$ と検索結果 0/1 との排他的論理和) に D_{m+1} の検索結果を付け加えることができる．

4. 実装

ASSE および DSSE を言語 Java8 で実装し，Eclispe と Tomcat で実行した．

4.1. 環境

実行環境は以下の通り

表 1　実行環境

プロセッサ	Intel(R) Core(TM) i7-3770 CPU 3.40GHz
メモリ	16.0 GB
OS	Windows 8.1 Enterprise
言語	Java 8
Eclispe	4.4
Tomcat	8.0.33

4.2. 結果

m はドキュメント数，m' は追加可能ドキュメント数，n はキーワード数を表す．

表 4　ASSE 実行時間 : ドキュメント数別 (キーワード数 : 10, 000)

$m+m'$	search	secure search	delete	add
2,000	3.0 ms	2.9 ms	3.6 ms	71 ms
20,000	3.5 ms	3.3 ms	3.9 ms	81 ms
200,000	3.3 ms	3.9 ms	3.6 ms	114 ms

表 2　ASSE インデックスサイズ

$(m+m') \times n$	size (MByte)
20,000,000	2.8
200,000,000	24.2
2,000,000,000	197.5

表 3　ASSE 実行時間：キーワード数別 (m : $10,000, m'$: $10,000$)

n	search	secure search	delete	add
1,000	3.2 ms	3.3 ms	3.4 ms	17 ms
10,000	3.5 ms	3.3 ms	3.9 ms	81 ms
100,000	3.3 ms	3.2 ms	3.4 ms	610 ms

表 5　DSSE 実行時間：1 キーワードに該当するドキュメント数別

documents per keyword	search
10	6.9 ms
100	9.2 ms
1,000	19.9 ms

DSSE の検索処理は，キーワードの検索結果を格納した暗号化線形リストを復号する処理であるため，線形リストの長さ（1 つのキーワードに該当するドキュメント数）に比例して実行時間が長くなる．

表 6　DSSE 実行時間 1 ドキュメントに該当するキーワード数別

keywords per document	delete	add
10	3.8 ms	9.8 ms
100	5.9 ms	19.9 ms
1,000	32.9 ms	62.6 ms

DSSE の削除・追加処理は，削除 / 追加するドキュメントに含まれるキーワードの数だけ線形リストのノードを削除 / 追加する処理であるため，1 つのドキュメントに該当するキーワード数に比例して実行時間が長くなる．

ASSE ではキーワード数およびドキュメント数が増えても，検索処理と削除処理は 4 ミリ秒以内で完了する．また，search と secure search の 2 種類の検索処理は実行時間にほとんど差がないことが分かる．つまり，今回の結果からは安全性の高い secure search のほうが優れていると言える．しかし，secure search のトラップドアサイズはドキュメント数と等しいためドキュメント数が増えるほどトラップドアサイズも大きくなる．一方，search のトラップドアはキーワード 2 つ分なのでドキュメント数やキーワード数の増減に影響されない．ドキュメントの数や通信環境によっては search よりも遅くなる可能性があるため，環境によって使い分ける必要がある．一方，追加処理にはドキュメント数に比例して処理時間が大きく増加し，最大で 610 ミリ秒かかった．しかし，これはキーワード数 1 万，ドキュメント数 10 万の場合であり，処理時間も 1 秒未満なので十分に実用に足ると考えられる．

ASSE の各処理はインデックスの構造上ドキュメント数とキーワード数の増加に比例して実行時間も増大する．一方 DSSE の検索処理はキーワードの検索結果に含まれるドキュメントの数に，削除処理と追加処理は 1 つのドキュメントに含まれるキーワードの数に比例して実行時間が長くなる．今回の実装では，それぞれの実行時間に比例する値を増加させて処理時間の変化を調べた．検索・削除処理に関しては，ASSE では値が増加しても処理時間はほぼ一定だが，DSSE では値が増えると実行時間も増加している．また，いずれの場合でも ASSE のほうが DSSE よりも実行時間が短い．逆に，追加処理に関しては，DSSE の方が実行時間が短い．これは，ASSE の追加処理のトラップドアサイズが DSSE より大きいことと，1 回の追加処理のなかで DSSE はユーザーとサーバ間の通信を 1 回しか行わないが，ASSE は 2 回通信を行うことが原因だと考えられる．しかし，DSSE の追加処理は追加するドキュメントに該当するキーワードの一覧が漏洩するリスクがあるため，ASSE の追加処理が DSSE の追加処理に安全性の面で優っている．以上のことから，ASSE は DSSE よりも実用的であると言える．

5. 結論

本論文では，インデックスの更新に対応した SSE の新方式である ASSE を提案した．また，ASSE を実装し，結果をもとに既存方式との比較を行い，ASSE を評価した．

既存方式と比較して，ASSE は検索・削除処理の実行時間で優れており，追加処理でも安全性で優っている．このことから，ASSE は既存方式よりも実用的であると言える．また，ドキュメントやキーワード数が 10,000 以上でも 100 ミリ秒とかからずに各処理が実行できることからも ASSE は実用的であると言える．

文　献

[1] R. Curtmola，J. Garay，S. Kamara，R. Ostrovsky，Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions，ACM Conference on Computer and Communications Security 2006，pp.79-88，Alexandria，U.S.A，2006.

[2] S. Kamara，C. Papamanthou，T. Roeder，Dynamic Searchable Symmetric Encryption，ACM Conference on Computer and Communications Security 2012，pp.965-976，Raleigh，U.S.A，2012.

[3] S. Kamara，C. Papamanthou，Parallel and Dynamic Searchable Symmetric Encryption，International Conference on Financial Cryptography and Data Security，pp.258-274，Okinawa，Japan，2013.