並列処理かつ動的データに向けた検索可能暗号の改良

三好 竜司[1]　山本 博章[2]

概要：近年、暗号化したデータに対して検索が可能なシステムが必要とされおり、より効率の良い手法の開発が進められている．過去に我々が提案した手法 [1] は、単純な配列を用いて索引を構成することで、索引サイズを大幅に削減する事ができた．しかし、この手法はリスト構造を用いてキーワードごとのデータを管理していた為、並列処理で検索することが不可能であった．そこで、本論文では、2 分木を用いてデータを管理することで、並列処理を可能にし、さらに、暗号化索引作成後にクライアントがドキュメントを追加できるよう改良した手法を提案する．本手法は、ランダムオラクルモデルにおいて適応的安全性、さらに、追加ドキュメントに対する安全性を満たしており、サーバに保存する暗号化索引のサイズは、セキュリティパラメータに依存しない．

キーワード：検索可能暗号，並列処理，動的データ

Improved Searchable Symmetric Encryption for Parallel Processing and Dynamic Data

Ryuji Miyoshi[1]　Hiroaki Yamamoto[2]

Abstract: From a view point of information security, researches on an encrypted search system have been done intensively. Such search systems are called symmetric searchable encryption (SSE). The main part of SSE is an encrypted index which affects security and efficiency. The existing SSE schemes use a Bloom filter or an inverted index to construct an encrypted index. A Bloom filter is simple data structure, while an inverted index is implemented by an advanced data structure such as hash or dictionary. In the previous paper[1], we proposed a new secure search scheme using Bloom filters and simple array, that is, an array of integers and an array of strings. Consequently, we can greatly reduce the index size, but in parallel because it uses list structure to build the index. In this paper, we propose an improved scheme which be able to search in parallel by building the index using binary tree. In addition, the scheme is improved for Dynamic SSE (DSSE) whichi can add documents.

Keywords: symmetric searchable encryption, parallel, dynamic data

1. はじめに

近年、メールサービスやストレージサービスといったクラウドコンピューティングの普及によって、クライアントがサーバに自分のデータを保存する機会が増加してきた．このようなサービスでは、信頼できないサーバに保存する際には、データの内容を隠す為、データを暗号化して保存する必要がある．しかし、この場合、クライアントはサーバに保存したデータに対し、サーバ上での検索は不可能となる．その為、暗号化したデータに対し、検索が可能なシステムが必要とされている．このようなシステムのことを検索可能暗号といい、近年、活発に研究が行われている [2][3][4][6][5][1]．検索可能暗号は、共通鍵暗号、もしくは、公開鍵暗号、どちらの方式を用いるかで区別されている．我々は、共通鍵暗号方式を用いた検索可能暗号の研究を行っている．本論文では、クライアントがデータ保有者とし、このデータの事をドキュメントと呼ぶ．サーバには、

[1] 信州大学院
Graduate school, Shinshu University
[2] 信州大学
Shinshu University

暗号化したクライアントのドキュメントが保存され、クライアントは、検索したいキーワードを暗号化してサーバに送信し、サーバから検索結果を受け取る．この時、サーバはアルゴリズム通りに動く．本論文の目的は、時間的、空間的により効率的な手法の開発である．

1.1 本論文の結果

今回我々が提案手法する手法は、三好 [1] の手法を改良した手法である．三好 [1] の手法は、ブルームフィルタと転置索引を組み合わせた手法であり、これによって索引サイズを大幅に削減することができた．しかし、索引である配列に、各キーワード w に対するデータをリスト構造で連結し、保存していた．その為，検索時、w に関するデータを一つずつ取得する必要があり、並列処理で検索を行うことができなかった．そこで、本論文では、2 分木を用いてデータを管理することで、並列処理を可能にし、さらに、暗号化索引作成後にクライアントがドキュメントを追加できるよう改良した手法を提案する．提案手法の性質は以下の通りである．

- 暗号化索引に登録したキーワード数を m、プロセッサ数を p とすると、任意のキーワード w を検索したときの検索時間は $O(\log p + \frac{n_w}{p})$ である．この時、n_w は w にマッチするドキュメント数である．つまり、並列処理で検索が可能である．
- サーバに保存する暗号化索引のサイズは $O(\Sigma_w\, n_w)$ であり、クライアントが保持する暗号化索引のサイズはサイズは $O(m)$ である．
- サーバに保存する暗号化索引のサイズは、セキュリティパラメータに依存しない．つまり、安全性を高める為にセキュリティパラメータを大きくしても、索引サイズは変化しない．
- ドキュメントの追加が可能である．
- ランダムオラクルモデルにおいて適応的安全性を満たし、追加するドキュメントに対する安全性も満たしている．

1.2 関連研究

検索可能暗号に関して、最初に Goh[2] がブルームフィルタを用いた検索法を提案した．彼の手法は、一つのドキュメントに含まれるすべてのキーワードを一つのブルームフィルタに登録していく．その為、検索の際はすべてのブルームフィルタを検索する必要があり、検索効率が悪い．しかし、この手法はブルームフィルタで構成されている為、索引サイズがセキュリティパラメータに依存しない．その後、Curtomola[3] は、転置索引を用いた手法を提案した．彼らは、Goh[2] のセキュリティモデルの欠点を指摘し、新たに非適応的，適応的安全性に関するセキュリティモデルを導入した．彼らは、これらのモデルに基づいて，非適応的安全な検索可能暗号 (SSE-1) 及び適応的安全な検索可能暗号方式 (SSE-2) を提案した．Curtomola[3] の手法 (SSE-2) は、ドキュメント ID とトラップドアをセットで保存する必要があった．トラップドアはキーワードをハッシュ関数になどに入力した値である為、セキュリティパラメータを大きくするとハッシュ関数の出力長が大きくなり、結果として索引サイズが大きくなってしまう．Kamara[4] は，ドキュメントの追加や削除にも対応する手法を提案した．この手法は、Curtomola らの手法を拡張したものであり、ランダムオラクルモデルにおいて適応的安全であり、索引サイズが、Curtomola[3] の手法 (SSE-1) と同等である．しかし、この手法も索引サイズがセキュリティパラメータに依存する．Kamara[4] 以降、ランダムオラクルモデルにおいて適応的安全性を満たす DSSE が多く提案され、追加するドキュメントに対しの安全性を満たす手法 [6][5] や、削除するドキュメントに対して安全性を満たす手法 [6] なども提案されている．

我々は、三好 [1] の手法を改良することで、ランダムオラクルモデルにおいて適応的安全性、更に、追加するドキュメントに対する安全性も満たしている DSSE を提案する．提案手法は、三好 [1] ではできなかった並列処理での検索、ドキュメントの追加を可能としているが、索引サイズはセキュリティパラメータに依存しないままである．しかし、クライアントが保持するデータサイズが増えてしまった．

これまでの手法と提案手法を比較した表を表 1 に示す．この表のクライアントデータとは、クライアントが保持しておくデータのサイズの事である．

2. 準備

$a||b$ を文字列 a と b を連結した文字列、集合 d において $|d|$ を d の要素数、更に、$\lfloor x \rfloor$ を x 以下の最大の整数とする．本論文において、d はドキュメント、$D = \{d_0, ..., d_{n-1}\}$ はドキュメントの集合を表している．また、$\mathtt{id}$ はドキュメント ID であり、$D(w) = \{\mathtt{id}_0, ..., \mathtt{id}_{n_w-1}\}$ はキーワード w を含むドキュメントの ID の集合である．更に、$K(D)$ は D に含まれる異なるキーワードの集合であり、$m = |K(D)|$ とする．本手法では、トラップドアの作成に疑似ランダム関数 F_{sk}、暗号化索引の作成にランダムオラクル H、ドキュメントの暗号化に CPA-安全性である共通鍵暗号 Enk_{sk} を使用する．この時、sk は使用される秘密鍵である．

3. 提案手法の定義

提案手法 $DSSE\_BT$ は，$DSSE\_BT$ = (Gen, Enc, Trapdr, BuildIndex, Search, Dec, MakeUpdateToken, Updata, MakeAddToken, Add) の 10 個の多項式時間アルゴリズムで構成されている．各アルゴリズムのは以下の役割を果たす．

定義 1

表 1　提案手法と他の手法の比較

手法	索引サイズ	クライアントデータ	検索時間	並列	追加
Curtomola[??](SSE-2)	$O(\lambda n s_{max})$	$O(1)$	$O(\frac{n_w}{p})$	Yes	No
Kamara[??]	$O(\Sigma_w \lambda n_w)$	$O(1)$	$O(n_w)$	No	Yes
Etemad[??]	$O(\Sigma_w \lambda n_w)$	$O(m)$	$O(\frac{n_w}{p})$	Yes	Yes
Miyoshi[1]	$O(\Sigma_w n_w)$	$O(1)$	$O(\log m + n_w)$	No	No
提案手法	$O(\Sigma_w n_w)$	$O(m)$	$O(n_a + \log p + \frac{n_w - n_a}{p})$	Yes	Yes

λ : セキュリティパラメータ, m : キーワード数, n : ドキュメント数, n_w : w を含むドキュメント数, p : プロセッサ数, s_{max} : 一番多くのキーワードを持つドキュメントのキーワード数, n_a : 検索されずに w にドキュメントが追加された回数.

- $SK \leftarrow \mathsf{KeyGen}(1^\lambda)$: クライアントによって実行され、セキュリティパラメータ λ を入力すると、秘密鍵 $SK = \{sk_1, sk_2\}$ を出力する.
- $\mathbf{c} \leftarrow \mathsf{Enk}(sk_1, D)$: クライアントによって実行され、秘密鍵 sk_1 とドキュメント集合 D を入力すると、暗号化されたドキュメント $\mathbf{c}$ を出力する. このアルゴリズムを $\mathsf{Enk}_{sk_1}(D)$ と書く.
- $(I_s, I_c) \leftarrow \mathsf{BuildIndex}(sk_2, D, \varepsilon)$: クライアントによって実行され、秘密鍵 sk_2、ドキュメント集合 D と暗号化索引サイズ ε を入力すると、暗号化索引 I_s、I_c を出力する. このアルゴリズムを $\mathsf{BuildIndex}_{sk_2}(D)$ と書く.
- $t \leftarrow \mathsf{Trapdr}(sk_2, w, SCnt)$: クライアントによって実行され、秘密鍵 sk_2、検索キーワード q、q の検索回数 $SCnt$ を入力すると、トラップドア t を出力する. このアルゴリズムを $\mathsf{Trapdr}_{sk_2}(w, SCnt)$ と書く.
- $\mathbf{c} \leftarrow \mathsf{Search}(I_s, t)$: サーバによって実行され、暗号化索引 I_s、トラップドア t を入力すると、暗号化ドキュメントの集合 $\mathbf{c}$ を出力する.
- $d \leftarrow \mathsf{Dec}(SK, c)$: クライアントによって実行され、秘密鍵 sk_1 と暗号化ドキュメント c を入力すると、ドキュメント d を出力する.
- $\mu \leftarrow \mathsf{MakeUpdateToken}(sk_2, D(w), w, I_c, \Phi)$: クライアントによって実行され、秘密鍵 sk_2、ドキュメント ID 集合 $D(w)$、w、暗号化索引 I_c、I_s の空要素のアドレスの集合 Φ を入力すると、更新トークン μ を出力する.
- $I_s^\# \leftarrow \mathsf{Updata}(I_s, \mu)$: サーバによって実行され、暗号化索引 I_s と更新トークン μ を入力すると、更新された暗号化索引 $I_s^\#$ が出力される.
- $\pi \leftarrow \mathsf{MakeAddToken}(SK, d_a, I_c, \Phi)$: クライアントによって実行され、秘密鍵 SK、追加するドキュメント d_a、暗号化索引 I_c と Φ を入力すると、追加トークン π を出力する. このアルゴリズムを $\mathsf{MakeAddToken}_{SK}(d_a)$ と書く.
- $I_s^\# \leftarrow \mathsf{Add}(I_s, \pi)$: サーバによって実行され、暗号化索引 I_s と追加トークン π を入力すると、更新された暗号化索引 $I_s^\#$ が出力される.

4. 暗号化索引の構成

DSSE_BT の暗号化索引 (I_s, I_c) は、検索結果のドキュメント ID が保存されている I_s と、各キーワードに関する情報が保存されいる I_c で構成されており、I_s はサーバが、I_c はクライアントが保持する. I_s は整数型配列、I_c は文字列型と 3 つの整数をセットで保存するようなデータ構造で構成する. I_s と I_c はアルゴリズム 1 を実行する事で得ることができる.

4.1 I_s の構成

I_s には、$K(D)$ に含まれるすべてのキーワード $w \in K(D)$ に関する、$D(w) = \mathtt{id}_0, ..., \mathtt{id}_{n_w - 1}$ が保存されている。各 $D(w)$ に含まれる id は、2 分木で管理されており、この木の事を ID 木と呼ぶ。ID 木は 2 分木である為、葉ノード以外の各ノードには左子ノードと右子ノードが存在し、各ノードには 1 つのドキュメント ID が格納されている. その結果、木の高さは $height = \lfloor n_w \rfloor$ となる。各ノードには階層番号 lev とノード ID$nodeID$ が割り当てられている. 根ノードから各ノードの深さを $depth$ とすると、$lev = height - depth$ である. つまり、根ノードの階層番号は $height$ で、一番深い位置の葉ノードの階層番号は 0 となる. また、あるノードのノード ID を x とすると、そのノードの左子ノードの ID は $2 \times x$、右子ノードの ID は $2 \times x + 1$ と定義する. そして、根ノードの ID を 0 として各ノードに ID を割り当てる. 例として、$n_w = 15$ の ID 木を図 1 に示す. I_s には、$w \in K(D)$ に関する ID 木のすべてのノードがランダムな位置に格納されている. T_w^{SCnt} を w のトラップドア t に格納されているランダム値とする. w の ID 木の各ノードに関して、$\mathtt{N}_{addr}$ をノードアドレス、L を左子ノードアドレス、R を右子ノードアドレスとすると、$I_s[\mathtt{N}_{addr}] = (\mathtt{id}, L, R) \oplus H(T_w^{SCnt} || lev || nodeID)$ のようにして、I_s が構成されている.

4.2 I_c の構成

I_c には、$K(D)$ に含まれるすべてのキーワード $w \in K(D)$ に関して、これまでの検索回数 $SCnt$、ID 木の根ノードア

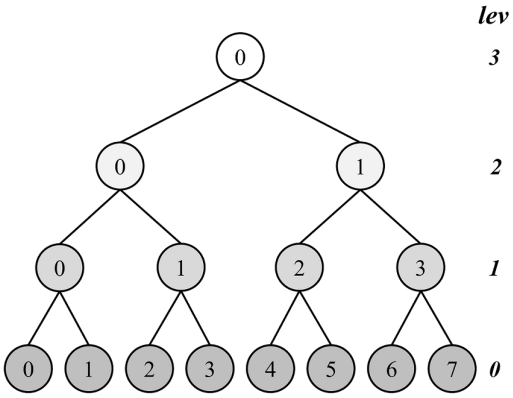

図 1 $n_w = 15$ の時の ID 木

ドレス $start$、ID 木の高さ ht が保存されており、w を入力する事でこれらの値にアクセスすることができる．つまり、$I_c[w] = \{SCnt, start, ht\}$ のようにして、I_c が構成されている．

5. 検索

検索は、クライアントが検索キーワード w と I_c からトラップドア $t = \{T_w^{SCnt}, I_c[w].start, T_c[w].ht\}$ を作成する．この時、$T_w^{SCnt} = F_{sk_1}(w||I_c[w].SCnt)$ である．この t を、サーバに送信する事で実行される．サーバは受け取った t と I_s を用いて検索を行い、$D(w)$ に一致する暗号化ドキュメントの集合 $\mathbf{c}$ をクライアントに返す．その後、I_s の $D(w)$ に関するデータを再構築する為に、クライアントが更新データを作成し、サーバに送信して更新を行う．

これより、アルゴリズム 2 について説明する．サーバは $\mathbf{t}$ から、T_w^{SCnt}、ID 木の根ノードアドレス $start$ と ID 木の高さ $height$ を取得し、$lev = height$ とする．$next$ には、次のノードのアドレス $\mathtt{N}_{addr}$ とノード ID$nodeID$ が保存されており、根ノードに関する $(start, 0)$ を $next$ に追加することで検索を開始する．$I_s[start] \oplus H(T_w^{SCnt}||lev||nodeID)$ を計算することで、$(\mathtt{id}_0, L, R)$ を取得する．次に、lev をデクリメントし、L と R の値が -1 でない場合、$(L, nodeID \times 2)$、$(R, nodeID \times 2 + 1)$ を $next$ に追加して、先ほどと同じ操作を繰り返す．この操作を $next$ の要素がなくなるまで繰り返し、$D(w) = \{\mathtt{id}_0, ..., \mathtt{id}_{\mathtt{n}_w}\}$ を取得する．この処理に関して、親ノードに対して 2 つの子ノードの処理は並列に行うことが可能である．つまり、$next$ に含まれる要素はすべて並列に処理することが可能である．そして、$D(w)$ に一致する暗号化ドキュメントの集合 $\mathbf{c}$ をクライアントに送信する．

これより、アルゴリズム 3 について説明する．$\mathbf{c}$ を受け取ったクライアントは、$D(w)$ そして I_s の空いている要素のアドレス集合 Φ をサーバから取得する．その後、$I_c[w].SCnt$ をインクリメントし、アルゴリズム 1 と同じように $D(w)$ に関する ID 木を作成する．そして、各ノードのアドレス $\mathtt{N}_{addr}$ と更新データ $(\mathtt{id}, L, R) \oplus H(T_w^{SCnt}||lev||nodeID)$ の集合 μ を作成し、$I_c[w].ht$ と $I_c[w].start$ を更新する．その後、μ をサーバに送信し．サーバは、受け取った μ を元に I_s を更新する．

6. 追加

追加は、クライアントが追加ドキュメント d_a と I_c から追加トークン π を作成し、サーバに送信する事で実行される．サーバは受け取った π を元に、I_s を更新する．

これより、アルゴリズム 4 について説明する．クライアントは、追加したいドキュメント d_a を暗号化して c_a を作成する．次に、d_a に含まれるすべての $w \in d_a$ に関して、I_s に保存されている $D(w)$ に $\mathtt{id}_a$ を追加していく．追加するノードは、追加する前の ID 木の根ノードの上に追加していく．つまり、追加するノードの左子ノードアドレスを以前の根ノードアドレス $I_c[w].start$、右子ノードアドレスを -1、階層番号を $I_c[w].ht+1$、ノード ID を 0 とし、追加ノードを作成する．よって、d_a に含まれているすべての w について、$(\mathtt{id}_a, I_c[w].start, -1) \oplus H(T_w^{SCnt}||I_c[w].ht + 1||0)$ を作成する．サーバから I_s の空いている要素のアドレス集合 Φ を取得し、Φ からランダムに $free$ を選択し、$free$ とセットで先ほどの追加ノードのデータを π に追加していく．そして、クライアントは、$I_c[w].start = free$ にし、$I_c[w].ht$ をインクリメントする．このようにして作成した Φ をサーバに送信し、サーバは Φ のデータを元に I_s を更新する．

7. セキュリティの解析

提案手法が、ランダムオラクルモデルにおいて適応的安全性を満たす事を示す．その為に、安全性を定義し、この定義を提案手法が満たす事を証明していく．まず初めに、無視可能関数を定義する．

定義 2 関数 f が、セキュリティパラメータ λ に対して無視可能であるとは、すべての多項式 $poly()$ と十分に大きい λ に対して、$f(\lambda) < \frac{1}{poly(\lambda)}$ になるときである．

7.1 漏れ関数の定義

安全性を定義する前に、提案手法の漏れ関数を定義する．漏れ関数とは、各動作を実行した際にサーバに漏れてしまう情報を定義したものであり、$\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2$ を定義する．

定義 3

- $(n, N, |d_0|, ..., |d_{n-1}|, |\Phi|) \leftarrow \mathcal{L}_0(D)$: ドキュメント集合 D を入力すると、ドキュメント数 n、I_s のサイズ N、各ドキュメントのサイズ $|d_0|, ..., |d_{n-1}|$、空いている要素数 $|\Phi|$ を出力します．
- $(n_w, D(w), SP) \leftarrow \mathcal{L}_1(D, w)$: D と検索キーワード w を入力すると、n_w、$D(w)$ そして、過去の検索キーワー

Algorithm 1 $(I_s, I_c) \leftarrow \mathsf{BuildIndex}(SK, D, \varepsilon)$

$m = |K(D)|$
I_s の上限 $N = \Sigma_w n_w + \varepsilon$ を設定する．
I_s, I_c を初期化する．
for all $w \in K(D)$ **do**
  $I_c[w].SCnt = 0$
  $T_w \leftarrow F_{sk_2}(w||I_c[w].SCnt)$
  $D(w) = \{\mathtt{ID}_0, ..., \mathtt{ID}_{n_w}\}$ とする．
  未使用の I_s の要素のアドレス $free$ を取得
  $I_c[w].start = free$
  $height = \lfloor \log n_w \rfloor$
  $I_c[w].ht = height$
  $lev = height$
  $nodeID = 0$
  for $j = 0$ to $n_w - 1$ **do**
    if $lev == 2^{(height-lev)}$ **then**
      $lev--$
      $nowID = 0$
    end if
    $nodeCnt = 2^{(height-lev+1)} + (2 \times nowID)$
    if 左子ノードが存在する場合 **then**
      未使用の I_s の要素のアドレス $free$ を取得
      $L = free$
    else
      $L = -1$
    end if
    if 右子ノードが存在する場合 **then**
      未使用の I_s の要素のアドレス $free$ を取得
      $R = free$
    else
      $R = -1$
    end if
    if $j == 0$ **then**
      $\mathtt{N}_{addr} = I_c[w].start$
    else
      親ノード作成時にこのノードに割り当てたアドレスを $\mathtt{N}_{addr}$ とする．
    end if
    $I_s[\mathtt{N}_{addr}] = (\mathtt{ID}_j, L, R) \oplus H(X||lev||nodeID)$
    $nowID++$
  end for
end for
output (I_s, I_c)

ドの集合 SP を出力する．

- $(|d_a|, m_a) \leftarrow \mathcal{L}_2(d_a)$ 追加ドキュメント d_a を入力すると、d_a のサイズ $|d_a|$ と d_a に含まれているキーワード数 m_a が出力される．

7.2 安全性の定義

適応的安全性の証明に向けて、2 つのゲーム $(\mathbf{Real}_{\mathcal{A}}, \mathbf{Sim}_{\mathcal{A},\mathcal{S}})$ を定義する．このゲームの参加者は攻撃者 $\mathcal{A}$、挑戦者 $\mathcal{C}$、シミュレータ $\mathcal{S}$ である．$\mathbf{Real}_{\mathcal{A}}$ は提案手法を実行する．$\mathbf{Sim}_{\mathcal{A},\mathcal{S}}$ は、$\mathbf{Real}_{\mathcal{A}}$ の暗号化索引、トラップドアを模倣する．

$\mathbf{Real}_{\mathcal{A}}$

最初、$\mathcal{C}$ は $\mathsf{KeyGen}(1^{\lambda})$ を実行し、SK を生成する．次に、$\mathcal{A}$ はドキュメント集合 D を出力する．その後、$\mathcal{C}$ が $\mathsf{Enc}_{sk_1}(D)$ と $\mathsf{BuildIndex}_{sk_2}(D)$ を実行することで、$\mathbf{c}$、I_c、I_s を生成し、

Algorithm 2 $D(w) \leftarrow \mathsf{Search}(I_s, t)$

$(T_w^{SCnt}, start, height) \leftarrow t$
$lev = height$
$next$ に $(start, 0)$ を追加
while $lev \geq 0$ **do**
  for all $(\mathtt{N}_{addr}, nodeID) \in next$ **do**
    $(\mathtt{id}, L, R) = I_s[\mathtt{N}_{addr}] \oplus H(T_w^{SCnt}||lev||nowID)$
    $D(w)$ に $\mathtt{id}$ を追加
    if $L \neq -1$ **then**
      $nextTemp$ に $(L, nodeID \times 2)$ を追加
    end if
    if $R \neq -1$ **then**
      $nextTemp$ に $(R, nodeID \times 2 + 1)$ を追加
    end if
    Φ に $\mathtt{N}_{addr}$ を追加
  end for
  $next = nextTemp$
  $nextTemp$ を初期化する．
end while
output $D(w)$

Algorithm 3 $\mu \leftarrow \mathsf{MakeUpdateToken}(D(w), \Phi, I_c, w, SK)$

$I_c[w].SCnt{+}{+}$
$T_w^{SCnt} \leftarrow F_{sk_2}(w||I_c[w].SCnt)$
Φ からアドレス $free$ を取得
$I_c.start = free$
$height = \lfloor \log n_w \rfloor$
$I_c[w].ht = height$
$lev = height$
$nodeID = 0$
for $j = 0$ to $n_w - 1$ **do**
  if $lev == 2^{(height-lev)}$ **then**
    $lev--$
    $nodeID = 0$
  end if
  $nodeCnt = 2^{(height-lev+1)} + (2 \times nodeID)$
  if 左子ノードが存在する場合 **then**
    Φ から未使用のアドレス $free$ を取得
    $L = free$
  else
    $L = -1$
  end if
  if 右子ノードが存在する場合 **then**
    Φ から未使用のアドレス $free$ を取得
    $R = free$
  else
    $R = -1$
  end if
  if $j == 0$ **then**
    $\mathtt{N}_{addr} = I_c[w].start$
  else
    親ノード作成時にこのノードに割り当てたアドレスを $\mathtt{N}_{addr}$ とする．
  end if
  μ に $(\mathtt{N}_{addr}], (\mathtt{ID}_j, L, R) \oplus H(X||lev||nodeID))$ を追加
  $nodeID{+}{+}$
end foroutput μ

Algorithm 4 $\pi \leftarrow \mathsf{MakeAddToken}(d_a, \Phi, I_c, SK)$

1: **for all** $w \in K(d)$ **do**
2: Φ から未使用のアドレス $free$ を取得
3: $T_w^{SCnt} \leftarrow F_{sk_2}(w||I_c[w].SCnt)$
4: $I_c[w].ht{+}{+}$
5: $lev = I_c[w].ht$
6: $L = I_c[w].start$
7: $R = -1$
8: π に $(frer, (\mathtt{id}, L, R) \oplus H(T_w^{SCnt}||lev||0))$ を追加
9: **end for**
10: output π

$\mathcal{A}$ に送信する．次に、$\mathcal{A}$ が、検索または追加を行う．検索の場合、検索キーワード w を選択し、$\mathcal{C}$ から $\mathsf{Trapdr}_{sk_2}(w)$ によって生成された t を取得する．追加の場合、追加するドキュメント d_a を選択し、$\mathcal{C}$ から $\mathsf{MakeAddToken}_{SK}(d_a)$ によって生成された π を取得する．その後、$\mathcal{A}$ は適応的に検索か追加を、先ほどと同様に実行する．この操作を多項式回 $\mathsf{poly}(\lambda)$ 実行する．最後に、ビット $b \in 0,1$ を出力する．

$\mathbf{Sim}_{\mathcal{A},\mathcal{S}}$:

最初、$\mathcal{C}$ は $\mathsf{KeyGen}(1^\lambda)$ を実行し、SK^* を生成し、S に送信する．$\mathcal{A}$ はドキュメント集合 D を作成し、$\mathcal{C}$ に送信する．$\mathcal{C}$ は、$\mathcal{S}$ に $\mathcal{L}_0(D)$ のみを伝え、$\mathcal{S}$ は $\mathbf{c}^*$、$\mathbf{I_s}^*$、$\mathbf{I_c}^*$ を生成する．そして、$\mathbf{c}^*$ と I_s^* を $\mathcal{A}$ に送信する．次に、$\mathcal{A}$ が、検索または追加を行う．検索の場合、検索キーワード w を選択し、$\mathcal{C}$ に送信する．$\mathcal{C}$ は、$\mathcal{S}$ に $\mathcal{L}_1(D,w)$ のみを伝え、$\mathcal{S}$ は t^* を生成する．追加の場合、$\mathcal{A}$ は追加するドキュメント d_a を選択し、$\mathcal{C}$ に送信する．$\mathcal{C}$ は、$\mathcal{S}$ に $\mathcal{L}_2(d_a)$ のみを伝え、$\mathcal{S}$ は π^* を生成する．その後、$\mathcal{A}$ は適応的に検索か追加を、先ほどと同様に実行する．この操作を多項式回 $\mathsf{poly}(\lambda)$ 実行する．最後に、ビット $b \in 0,1$ を出力する．

定義 3 もし検索システムが次の条件を満たすならば，それは適応的安全な検索システムである．確率的多項式時間で動作するすべての攻撃者 A に対し, 次の式を満足する確率的多項式時間シミュレータが存在する．$|Pr(\mathbf{Real}_{\mathcal{A}}$ で $\mathcal{A}$ は b=1 を出力) − Pr($\mathbf{Sim}_{\mathcal{A},\mathcal{S}}$ で $\mathcal{A}$ は b=1 を出力)| は無視可能である．

定理 提案手法 *DSSE_BT* は、適応的安全性を満たしている．

証明 $\mathbf{Real}_{\mathcal{A}}$ と区別がつかない $\mathbf{Sim}_{\mathcal{A},\mathcal{S}}$ を構築していく．

- $\mathbf{c}$: $\mathcal{L}_0(D)$ より、$|d_0|, ..., |d_{n-1}|$ を知っている為、同じ長さのランダムなビット列 $r_0, ..., r_{n-1}$ を用意し、D^* とし、$\mathbf{Real}_{\mathcal{A}}$ と同様に、$\mathbf{c}^* = Enc_{sk1^*}(D^*)$ を作成する．Enk_{sk} は CPA-安全性である為、$\mathbf{c}$ と $\mathbf{c}^*$ は区別できない．
- I_s: $\mathcal{L}_0(D)$ より、I_s のサイズ N、Φ を知っている．まず、サイズ N の I_s^* を準備し、$N - |\Phi|$ 個ランダムに要素を選択し、ランダム値を格納する．選択されなかった要素のアドレス集合を Φ とする．$\mathbf{Real}_{\mathcal{A}}$ の I_s の空でない要素には、$(\mathtt{id}, L, R) \oplus H(T_w^{SCnt}||lev||nodeID)$ が格納されている．ランダムオラクルの入力値 $(T_w^{SCnt}||lev||nodeID)$ は、一つ一つ異なる値である為、$(\mathtt{id}, L, R) \oplus H(T_w^{SCnt}||lev||nodeID)$ は一様ランダムな値である．よって、I_s と I_s^* は区別できない．
- t: $\mathcal{L}_1(D,w)$ より、$(n_w, D(w), SP)$ を知っている．SP より、w が過去に検索・追加動作を実行したかチェックする． (1) もし過去に w を検索がされていた場合、前回使用した w^* を用いて、I_c^* から、$SCnt$、$start$、$height$ を取り出す．$\mathbf{Real}_{\mathcal{A}}$ と同様に、$T_w^{SCnt*} = F_{sk2^*}(w^*||I_c[w^*].SCnt)$ とする． (2) もし、過去に追加のみさていた場合、ランダムな文字列を w^* とし、$I_c[w^*].SCnt = 0$ にする．そして、$\mathbf{Real}_{\mathcal{A}}$ と同様に、$T_w^{SCnt*} = F_{sk2^*}(w^*||I_c[w^*].SCnt)$ とする．次に、I_s^* で空要素でない要素から、未使用の要素を $n_w - y$ 個選択する．そのアドレス、 T_w^{SCnt*}、$D(w)$ の中で追加されたドキュメント以外の集合 $D(w)'$ を用いで ID 木を作成する．$I_s[\mathtt{N}]_{addr}$ に階層番号 lev、ノード ID$nodeID$、左子ノードアドレス L、右子ノードアドレス R、格納するドキュメント ID$\mathtt{id}$ であるノードを保存する場合、ランダムオラクルを $H(T_w^{SCnt*}||lev||nodeID) = I_s[\mathtt{N}]_{addr} \oplus (\mathtt{id}, L, R)$ にセットする．この操作を、すべてのノードに対して行い、作成した ID 木の根ノードのアドレスを top とする．次に、追加された回数を y とすると、$height = \lfloor n_w - y \rfloor + y$ とする．追加したドキュメントのアドレスを最後から順に $a_0, ..., a_{y-1}$ とし、$a_y = top$ する．これらのノードをリスト構造で繋いでいく為、ランダムオラクルを次のように設定していく．$H(T_w^{SCnt*}||height{-}i||0) = (\mathtt{id}_i, a_{i+1}, -1) \oplus I_s[a_i] (0 \leq i \leq y-1)$．そして、$start = y_0$ とし、$start$ と $height$ を $I_c^*[w^*]$ に格納する．(3) もし過去に w が検索・追加されていなかった場合、ランダムな文字列を w^* とし、$I_c[w^*].SCnt = 0$ にする．そして、$\mathbf{Real}_{\mathcal{A}}$ と同様に、$T_w^{SCnt*} = F_{sk2^*}(w^*||I_c[w^*].SCnt)$ とする．I_s^* で空要素でない要素から、未使用の要素を n_w 個選択し、そのアドレス、T_w^{SCnt*}、$D(w)$ を用いで ID 木を作成する．$I_s[\mathtt{N}]_{addr}$ に階層番号 lev、ノード ID$nodeID$、左子ノードアドレス L、右子ノードアドレス R、格納するドキュメント ID$\mathtt{id}$ であるノードを保存する場合、ランダムオラクルを $H(T_w^{SCnt*}||lev||nodeID) = I_s[\mathtt{N}]_{addr} \oplus (\mathtt{id}, L, R)$ にセットする．この操作を、すべてのノードに対して行う．作成した ID 木の根ノードのアドレスを $start$、高さを $height$ とし、$start$ と $height$ を $I_c^*[w^*]$ に格納する．

そして、$t^* = \{T_w^{SCnt*}, start, height\}$ を作成する．$\mathbf{Real}_{\mathcal{A}}$ が t^* を用いて、検索を行った場合、先ほど

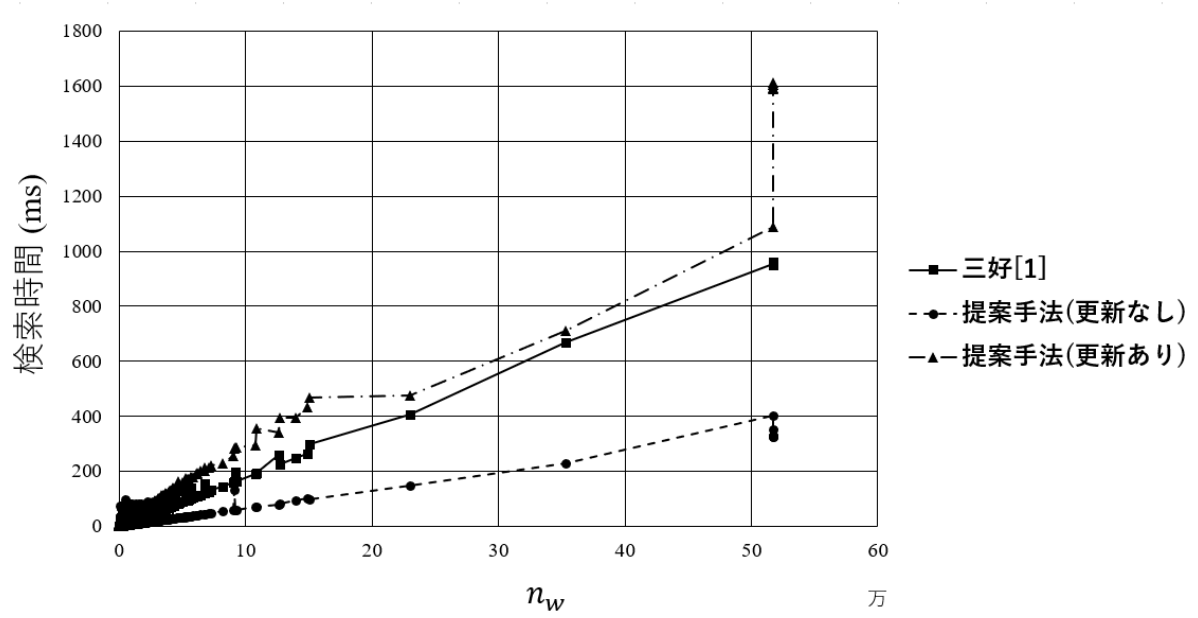

図 2 検索時間の比較

ランダムオラクルを設定した為、t と同じ動作になる．しかし、$\mathcal{S}$ がランダムオラクル $H(x)$ を設定する前に、$\mathcal{A}$ が $H(x)$ にアクセスした場合、t と t^* が識別されてしまう．T_w^{SCnt} の作成に使用する疑似ランダム関数 F_{sk_2} の出力の長さは 2^λ であり、$\mathcal{A}$ は多項式時間で動作する．その為、$\mathcal{A}$ がランダムオラクルに事前にアクセスする確率は $\frac{poly(\lambda)}{2^\lambda}$ である．よって、λ が十分大きければ、$\frac{poly(\lambda)}{2^\lambda} < \frac{1}{poly(\lambda)}$ になる為、この確率は無視できる．よって、t と t^* は区別できない．

- π: $\mathcal{L}_2(d_a)$ より、$(|d_a|, m_a)$ を知っている．$|d_a|$ と同じ長さのランダムなビット列 r_a を作成し、$\mathbf{Real}_\mathcal{A}$ と同様に暗号化して c_a^* を作成する．Φ から未使用の要素アドレスを m_a 個取り出し、ランダム値 r とアドレス $\mathtt{N}_{addr}$ のペア $(r, \mathtt{N}_{addr})$ を m_a 個作成する．作成したすべてのペアと c_a^* を π^* とする．c_a と c_a^* は、Enk_{sk} が CPA-安全性である為、区別できない．また、$\mathbf{Real}_\mathcal{A}$ の π に含まれる I_s に保存されるデータは、$(\mathtt{id}_a, start, -1) \oplus H(T_w^{SCnt}||lev||nodeID)$ である．ランダムオラクルの入力値 $(T_w^{SCnt}||lev||nodeID)$ は、一つ一つ異なる値である為、$(\mathtt{id}_a, start, -1) \oplus H(T_w^{SCnt}||lev||nodeID)$ は一様ランダムな値である．よって、π と π^* は区別できない．

このように、$\mathbf{Sim}_{\mathcal{A},\mathcal{S}}$ を構成すると、$\mathbf{Real}_\mathcal{A}$ と $\mathbf{Sim}_{\mathcal{A},\mathcal{S}}$ は区別できる確率は、無視可能である．よって、提案手法はランダムオラクルモデルにおいて適応的安全である．更に、ドキュメントの追加の際に漏れる情報は、$(|d_a|, m_a) \leftarrow \mathcal{L}_2(d_a)$ より、追加するドキュメントに含まれるキーワード $w \in d_a$ に関する情報は漏れない．よって、追加するドキュメントに対しての安全性も満たしている事がわかる．

8. 実験

今回、提案手法について実験的に評価した為、その結果を示す．実験では、ドキュメント集合として Enron Email データセット [7] を使用した．このデータセットには、51,7431 個のメールデータが保存されており、各メールデータから高々 500 個のキーワードを抽出したデータを使用した．結果として、この実験データの異なるキーワード数は 307,830 個であった．実験に使用した言語は Java8、OS は Windows10、CPU は 4 コア 8 スレッドの i7-6700、メモリは 16GB、共通鍵暗号は AES-256、疑似ランダム関数とランダムオラクルとして SHA256 を使用した．

実験では、並列処理による検索が可能な提案手法と三好 [1] の手法と比較した．実験は、追加操作を行われていない状況で行った．2 つの手法とも、サーバにトラップドア t を入力してから検索結果の $D(w) = \{\mathtt{id}_0, ..., \mathtt{id}_{n_w-1}\}$ が出力されるまでの時間を計測し、比較した．また、提案手法は検索後に I_s の更新が必要である為、クライアントが更新トークン μ を作成し、サーバが I_s を更新するまでの時間も計測した．また、検索時間には、クライアント・サーバ間の通信時間は含まれていない．実験の結果を図 2 に示す．今回の実験では、並列処理を ID 木の階層ごとで実行した．つまり、同じ階層のノードについて並列処理し、その階層の処理が終了したら次の階層の処理に移るようにプログラムを組んだ．結果より、提案手法は検索時間は 3 分の 1 程度になった事がわかった．これより、提案手法は並列処理で高速化が可能である事が分かった．しかし、検索後に実行される I_s 更新の時間を加えると、三好 [1] の手法よりも遅くなってしまった．これは、ID 木の再構築に

時間がかかる為だと思われる．

9. まとめ

本論文では、三好 [1] の手法を、並列処理・動的データ向けに改良した手法を提案した．提案手法は、ランダムオラクルモデルにおいて適応的安全性、追加するドキュメントに対する安全性を満たしており、索引サイズはセキュリティパラメータに依存しない．今後の課題として、ドキュメント削除への対応があげられる．削除リストの作成によって、$D(w)$ から削除されたドキュメント ID を削除する事は可能であるが、削除されたドキュメントに関するデータは検索されるまで保持しておく必要がある．今後の改良で、仮想的なデータの削除ではなく、削除されたドキュメントのデータを物理的に削除できるよう改良をしたいと考えている．

謝辞 本研究は，JSPS 科研費 JP17K00183 の助成を得て行われた．

参考文献

[1] R. Miyoshi, H. Yamamoto and H. Fujiwara, Improved Searchable Symmetric Encryption for Reducing Space, IEICE-Shinetsu, 2017.
[2] EJ. Goh, Secure Indexes, Stanford Univ. Technical Report, In IACR ePrint Cryptography Archive, 2003, See http://eprint.iacr.org/2003/216.
[3] R. Curtmola, J. Garay, S. kamara and R. Ostrovsky, Searchable symmetric encryption: Improved definitions and efficient constructions, Journal of Computer Security, pp.895-934, 2011.
[4] S. Kamara, C. Papamanthou and T. Roeder, Dynamic Searchable Symmetric Encryption, Proc. of CCS'12, pp.95-976, 2012.
[5] M. Etemad, A. Küpçü, C. Papamanthou and D. Evans, Efficient Dynamic Searchable Encryption with Forward Privacy, PETS 2018, pp.5-20, 2018.
[6] A. Yavuz and J. Guajardo, Dynamic Searchable Symmetric Encryption with Minimal Leakage and Efficient Updates on Commodity Hardware, SAC 2015, LNCS 9566, pp.241– 259, 2016.
[7] W. W. Cohen, The enron email dataset, http://www.cs.cmu.edu/enron/.