# 検索可能暗号の検索応答時間を一定にする簡潔データ構造を用いた索引手法

北村 優汰[†1]　毛利 公美[†2]　中井 敏晴[†3]　白石 善明[†4]　岩田 彰[†1]

暗号化されたデータを復号することなくキーワード検索できる検索可能暗号がある．識別不可能性を満たす検索可能暗号はタグに対する索引を生成できず，検索時間はタグの個数に比例する．キーワードのハッシュ値が数ビット漏れることを許容し，索引を生成する方式が提案されている．しかし，安全性の面から，用いることのできるキーワードのハッシュ値のビット数には制限があり，ハッシュ値のビット数を最大の値に設定しても，システムに要求される検索の応答時間を達成できない場合がある．検索の応答時間を一定に保つために，ハッシュ値が一致したタグの中から一定数のタグのみを検索する方法が考えられる．計算機資源に制約がある中で検索の応答時間を一定に保つためには，索引のサイズについて考慮しなくてはならない．使用可能なメモリ領域と比べ索引のサイズが大きいと補助記憶へのアクセスが頻繁に発生し，応答時間を一定に保てなくなる．本稿では，省メモリな索引を生成するために簡潔データ構造を用いた索引手法を提案する．提案手法により生成された索引を用いることで補助記憶へのアクセス回数を減らし，検索の応答時間を一定に保てることを確認している．

# An Index Method Using Succinct Data Structure for Searchable Encryption to Keep Retrieval Time Constant

YUTA KITAMURA[†1]　MASAMI MOHRI[†2]
TOSHIHARU NAKAI[†3]　YOSHIAKI SHIRAISHI[†4]　AKIRA IWATA[†1]

## 1. はじめに

暗号化されたデータを検索するには一度データを復号しなくてはならない．暗号化されたデータを復号することなくキーワード検索できる技術として検索可能暗号がある．

Boneh らによって公開鍵暗号に基づく検索可能暗号が提案されている[1]．この方式は確率的な暗号であり，識別不可能性（IND）という安全性を満たす．タグからキーワードに関する情報が 1 ビットも漏れないため，タグに対する索引を生成することができない．検索は線形探索となり，データ件数に比例した検索時間を要する．

検索の計算量がデータ数の log オーダである検索可能暗号が Bellare らによって提案されている[2]．この方式は，キーワード空間のエントロピーが大きいことを仮定した場合に PRIV セキュリティという安全性を満たすことが証明されている．しかし，現実のシステムに適用する際に，そのような仮定を満たすことは困難であるとの指摘がある[3]．

松田らはキーワードのハッシュ値が数ビット漏れることを許容し，得られたハッシュ値を用いて索引を生成する方式を提案している[4]．検索時に索引を参照し，検索キーワードのハッシュ値と同じハッシュ値を持つタグを求め，そのタグに対してのみ一致判定処理を行うことで検索対象が絞り込まれ検索が高速化される．ハッシュ値のビット数を大きくすることで検索が高速化されるが，キーワードの部分情報がより多く漏れるため，安全性は低下する．データ件数が多い場合，安全性の面で許容できる最大のビット数までハッシュ値のビット数を大きくしても，ハッシュ値が一致したタグをすべて検索していてはシステムに求められる検索の応答時間を達成できない場合がある．

データ件数に依存しない応答時間にする検索方法として，ハッシュ値が一致したタグの中から一定数のタグのみを検索する方法が考えられる．検索されないタグがあることを許容し，一定数のタグに対してのみ一致判定処理を行うことで，計算機資源に制限がなければ応答時間を一定に保つことができる．

計算機資源に制約がある中で検索の応答時間を一定に保つためには，索引のサイズについて考慮しなくてはならない．なぜなら，データ件数の増加に伴い索引のサイズも増大していくからである．一致判定処理の回数が一定であったとしても，使用可能なメモリ領域と比べ索引のサイズが大きいと，読み書きの遅い補助記憶へのアクセスが頻繁に発生し，要求される応答時間を達成できなくなる．したがって，補助記憶へのアクセス頻度を少なくするために，サ

†1 名古屋工業大学
Nagoya Institute of Technology
†2 岐阜大学
Gifu University
†3 国立長寿医療研究センター
National Center for Geriatrics and Gerontology
†4 神戸大学
Kobe University

イズの小さな索引の生成が求められる．

本稿では，省メモリな索引を生成可能な，簡潔データ構造を用いた索引手法を提案する．簡潔データ構造は，計算量を保ったままサイズを小さくできるデータ構造である．提案手法によって生成された索引を用いることで，補助記憶へのアクセス回数を減らし，検索の応答時間が一定に保たれることを確認する．

## 2. 検索可能暗号

### 2.1 検索可能暗号の概要と課題

暗号文を検索する際には一度復号しなくては検索できない．第三者が管理するサーバにデータの保管および検索を依頼するサービスでは，データの検索時に復号されたデータをサーバ側で閲覧可能となる．復号することなくキーワード検索できる検索可能暗号がある．

Boneh らによって公開鍵暗号に基づく検索可能暗号が提案されている[1]．この方式は確率的な暗号であり，識別不可能性を満たす．検索対象が平文である場合は，B-木やハッシュテーブルといったデータ構造を構築し，これを索引とすることで検索を高速化できるが，Boneh らの方式のような識別不可能性を達成する方式は，タグからキーワードに関する情報が 1 ビットも漏れないため，タグに対する索引を生成できず，検索に要する時間はタグの個数に比例する．識別不可能性を達成する方式は検索の高速化が課題として挙げられる．

### 2.2 検索可能暗号の高速化

確定的な暗号を用いることで，データ件数の log オーダで検索できる方式が Bellare らによって提案されている[2]．Bellare らは PRIV セキュリティという安全性を定義し，キーワード空間のエントロピーがきわめて大きいことを仮定した場合には PRIV セキュリティを満たすことを証明している．しかし，現実のシステムでそのような仮定を満たすことは困難であるとの指摘がある[3]．

松田らは，Boneh らの方式から若干の情報漏れを許容することで，サーバ側で索引を生成し，検索を高速化する方式[4]を提案している．この方式では，索引開示鍵という特別な鍵を利用者が開示した場合に限って，キーワードのハッシュ値をタグから取り出すことができ，得られたハッシュ値を用いてサーバ側で索引を生成する．検索時には，索引を参照し，検索キーワードのハッシュ値と同じハッシュ値をもつタグを求める．ハッシュ値が一致するタグに対してのみ一致判定処理を行うことで，検索対象が絞り込まれ，検索が高速化される．この方式の 3 つのエンティティは次のように定義されており，この方式の処理の流れを図 1 に示す．なお，図 1 は利用者によってハッシュ値が 4 ビット公開されている場合の例である．

【エンティティ】

[登録者]：データの登録をサーバに依頼する

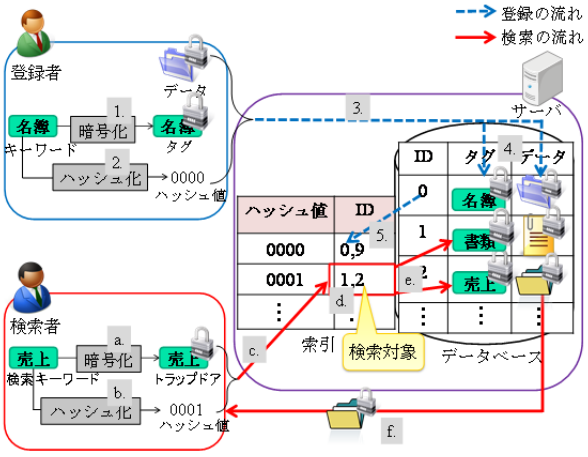


図 1 索引生成可能な検索可能暗号[4]の処理の流れ

[サーバ]：データの登録，および検索を行う

[検索者]：データの検索をサーバに依頼する

【データ登録時の処理の流れ】

1. 登録者はキーワードを暗号化してタグを生成する
2. 登録者はキーワードのハッシュ値を生成する
3. 登録者はデータ，タグ，ハッシュ値をサーバに送信する
4. サーバはデータ，タグを同一レコードとして登録する
5. サーバはハッシュ値が衝突したタグ同士をグルーピングすることで索引を生成する

【データ検索時の処理の流れ】

a. 検索者は検索キーワードを暗号化してトラップドアを生成する
b. 検索者は検索キーワードのハッシュ値を生成する
c. 検索者はトラップドアとハッシュ値をサーバに送信する
d. サーバは索引を参照することで，検索キーワードのハッシュ値と同じハッシュ値を持つタグを求める
e. サーバは d.で求めたタグに対して一致判定処理を行う
f. サーバはキーワードが一致すると判定したタグと同一のレコードにあるデータを検索者に送信する

松田らの方式は，ハッシュ値のビット数を大きくすることで検索が高速化されるが，それに伴い安全性は低下する．タグの個数が多くなると，ハッシュ値のビット数を安全性の面で許容される最大の値に設定してもシステムに求められる検索の応答時間を達成できない場合がある．

タグの個数に依存しない応答時間にする検索方法として，一致判定処理を行うタグの個数をあらかじめ設定しておく方法が考えられる．この検索方法は，検索されないタグがあることを許容するが，ハッシュ値のビット数を増やすことで，検索対象の中に検索キーワードと同一のキーワードのタグが存在する確率を高めることができる．

計算機資源に制約がある場合，タグの個数が増加しても

検索の応答時間を一定に保つためには索引のサイズについて考慮しなくてはならない．なぜなら，タグの個数が増加すると索引のサイズも増大するからである．一致判定処理の回数が一定であったとしても，使用可能なメモリ領域に比べ索引のサイズが大きいと，補助記憶へのアクセスが頻繁に発生し，要求される応答時間を達成できなくなる．補助記憶へのアクセス回数を少なくするために，サイズの小さな索引の生成が求められる．本稿では，省メモリな索引を生成するために，簡潔データ構造を用いた索引手法を提案する．

## 3. 簡潔データ構造

### 3.1 準備

#### 3.1.1 計算モデル

本稿では計算モデルとして word-RAM を用いる．word-RAM は次の性質を満たす実際の計算機に近いモデルである[5].

- 計算機は$U$ビットのメモリを持つ．メモリのアドレスは$[0, U-1]$の整数で指定する
- 計算機の語長は$\lg U$ビットである．つまり，$\lg U$ビットの数の算術・論理演算および$\lg U$ビットのメモリアクセスが 1 単位時間で行える
- メモリから読み込んだ値は$[0, U-1]$の整数とみなす

#### 3.1.2 情報理論的下限

あるデータを表現するために必要なビット数の情報理論的下限を定義する．基数$L$のある集合に含まれる一つの要素を表現するのに必要なビット列の情報理論的下限を$\lg L$ビットと定義する．例えば，長さ$n$のビット列の集合の基数は$2^n$であるため，情報理論的下限は$\lg 2^n = n$ビットである．

### 3.2 簡潔データ構造とは

データ構造を情報量や計算量を保ったままサイズを小さくできる技術として簡潔データ構造がある．[5]では，あるデータに対する簡潔データ構造は，次の性質を持つ抽象データ型であると定義されている．

- データ構造のサイズが，データサイズの情報理論的下限に漸近的に一致する
- データに対する問い合わせが，word-RAM 上で従来のデータ構造と同じ計算量で行える

サイズを縮小する技術としては代表的なものにデータ圧縮があるが，圧縮されたデータに対して問い合わせを行うには，一度データをすべて復元しなくてはならない．一方，簡潔データ構造は復元操作を必要としないため，データ圧縮に比べ高速な処理が実現可能となる．

簡潔データ構造の基本的なデータ構造として Rank/Select 辞書がある．ビット列$B[0, n-1]$，$B[i] \in \{0,1\}$に対して次の操作を備えたデータ構造を Rank/Select 辞書と呼ぶ．

- $\mathrm{rank}_b(B, i)$：$B[0, i-1]$の中の$b \in \{0,1\}$の数を返す

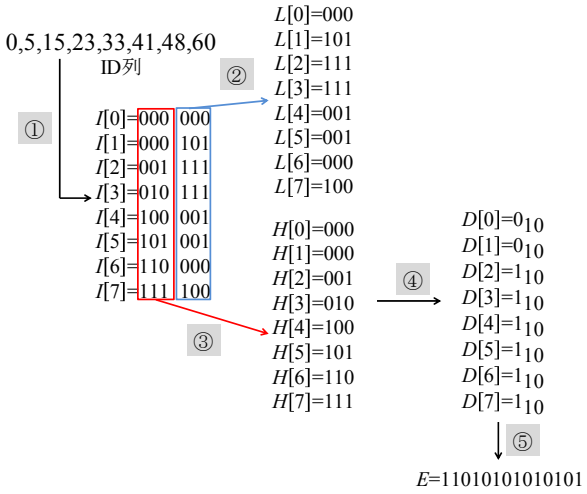


図 2 ID 列のサイズ縮小の例

- $\mathrm{select}_b(B, i)$：$B$中で先頭から見て$i+1$番目に出現した$b \in \{0,1\}$の位置を返す

Rank/Select 辞書は多くの簡潔データ構造の内部表現として使われており，rank，select 演算を組み合わせて使うことで様々な操作を実現することができる．

## 4. 提案手法

### 4.1 簡潔データ構造を用いた索引手法

本稿で提案する索引手法は，[4]の方式と同様にキーワードのハッシュ値を得ることによりタグに対する索引を生成する．タグにはユニークな整数値（ID）が割り振られており，ハッシュ値が$l$ビット得られるものとする．

配列を$2^l$個確保し，ハッシュ値が一致したタグの ID を同じ配列の要素として順次格納していくことで，索引を生成することができる．構成される各配列のことを `ID 列’と呼ぶこととする．検索時には索引を参照することで，検索キーワードのハッシュ値と同じハッシュ値を持つタグを求めることができる．索引は ID 列によって構成されているので，索引のサイズ縮小を図る場合，ID 列のサイズを縮小できればよい．そこで，提案手法では，文献[6]に示されている手法を用いて，ID 列のサイズ縮小を図る．ID 列のサイズ縮小の手順を図 2 に示す．

① ID 列の要素数を$m$とする．ID 列の各要素を配列$I[0, m-1]$に格納する

② $p = \lceil \lg(n/m) \rceil$とし，$I[i]$の下位$p$ビットを配列$L[i]$に格納する

③ $I[i]$の残りの上位ビットを$H[i] = I[i]/2^p$と格納する．このとき ID 列の$i$番目の要素は，$I[i] = 2^p H[i] + L[i]$で求められる

④ $D[i] = H[i] - H[i-1]$となる配列$D[0, m-1]$を構成する．ただし，$H[-1] = 0$と定義する．このとき，$D$は非負整数列となり，$H[i] = \sum_{k=0}^{i} D[i]$と表せる

⑤ $D$の各要素を単進符号で表現したものを連結させ

たビット列$E$を構成する．このときビット列$E$の長さは$2m$であり，$H[i] = \sum_{k=0}^{i} D[i] = \mathrm{select}_1(E, i) - i$となる

もとの ID 列の$i$番目の要素は配列$L$とビット列$E$を用いて，

$$I[i] = 2^p(\mathrm{select}_1(E, i) - i) + L[i] \quad (1)$$

で求められる．もとの ID 列の要素が格納された配列$I$を保持しておかなくても，配列$L$とビット列$E$を保持しておくことで，ID 列の各要素を求めることができる．

### 4.2 ID の取得に要する計算量と ID 列を保持するのに必要なビット数

検索時には演算(1)を実行することで検索対象となるタグの ID を取得する．ビット列$E$に対する select 演算が定数時間で実現可能であれば，演算(1)の計算量は定数時間となる. Kim らによって, 長さ$k$のビット列に対して, $k + o(k)$ビットの補助データ構造を用いて，select 演算を定数時間で実現可能な手法が提案されている[7]．ビット列$E$に対して Kim らの手法を適用することで，演算(1)は定数時間で実行可能となる．

ビット列$E$は長さ$2m$であるので，Kim らの手法を用いた際に構築される補助データ構造のサイズは$2m + o(m)$ビットである. もとの ID 列を表現するのに必要なビット数は，配列$L$，ビット列$E$，補助データ構造の合計のサイズである$m\lg(n/m) + 4m + o(m)$ビットである．

## 5. 評価

### 5.1 索引の実装

提案手法を適用していない索引を比較対象とし，提案手法を適用した索引を用いることで ID の取得に要する時間がどのように変化するかを計測する．

データ構造を構築する際には，理論的には可能であっても，実装する際には必ずしも実現できない点がある．0から$n-1$までの整数値を格納する配列の一つの要素は，理論的には$\lg n$ビットで表現できる．例えば，$n = 2^{26}$であった場合，配列の一つの要素は 26 ビットあれば良い．しかし，実装する際には, 一つの要素が 26 ビットの配列を構築することは困難であるため，unsigned int 型(32 ビット符号なし整数)の配列などで実装することになる．

比較対象の索引の ID 列は unsigned int 型の配列で実装した．提案手法の索引の配列$L$は unsigned char 型(8 ビット符号なし文字型)で実装した. $p$が 9 以上になると配列$L$の一つの要素は 8 ビットで表現できなくなるが, そうなった場合，配列$L$に格納するのは$I$の下位 8 ビットまでとし，残りの上位ビットを配列$H$に格納するよう実装した．

### 5.2 実験内容

実験用プログラムのコンパイルおよび実行は仮想マシン上で行った．物理マシンと仮想マシンの環境をそれぞれ表 1，表 2 に示す．

次の二つの検索方法を想定し，それぞれの方法を行うのに要する時間を測定した．

- 検索方法 1 ：ハッシュ値が一致したタグの ID をすべて取得する方法
- 検索方法 2：ハッシュ値が一致したタグの ID を一定数取得する方法

表 1　物理マシンの環境

| | |
|---|---|
| OS | Windows7 Professional(64bit) |
| CPU | Intel Core i5-3210M 2.50GHz |
| メモリ | 8GB |
| HDD | 250GB |
| | 7200[rpm] |
| 仮想化ソフト | VMware Player 6.0.2 |

表 2　仮想マシンの環境

| | |
|---|---|
| OS | CentOS 6.5 |
| CPU | Intel Core i5-3210M 2.50GHz |
| メモリ | 512MB |
| 仮想 HDD | 20GB |
| コンパイラ | gcc 4.4.7 |

実験は，索引生成フェーズと ID 取得フェーズの二つから成る．

【索引生成フェーズ】

1. 配列を$2^l$個確保する．各配列が ID 列となる
2. 0 から$n-1$までの整数値を 1.で確保した配列のいずれかの末尾にランダムに挿入する
3. （提案手法のみ）4.1 節で述べた手法を用いて ID 列を配列$L$とビット列$E$に変換する

【ID 取得フェーズ】

1. $2^l$個の ID 列から検索対象とする ID 列を一つランダムに決定する
2. （検索方法 1 の場合）1.で決定された ID 列の先頭から末尾まで順にアクセスし, ID 列中のすべての ID を取得する
   （検索方法 2 の場合）1．で決定された ID 列の中から一定数ランダムに ID を取得する
3. 上記 1., 2.を各 ID 列が平均 10 回アクセスされるように，すなわち$2^l \times 10$回繰り返す

ID 取得フェーズの 1．から 3．を実行するのに要する時間を計測し，その時間を ID 取得フェーズ 3．で繰り返した回数で割ることで各検索方法を実行するのに要した平均時間を求める．

### 5.3 実験結果

#### 5.3.1 検索方法 1 ：ハッシュ値が一致したタグの ID をすべて取得する方法

検索方法 1 では ID 列からすべての ID を取得するため，一般に，ID 列の要素数が多いほど検索に要する時間は長くなる．ハッシュ値が小さい場合，タグの個数が多い場合にそれぞれ ID 列の要素数は多くなる．

ID 列からすべての ID を取得するのに要した時間を表すグラフを図 3 に示す．図 3 において，比較対象，提案手法ともにハッシュ値のビット数が小さいときほど時間が長くなっている．タグの個数が増加した場合もそれに伴い時間が長くなっている．

各ハッシュ値のビット数ごとに比較対象と提案手法の実測時間を比べると，すべてのビット数において比較対象の方が短い時間で ID の取得が可能であるという結果となった．

#### 5.3.2 検索方法 2 : ハッシュ値が一致したタグの ID を一定数取得する方法

検索方法 1 では ID 列中の要素数が増加するとそれに伴い検索に要する時間も長くなったが，検索方法 2 は取得する ID の個数が決まっているため，補助記憶へのアクセスが発生しない限り，ハッシュ値のビット数やタグの個数が変化しても時間は一定となる．ハッシュ値のビット数やタグの個数を変化させて，ID を一定数取得する時間に変化があった場合は補助記憶へのアクセスが影響を及ぼしていると言える．

ID 列から ID を 1000 個取得するのに要した時間をあらわすグラフを図 4 に，10000 個取得するのに要した時間をあらわすグラフを図 5 にそれぞれ示す．図 4，図 5 から比較対象はタグの個数が$1 \times 10^8$個を超えると，急激に時間が増加していることが確認できる．これは，使用可能なメモリ領域に対して索引のサイズが大きいため，頻繁に補助記憶へのアクセスが発生していることを意味する．

提案手法はタグの個数が$2 \times 10^8$個を超えると時間が増加しているが，比較対象に比べ，短い時間で ID の取得ができている．

ただし，図 4 において$l = 2$のとき，タグの個数が$4 \times 10^8$個を超えると，提案手法を用いた場合の方が ID の取得に要する時間は長くなる．

### 5.4 考察

実験の結果，

- 検索方法 1 を行う場合は比較対象の方が高速である
- 検索方法 2 を行う場合は提案手法の方が高速である．ただし，一部比較対象の方が高速な場合もある

ことが確認された．

OS のページングの性質上，ある ID 列の要素をメモリ上にロードするとその周辺の要素も同時にロードされる．検索方法 1 では，ある ID 列の要素を取得するためにその要素をメモリ上にロードした場合，次にアクセスされる要素もメモリ上にロードされる確率が高く，ID の取得回数に対し補助記憶へのアクセス頻度が少ない．ID の取得操作 1 回に要する時間は比較対象の索引の方が短いため，検索方法 1 では比較対象の方が高速になる．

検索方法 2 では，ある ID 列の要素がメモリ上にロードされても，次に取得される要素がその周辺に存在するとは

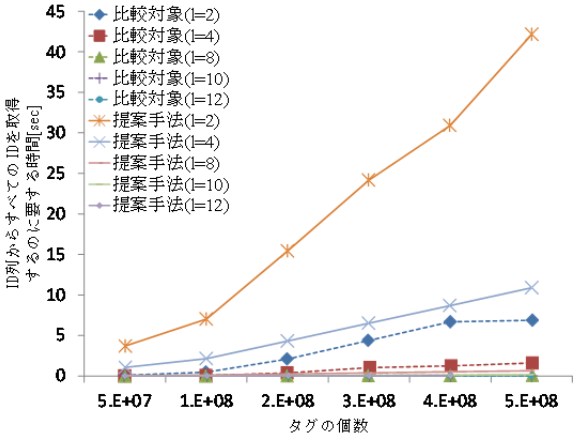


図 3 ID 列からすべての ID を取得するのに要した時間

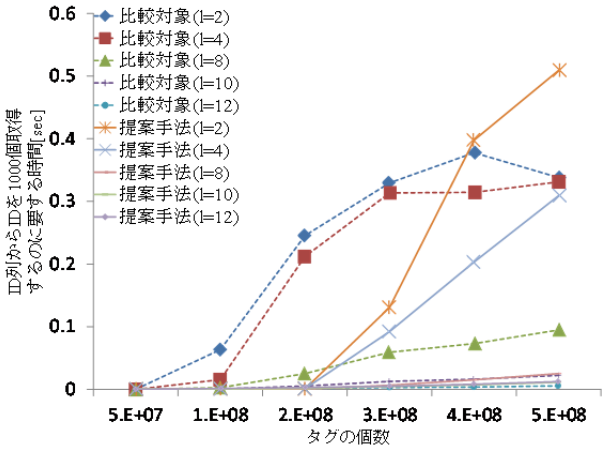


図 4 ID 列から ID を 1000 個取得するのに要した時間

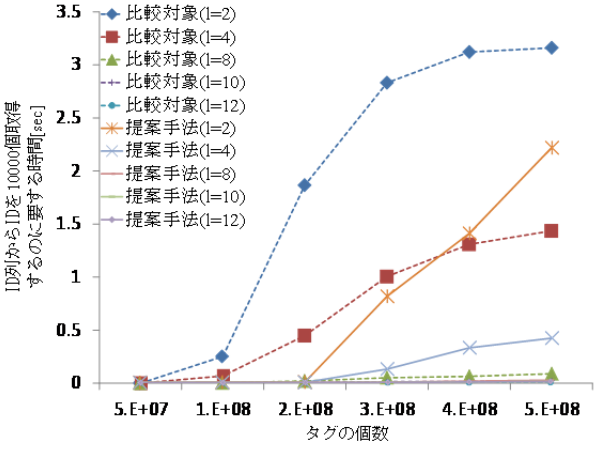


図 5 ID 列から ID を 10000 個取得するのに要した時間

限らないため，補助記憶へのアクセス頻度は多くなる．索引のサイズは提案手法の方が小さく，補助記憶へのアクセスが発生する確率が低いため，検索方法 2 では提案手法の索引の方が高速になる．

検索方法 2 を行う目的はタグの個数が増加しても検索の応答時間を一定に保つことである．実験環境において，タグの個数が$5 \times 10^7$から$2 \times 10^8$個の区間ではタグの個数の増加による ID の取得に要する時間の増加は無視できる程度であり，応答時間を一定に保つという目的は達成されている．

## 6. おわりに

検索可能暗号の高速化という課題に対して，キーワードのハッシュ値を用いて索引生成する方式が提案されている．検索キーワードのハッシュ値と同じハッシュ値を持つタグを，索引を参照して求めることで，検索対象を絞り込み高速化できることが示されている．しかし，安全性の面から，用いることのできるキーワードのハッシュ値のビット数には制限があり，ハッシュ値のビット数を最大の値に設定しても，システムに要求される検索の応答時間を達成できない場合がある．

検索の応答時間を一定に保つために，ハッシュ値が一致したタグから一定数を検索する方法について検討したところ，あらかじめ検索するタグの個数を設定しておくことで一致判定処理の回数を一定に保つことができるが，計算機資源に制約がある中で検索の応答時間を一定に保つためには索引のサイズについて考慮しなくてはならない．

本稿では，簡潔データ構造を用いた索引手法を提案した．提案手法を用いることで，検索に要する計算量を保ったままサイズの小さな索引を生成することができる．

ハッシュ値が一致したタグから一定数を検索する方法において，タグのある個数までであれば，提案手法によって生成された索引を用いることで検索の応答時間を一定に保てることを確認した．

## 参考文献


1) Boneh, D., Crescenzo, G.D., Ostrovsky, R. and Persiano, G.: Public Key Encryption with Keyword Search, *EURO-CRYPT*'04, LNCS, Vol.3027, pp.506-522 (2004).

2) Bellare, M., Fischlin, M., O' Neill, A. and Ristenpart, T.: Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles, *CRYPTO 2008*, LNCS, vol.5157, pp.360-378 (2008).

3) 松田規，服部充洋，平野貴人，森拓海，伊藤隆，川合豊，坂井祐介，太田和夫：安全性と高速性の両立を目指した検索可能暗号(1), 2012 年暗号と情報セキュリティシンポジウム(SCIS2012), 1A3-1（2012）.

4) 松田規，伊藤隆，柴田秀哉，服部充洋，平野貴人：検索可能暗号の高速化と Web アプリケーションへの適用方式に関する提案, マルチメディア，分散，協調とモバイル（DICOMO2013）シンポジウム，pp.2067-2074（2013).

5) 定兼邦彦：超簡潔データ構造，電子情報通信学会誌 Vol.92, No.2，pp.97-104（2009）.

6) Okanohara, D., Sadakane, K.: Practical Entropy-Compressed Rank/Select Dictionary, Proc. ALENEX (2007).

7) Kim, D.K, Na, J.C., Kim, J.E., and Park, K.: Efficient Implementation of Rank and Select Functions for Succinct Representation, WEA 2005, LNCS 3503, pp.315-327 (2005).