6L-1

検索可能暗号を高速化するための簡潔データ構造を用いた索引手法

北村　優汰[†]　福田　洋治[‡]　廣友　雅徳[††]　毛利　公美[‡‡]　中井　敏晴[†††]　白石　善明[‡‡‡]

†名古屋工業大学　‡愛知教育大学　††佐賀大学

‡‡岐阜大学　†††国立長寿医療研究センター　‡‡‡神戸大学

1.はじめに

暗号化されたデータを検索する際には，一度データを復号しなければ検索できない．つまり，暗号化されたデータの保管，および検索を第三者が管理するサーバに依頼する場合，検索時にはデータの内容がサーバ側で閲覧可能になる．サーバが閲覧できないようにするために，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．検索可能暗号は，暗号化されたキーワード(タグ)と暗号化された検索キーワード(トラップドア)が一致しているかを復号せずに判定することができる．検索可能暗号を利用することで，例えば自社のサーバに保管しきれない大量の業務データの保管，検索を第三者のサーバに依頼する場合でも，第三者に顧客情報を漏らすことなく検索を行えるようになる．

検索可能暗号に限らず，大量のデータに対して検索を行う場合，検索速度の向上が課題となる．検索対象が平文である場合，検索を高速化する方法として索引生成がある．代表的なものに，ハッシュテーブルや B-木等のデータ構造を用いた索引があり，検索時には索引を参照して検索対象を絞り込むことで検索の高速化を図る．しかし，Boneh らによって提案された検索可能暗号[1]は，タグからキーワードに関する情報が 1 ビットも漏れない安全性(識別不可能性)を満たすものであり，タグに対する索引を生成することができない．そこで，安全性を若干低下させてタグに対する索引を生成することで検索可能暗号を高速化する方式が提案されている[2]．索引を生成する場合，検索対象であるタグに加えて索引もメモリ上に格納する必要がある．索引が占めるメモリ領域が大きくなると，タグを格納できるメモリ領域が小さくなり，読み書きの遅い補助記憶へのアクセス回数が増えるため，検索に要する時間も長くなる．生成される索引のサイズを小さくすることができれば，検索の更なる高速化を図ることができる．

本稿では，簡潔データ構造を用いた索引手法を提案し，省メモリな索引を生成することで検索可能暗号の高速化を図る．簡潔データ構造は，データ構造を情報量や検索速度を保ったままサイズだけを小さくできるデータ構造である．提案する索引手法では，キーワードから得られるハッシュ値の衝突を利用したタグのグループを簡潔データ構造で表現することで，情報量を保ちつつ省メモリな索引を生成する．省メモリな索引を用い，補助記憶へのアクセス回数を減らすことで検索が高速化される．

2. 検索可能暗号

検索可能暗号では，暗号化したキーワード(タグ)と暗号化した検索キーワード(トラップドア)が同一かどうかを，復号することなく判定することができる．Boneh らによって提案された楕円曲線上のペアリング演算を用いる検索可能暗号[1]は，確率的な公開鍵暗号であり，識別不可能性という安全性を満たし，タグからキーワードに関する情報が 1 ビットも漏れないことを保証する．これは検索対象であるタグに対する索引を生成できないことを意味しており，検索時にはすべてのタグに対して一致判定処理を行う必要があるため，検索時間はデータ件数に比例する．さらに，一致判定の際に用いられるペアリング演算を実行するのに要する時間は文献[2]によると 0.41[msec]とされており，例えば，数百万件分の業務データを検索する場合に，すべてのタグに対して一致判定処理を行うと 1 回の検索に数十分程度の時間を要することになる．このことから，検索可能暗号は検索の高速化が課題として挙げられる．

タグから数ビットの情報漏れを許容することで，タグに対する索引を生成し，検索を高速化する方式が提案されている[2]．この方式ではタグからキーワードのハッシュ値を得ることで，同じハッシュ値をもつタグを同一グループとみなしてグルーピングする．検索時には，検索キーワードのハッシュ値を求め，同じハッシュ値をもつタグのグループに対してのみ一致判定処理を行うことで，一致判定処理の対象となるタグの個数を減らすことができ，検索を高速化している．以下に三つのエンティティを定義し，この方式の概念図を図 1 に示す．

[登録者]：データの登録をサーバに依頼する
[サーバ]：データの登録，索引生成，データの検索を行う
[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
1. 登録者はキーワードを暗号化することでタグを生成
2. 登録者はキーワードのハッシュ値を生成
3. 登録者はデータ，タグ，ハッシュ値をサーバに送信
4. サーバはデータ，タグを同一レコードとして登録
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

■ データ検索時の処理
a. 検索者は検索キーワードを暗号化しトラップドアを生成
b. 検索者は検索キーワードのハッシュ値を生成
c. 検索者はトラップドアとハッシュ値をサーバに送信
d. サーバは索引を参照することで，受信したハッシュ値と同じハッシュ値をもつタグを求める
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. サーバは一致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

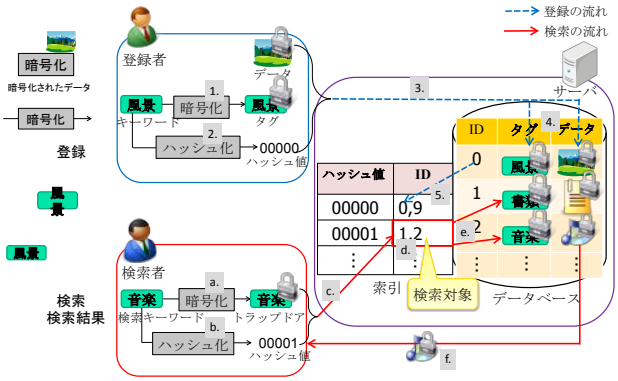

図 1 索引生成可能な検索可能暗号の概念図

このアプローチは，索引のサイズが増大したときに，メモリの空き容量不足による補助記憶へのアクセス回数の増加で検索速度が低下する．つまり，生成される索引のサイズは可能な限り小さいことが望ましい．そこで，本稿では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

計算量を評価する計算モデルとして word-RAM がある．word-RAM は以下の性質を持つ実際の計算機に近い計算モデルである．

■ 計算機はUビットのメモリをもつ．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・論理演算及び$\lg U$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

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 1 つの要素を表現するビット列のサイズの情報理論的下限を$\lg L$ビットと定義する．

3.2 簡潔データ構造

データ構造を情報量や検索速度を保ったまま，サイズを小さくする技術として簡潔データ構造[3]がある．あるデータに対する簡潔データ構造は，次の性質をもつ抽象データ型である．

■ 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

An Index Method using Succinct Data Structure for Efficient Searchable Encryption
† Yuta KITAMURA・Nagoya Institute of Technology
‡ Youji FUKUTA・Aichi University of Education
†† Masanori HIROTOMO・Saga University
‡‡ Masami MOHRI・Gifu University
††† Toshiharu NAKAI・Research Institute National Center for Geriatrics and Gerontology
‡‡‡ Yoshiaki SHIRAISHI・Kobe University

ビット列$B[0,n)$，$B[i] \in \{0,1\}$に対して次の操作を備えたデータ構造を完備辞書と呼ぶ．

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

完備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，完備辞書の三つの演算を組み合わせて用いる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

本手法では，関連研究[2]と同様に，キーワードのハッシュ値の衝突を利用したグルーピングにより索引を生成する．グルーピングによって構成されたタグの集合を簡潔データ構造で表現することで，索引のサイズを縮小することができる．サイズを縮小する技術としては他に代表的なものとしてデータ圧縮があるが，圧縮されたデータを扱う際には一度復元操作を行わなければならず，高速な処理には向かない．一方，簡潔データ構造を用いて索引生成した場合，復元操作の必要がなく，検索速度を保ちつつ省メモリな索引を生成することができる．

4.2 索引生成の手順

索引生成の手順を説明する．タグにはユニークな整数値(ID)が割り振られており，タグからはキーワードのハッシュ値がlビット得られるものとする．

1. キーワードのハッシュ値が一致するタグ同士を同一のグループとみなし，タグを2^l個のグループに分類する
2. グループごとに，タグの ID を昇順に並べた整数列(これを ID 列と呼ぶこととする)を構成する
3. ID 列に対して，文献[4]に示されている手法を適用し，ID 列のサイズを縮小する

このとき，タグの個数をn，ID 列の要素数をmとすると，ID 列のi番目の要素$P[i]$は，長さ$2m$のビット列Eとサイズ$m\lg(n/m)$の配列$L[0,m)$を用いて，

$$P[i] = (n/m) \times (\mathrm{select}_1(E,i) - i) + L[i] \quad (1)$$

で表せる．長さkのビット列に対し，$k+o(k)$ビットの補助データ構造を用いて定数時間でselect演算を実現可能な手法[5]が提案されており，この手法をビット列Eに適用することで，式(1)の計算量は定数時間となる．このとき，ID 列を表現するのに必要なビット数は$m\lg(n/m)+4m+o(m)$ビットである．2 章で示した概念図の処理 d.で式(1)を計算することで一致判定を行うタグの ID を定数時間で求めることができる．

5.評価

タグの個数をnとするとタグに割り振られる識別子を表すのに必要なビット数は$\lg n$ビットである．これをそのまま索引に記録した場合，索引のサイズは$n\lg n$ビットである．本章では，これを比較対象とし，提案手法によって生成された索引を用いることで，検索がどの程度高速化されるかを評価する．

5.1 評価内容

以下の操作を定義し，検索のアルゴリズムを図 2 に示す．

- seeIndex(i)：索引を参照し，検索対象となるi番目のタグの ID を求める
- isThereTag(ID)：入力された ID のタグがメモリ上に存在するか判定する．なお，この操作に要する時間は無視できるものとする
- test(ID)：入力された ID のタグに対して，検索可能暗号の一致判定処理を行う
- getTag(ID)：入力された ID のタグを補助記憶から読み込み，メモリ上に格納する

mは一致判定処理を行うタグの個数，すなわち検索キーワードのハッシュ値と一致したタグの個数である．ここでは，簡単のため，n個のタグが2^l個のグループに同じ数ずつグルーピングされていることとする．つまり，$m = n/2^l$となる場合を考える．このとき，提案手法によって生成される索引のサイズは$2^l(m\lg(n/m)+4m+o(m)) = (l+4)n$ビットで表せる．ここでは$o(m)$は無視できるものとする．

各操作に要する時間を考える．seeIndex はタグの ID をそのまま格納した索引を用いた検索では，メモリ上にある整数値を読み込む操作であり，提案手法においては式(1)の計算がそれに当る．どちらの操作もメモリ上で定数時間で実現可能である．test の実行に要する時間は，楕円曲線上のペアリング演算に要する時間と考えてよく，操作に要する時間は 0.41msec とする．getTag に要する時間は補助記憶へのアクセス時間と考えてよい．[6]によると，一般的な HDD のスループットは 180IOPS 程度とされており，ディスクへの一回のアクセスに 5.5msec ほど要することになる．ここでは，getTag を実行するのに要する時間を 5.5[msec]と定義する．

次に各操作の実行回数を考える．seeIndex と test の実行回数はm回である．getTag の実行回数の期待値は$(1-h)m$回である．hはあるタグがメモリ上に存在する確率であり，使用可能なメモリ領域をM，索引のサイズをS_I，タグ一つあたりのビット数をS_Tとすると，$(M-S_I)/nS_T$で求められる．ただし，$(M-S_I) > nS_T$のとき，$h=1$とする．

seeIndex と test はどちらもm回実行されるが，seeIndex に要する時間は test に要する時間と比べて無視できると考えられる．したがって，検索時間は$5.5 \times (1-h)m + 0.41m$[msec]で求められる．

```
function search(m)
  for i←0, m do
    ID←seeIndex(i)
      if(isThereTag(ID)) then
        test(ID)
      else
        getTag(ID)
        test(ID)
      end if
  end for
end function
```

図 2 検索アルゴリズム

5.2 評価結果

タグの識別子をそのまま格納した索引を用いた場合の検索時間をT_A，提案手法によって生成された索引を用いた場合の検索時間をT_Bとして，検索速度の速度比rを，

$$r = \frac{T_A}{T_B}$$

と定義する．縦軸に速度比，横軸にタグの個数をとり，タグの個数によって検索がどの程度高速化されるかを確認する．

図 3 にハッシュ値のビット数を変化させた場合の速度比のグラフを示す．$M = 2^{38}$ビット(32G バイト)，$S_T = 512$ビットとした．速度比が増加している区間において，提案手法ではメモリ上にすべてのタグを格納できているが，比較対象ではタグをメモリ上に格納しきれず，補助記憶上にも格納しているため，補助記憶へのアクセスが発生する．このことから，補助記憶へのアクセスが検索速度低下の要因となることが確認できる．提案手法の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，検索を高速化できることが示された．ハッシュ値のビット数に注目すると，ビット数が 2 ビットである場合が，最も速度比の値は大きくなっている．情報漏れを抑えつつ高速化を図ることが求められる場合，提案手法は有効であると言える．

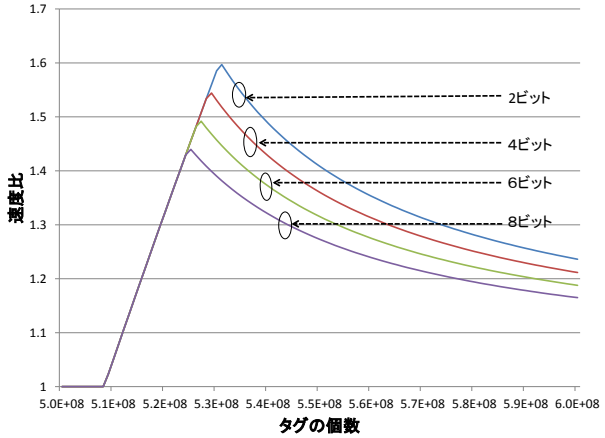

図 3 ハッシュ値のビット数を変化させた場合の速度比

6. おわりに

本稿では，検索可能暗号を高速化するために，簡潔データ構造を用いた索引手法を提案した．本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高速化が図られることを示した．

本稿では，キーワードのハッシュ値が漏れることで，どの程度安全性の低下するかについては言及していない．ハッシュ値のビット数と安全性の関係についての検討を今後の課題とする．

参考文献

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

[2] 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).

[3] 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).

[4] 岡野原大輔：高速文字列解析の世界，p.140，岩波書店 (2012).

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

[6] VDI ストレージソリューション-Data Core，入手先 http://www.datacore.com/Libraries/Japan_Documents/20120424_SANsymphony-V_VDI.sflb.ashx (参照 2014-01-12).