暗号化された情報の大小比較可能な検索処理の Trie 型分岐構造管理による高速化手法

浅野　将希† 　Hieu Hanh Le† 　横田　治夫†

† 東京工業大学　〒 152-8550 東京都目黒区大岡山 2-12-1
E-mail: †{asano,hanhlh}@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　近年，データベースに蓄積されるプライバシーに関わる情報が増加しており，データを暗号化して保存することが求められている．権限者のみがデータの利用を可能とするためにプロキシ再暗号化されたデータを蓄積する手法が提案されている．プロキシ再暗号化を用いた先行研究では，大小比較を可能とするデータの検索に用いるため，格納されるデータの数値を予めブロック毎に比較した結果を持つ検索タグの作成方法を提案している．しかし，先行研究の検索では一度の問い合わせにおいて多くのブロックとの比較結果を必要とし非効率である．そこで本研究では Trie 型分岐構造に検索タグの上位ブロックの同じものをまとめ，一方向性のハッシュと検索タグの比較回数を減らすことで計算量を抑える．また，機密性を考慮して各エントリとの紐付けや格納されたデータとの紐付けにハッシュを用いる．提案手法を評価するため，RDF で表現される情報から検索用タグを作成して実験を行なった．上位ブロックをまとめることによるオーバーヘッドは高々7.40%であり，検索の性能においては，インデックスの取得の実行時間では最大で 99.7%，クエリを実行してから復号するまでの実行時間では最大で 70.9%の削減がされた．

キーワード　データ共有，範囲クエリ，Trie木，プロキシ再暗号化

1 はじめに

近年，災害の復旧や復興を目的とした情報技術の活用が注目されており，安否確認，情報共有，被災者支援をするためのシステムが利用されている．その代表として，災害用伝言サービス [1] や安否情報を共有するためのクラウドサービス [2] などが挙げられる．このようなシステムでは，被災者の個人情報などプライバシーやセキュリティに関わる情報がデータベースに蓄積されることがある．これらの情報を蓄積するときには，情報の機密性を保つ必要がある．データの不正利用を防ぐためには適切なアクセス制御が必要であり，システム全体が信頼できない場合にはデータを暗号化して保存することが有効である．

従来の暗号化手法よりユーザやデータの追加および削除を効率的に行うため，プロキシ再暗号化可能な暗号 [3] を利用する手法を提案した児玉らの研究 [4] が存在する．この研究には，範囲クエリに対応できないという課題があった．平田らは，格納されるデータの数値を予めブロック毎に比較した結果を持つ検索タグの作成方法を提案することで，この課題の解決を図った [5]．

しかし，先行研究の検索では一度の問い合わせにおいて多くのブロックとの比較を必要とするため非効率であることが問題点として挙げられる．

そこで本研究では Trie 型分岐構造に検索タグの上位ブロックの同じものをまとめる手法を提案する．このことで，問い合わせの数値を暗号化した一方向性のハッシュと検索タグの比較回数を減らすことで計算量を抑える．また，機密性を考慮して各エントリとの紐付けや格納されたデータとの紐付けにハッシュを用いる．

提案手法を実装し，Trie 型分岐構造の構築のオーバーヘッドが高々7.40%であり，Trie 型分岐構造を用いた検索の性能はインデックスの取得の実行時間では最大で 99.7%，クエリを実行してから復号するまでの実行時間では最大で 70.9%の削減がされたことを確認した．

2 前提知識

本節では，前提知識として順序比較可能暗号と Trie 木について述べる．

2.1 順序比較可能暗号

順序比較可能暗号は，平文 $x = x_1x_2 \dots x_n$ の暗号文 $ct_L^{(x)}$ と平文 $y = y_1y_2 \dots y_n$ の暗号文 $ct_R^{(y)}$ から x と y の一致比較，大小比較の結果を得ることができる暗号化手法である．本節では，先行研究における検索タグの作成方法の基となる Lewi らの順序比較可能暗号 [6] について述べる．

2.1.1 擬似ランダム関数と擬似ランダム置換

まずは順序比較可能暗号において利用する擬似ランダム関数 [7] と擬似ランダム置換について述べる．$F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ において，ランダムに選ばれた $k \leftarrow \mathcal{K}$ に対する $F(k, \cdot)$ の出力と $\mathcal{X}$ から $\mathcal{Y}$ に写像する真にランダムな関数 $f(\cdot)$ の出力を区別することができないときに，F を擬似ランダム関数 (PRF) という．同様に，$F : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ において，$\forall k \in \mathcal{K}$ に対する $F(k, \cdot)$ の出力と真にランダムな置換 $\pi(\cdot)$ の出力を区別することができないとき，擬似ランダム置換 (PRP) という．

2.1.2 構成方法

セキュリティパラメータを $\lambda \in \mathbb{N}$，メッセージ空間の大きさを $N > 0$，$d^n \geqq N$ となるように整数 $d, n > 0$ を定める．また，$F : \{0,1\}^\lambda \times \mathbb{Z}_N \to \{0,1\}^\lambda$ を PRF，$H : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_3$ をハッシュ関数，$\pi : \{0,1\}^\lambda \times \mathbb{Z}_d \to \mathbb{Z}_d$ を PRP とする．x をビット表現し，固定の長さにするために 0 埋めをする．その後に n 等分し，ブロックごとに 10 進数で表したものを $x_1 x_2 \ldots x_n$ とする．$x_{|i}$ を x の先頭から i 番目までのブロックのプレフィックスとする．ただし，$x_{|0}$ を空のプレフィックスとする．$a||b$ を a と b のビット連結とし，a と b の大小判定をする関数 CMP(a,b) を以下のように定義する．

$$CMP(a,b) = \begin{cases} 2 & (a < b) \\ 0 & (a = b) \\ 1 & (a > b) \end{cases}$$

- セットアップ

$k_1, k_2 \leftarrow \{0,1\}^\lambda$ をランダムに選ぶ．秘密鍵は $sk = (k_1, k_2)$ とする．

- 右暗号化

乱数 $r \leftarrow \{0,1\}^\lambda$ をランダムに選ぶ．平文 $y = y_1 y_2 \ldots y_n$ において，$\forall i \in \{1,\ldots,n\}$ と $\forall j \in \{1,\ldots,d\}$ に対して，$j^* = \pi^{-1}(F(k_2, y_{|i-1}), j)$ を計算し，$z_{ij} = CMP(j*, y_i) + H(F(k_1, y_{i-1}||j), r)(mod\ 3)$ とする．$ct_R^{(y)} = (r, z_{11}, \ldots, z_{1d}, z_{21}, \ldots, z_{2d}, \ldots, z_{n1}, \ldots, z_{nd})$ を右暗号文とする．

- 左暗号化

平文 $x = x_1 x_2 \ldots x_n$ において，$\forall i \in \{1,\ldots,n\}$ に対して，$\tilde{x}_i = \pi(F(k_2, x_{|i-1}), xi)$，$u_i = (F(k_1, x_{|i-1}||\tilde{x}_i), \tilde{x}_i)$ を計算する．$ct_L^{(x)} = (u_1, \ldots, u_n)$ を左暗号文とする．

- 比較

x の左暗号文 $ct_L^{(x)}$ と y の右暗号文 $ct_R^{(y)}$ から x と y の比較結果を得る．$i \in \{1,2,\ldots,n\}$ に対して，$z_{i\tilde{x}_i} - H(F(k_1, x_{|i-1}||\tilde{x}_i), r) \neq 0(mod\ 3)$ となる最小のインデックス i を見つけ，これを ℓ とし，$z_{\ell\tilde{x}_\ell} - H(F(k_1, x_{|\ell-1}||\tilde{x}_\ell), r)(mod\ 3)$ を出力する．そのような ℓ が存在しない場合は 0 を出力する．

$d = 3, n = 4$ のときの具体例を考える．$x = 41 = (101100)_2 = 2_{10}3_{10}0_{10}, y = 32 = (100000)_2 = 2_{10}0_{10}0_{10}$ とし，$CMP(x,y)$ を考える．y を右暗号化したものを図 1 に示す．$i = 1$ のとき，$z_{1\tilde{x}_1} - H(F(k_1, x_{|0}||\tilde{x}_1), r) = z_{1\pi(F(k_2,x_{|0}),2)} - H(F(k_1, x_{|0}||\pi(F(k_2, x_{|0}), 2)), r) = z_{10} - H(F(k_1, x_{|0}||0), r) = 2 - 2 = 0$ となるので i をインクリメントする．$i = 2$ のとき，$z_{2\tilde{x}_2} - H(F(k_1, x_{|1}||\tilde{x}_2), r) = z_{2\pi(F(k_2,x_{|1}),3)} - H(F(k_1, x_{|1}||\pi(F(k_2, x_{|1}), 3)), r) = z_{21} - H(F(k_1, x_{|1}||1), r) = 0 - 2 = 1$ となり，$CMP(x,y) = 1$，すなわち $x > y$ と判定される．

2.1.3 セキュリティ

$ct_L^{(x)}$ と $ct_R^{(y)}$ からは x と y の大小と，$x_i \neq y_i$ となる最初のブロックのインデックス i 以外の情報は漏れない．特に，$ct_R^{(y)}$ と平文 y' の暗号文 $ct_R^{(y')}$ から y と y' の大小を得ることはできない．

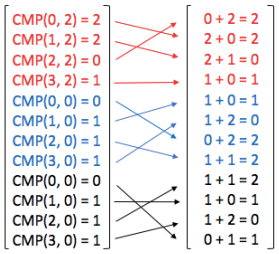

図 1　y の右暗号化

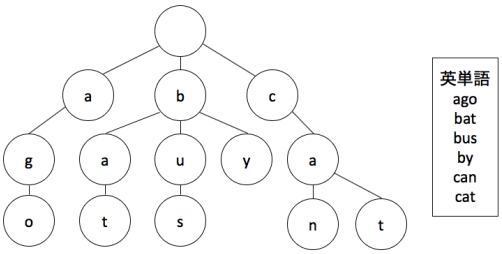

図 2　Trie木

2.2 Trie木

Trie 木とは，文字列を格納するためのデータ構造である．共通のプレフィックスを同じノードにまとめて，異なる部分を後続のノードとする．

図 2 に Trie 木の例を示す．この例では，1 つのノードにつき 1 つの文字が格納されるとする．“bat”,“bus”,“by” は ‘b’ が共通のプレフィックスであり，“at”,“us”,“y” が異なる部分である．‘b’ が ‘a’,‘u’,‘y’ へのポインタを持ち，後続の文字においても同様にポインタを持つことで先頭から検索をしたときにそれぞれの単語がヒットするようになる．

3 関連研究

本節では，本研究の基となる研究とその課題を述べる．

3.1 概　　要

ユーザのアクセス権限削除を効率に行うために，プロキシ再暗号化可能な暗号を用いた先行研究 [4] に対し，平田らは範囲クエリを考慮して，順序比較可能暗号を用いて検索用タグを作成する手法を提案した [5]．データの蓄積と検索の流れは図 3 と図 4 に示す．

児玉らのプロキシ再暗号化可能な暗号を用いた手法では，BBS 暗号 [3] を利用しており，暗号文から元の平文の大小比較の結果を得ることができない．

平田らの手法では，データの暗号文とは別にデータに含まれている数値から検索用タグを作成し，これを順序比較可能暗号を用いて暗号化するため，大小比較が可能である．

3.2 検索用タグの作成と検索

Lewi らの順序比較可能暗号 [6] を利用して，検索用タグの作成や検索を行なっている．検索用タグの作成方法や検索方法について簡単に述べる．

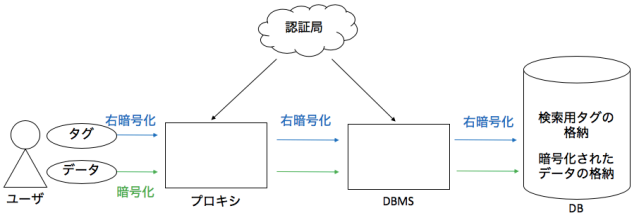

図 3　データの蓄積

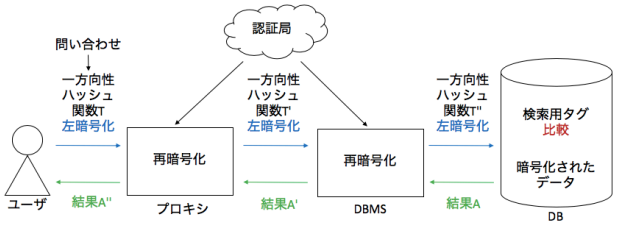

図 4　データの検索

3.2.1　検索用タグの作成

検索用タグの作成において，2.1 節で述べたように，パラメータや関数を定める．認証局において，$k_1, k_2 \leftarrow \{0,1\}^\lambda$ をランダムに選び，$F(k_1,\cdot), F(k_2,\cdot)$ をユーザ用の PRF とする．同様に，アクセスレベル l ごとに $F(k_{1,l}^{(P)},\cdot), F(k_{2,l}^{(P)},\cdot)$ としてプロキシ用の PRF を作成し，$F(k_{1,l}^{(D)},\cdot), F(k_{2,l}^{(D)},\cdot)$ を DBMS 用の PRF として作成する．

（1）　ユーザによるタグの作成

ユーザ用の PRF を用いて 2.1 節の右暗号化と同じ手順で入力 y に対して計算したものを $t_y^{(A)}$ とし，$\tilde{y} = \tilde{y}_1\tilde{y}_2\ldots\tilde{y}_n(\tilde{y}_i = \pi(F(k_2, y_{|i-1}), y_i))$ と Pol と併せてプロキシに送る．

（2）　プロキシによるタグの変換

Pol を見て，アクセスレベルに紐づけられたプロキシ用の PRF である $F(k_{1,l}^{(P)},\cdot), F(k_{2,l}^{(P)},\cdot)$ を取得する．

$\forall k \in \{1,\ldots,n\}$ に対して，$w_{ik} = z_{i\pi^{-1}(F(k_{2,l}^{(P)}, \tilde{y}_{|i-1}), k)} + H(F(k_{1,l}^{(P)}, \tilde{y}_{|i-1}||k), r)(mod\ 3)$ を計算し，プロキシ作成タグを $t_{y,l}^{(P)} = (r, w_{11},\ldots,w_{1d}, w_{21},\ldots,w_{2d},\ldots,w_{n1},\ldots,w_{nd})$ とする．$t_{y,l}^{(P)}$ と $\hat{y} = \hat{y}_1\hat{y}_2\ldots\hat{y}_n(\hat{y}_i = \pi(F(k_{2,l}^{(P)}, \tilde{y}_{|i-1}), \tilde{y}_i))$ と Pol を DBMS に送る．

（3）　DBMS によるタグの変換

Pol を見て，アクセスレベルに紐づけられた DBMS 用の PRF である $F(k_{1,l}^{(D)},\cdot), F(k_{2,l}^{(D)},\cdot)$ を取得する．

$t_{y,l}^{(P)}$ に対して，DBMS 用の PRF を用いてプロキシによるタグの変換と同様の計算を行い，$t_{y,l}^{(D)} = (r, s_{11},\ldots,s_{1d}, s_{21},\ldots,s_{2d},\ldots,s_{n1},\ldots,s_{nd})$ を作成してこれをデータベースに保存する．

3.2.2　タグの検索

（1）　ユーザによる落とし戸の作成

ユーザ用の PRF を用いて 2.1 の左暗号化と同じ手順で入力 x に対して計算し，$\tilde{x}_i, v_i = F(k_1, x_{|i-1}||\tilde{x}_i)$ を求め，$q_x^{(B)} = ((v_1,\tilde{x}_1),\ldots,(v_n,\tilde{x}_n))$ とし，これをプロキシに送る．

（2）　プロキシによる落とし戸の変換

B に紐づけられたプロキシ用落とし戸変換鍵 $tk_B^{(P)} = \{F(k_{1,l}^{(P)},\cdot), F(k_{2,l}^{(P)},\cdot) | l \in AL_B\}$ を認証局から取得する．

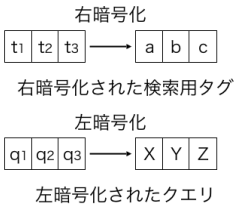

図 5　暗号化されたタグとクエリ

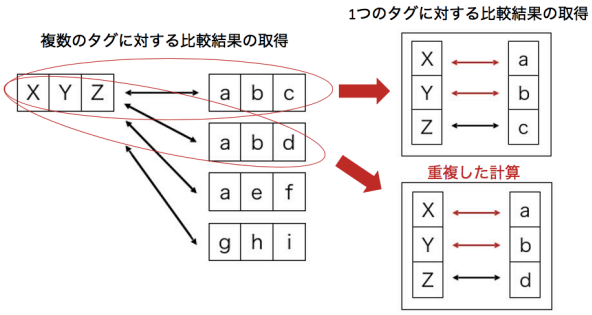

図 6　先行研究の重複した計算

$q_x^{(B)}$ から $\tilde{x}_i$ を取り出し，これを入力としてプロキシ用落とし戸変換鍵を用いて，ユーザによる落とし戸の作成と同様に左暗号化の計算を行い，$\hat{x}_i, \tilde{v}_i$ を求める．$q_{x,l}^{(P)} = ((v_1,\tilde{v}_1,\hat{x}_1),\ldots,(v_n,\tilde{v}_n,\hat{x}_n))$ とし，これを DBMS に送る．

（3）　DBMS による落とし戸の変換と比較 B に紐づけられた DBMS 用落とし戸変換鍵 $tk_B^{(D)} = \{F(k_{1,l}^{(D)},\cdot), F(k_{2,l}^{(D)},\cdot) | l \in AL_B\}$ を認証局から取得する．

DBMS 用落とし戸変換鍵を用いて，プロキシによる落とし戸の変換と同様に $\check{x}, \hat{v}$ を計算し，$q_{x,l}^{(D)}$ を求める．

$t_{y,l}^{(D)}$ と $q_{x,l}^{(D)}$ から一致比較，大小比較を行う．$s_{ix_i} - H(v_i,r) - H(\tilde{v}_i,r) - H(\hat{v}_i,r) \neq 0(mod\ 3)$ となる最小のインデックス i を見つけ，これを ℓ として $s_{\ell x_\ell} - H(v_\ell,r) - H(\tilde{v}_\ell,r) - H(\hat{v}_\ell,r)(mod\ 3)$ を結果として出力する．そのような ℓ が存在しなければ 0 を出力する．

3.3　課　　題

平田らの手法 [5] では，一致比較，大小比較を行う際に最小のインデックス i を見つけるため，$t_{y,l}^{(D)}$ と $q_{x,l}^{(D)}$ の上位ブロックから計算を行う．全ての検索用タグに対して，$\forall i \in \{1,\ldots,\ell\}$ において，$s_{ix_i} - H(v_i,r) - H(\tilde{v}_i,r) - H(\hat{v}_i,r)(mod\ 3)$ を計算する必要があり，上位ブロックの同じ検索用タグに対し，上位ブロックにおいて同じ計算を繰り返しており，非効率である．

右暗号化された検索用タグと左暗号化されたクエリを図 5 に，重複して計算を行なっている検索用タグの例を図 6 に示す．初めに，右暗号化された検索用タグ abc と左暗号化されたクエリ XYZ から数値 $t_1t_2t_3$ と数値 $q_1q_2q_3$ の大小比較結果を取得する．上位ブロックである a と X を用いた計算を行い，t_1 と q_1 の値が等しければ，b と Y を用いて比較結果の取得を行う．次に $t_1t_2t_4$ を右暗号化した abd と左暗号化されたクエリ XYZ から大小比較結果を取得する．このとき，a と X を用いた計算をしなければならず，先程の計算と重複している．更に，t_1 と q_1 の値が等しいときには，b と Y を用いた計算を行わなければならず，こちらも重複している．

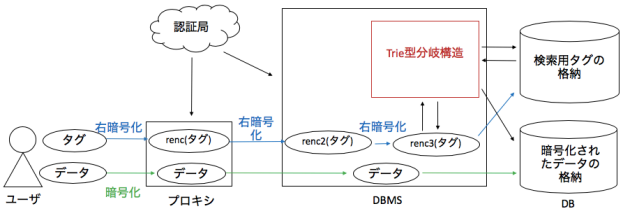

図 7　本研究における蓄積

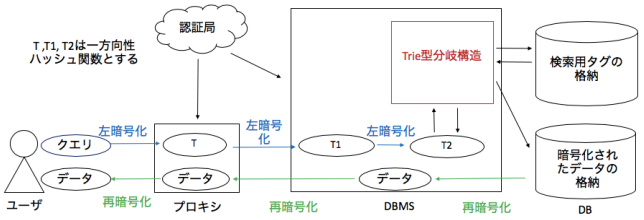

図 8　本研究における検索

4　提 案 手 法

本節では，本研究の目的について述べ，検索用タグの Trie 型分岐構造的管理を行う手法を提案する．また，先行研究と本研究の計算量やセキュリティについて考察をする．

4.1　研 究 目 的

本研究では，3.3 節で述べた課題を解決する．上位ブロックの同じ検索用タグに対して，上位ブロックにおける同じ計算を省くことでデータの検索における時間効率の改善を行う．

4.2　提案手法の概要

本研究でも先行研究 [5] と同様のモデルを想定し，ユーザがプロキシサーバに対して，保持する情報の蓄積や蓄積されているデータの検索を要求することができる．認証局がユーザのクラスやクラスのアクセスレベルを保持する．検索用タグにおけるブロックとは，3.2.1 節で作成した検索用タグ $t_{y,l}^{(D)}$ のうちの $(s_{i1},\ldots,s_{id})(\forall i \in \{1,\ldots,n\})$ を指す．また，3.2.2 説における落とし戸をここでは一方向性ハッシュ関数とする．

4.2.1　データの蓄積

データの蓄積の流れを図 7 に示す．

（1） 公開鍵で蓄積するデータを暗号化する．また，蓄積するデータにおける数値を検索用タグとし，これを右暗号化する．暗号文と検索用タグをプロキシに送信する．

（2） 認証局から検索用タグを変換するための PRF を取得し，ユーザから受け取った検索用タグを右暗号化する．検索用タグとユーザから受け取った暗号文をそのまま DBMS に送信する．

（3） 認証局から検索用タグを変換するための PRF を取得し，プロキシから受け取った検索用タグを右暗号化する．これと蓄積されている検索用タグを用いて Trie 型分岐構造の分岐ノードにインデックスを格納する．検索用タグのインデックスのハッシュ値をプロキシから受け取った暗号文に付与してからデータベースに蓄積する．また，検索用タグもデータベースに蓄積する．

4.2.2　データの検索

データの検索の流れを図 8 に示す．

（1） 検索に使用する値を左暗号化し，これとユーザの ID をプロキシに送信する．

（2） 認証局から一方向性ハッシュ関数を変換するための鍵を取得し，ユーザがアクセス可能なレベルに対してそれぞれに一方向性ハッシュ関数を左暗号化して DBMS に送信する．

（3） 認証局から一方向性ハッシュ関数を変換するための鍵を取得して左暗号化する．これと蓄積されている検索用タグ，Trie 型分岐構造を用いて暗号文のインデックスを取得する．

（4） インデックスから暗号文の取得をする．その後に認証局から DBMS 用の再暗号化鍵を取得し，暗号文を変換する．これをプロキシに送信する．

（5） 認証局からプロキシ用の再暗号化鍵を取得し，暗号文を変換する．これをユーザに送信する．

（6） ユーザの秘密鍵によって暗号文を復号する．

本研究におけるデータの暗号化手法，検索用タグや一方向性ハッシュ関数の変換，比較に用いる暗号化手法も先行研究と同様とする．

4.3　Trie 型分岐構造の概要

まず，検索用タグが n ビットずつ d 個のブロックに分けられているとする．本研究における Trie 型分岐構造では，根ノードから末端ノードまでの距離は必ず d となるようにする．分岐ノードは，子ノードへのポインタに加えて，逆順に辿ることを可能にするために親ノードへのポインタを持つ．また，全ての分岐ノードを辿ることを考慮し，親ノードに対して何番目の子ノードであるかという情報も持つ．ここで検索用タグに関する情報は，インデックスのみを保持する．このインデックスは，検索用タグの順番をランダムにするためにハッシュ化する．Trie 木との違いとして，Trie 木ではノードで値を持つのに対して，この Trie 型分岐構造の分岐ノードでは検索用タグやブロックを保持しないことが重要である．

4.4　Trie 型分岐構造の構築

検索用タグの上位ブロックの同じものをまとめていくことで Trie 型分岐構造を構築する．分岐ノードは検索用タグを保持していないため，分岐ノードに格納されているインデックスから蓄積されている検索用タグを取り出す．

Trie 型分岐構造の構築手順を以下に示す．

（1） 分岐ノードのポインタ p_1, p_2 を用意する．それぞれの指す分岐ノードは根ノードとする．

（2） p_2 を末端ノードに移動し，分岐ノードに格納されているタグから検索用タグを取得する．

（3） 蓄積する検索用タグと蓄積されている検索用タグが等しいか上位ブロックから判定する．一致したブロックの最後のブロックの深さを i とする．

（4） i がブロックの数 d と等しい場合は p_1 から p_2 の分岐ノードにインデックスを格納し，終了とする．

（5） p_1 は i 番目の分岐ノードまで，インデックスを格納し

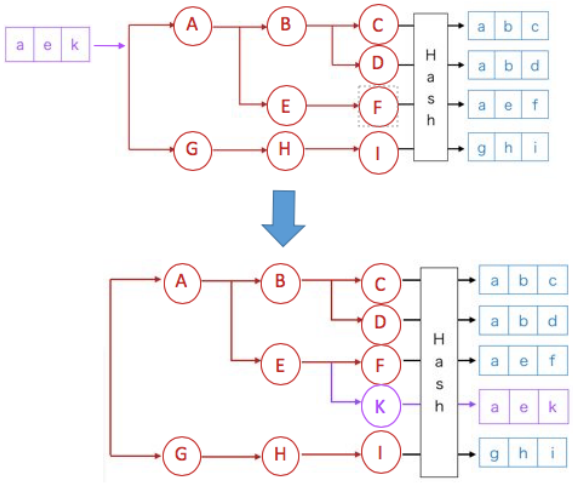

図 9　新しく生成された Trie 型分岐構造

ながら移動する．

（6） p_2 は i+1 番目の分岐ノードまで戻り，何番目の子ノードか確認し，p_2 の指す分岐ノードに辿っていない兄弟ノードがあれば (2) に戻る．

（7） (6) において辿っていない兄弟ノードがなければ p_1 の指す分岐ノードに対して子ノードを作成し，インデックスを格納して p_1 をその子ノードに移動させる．これを d-i 回繰り返して終了する．

例を図 9 に示す．この例ではまず p_2 は C の分岐ノードまで辿り，検索用タグ abc を取り出す．最上位ブロック a は等しいが次のブロックが異なるため，p_1 は A の分岐ノードに移動する．p_2 は B の分岐ノードまで戻って兄弟ノードを辿り，末端ノードである F の分岐ノードに行く．F の分岐ノードで検索用タグ aef を取り出し，2 番目のブロックから比較をする．3 番目のブロックが異なることから，p_1 は A の分岐ノードにインデックスを格納した後に E の分岐ノードに行く．このとき，F の分岐ノードに兄弟ノードが存在しないため，E の子ノード K を新しく生成する．そして，E の分岐ノードと K の分岐ノードにインデックスを格納し，検索用タグ aek を紐づける．

4.4.1　インデックスと暗号化されたデータの紐付け

検索用タグのインデックスと暗号化されたデータのインデックスが同じ値の場合，検索用タグと暗号化されたデータの対応を容易に把握することができる．よって，検索用タグのインデックスをハッシュ化した値を暗号化されたデータのインデックスとする．

構築された Trie 型分岐構造と検索用タグとの紐付け，また，検索用タグと暗号化されたデータとの紐付けを図 10 に示す．この例では，暗号化されたデータ D_1 に対する検索用タグ $t_1t_2t_3$ を右暗号化したものを abc とする．

4.5　Trie 型分岐構造におけるタグの検索

Trie 型分岐構造の分岐ノードからインデックスを取得する手順を以下に示す．

（1） 分岐ノードのポインタ p_1, p_2 を用意する．それぞれの指す分岐ノードは根ノードとする．

（2） p_2 を末端ノードに移動し，分岐ノードに格納されているインデックスから検索用タグを取得する．

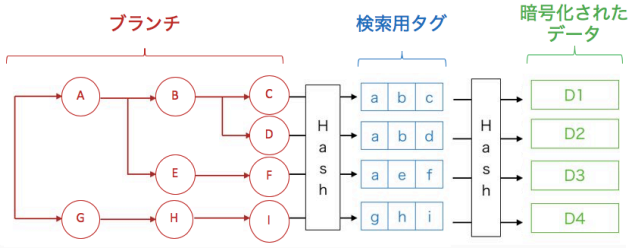

図 10　Trie 型分岐構造と紐付けされた検索用タグ，暗号化されたデータ

（3） 一方向性ハッシュ関数と検索用タグの大小比較結果の取得を上位ブロックから行う．比較結果を取得できたブロックの深さを i とする．

（4） p_1 を i 番目の分岐ノードまで移動させ，比較結果が取得したいものであればインデックスの取得を行う．

（5） p_1 が何番目の兄弟ノードか確認をし，辿っていない兄弟ノードがあれば p_1 をその分岐ノードに移動させて (2) に戻る．

（6） (5) において辿っていない兄弟ノードがなければ，p_1 を親ノードに移動させて (5) に戻る．このとき，p_1 の親ノードが根ノードであれば終了する．

図 9 において，新しく生成された Trie 型分岐構造を例とする．検索用タグやクエリの数値の対応表を表 1 とする．また，$q_1 = t_1, q_1 > t_8, q_2 < t_2, q_2 = t_5, q_3 > t_6, q_3 < t_7$ と仮定する．$q_1q_2q_3$ より大きい数値のインデックスを取り出す．

まずは p_2 は C の分岐ノードに移動し，分岐ノードに格納されているインデックスから検索用タグ abc を取り出す．2 番目のブロック，Y と b によって $q_2 < t_2$ であることが分かるため p_1 を B の分岐ノードに移動させる．このとき，取り出したい結果ではなかったためインデックスの取得は行わない．p_1 は E の分岐ノードに移動し，p_2 は F の分岐ノードに移動する．F の分岐ノードから検索用タグ aef を取り出し，2 番目のブロックから比較結果の取得を行う計算をする．3 番目のブロック，Z と f によって $q_3 > t_6$ であることが分かるため，p_1 は F の分岐ノードに移動し，インデックスの取得を行う．p_1, p_2 は K の分岐ノードに移動し，検索用タグ aek を取り出し，3 番目のブロックから比較結果の取得を行う計算をする．$q_3 < t_7$ であるため，インデックスは取得しない．兄弟ノードがあるか確認しながら p_1 は親の方向に進み，A の分岐ノードに到達後，兄弟ノードである G の分岐ノードに移動する．p_2 は I の分岐ノードに移動し，検索用タグ ghi を取り出し，X と g から $q_1 > t_8$ であることが分かるため，G の分岐ノードからインデックスを取得する．p_1 が兄弟ノードが存在するか確認しながら親の方向に進めば根ノードに移動するため，ここで終了とする．

4.5.1　インデックスから暗号化されたデータの取得

分岐ノードから取得したインデックスに対して，ハッシュをかけて暗号化されたデータのインデックスを取得する．このインデックスから暗号化されたデータを取得する．

4.6　計算量の考察

検索用タグの数を N，1 つの検索用タグのブロックの数を M

表 1　図 9 における検索用タグとクエリの数値

暗号化前	暗号化後
$t_1t_2t_3$	abc
$t_1t_2t_4$	abd
$t_1t_5t_6$	aef
$t_1t_5t_7$	aek
$t_8t_9t_{10}$	ghi
$q_1q_2q_3$	XYZ

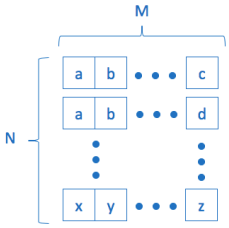

図 11　先行研究における検索用タグ

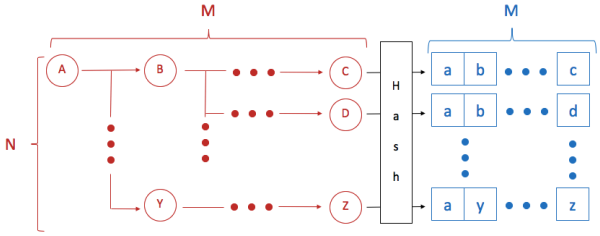

図 12　本研究における時間が最短となり得る Trie 型分岐構造

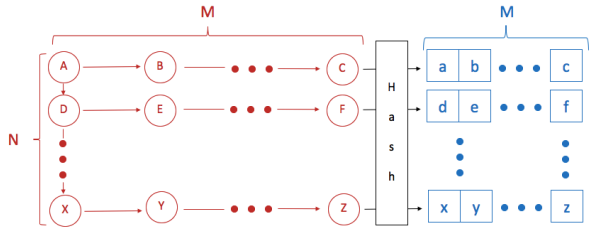

図 13　本研究における時間が最長となり得る Trie 型分岐構造

とする．

先行研究では，検索用タグはブランチによってまとめられておらず，図 11 のようになっている．全ての検索用タグにおいて，最上位ブロックで大小比較の結果を得られるときに計算量は最も少なくなり，N となる．また，全ての検索用タグにおいて最後のブロックで大小比較の結果を得られるときに，計算量は最も多くなり，$N \times M$ となる．

本研究では，検索用タグはブランチによってまとめられているが，検索用タグの数値の偏りによってブランチの数が変わるため，時間が最短となり得るとき (図 12) と最長となり得るとき (図 13) を考慮する．最短となり得るのときでは図 12 において，A のブランチで判断できるときであり，末端ノードまで辿るため計算量は $M+1$ となる．最悪となり得るときでは全てのブロックを辿る必要があるため，計算量は $N \times M$ となる．

先行研究と本研究の最も計算量が少ないときを比較する．ブロックの数 M はセットアップで固定し，値は大きくならない．それに対し，検索用タグの数 N は可変であり，大きくなるにつれて本研究の手法の方が計算量が少ない．

表 2　生成されたデータセット

データセット	都道府県	距離 [km]	人数	データサイズ [triples]
A	青森	0.8	46	643
B	福島	0.8	271	3,853
C	福島	1.5	2,672	37,736

4.7　セキュリティの考察

データベース管理者による不正なアクセスや外部から DBMS への攻撃について考える．データベース管理者が Trie 型分岐構造やインデックス，データや検索用タグの改竄や削除をするなどの攻撃は可能だが，これらは今回は考えないものとする．

検索用タグや一方向性ハッシュ関数は順序比較可能暗号 [6] の安全性から秘匿性が守られており，秘匿検索が可能である．

Trie 型分岐構造の構築を行う際には，検索用タグ同士を比較する．しかし，2.1 節のセキュリティから右暗号文同士の比較では大小の比較結果を得ることはできず，右暗号文のみでは数値としての意味はなさない．よって右暗号化されている検索用タグ同士からは大小の情報を得ることができない．

Trie 型分岐構造を解析する場合，Trie 型分岐構造の分岐ノードには，検索タグのインデックスや親ノードと子ノードのポインタなどが格納されているものの検索タグのブロックに関する情報は保持していない．つまり，図 10 において分岐ノード A は検索用タグのブロック a を保持していないため，漏洩することはない．

また，検索用タグのインデックスにハッシュをかけて暗号化されたデータのインデックスとすることで位置関係による検索用タグと暗号化されたデータの対応関係の情報の漏洩を防ぐ．

5　実　　験

提案手法の実装に，Apache により開発されている Java 用のオープンソースのセマンティックウェブフレームワークである Jena [8] を利用する．

5.1　実 験 環 境

今回は 1 つのサーバのみで実験を行なった．実験には以下の環境を用いた．

- OS: MacOS High Sierra ver10.13.6
- プロセッサ: 1.8GHz Intel Core i5
- メモリ: 8GB 1600MHz DDR3
- Java: 11.0.1
- Jena: 2.12.1

5.2　データセット

実験データには，RDF [9] データベース用ベンチマークツールとして，Nguyen らの研究 [10] で提案された SIBM(Shelter Information Benchmark) を利用する．SIBM では，東日本大震災時の情報を基にデータセットを生成することができる．データセットは氏名，年齢，性別などの個人情報と住所，避難場所名などの避難場所を含む．被災地の都道府県と中心からの距離をパラメータとして以下のデータセットを生成した．

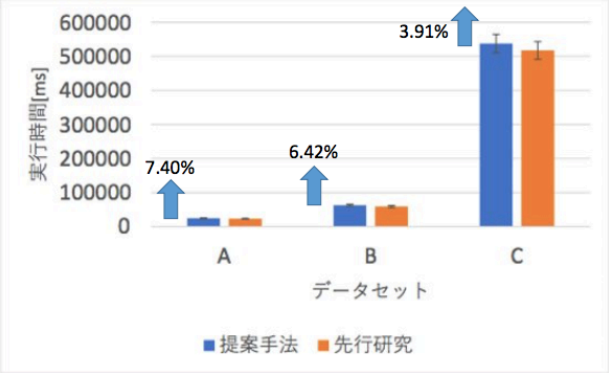

図 14　データを暗号化したときのオーバーヘッドのグラフ

5.3　データの挿入におけるオーバーヘッド

Trie 型分岐構造を構築するためのオーバーヘッドを計測した．検索用タグの数字の範囲は 3 から 85 である．実験はそれぞれ 10 回ずつ行い，その平均を表 3 と図 14 に示す．図 14 以降に出てくるグラフのエラーバーは 95%信頼区間とする．オーバーヘッドは高々7.40%であった．データの暗号化に対して Trie 型分岐構造の構築のコストは非常に小さいと考えられる．

表 3　データを暗号化したときのオーバーヘッド

データセット	提案手法 [ms]	先行研究 [ms]	オーバーヘッド [%]
A	2.48×10^4	2.31×10^4	7.40
B	6.29×10^4	5.91×10^4	6.42
C	5.38×10^5	5.17×10^5	3.91

5.4　検索における性能向上の評価

Trie 型分岐構造による検索を行う場合と Trie 型分岐構造を利用せずに検索を行う場合を計測した．

5.4.1　データサイズの変更

データサイズを変更した場合の検索時間を計測した．検索用タグの数字の範囲は 3 から 85 である．また，アクセスレベルは最も低いレベルとし，数値が $x > 30$ を満たす検索用タグに紐付くデータを取得する．それぞれのデータサイズ，10 人，271 人，2672 人に対し，該当した検索用タグの数は 4 個，150 個，1416 個である．実験はそれぞれ 10 回ずつ行なった．結果は表 4,5,6 と図 15 である．

データの復号の計算においては本研究と先行研究において同じ手法で計算しているため差異はない．インデックスの取得においては，データサイズが大きくなるにつれて大きな効果が得られている．これはデータの範囲が原因であると考えられる．検索用タグの数字の範囲が 3 から 85 であることから，Trie 型分岐構造における末端ノードの数は高々83 個である．同じ数値のものは同じ分岐ノードに格納されるため，計算を抑えることができたと考えられる．しかしながら，クエリを実行してから復号するまでの時間で比較すれば，データサイズとの相関性はない．Trie 型分岐構造からインデックスを取得するまでの時間に対して，復号の時間の方がかかるからであると考えられる．

5.4.2　検索用タグの数値の範囲とクエリの変更

検索用タグの数値の範囲とクエリの数値の変更をした場合の時間を計測した．アクセスレベルは最も低いレベルとした．

表 4　データの復号における実行時間

データサイズ	提案手法 [ms]	先行研究 [ms]
643triples	5.19×10^2	5.43×10^2
3853triples	2.07×10^3	2.28×10^3
37736triples	1.13×10^4	1.11×10^4

表 5　データサイズを変更したときのデータのインデックスの取得における実行時間

データサイズ	提案手法 [ms]	先行研究 [ms]	実行時間の削減率 [%]
643triples	1.52×10	2.64×10^2	94.2
3853triples	1.16×10	1.10×10^3	98.9
37736triples	1.49×10	5.73×10^3	99.7

表 6　データサイズを変更したときのクエリを実行してから復号するまでの時間

データセット	提案手法 [ms]	先行研究 [ms]	実行時間の削減率 [%]
A	5.88×10^2	8.65×10^2	32.0
B	2.21×10^3	3.54×10^3	37.5
C	1.25×10^4	1.82×10^4	31.4

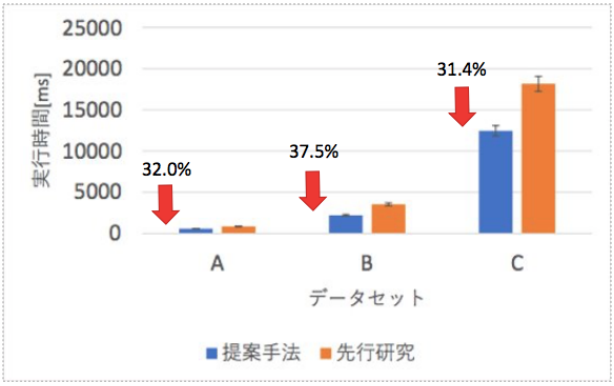

図 15　データサイズを変更したときのクエリを実行してから復号するまでの時間のグラフ

データサイズは人数が 271 人であり，3,853 triples とする．実験はそれぞれ 10 回ずつ行なった．表 7 と図 16 は数値の範囲を 3 から 85 とし，表 8 と図 17 は数値の範囲を 3 から 1000 としたときの結果である．

クエリを変更したときの差はインデックスの取得以上にクエリを実行してから復号するまでの時間において如実に現れている．これは復号するデータの数に差があるからだと考えられる．数値の範囲を 3 から 1000 にしたときには数値の偏りが存在したために，復号するデータの数に差があったことから数値の範囲を 3 から 85 にしたとき以上にクエリの変更による差が大きい．クエリを実行したから復号するまでの時間においては，実行時間の削減率は復号するデータの数と相関関係がある．

表 7　数値の範囲が 3-85 のときのデータのインデックスの取得における実行時間

クエリと該当する検索用タグ	提案手法 [ms]	先行研究 [ms]	実行時間の削減率 [%]
クエリ:$x > 30$ 検索用タグ:150	1.16×10	1.10×10^3	98.9
クエリ:$x < 18$ 検索用タグ:54	1.70×10	9.90×10^2	98.3

6　おわりに

6.1　まとめ

先行研究 [5] では，プロキシ再暗号化可能な暗号を用いた研究

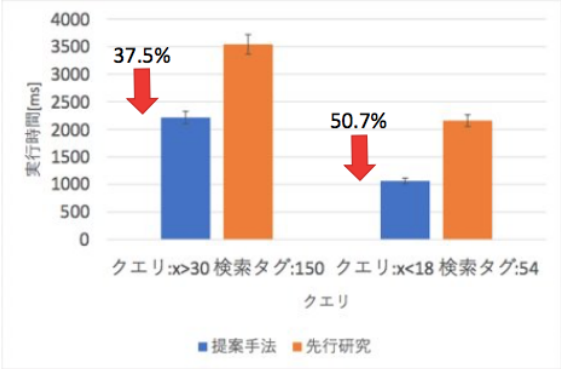

図 16　数値の範囲が 3-85 のときのクエリを実行してから復号するまでの時間のグラフ

表 8　数値の範囲が 3-1000 のときのデータのインデックスの取得における実行時間

クエリと該当する検索タグ	提案手法 [ms]	先行研究 [ms]	実行時間の削減率 [%]
クエリ:$x > 300$ 検索タグ:19	8.94	1.16×10^3	99.2
クエリ:$x < 200$ 検索タグ:317	1.70×10	1.20×10^3	98.6

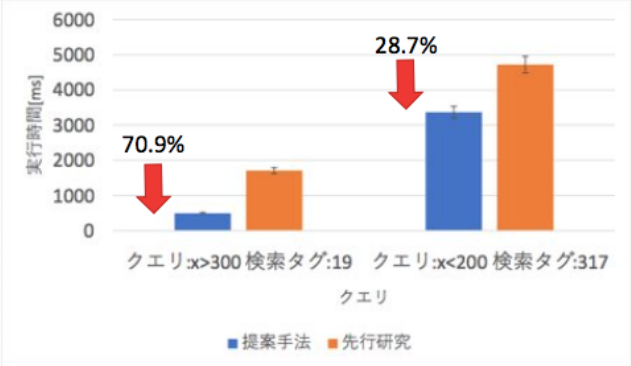

図 17　数値の範囲が 3-1000 のときのクエリを実行してから復号するまでの時間のグラフ

に対し，順序比較可能暗号を用いることで範囲クエリに対応できないという課題の解決を図った．しかし，一度の問い合わせにおいて多くのブロックとの計算が必要であり非効率であるという課題があったため，本研究ではこれを解決するために Trie 型分岐構造に検索タグの上位ブロックの同じものをまとめる手法を提案した．

また，先行研究と本研究の手法における計算量の考察をし，検索用タグからの大小の情報や Trie 型分岐構造から検索用タグの情報が漏洩しないようなセキュリティを持つことを確認した．

更に，SIBM によって生成されるデータセットを用いて，データの蓄積や検索の実験を行なった．Trie 型分岐構造の構築におけるオーバーヘッドは高々7.40%であり，検索においては復号するデータの数に依存はするが，インデックスの取得の実行時間では最大で 99.7%，クエリを実行してから復号するまでの実行時間では最大で 70.9%の削減がされた．

6.2　今後の課題

今後の課題として，以下が挙げられる．

6.2.1　追 加 実 験

今回行なった実験におけるデータサイズは最大で 37736triples であり，高々2.8MB 程度のものであるため，更にデータサイズを大きくしてクエリを変更したときの Trie 型分岐構造生成のオーバーヘッドや検索性能の実験を行なって評価をすることが必要である．

6.2.2　プロキシサーバと認証局サーバの実装

本研究における実験では，暗号化，復号化の計算を行うことはできているものの，プロキシサーバと認証局を介して実験を行うことができていない．クライアント，プロキシサーバ，データベースサーバ，認証局サーバとそれぞれ分けて実装をすることで本研究における評価を正確に行うことができる．

6.2.3　データ削除時の検索用タグの削除方法の検討

検索用タグと暗号化されたデータの紐付けにハッシュを用いているため，暗号化されたデータのインデックスから検索用タグのインデックスを求めることは不可能である．よって，削除したデータから削除対象の検索用タグを求めることができない．データの格納や検索を行なっていないときに検索用タグと紐付くデータが存在するか調べることで検索用タグの削除は可能である．

6.2.4　検索用タグとデータの紐付けの見直し

暗号化されたデータのインデックスを取得することはできているものの，SPARQL によってデータを取得している．そのため，インデックスと全てのデータを照合しなければならない点で効率が悪い．SPARQL を介することなくインデックスからデータの取得をすることができれば，クエリの応答時間を短縮できると考えられる．

文　　献

[1] 総務省関東総合通信局防災対策推進室．災害時に活用できる情報伝達手段．Retrieved December 27, 2019.
[2] 独立行政法人情報処理推進機構．災害に対応する IT システム検討プロジェクトチーム活動報告．January 2013. Retrived December 27, 2019.
[3] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In Proceedings of Eurocrypt ‘98. Vol. 1403. 127-144.
[4] 児玉快，横田治夫．データやユーザの効率的な追加・削除が可能 な秘匿情報アクセス手法. 第 7 回データ工学と情報マネジメン トに関するフォーラム，2015.
[5] 平田拓三，宮沢駿輔，Hieu Hanh Le，横田治夫．プロキシ再暗号化と検索タグを用いた範囲クエリ可能な秘匿情報の耐結託共有手法，2018.
[6] K. Lewi and D. J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In ACM CCS, 2016.
[7] O. Goldreich, S. Goldwasser, S. Micali. How to construct random functions. J. ACM, 1986.
[8] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the semantic web recommendations. In Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, WWW Alt. ’04, pp. 74-83. ACM, 2004.
[9] RDF Working Group. RDF - Semantic Web Standards. Retrieved January 7, 2020, from: https://www.w3.org/RDF/
[10] Nguyen Hoai Nam, Yoshitaka Arahori, and Haruo Yokota. SIBM: 避難場所情報に対する RDF データセットベンチマークツール. 第 7 回データ工学と情報マネジメントに関するフォーラム, 2015.