プライバシー保護を考慮した効率的な検索のための安全な索引構造

大井篤† 山本博章† 山下智穂‡ 中村伸一† 白井啓一郎† 岡本正行†

†信州大学工学部 ‡アヴァシス株式会社
380-8553 長野市若里 4-17-1

あらまし プライバシーや機密データの保護の観点から，コンピュータに保存するデータの安全な管理に対する要求が高まっている．これを実現するための最も効果的な方法の一つはデータの暗号化である．しかし，データを暗号化することによって，効率的に検索することが難しくなってしまう．そのため，暗号化されたデータを暗号化したまま効率的に検索する技術の開発が望まれている．本論文は，効率的な検索を提供する安全な索引構造について提案する．提案する索引構造はブルームフィルタを階層的に構成することによって実現されており，暗号化データの検索を暗号化したまま効率的に行うことができる．

A Secure Index for Efficient Search Preserving Privacy

Atsushi Ooi† Hiroaki Yamamoto† Chiho Yamashita‡ Shin-ichi Nakamura†
Keiichiro Shirai† Masayuki Okamoto†

†FacultyEngineering,Shinshu University ‡AVASYS CORPORATION
4-17-1 Wakasato, Nagano-shi 380-8553, JAPAN

Abstract From a view point of privacy-preserving and security of sensitive data, a need to manage securely data stored in remote storage servers rapidly grows . One of most effective ways is to encrypt data. However, encryption of data makes the search difficult. Therefor it is desirable to develop efficient search methods on encrypted data. In this paper, we present an efficient and secure method to search for encrypted documents. Given a keyword, the proposed method find out documents containing the keyword without decrypting documents. To achieve the efficiency and the security for search, we construct a secure index using bloom filters with a hierarchical structure.

1 はじめに

インターネットが普及し，多くのサービスがネットワークを通して行なわれるようになってきた．このような情報化社会において，プライバシーの保護，機密情報の保護は非常に重要な課題となっている．情報検索においても，データを保護するためのセキュリティの重要性が益々増してきている．例えば，データベースのシステム管理者とデータそのものの管理者とが異なる場合が多々発生する．その場合，システムの管理者は信頼がおけるものと仮定して運用する．しかし，第３者である管理者にこのような仮定を置くことは，セキュリティの観点から見ると十分とは言えない．より安全に運用することが望まれる．セキュリティを高める一つの方法は，データを暗号化して保存することである．こうすれば，システム管理者はデータへのアクセスはできるが，その内容まで知ることはできない．問題点は，暗号化によって，検索が難しくなってしまうことである．そのため，暗号化データに対する検索手法に関する研究が活発に行なわれるようになってきた．[1, 5, 6, 7, 8]

本論文は，暗号化されたドキュメントを効率的に検索するための索引構造を提案する．本索引構造はブルームフィルタ [2] を利用して構成され，かつ，与えられたキーワードに対し，第 3 者に一切の情報を漏らすことなく，暗号化されたドキュメントからキーワードにマッチするものを効率的に検索機能を

提供する．

従来の手法は，各ドキュメントに対し，そのドキュメントのキーワードを登録するためのブルームフィルタを構成するという形をとっている．したがって，与えられたキーワードが含まれるかチェックするとき，各ドキュメントのブルームフィルタをチェックする必要がある．この場合，検索時間がドキュメントの数に比例してしまい効率が悪い．Goh [5] は，改善案として，ブルームフィルタを 2 分木で管理する方法を提案している，しかし，彼の提案手法は，検索速度を改善できるが，安全性を犠牲にしている．すなわち，彼は，ドキュメントごとにブルームフィルタを割り当て，それを 2 分木に管理するため，安全性の重要な要素であった 2 段階暗号化を行なっていない．これを改良した手法として，山下・山本 [9] は，2 段階暗号化を取り入れ，ブルームフィルタを階層化した索引構成法を示した．彼らの手法は，キーワードにマッチするドキュメント数に比例した時間で検索を行なうことができるため，マッチするドキュメントの数が少ないほど早く動作する．しかし，マッチするドキュメントが多くなると，従来法より効率が悪くなる．

本論文では，山下・山本の構成法を改善し，より効率的な検索を可能にする暗号化索引構成法を示す．この改善によって，従来法と比較し，最悪の場合でも，同程度の時間で検索ができるようになる．なお，安全性については，山下・山本と同じである．

2 ブルームフィルタ

ブルームフィルタは，Bloom [2] によって開発されたデータ構造で，ビット列で構成され，ある要素が集合に含まれるかどうかを効率的にチェックできる．ブルームフィルタ BF の概要を説明する．今，$W = \{w_1, \ldots, w_l\}$ をキーワードの集合とし，$h_1, \ldots, h_k$ を W 上のキーワードを $[0, m-1]$ の整数へ変換するハッシュ関数とする．そのとき，m ビットのブルームフィルタは，各 w_i に対し，$h_1(w_i), \ldots, h_k(w_i)$ に対応する BF のビットを 1 にセットすることで構成される．与えられた語 w が W に入っているかどうかは，$h_1(w), \ldots, h_k(w)$ を計算し，BF で対応する位置のビットがすべて 1 かどうかで判定できる．欠点としては，キーワードでない文字列 v に対し，$h_1(v), \ldots, h_k(v)$ のすべての位置のビットが 1 になってしまう場合がある．この場合は，間違った答えを得てしまう．これは正の誤り (false positive) と呼ばれる．なお，W 内のキーワードに対しては必ず正しい答えを返す．

3 ブルームフィルタを用いた暗号化索引

まず，従来の非階層的な構成方法を説明し，それから，階層的な構成方法について述べる．暗号化索引（暗号化インデックス）を用いた検索システムの概略を図 1 に示す．ユーザは暗号化したまま検索することができる．

3.1 従来の非階層的構成方法

一般に暗号検索で用いられる，ブルームフィルタを用いた安全な索引の構成について説明する．本論文の暗号化索引は，この構成法を拡張したものになっている．$D = \{d_0, \ldots, d_{n-1}\}$ をドキュメントの集合，$W = \{w_1, \ldots, w_l\}$ をドキュメントに含まれるキーワードの集合とする．E_p, E_s をそれぞれ，鍵 p, s による暗号化関数とする．ここでは，共通鍵暗号化アルゴリズムを用いる．暗号化索引のためのブルームフィルタを BF としよう．そのとき，BF は，すべてのキーワード w_i に対し，次を実行することにより構成される．

1. キーワード $w_i \in W$ に対し，$E_s(w_i)$ を計算する．
2. 各ドキュメント d_{id} に対し，もし w_i が d_{id} に出現するなら以下を実行する：
   (a) $X_w = E_p(id, E_s(w))$ を計算する．
   (b) k 個のハッシュ関数 $h_1(X_w), \ldots, h_k(X_w)$ を計算し，BF の対応するビットを 1 にセットする．

上の手法で，E_p および E_s を使って 2 回暗号化を行なっている．これは 2 段階暗号化方式と呼ばれ，キーワードの出現頻度を隠すために用いられる方法である．すなわち，ドキュメント ID id とペアで暗号化することによって，キーワードが同じでもドキュメント ID が違うため，暗号結果が異なり，キーワードの出現頻度を隠すことができる．

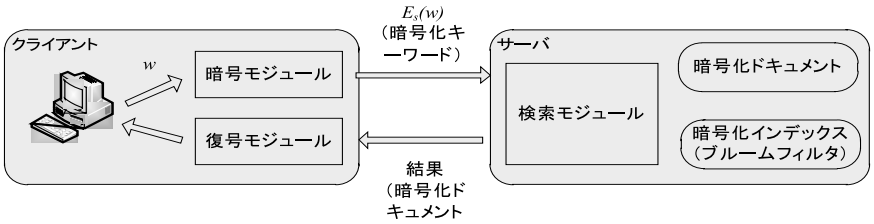

図 1: システム構成図

3.2 階層的ブルームフィルタによる暗号化索引

Procedure *MakeIndex2*($\mathcal{D}$);

すべてのキーワード $w \in W$ に対し，以下を実行する．

(1) for $h = 0$ to $\lceil \log n \rceil$

(2) for $D_i^h \in \mathcal{D}^h = \{D_0^h, D_{2^\alpha}^h, D_{2\cdot 2^\alpha}^h, \ldots, D_{(2^h-1)\times 2^\alpha}^h\}$

(3) if $w \in \mathcal{K}(D_i^h)$, then $h_1(E_p(i, E_s(w))), \cdots,$

(4) $h_k(E_p(i, E_s(w)))$ を計算し，BF_h に登録する．

(5) for-end.

(6) for-end.

図 2: 改良暗号化索引作成アルゴリズム

本節では，階層的に構成する方法を示す．$D = \{d_0, \ldots, d_{n-1}\}$ をドキュメントの集合とする．このとき，もし n が 2 のべき乗でなければ，ダミードキュメントを追加し，個数を 2 のべき乗にする．ダミードキュメントは空のドキュメントであり，以下の分割を効果的に行なうために用いる．さらに，ダミードキュメントによって，新たなドキュメントの追加による暗号化索引の更新が効率的に行なえるようになる．さて，ダミーを追加したドキュメントの集合（拡張ドキュメント集合と呼ぶ）を $\mathcal{D} = \{d_0, \ldots, d_{n-1}, d'_n, \ldots, d'_{N-1}\}$ とする．ここで，d'_n, $\ldots$, d'_{N-1} がダミーであり，$2^{l-1} < n \leq 2^l$ ならば $N = 2^l$ である．山下・山本は，$\mathcal{D}$ を以下のように分割して，暗号化索引を構成した．ここで，D_i^h は，階層 h における i 番目のドキュメント集合を示す．番号 i を D_i^h の ID とよぶ．

1. $D_0^0 = \mathcal{D}$,
2. 任意の $0 \leq h \leq \log N - 1$ に対し，$|D_i^h| \geq 2$ ならば D_i^h を D_{2i}^{h+1} と D_{2i+1}^{h+1} に等分に分割する．

分割の定義から，$0 \leq h \leq \log N$ となる．ここで，$\log N = \lceil \log n \rceil$ であることに注意する．

暗号化索引作成時に，D_i^h の i をドキュメント ID として使う．上記の分割では，ID を 0 から順に割り振ることをしているため，同じ ID とキーワードで何回も暗号化を行うことになり効率が悪い．そこで，本論文では，最悪の検索時間を小さくするため，以下のように，暗号化索引を作成するときの番号の割り振り方を工夫する．

1. $D_0^0 = \mathcal{D}$,
2. 任意の $0 \leq h \leq \log N - 1$ に対し，$|D_i^h| \geq 2$ ならば D_i^h を D_i^{h+1} と $D_{i+2^{\lceil \log n \rceil - (h+1)}}^{h+1}$ に等分に分割する．

上の分割において，D_i^h を D_i^{h+1} と $D_{i+2^{\lceil \log n \rceil - (h+1)}}^{h+1}$ の親とみなせば，拡張ドキュメントの集合の分割は，D_i^h をノードとする 2 分木を構成する．これをドキュメント木と呼ぶことにする．根は D_0^0 となる．また，葉は 1 個のドキュメント d_i だけからなるドキュメントの集合である．我々は，各階層 $0 \leq h \leq \lceil \log n \rceil$ に対し，m ビットのブルームフィルタ BF_h を用意する．改良された方法では，階層 h において，D_i^h の ID を 0 から始め，$2^{\lceil \log n \rceil - h}$ の間隔で順に割り振っていく．親とその左の子が同じ ID をもつので，暗号化とハッシュ関数の計算については，親で計算すれば，左の子をチェックするとき計算する必要がない．そのため，計算量をかなり減らすことができる．

さて，$\alpha = \lceil \log n \rceil - h$ とおいたとき，$\mathcal{D}^h = \{D_0^h, D_{2^\alpha}^h, D_{2\cdot 2^\alpha}^h, \ldots, D_{(2^h-1)\times 2^\alpha}^h\}$ と定義する．また，E_p を，p を鍵とする暗号化関数とする．そのとき，暗号化索引を作成するアルゴリズム *MakeIndex2* ($\mathcal{D}$) を図 2 に与える．ここで，$\mathcal{K}(D_i^h)$ は D_i^h 内のドキュメントのキーワードの集合とする．

例： 分割とドキュメント木の例を与える．5 個のドキュメントに集合 $D = \{d_0, d_1, d_2, d_3, d_4\}$ を考える．

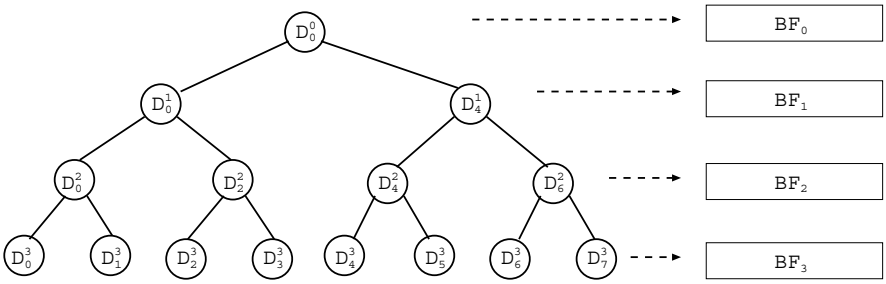

図 3: ドキュメント木とブルームフィルタ

ダミードキュメントを3つ追加し，ドキュメント数を $8 = 2^3$ とする．$\mathcal{D} = \{d_0, d_1, d_2, d_3, d_4, d'_5, d'_6, d'_7\}$ に対するドキュメント木は図 3 となる．ここで，BF_0, BF_1, BF_2, BF_3 は各階層のノードに対するブルームフィルタで，各ノードは以下のようになる：
$D_0^0 = \{d_0, d_1, d_2, d_3, d_4, d'_5, d'_6, d'_7\}$,
$D_0^1 = \{d_0, d_1, d_2, d_3, d_4\}$, $D_4^1 = \{d_4, d'_5, d'_6, d'_7\}$,
$D_0^2 = \{d_0, d_1\}$, $D_2^2 = \{d_2, d_3\}$, $D_4^2 = \{d_4, d'_5\}$,
$D_6^2 = \{d'_6, d'_7\}$,
$D_0^3 = \{d_0\}$, $D_1^3 = \{d_1\}$, $D_2^3 = \{d_2\}$, $D_3^3 = \{d_3\}$,
$D_4^3 = \{d_4\}$, $D_5^3 = \{d'_5\}$, $D_6^3 = \{d'_6\}$, $D_7^3 = \{d'_7\}$.

3.3 検索アルゴリズム

本節では，階層的ブルームフィルタによる暗号化索引を用いた検索アルゴリズムを与える．検索アルゴリズム $Search2(E_s(w))$ を図 4 に示す．アルゴリズムの中で使われている $ActiveID$ はチェックすべき ID とブルームフィルタのレベルのペアを保存するためのスタックである．基本的にアルゴリズムは，深さ優先探索でブルームフィルタをチェックし，葉に相当するブルームフィルターでチェックが成功すればその ID がキーワードを含むドキュメントの ID となる．

ハッシュ関数の計算を計算するコストを C_h，暗号化関数を計算するコストを C_e とする．ブルームフィルタのチェックでは，1回暗号化関数 E_p を適用し，その後，k 個のハッシュ関数を計算する．したがって，ブルームフィルタ1回当たり $O(k \cdot C_h + C_e)$ 時間掛かる．今，キーワードにマッチするドキュメント数を t とすると，以下の定理を得る．

定理 1 任意のキーワード w に対し，検索アルゴリズム $Search(E_s(w))$ は，$O((k \cdot C_h + C_e) \cdot L)$ 時間で動作する．ここで，$L = \min\{2t \log n, 2n\}$.

Algorithm $Search2(E_s(w))$;
入力：$E_s(w)$ (検索キーワード w の暗号化)
出力：w を含むドキュメント ID

(1) $ActiveID = \{(0, 0)\}$, //$ActiveID$ はスタック
(2) $tempID = -1$; //すでに計算済の ID を記録
(3) while $ActiveID$ が空でない do
(4) $ActiveID$ のトップの値を $(id, level)$ にセットする;
(5) if $tempID == id$ then
(6) $b_1 = hash[1], \cdots, b_k = hash[k]$;
(7) else {
(8) $T(w) = E_p(id, E_s(w))$;
(9) $b_1 = h_1(T(w)), \cdots, b_k = h_k(T(w))$;
(10) $hash[1] = b_1, \cdots, hash[k] = b_k$;
(11) $tempID = id$;
(12) };
(13) if BF_{level} 内の $b_1, \ldots, b_k$ の位置に対応するすべてのビットが 1, then
(14) if $level == (\log N)$ then ID を出力;
(15) else $(id, level + 1)$ および $(id + 2^{\lceil \log n \rceil - level - 1}, level + 1)$ を $ActiveID$ に積む;
(16) while-end

図 4: 改良暗号化索引を使った検索アルゴリズム

もし t が定数ならば，$Search2(E_s(w))$ は，$O((k \cdot C_h + C_e) \cdot \log n)$ 時間で動作する．また，最悪の場合として，$t = n$（すなわち，キーワードがすべてのドキュメントに出現する）ならば，L は $2n$ となり，$O((k \cdot C_h + C_e) \cdot n)$ 時間で動作する．ブルームフィルタを階層的にしない場合，すべてのドキュメントに対しブルームフィルタをチェックする必要がある．そのため，ブルームフィルタのチェック回数は n となり，$O((k \cdot C_h + C_e)n)$ 時間掛かる．これより，我々のアルゴリズムは，キーワードにマッチするドキュメントが少ないほど高速に動作する．特

に，t が定数の場合は，非階層に比べ指数的な高速化を実現する．また，最悪の場合において，ブルームフィルタをチェックする回数は非階層の高々 2 倍になるが，ハッシュ関数の計算および暗号化の計算が省略できるため，実際には，ほとんど同程度の時間で計算できる．

3.4 安全性について

検索システムの安全性について考察する．今回は 2 段階暗号化を行っており，共通鍵暗号方式を 2 回適用する．そのため，2 つの鍵 s と p を用いている．鍵 s は秘密鍵でクライアントしか知らない．サーバ上のすべてのデータは s によって暗号化されており，サーバは内容を知ることができない．

鍵 p は，s で暗号化されたキーワードをブルームフィルタへ登録するとき，および検索時に用いられる鍵で，これはサーバに公開している．秘密鍵 s で暗号化されたキーワード $E_s(w)$ と ID のペアを p で暗号化し，ブルームフィルタに登録することにより，そのキーワードの出現頻度を隠すことができる．すなわち，ドキュメントより ID を変えているため，ID が違うと違った 1 のパターンが登録されることになる．そのため，各キーワードに対し，どの階層のブルームフィルタにおいても，鍵 s が分からない限り，ドキュメント間に渡ってのそのキーワードの出現頻度を直接推測することはできない．サーバ上での検索はすべて暗号化された世界で行われ，結果も暗号化データが返されるため，これも秘密鍵 s を知らない限り，内容を知ることができない．

4 実装と実験結果

検索時間を調べるため，検索部を実際に実装し，実験的に性能を評価した．結果について述べる．

4.1 実装

Java を使って実験的に検索アルゴリズムを実装した．ブルームフィルタは 32768 ビット分のフィルタを用いて暗号化索引を作成した．非階層については，1 個のブルームフィルタでよいが，階層型の場合は，階層ごとにブルームフィルタを用意する必要がある．そのため，索引のサイズは，階層のほうが大きくなっている．

ハッシュ関数については 5 個のハッシュ関数を用いた．ハッシュ関数は，ユニバーサルハッシュ関数 [4] として知られているものの一つを利用した．今回は，文献 [4] で提案されている排他的論理和を使った関数を用いた．暗号化は Java の暗号化拡張機能を利用し，暗号アルゴリズムとして AES を用いた．

今回は階層化されたブルームフィルタ（階層型）と階層化されていないブルームフィルタ（非階層型）の検索時間を比較した．非階層型は，1 つのブルームフィルタにすべてのドキュメントを登録するため，基本的に，階層型の葉に相当するブルームフィルタを利用した検索方式となる．本論文では，1024 個のドキュメント群に対し，ドキュメント当り 6 個のキーワードを抽出しブルームフィルタに登録した．

4.2 実験結果

実験は，6 個のキーワードに対して行った．実験を行ったマシンは，Core i7 870@2.93GHz, メモリ 12GB，Windows 7(64 ビット) である．Java のバージョンは JavaSE-1.6 である．

表 1 は，階層型にとってドキュメントの配置を最悪にした場合とドキュメントをランダムに配置した場合のキーワードセットに対する検索時間（ミリ秒）を示している．5 回行った平均をとっている.「一致するドキュメント数 t」は，6 個のキーワードは，それぞれ皆同じ t 個のドキュメントに一致するようにドキュメントを構成してある．均等配置（最悪のケース）は，一致するドキュメントが等間隔に出現するようにしてある．たとえば，$t = 2$ の場合，ドキュメント 0 番と 512 番，1 番と 513 番が同じキーワードをもつように配置した．同様に，$t = 4$ の場合は，0 番, 256 番, 512 番, 768 番のドキュメントが同じキーワードをもつ．非階層はどのような配置でもチェックするドキュメントの数は同じだが，階層型の場合，チェックするドキュメント数は，等間隔で出現する場合が一番多くなる．ランダム配置は，最悪のケースを軽減するため，乱数を用いて各ドキュメントに ID を割り振り，索引を構成した．表 1 において,「改良型階層」が本論文で提案した構成法であり,「階層」が山下・山本らの手法である．キーワードに一致するドキュメント数が少ない場合は，非階層と比べ，どちらも早いが，一致するドキュメント数が多くなるにつれて，改良型は時間の増加が抑えられている．512 の場合でも，非階層とほとんど同じ時間で検索

表 1: 検索時間（msec) の比較

一致するドキュメント数 t	均等配置（最悪ケース)			ランダム配置	
	非階層	階層	改良型階層	階層	改良型階層
0	75	10.2	10	10.2	10
2	79.8	14.2	9.6	16	10
4	74	18.6	14.2	18.2	15
8	77.8	24.2	19.4	24.4	18.2
16	83	34.8	25.8	33	25.4
32	74.6	41	34	38	33.4
64	76.2	53.6	39	47.8	38.4
128	79.4	65.8	51.8	58	53.4
256	74.6	81	65.8	77.4	60.6
512	79	99.6	77.8	79.8	75.6

表 2: 階層型に対し最適な配置の場合（一致ドキュメント数 512）

非階層	階層	改良型階層
75	74.4	56.2

できる．ランダム配置にすると少し改善される．このことから，ランダムに ID を割り振ることは有効であると考えられる．

表 2 は，階層型にとってドキュメントの配置を最適にした場合の検索時間である．階層型は，同じキーワードを含むドキュメントが近くにあるほど効率がよくなる．したがって，最適な配置は，0 番から 511 番までが一致し，512 番から 1023 番までに一致しないドキュメントを配置して，性能をチェックした．表 1 と比べると，改良型はかなり改善されていることがわかる．このことから，同じようなキーワードを含むドキュメントには近い値の ID を割り振って索引を構成していくことが有効と思われる．

5 まとめ

階層型において，同じようなキーワードを含むドキュメントには近い値の ID を割り振って索引を構成していくことが有効と思われるが，これを効率的に行う手法については今後の課題である．

参考文献

[1] 新井裕子, 渡辺智恵美，*DAS* **モデルにおけるプライバシー保護に考慮した範囲検索法**, 日本データベース学会論文誌, 7,1(2008), 7–12.

[2] B.H. Bloom, *Space/Time Trade-offs in Hash Coding with Allowable Errors*, Comm. of the ACM, 13(1970), 422–426.

[3] A. Broder and M. Mitzenmacher, *Network Applications of Bloom Filters: A Survey*, Internet Mathmatics, 1,4(2004), 485–509.

[4] J.L. Carter and M.N. Wegman, *Universal classes of hash functions*, JCSS, 18(1979), 143–154.

[5] E-J. Goh, *Secure Indexes*, Stanford Univ. Technical Report, In IACR ePrint Cryptography Archive, (2003), See http://eprint.iacr.org/ 2003/216.

[6] P. Golle, J. Staddon and B. Waters, *Conjuctive Keyword Searcg over Encrypted Data*, Proc. of ACNS 2004, LNCS 3089(2004), 31–45.

[7] H. Hacüigumüs, B. Hore, B. Iyer, and S. Mehrotra, *Search on Encrypted Data*, Advances in Information Security, 33(2007), 383-425.

[8] D.X. Song, D. Wagner and A. Perrig, *Techniques for Searchers on Encrypted Data*, IEEE Symposium on Security and Privacy, (2000), 44–55.

[9] 山下智穂, 山本博章, **階層化されたブルームフィルタを用いた安全で効率的な検索システム**, SITA2010, (2010) .