Computer Security Symposium 2014
22 - 24 October 2014

階層型ブルームフィルタを用いた暗号化検索法の改良

渡邊 尊司 †　　山本 博章 ‡

† 信州大学大学院理工学研究科
380-8553 長野市若里 4-17-1

‡ 信州大学工学部
380-8553 長野市若里 4-17-1
yamamoto@cs.shinshu-u.ac.jp

あらまし 情報セキュリティの観点から，暗号化したデータを暗号化したまま効率的に検索する暗号化検索法の開発が進められている．これに対し，我々は階層型ブルームフィルタを用いた手法を提案した．本論文では，我々が提案した手法を改良し，大規模データを用いてその性能を評価する．

Improvement of an Encrypted Search Method using Hierarchical Bloom Filters

Takashi Watanabe†　　Hiroaki Yamamoto‡

†Shinshu University.
4-17-1, Wakasato, Nagano-shi, Nagano 380-8553, JAPAN
‡Shinshu University
4-17-1, Wakasato, Nagano-shi, Nagano 380-8553, JAPAN
yamamoto@cs.shinshu-u.ac.jp

Abstract From a view point of information security, researches on an encrypted search system have been done intensively. For this problem, we developed a method which makes use of hierarchical Bloom filters. In this paper, we improve our method and evaluate its performance using a large dataset.

1 はじめに

クラウドコンピューティングが普及する中，ストレージサービス，メールサービスなど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，個人情報の保護及び機密情報の保護は非常に重要な課題である．情報検索においても，安全な検索法として，データを暗号化したまま検索する手法の開発が進められている．一般に，データベースやメールシステムなどにおいて，サーバ管理者とデータの所有者とが異なる場合が多い．サーバ管理者にも内容を知られずに検索するには，データを暗号化して保存し，暗号化したまま検索する技術が要求される．そのため，暗号化データに対する効率的な検索手法に関する研究が活発に行なわれてきた [3, 4, 6, 7, 8, 9, 10, 11, 12, 14, 13].

本論文は，ドキュメントに対するキーワード検索を暗号化された世界で効率的に行う手法について議論する．すなわち，暗号化されたキーワードと暗号化されたドキュメントの集合が与えられたとき，そのキーワードを含むドキュメントを暗号化したまま効率的に検索するための新たな暗号化索引構造と検索アルゴリズムを与える．暗号化検索法として，共通鍵暗号方式または公開鍵暗号方式を用いた手法が研究されて

いるが，ここでは共通鍵暗号方式を用いた手法を提案する．提案する暗号化索引構造は，ブルームフィルタ（Bloom filter）を用いて構成する．

ブルームフィルタを用いた手法として，Goh [7] は，各ドキュメントに対し検索用のブルームフィルタを用意し，そのドキュメントに含まれるキーワードを暗号化して暗号化索引を構成する手法を提案した．山本他 [16] は，ドキュメントを 2 分木構造で管理し，2 分木の各階層にブルームフィルタを割り当てることにより，効率的な検索を可能にする安全な索引構造を提案した．彼らは，理論的に検索時間を解析するとともに実験的に性能を評価した．

本論文では，山本らの手法に対し，以下の改善点を導入した手法を提案する．

- 山本らの手法は，キーワードのマッチする文書数が少ないほど高速に動作する．そこで，複数キーワードの AND 検索が可能になるよう拡張する．
- 山本らの手法は，高速化のためブルームフィルタを階層化した．しかし，階層数分のブルームフィルタを用意するため，暗号化索引のサイズが，階層化しない場合より増える．そこで，文書の木構造管理に k 分木を導入し，サイズの増加を抑える．分岐数 k が大きくなれば階層数は少なくなるが，検索時間は増える可能性がある．Goh の構成は，k が全体の文書数になる場合である．

本論文は，2 章で検索システムの概要，3 章でブルームフィルタの概要，4 章で提案法，5 章で実験結果について述べる．

2　検索問題とシステムの概要

本論文では，次のような検索問題を考える．今，$D = \{d_0, \ldots, d_{n-1}\}$ をドキュメントの集合とする．各ドキュメント d_i $(0 \leq i \leq n-1)$ はテキストで，識別番号 i が割り振られている（i をドキュメント ID と呼ぶ）．各ドキュメント d_i に対し，$\mathcal{K}(d_i)$ を，d_i から抜き出したキーワードの集合とする．キーワードとは語（すなわち，文字列）である．そのとき，与えられたキーワードの集合 $Q = \{w_1, \cdots, w_q\}$（クエリと呼ぶ）に対し，Q のすべてのキーワード w を持つドキュメント d_i をすべて見つける問題を考える．このようなドキュメント d_i を Q に一致するドキュメントと言う．ここでは，これを暗号化したまま行う方法について提案する．

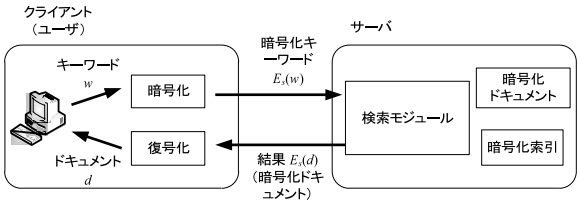

図 1: 安全な検索システムの概要

検索システムの概要を図 1 に示す．システムは，クライアント（ユーザ）とサーバからなり，クライアントがドキュメントの所有者で，サーバはクライアントからの暗号化キーワードに対する検索を行う．本システムでは，データの暗号化のために，共通鍵暗号方式を用いる．このとき，E_s によって，鍵 s を用いた暗号化関数を表す．さらに我々は，ハッシュ関数として擬似ランダム関数を使う．擬似ランダム関数 $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^m$ に対し，$F(K,x)$ を $F_K(x)$ と書く．以下で検索手順の概要を示す．

(1) クライアントは，自分の秘密鍵 s を用いてドキュメントを暗号化する．さらに，ドキュメントから抽出したすべてのキーワード w から鍵 s を用いて暗号化索引を作成する．ここで，p として $E_s(w)$ を使う．それから，暗号化ドキュメントと暗号化索引をサーバに保存する．

(2) クエリ Q の検索では，クライアントは，秘密鍵 s を用いて暗号化クエリ $E_s(Q) = \{E_s(w_1), \cdots, E_s(w_q)\}$ を作成し，それをサーバに渡す．

(3) サーバは，クライアントから暗号化クエリ $E_s(Q)$ を受け取ると，暗号化クエリと暗号化索引を使って，Q に一致するドキュメントを探す．一致するドキュメントの暗号化ドキュメント $E_s(d)$ をクライアントに返す．

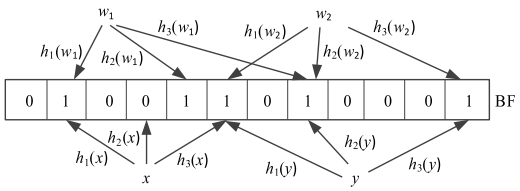

図 2: ブルームフィルタの例

(4) クライアントは $E_s(d)$ を秘密鍵 s で復号化することによって，目的のドキュメント d を得る．

3 ブルームフィルタ

ブルームフィルタは，Bloom [1] 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．サイズ m ビットのブルームフィルタ BF の概要について説明する．今，$W = \{w_1, \ldots, w_l\}$ を l 個の語からなる集合，$h_1, \ldots, h_k$ を語から $[0, m-1]$ の整数へのハッシュ関数とする．そのとき，各 $w_i \in W$ に対し，BF 内で，$h_1(w_i)$, ..., $h_k(w_i)$ の位置にあるビットを 1 にセットする．与えられた語 w が W に入っているかどうかは，$h_1(w)$, ..., $h_k(w)$ を計算し，BF で対応する位置のビットがすべて 1 ならば $w \in W$，そうでなければ $w \notin W$ と判定する．欠点としては，W にない語 $v \notin W$ に対し，$h_1(v)$, ..., $h_k(v)$ 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，$v \in W$ という間違った答えを得てしまう．これは偽陽性 (false positive) と呼ばれる．なお，W 内の語に対しては必ず正しい答えを返す．例を図 2 に示す．$W = \{w_1, w_2\}$，h_1, h_2, h_3 をハッシュ関数とする．そのとき，W に対する BF は，w_1 に対しては，$h_1(w_1), h_2(w_1), h_3(w_1)$ の位置のビットが 1 にセットされ，w_2 に対しては，$h_1(w_2), h_2(w_2), h_3(w_2)$ の位置のビットが 1 にセットされる．この BF に対し，語 x を与えると，$h_2(x)$ の位置のビットが 0 より，$x \notin W$ と判定される．また，語 y に対しては，3 箇所すべての位置のビットが 1 より，$y \in W$ と判定される．ブルームフィルタの概要と応用については，Broder と Mitzenmacher [2] の中でもよくまとめられている．

4 階層型ブルームフィルタを用いた暗号化検索

4.1 階層型暗号化索引の構成

本節では，ブルームフィルタを階層的に用いた新たな暗号化索引の構成法について述べる．これを，階層型暗号化索引と呼ぶ．前と同様に，$D = \{d_0, \ldots, d_{n-1}\}$ をドキュメントの集合とする．また，$\lceil x \rceil$ は x 以上の最小の整数を表し，$\lfloor x \rfloor$ は x 以下の最大の整数を表す．我々は，ドキュメントの集合を管理するため，k (≥ 2) 分木を導入する．ここで，$h = \lceil \log_k n \rceil$ と定義する．本手法では，もし $n < k^h$ ならば，仮想的に空ドキュメントを追加し，完全 k 分木を考える．さて，$0 \leq lev \leq h$ に対し，階層 lev における $\mathcal{D}$ の k-分割 $\mathcal{D}^{lev} = \{D_0^{lev}, D_{k^\alpha}^{lev}, D_{2\cdot k^\alpha}^{lev}, \ldots, D_{(k^{lev}-1)\cdot k^\alpha}^{lev}\}$ を以下のように定義する．ここで，$\alpha = h - lev$ とする．

定義 1 $i = 0, k^\alpha, 2\cdot k^\alpha, 3\cdot k^\alpha, \ldots, (k^{lev}-1)\cdot k^\alpha$ に対し，$D_i^{lev} = \{d_i, d_{i+1}, \ldots, d_{i+k^\alpha-1}\}$ とする．ここで，もしドキュメント d_j の j が $n-1$ より大きければ，そのドキュメントは空ドキュメントとなる．

階層 lev において，各ノード D_i^{lev} の ID i は，0 から始め，k^α の間隔で順に割り振られていることに注意する．階層 lev の分割において，各 D_i^{lev} は，k^α 個の要素からなる $\mathcal{D}$ の部分集合であり，番号 i は D_i^{lev} の識別番号（ID）と呼ばれる．また，$\mathcal{K}(D_i^{lev}) = \bigcup_{d \in D_i^{lev}} \mathcal{K}(d)$ と定義する．

分割において，D_i^{lev} $(0 \leq lev \leq h)$ に対し，$D_i^{lev+1}, \cdots, D_{i+(k-1)k^{h-(lev+1)}}^{lev+1}$ を D_i^{lev} の k 個の子供とみなせば，拡張ドキュメント集合の分割は，D_i^{lev} をノードとし，D_0^0 を根とする k 分木を構成する．この木をドキュメント木と呼ぶことにする．ドキュメント木において，葉 D_i^h は 1 個のドキュメント d_i だけからなる集合 $\{d_i\}$ となる．任意のノード D_i^{lev} に対し，D_i^{lev} が空ドキュメントだけからなるとき，そのノードを空ノー

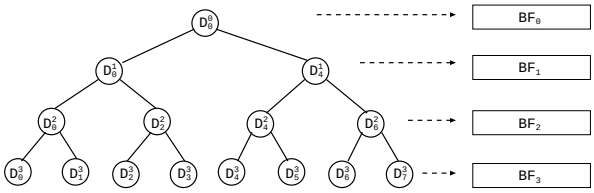

図 3: ドキュメント木とブルームフィルタ

ドと言う．木の用語を使うならば，各ノードの階層 lev はそのノードの深さに対応し，木の高さが h となる．階層型は，各階層 $0 \leq lev \leq h$ に対し，1 個のブルームフィルタ BF_{lev} を用いる．したがって，全体で $h+1$ 個のブルームフィルタを用いる．

ドキュメント木において，各ノードの ID は，その最も左の子の ID と同じになることに注意する．これによって，検索時に，暗号化とハッシュ関数の計算を減らすことができる．

例 1 ドキュメント木の例を与える．ここで，$k = 2$ とする．5個のドキュメントの集合 $\mathcal{D} = \{d_0, d_1, d_2, d_3, d_4\}$ を考える．3つの空ドキュメント d'_5, d'_6, d'_7 を追加し，ドキュメント数を $8 = 2^3$ とし，拡張ドキュメント集合 $\mathcal{D} = \{d_0, d_1, d_2, d_3, d_4, d'_5, d'_6, d'_7\}$ を構成する．そのとき，$\mathcal{D}$ に対するドキュメント木は図 3 となる．図 3 で，BF_0, BF_1, BF_2, BF_3 は各階層に対するブルームフィルタを表し，各ノードは以下のような拡張ドキュメント集合の部分集合になる．ここで，D_6^2, D_5^3, D_6^3, D_7^3 が空ノードである．
$D_0^0 = \{d_0, d_1, d_2, d_3, d_4, d'_5, d'_6, d'_7\}$,
$D_0^1 = \{d_0, d_1, d_2, d_3\}$, $D_4^1 = \{d_4, d'_5, d'_6, d'_7\}$,
$D_0^2 = \{d_0, d_1\}$, $D_2^2 = \{d_2, d_3\}$, $D_4^2 = \{d_4, d'_5\}$, $D_6^2 = \{d'_6, d'_7\}$,
$D_0^3 = \{d_0\}$, $D_1^3 = \{d_1\}$, $D_2^3 = \{d_2\}$, $D_3^3 = \{d_3\}$, $D_4^3 = \{d_4\}$, $D_5^3 = \{d'_5\}$, $D_6^3 = \{d'_6\}$, $D_7^3 = \{d'_7\}$.

さて，暗号化索引作成アルゴリズム *MakeIndex* を図 4 に示す．2 段階暗号化によって暗号化索引を構成している．なお，パラメータ ϵ は偽陽性のためのもので，ブルームフィルタのサイズを決定する．これが大きいほどサイズが大きくなり，偽陽性率を下げる．5 章の実験では，$\epsilon = 10$ 及び $\epsilon = 30$ としている．

Algorithm *MakeIndex*$(s, \mathcal{D}, \epsilon)$;
(1) for $lev = 0$ to h do
(2) $m_1 = \sum_{D_i{}^{lev} \in D^{lev}} |\mathcal{K}(D_i^{lev})|$,
(3) サイズ ϵm_1 ビットのブルームフィルタ BF_{lev} を初期化する．
(4) for all $D_i^{lev} \in \mathcal{D}^{lev} = \{D_0^{lev}, D_{k^\alpha}^{lev}, D_{2\cdot k^\alpha}^{lev}, \ldots, D_{(k^{lev}-1)\times k^\alpha}^{lev}\}$ do
(5) for all $w \in \mathcal{K}(D_i^{lev})$ do
(6) $X = E_s(w)$, $p = E_s(w)$,
(7) $T(w) = F_p((X||i))$ を計算する;
(8) $h_1(T(w)), \cdots, h_k(T(w))$ を計算し，BF_{lev} のビットをセットする;
(9) for-end
(10) for-end
(11) for-end

図 4: 階層型暗号化索引作成アルゴリズム

4.2 階層型暗号化索引を用いた検索アルゴリズム

前節で示した階層型暗号化索引を用いた検索アルゴリズム *Search* を図 5 に与える．ここでは，階層型暗号化索引を用いた検索法を階層型と言う．アルゴリズム *Search* の中で使われている *ActiveID* は，チェックすべきノードの ID と階層のペアを保存するために使われるスタックである．*Search* はサーバ上で実行され，クエリ Q に対する暗号化 $E_s(Q)$ をクライアントから受け取ると，ドキュメント木上を深さ優先探索で，各ノードに対応するブルームフィルタを順にチェックしていく．すなわち，各ノード D_i^{lev} において，そのノードに含まれるドキュメントに Q に一致するものがあるかどうかを $E_s(Q)$ と BF_{lev} を使ってチェックする．もしなければ，その子供以下のノードをチェックする必要がないので，チェックの対象から外す．もしあれば，すべての子供をチェックするために，子供の ID をスタックに入れる．このとき，空ノードかどうかチェックし，空ノードでないときのみスタックに入れる．最も左の子については，親が空でなければ，空ノードにはならないので常にスタッ

クにそのIDを入れる．もし葉に対応するブルームフィルタで一致と判定されれば，その葉のIDがキーワードに一致するドキュメントのIDとなるので，*Search* はそのIDを出力する．サーバは，そのIDを持つ暗号化ドキュメントをクライアントに返せばよい．このように，検索は暗号化された世界で行われる．なお，*Search* は，右の子から順にスタックにIDを格納するため，親と同じIDを持つ左の子から検索するようになっていることに注意する．これによって，暗号化およびハッシュ関数の計算を減らすことができる．

4.3 検索時間の解析

さて，検索アルゴリズム $Search(E_s(Q))$ の計算時間（検索時間）を評価する．検索時間に関しては，非階層型及び階層型は，どちらもその大部分を暗号化とハッシュ関数の計算で占められているため，これらの計算時間を評価することにする．なお，一般に，ブルームフィルタでは偽陽性が発生する可能性があるが，以下の計算時間の解析では，簡単のために，誤り率を0として計算時間を評価する．次の6章において，実際に暗号化索引と検索アルゴリズムを実装して実験的に評価し，理論的結果との比較を行う．

今，クエリ w が t 個のドキュメントに一致するとする．そのとき，ドキュメント木上で t 本のパスを根から葉へたどることになる．アルゴリズム $Search(E_s(Q))$ は，パス上にあるノードに対し，対応するブルームフィルタをチェックしていくから，ブルームフィルタがチェックされるノード（単に，チェックされるノードとも言う）がどれくらいの数になるのかを調べることにする．このとき，t 本のパスで共通するノードについては1回のチェックでよいことに注意すれば，チェックされるノード数が最大になる（すなわち，計算時間が最悪になる）のは，t 本のパスができるだけ共通のノードを持たないように，t 個のドキュメントがドキュメント木の葉に配置されている場合である．これは，ちょうど，t 個のドキュメントができるだけ等間隔で葉に出現する場合となる．すなわち，以下の補題が成り立つ．

表 1: 暗号化索引のサイズ

分岐数	サイズ 10 倍 ($\epsilon = 10$)	サイズ 30 倍 ($\epsilon = 30$)
2	296MB	888MB
4	159MB	476MB
8	113MB	340MB

補題 1 ノード D を，ドキュメント木の葉以外の任意のノードとし，その k 個の子供を $D_1, \cdots, D_k$ とする．また，D には検索キーワードに一致するドキュメントが t' 個含まれているとする．そのとき，一致するドキュメントを $D_j (1 \leq j \leq k)$ に $\lceil t'/k \rceil$ 個または $\lfloor t'/k \rfloor$ 個配置すれば，チェックされるノード数が最大となる．

最悪の場合を考慮すると，検索時間に関しては以下の定理が成り立つ．

定理 1 任意のクエリ Q に対し，検索アルゴリズム $Search(E_s(Q))$ は，$O(\sum_{i=0}^{h} \gamma_i)$ 時間で走る．ここで，$\gamma_i = \min\{t, \text{レベル } i \text{ のノード数}\}$, t は Q に一致するドキュメント数，$h = \lceil \log_k n \rceil$.

定理1から分かるように，t が小さいほど検索時間は速い．最良でドキュメント木の高さ h，最悪でドキュメント木のノード数となる．

4.4 安全性について

提案手法の安全性は，Goh [7] の手法と同じである．すなわち，ドキュメントの所有者でないサーバ管理者を含めた第3者に対し，暗号化キーワードとその検索結果以外の情報を漏らさない．このような安全性の設定は，他の暗号化キーワード検索の手法 [4, 7, 8, 11, 14] でも用いられている．

5 実験

我々は提案法を実装し，実験的に評価した．評価用のデータとして，Enron Emai データセット [5] を用いた．Enron Email データセットで公開されている517431個のメールデータを使い，これらのメールデータから取り出した40115527

Algorithm $Search(E_{sk}(Q))$;
入力：$E_s(Q)$ (検索キーワード w の暗号化)
出力：Q に一致するドキュメント ID

(1) $(0,0)$ をスタック $ActiveID$ に入れる
(2) $tempID = -1$; // $tempID$ は ID を保持
(4) while $ActiveID$ が空でない do
(5) スタック $ActiveID$ からトップの値を取り出し，(id, lev) にセットする
(6) if $tempID = id$ then {
(7) $match$:= `true`
(8) for $i = 1$ to q do
(9) $b_1 = hash[i,1]$,$\cdots$,$b_k = hash[i,k]$;
(10) if BF_{lev} の $b_1, \ldots, b_k$ の位置に 0 ビットが存在する
then $match$:= `false`;
(11) for-end
(12) }
(13) else {
(14) $match$:= `true`
(15) $tempID = id$;
(16) for $i = 1$ to q do
(17) $p = E_{sk}(w_i)$, $T(w_i) = F_p(E_s(w_i)||id)$;
(18) $b_1 = h_1(T(w_i))$,$\cdots$,$b_k = h_k(T(w_i))$;
(19) $hash[i,1] = b_1$, $\cdots$, $hash[i,k] = b_k$;
(20) if BF_{lev} の $b_1, \ldots, b_k$ の位置に 0 ビットが存在する
then $match$:= `false`
(21) for-end
(22) };
(23) if $match$ = `true`, then {
(24) if $lev = h$ (すなわち, BF_{lev} は葉に対応する),
(25) then id を出力;
(26) else {
(27) $\{(id + (k-1)k^{h-lev-1}, lev+1)$, $\{(id + (k-2)k^{h-lev-1}, lev+1)$, $\cdots$, $(id, lev+1)$ の k 個の ID をスタック $ActiveID$;$\}$ に格納
(29) }
(30) }
(31) while-end

図 5: 検索アルゴリズム *Search*

個のキーワードで暗号化索引を構成した．提案法の実装は Linux 上で C++を用いて行った．共通鍵暗号方式としては鍵長 128 ビットの AES を用い，ハッシュ関数は HMAC-SHA512 を用いた．実験は，登録キーワード数の 10 倍及び 30 倍のサイズのブルームフィルタを用いて，2 分木で構成した暗号化索引，4 分木で構成した暗号化索引，8 分木で構成した 6 種類の暗号化索引を作成した．暗号化索引のサイズについては表 1 に示す．したがって，計 6 種類の暗号化索引を作成している．各暗号化索引に対し，2 キーワードからなるクエリと 4 キーワードからなるクエリの 2 種類のクエリの偽陽性率と検索時間を評価した．各クエリについては，キーワード

表 2: 平均偽陽性率及び平均検索時間 (サイズ 10 倍，2 キーワード)

マッチ数	2 分木		4 分木		8 分木	
	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）
1	0	0.005	0	0.005	0	0.005
2-10	1.01×10^{-6}	0.03	1.57×10^{-6}	0.03	2.45×10^{-6}	0.05
11-100	8.97×10^{-6}	0.12	2.46×10^{-5}	0.13	2.08×10^{-5}	0.24
101-1000	9.30×10^{-5}	0.54	1.47×10^{-4}	0.6	2.2×10^{-4}	0.97
1001-10000	9.17×10^{-4}	1.87	1.36×10^{-3}	1.99	1.99×10^{-3}	2.95
10001-100000	6.49×10^{-3}	5.73	1.14×10^{-2}	3.65	1.68×10^{-2}	7.44
100001 以上	4.08×10^{-2}	14.66	6.5×10^{-2}	12.32	1.2×10^{-1}	13.79

表 3: 平均偽陽性率及び平均検索時間 (サイズ 10 倍，4 キーワード)

マッチ数	2 分木		4 分木		8 分木	
	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）
1	0	0.17	0	0.21	0	0.32
2-10	1.76×10^{-6}	0.25	2.41×10^{-6}	0.34	3.98×10^{-6}	0.68
11-100	1.89×10^{-5}	0.66	2.66×10^{-5}	0.97	3.81×10^{-5}	2.01
101-1000	1.77×10^{-4}	1.67	2.57×10^{-4}	2.21	3.43×10^{-4}	4.24
1001-10000	1.48×10^{-3}	3.65	2.15×10^{-3}	4.53	2.94×10^{-3}	8.60
10001-100000	1.48×10^{-2}	3.65	1.85×10^{-2}	8.44	2.09×10^{-2}	12.45
100001 以上	–	–	–	–	–	–

をランダムに選び，それぞれ 1000 個のクエリを作成し，評価した．表 2，表 3，表 4，表 5 に実験結果を示す．マッチ数は，1000 個のクエリがマッチしたドキュメント数を表している．偽陽性率及び検索時間はマッチ数で示した範囲に入るクエリの平均値である．表 2 は，サイズ 10 倍，2 個のキーワードからなるクエリの実験結果である．暗号化索引を 2 分木で構成した場合，4 分木で構成した場合，8 分木で構成した場合について示している．表 1 からわかるように，一般に分岐数を増やすと階層数が減るため暗号化索引のサイズを減らすことができる．しかし，各ノードの子供の数が増えるため，時間が増加する．実験結果からもこの傾向が見られる．しかし，検索時間にそれほどの増加がなく，また，暗号化索引のサイズの削減効果が大きいことから k-分木の導入は有効と思われる．

参考文献

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

[2] A. Broder and M. Mitzenmacher, Network Applications of Bloom Filters: A Survey, Internet Mathematics, 1, 4, pp.485–509, 2004.

[3] N. Cao, C. Wang, M. Li, K. Ren and W. Lou, Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data, Proc. of INFOCOM 2011, pp.829–837, 2011.

[4] Y.-C. Chang and M. Mitzenmacher, Privacy Preserving Keyword Searches on Remote Encrypted Data, Proc. of ACNS 2005, LNCS 3531, pp.442–455, 2005.

[5] W. W. Cohen, Enron Email Dataset, http://www.cs.cmu.edu/ enron/.

[6] 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.

[7] E.-J. Goh, Secure Indexes, Stanford

表 4: 平均偽陽性率及び平均検索時間 (サイズ 30 倍，2 キーワード)

マッチ数	2 分木		4 分木		8 分木	
	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）
1	0	0.028	0	0.028	0	0.042
2-10	0	0.05	0	0.05	0	0.09
11-100	0	0.23	0	0.26	0	0.42
101-1000	0	0.97	5.77×10^{-9}	1.08	0	1.62
1001-10000	6.45×10^{-8}	3.29	9.96×10^{-8}	3.42	9.38×10^{-8}	4.67
10001-100000	1.9×10^{-7}	10.18	5.69×10^{-7}	9.96	6.95×10^{-7}	11.79
100001 以上	0	25.06	0	20.21	0	19.76

表 5: 平均偽陽性率及び平均検索時間 (サイズ 30 倍，4 キーワード)

マッチ数	2 分木		4 分木		8 分木	
	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）	偽陽性率	検索時間（秒）
1	0	0.25	0	0.31	0	0.53
2-10	0	0.45	0	0.56	0	0.95
11-100	0	1.11	0	1.39	0	2.35
101-1000	0	2.65	0	3.10	1.52×10^{-7}	4.78
1001-10000	6.78×10^{-8}	5.73	0	6.19	1.25×10^{-7}	8.42
10001-100000	0	13.6	0	11.8	0	13.55
100001 以上	–	–	–	–	–	–

Univ. Technical Report, In IACR ePrint Cryptography Archive, 2003, See http://eprint.iacr.org/ 2003/216.

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

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

[10] L. Liu and J. Gai, Bloom Filter Based Index for Query over Encrypted Character Strings in Database, Proc. of CSIE 2009, pp.303–307, 2009.

[11] Q. Liu, G. Wang and J. Wu, An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing, Proc. of CSE 2009, pp.715–720, 2009.

[12] R.A. Popa, C.M.S. Redfield, N. Zeldovich and H. Balakrishnan, CryptDB: Processing Queries on an Encrypted Database, Commun. ACM, 55, 9, pp.103–111, 2012.

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

[14] T. Suga, T. Nishida and K. Sakurai, Secure Keyword Search Using Bloom Filter with Specified Character Positions, ProvSec 2012, LNCS 7496, pp.235–252, 2012.

[15] C. Wang, N. Cao, J. Li, K. Ren and W. Lou, Secure Ranked Keyword Search over Encrypted Cloud Data, Proc. of ICDCS 2010, pp.253–262, 2010.

[16] 山本博章, 山下智穂, 大井篤, 中村伸一, 白井啓一郎, 宮崎敬, 階層化的ブルームフィルタを用いた安全で効率的なキーワード検索法, 電子情報通信学会論文誌, Vol.J96-D, No.12, pp.3030–3043, 2013.