情報処理学会論文誌 Vol.56 No.10 1977–1987 (Oct. 2015)

クラウド上の安全で高速なキーワード検索アルゴリズムの提案

清 雄一[1,a)] 竹之内 隆夫[2] 大須賀 昭彦[1]

受付日 2015年1月3日, 採録日 2015年7月1日

概要：データを外部のストレージ事業者に預けることが多くなっているが，プライバシや機密情報管理の観点から問題が生じる場合がある．データおよびその索引を暗号化する手法が有効であるが，検索等，データ処理の効率性を低下させることは避けたい．このような課題に対し，Bloom Filter というデータ構造を用いる情報管理エージェントが提案されている．しかし，安全性を担保するためには，検索速度が悪化するという問題がある．これは検索時にクラウド上のデータ数に比例した回数だけハッシュ値を計算する必要が生じるためである．提案手法では，Bloom Filter を利用し，ハッシュ値の計算のみではなく，素数による MOD 演算を併用することで，これまでと同レベルの安全性を保持したうえで検索速度を向上させる．793 万ドキュメントを利用したシミュレーション評価により，従来約 30.2 秒必要だった検索が 0.7 秒程度でできることを示す．

キーワード：プライバシ，ドキュメント検索，暗号化索引，Bloom Filter，クラウドコンピューティング

An Efficient Algorithm for Encrypted Text Searching in Cloud Computing

Yuichi Sei[1,a)] Takao Takenouchi[2] Akihiko Ohsuga[1]

Received: January 3, 2015, Accepted: July 1, 2015

Abstract: Although cloud storage services are becoming popular these days, the cloud service provider may violate users' privacy. We can use encryption techniques, but the performance of data processing such as searching should not be degraded. Existing studies use a data structure named Bloom Filter to deal with this challenge. However, it takes relatively a long time to search data by a keyword in their techniques. This is because they need to calculate hash values as many as the number of data. We propose a novel technique which uses not only hash values but also MOD operation by a prime number. Our goal is to increase the performance of searching data while maintaining a security level. By conducting experiments with 789 K real data, we show that our technique can search data within 0.7 seconds whereas existing studies need 30.2 seconds.

Keywords: privacy, text searcing, encrpted index, bloom filter, cloud computing

1. はじめに

近年，モバイル端末で取得したデータをクラウドへ保管したり，企業で日々生成される書類や日報をクラウド上へ保管したりすることが多くなってきた．これは，モバイル端末を紛失した場合における情報流出を抑制する効果もある．しかしデータを管理するクラウドが外部の第三者である場合，故意または不注意により，クラウド上の情報が漏洩する危険性も考えられる．今後，プライバシを考慮したうえで情報を管理する情報管理エージェントの存在が必要となってくると考えられる．ただし，クラウド上でのデータ検索等の処理にかかる時間が増加することは避けなければならない．本論文では，キーワードによるデータ検索にフォーカスし，データの安全性を担保したうえで，第三者が管理するクラウド上で，高速にキーワード検索を行うこ

[1] 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[2] 日本電気株式会社クラウドシステム研究所
Cloud System Research Laboratories, NEC Corporation, Kawasaki, Kanagawa 211–8666, Japan
[a)] sei@is.uec.ac.jp

© 2015 Information Processing Society of Japan

とを目標とする．

単純な仕組みとしては，データおよび検索用のキーワードをそれぞれを暗号化して，クラウドに登録することが考えられる．ユーザは暗号化したキーワードをクラウドに送り，クラウドはそれに合致するデータの集合を返す．しかしこの場合，クラウド側は，各キーワードの出現頻度を計算でき，また，各データ間で，共通するキーワードの数が多いものや少ないものを特定することができる．この結果，暗号化されたキーワードの中身をクラウド管理者が推測できてしまうリスクが高まってしまう．

近年，Bloom Filter（BF）というデータ構造を用いてこの問題に取り組む研究がさかんに行われている [4], [13], [14], [15]．しかし，ユーザ側に各データに紐づく検索用の全キーワードを管理させることを避ける場合，検索性能は十分なレベルに達していない．これは検索時にクラウド上のデータ数に比例した回数だけハッシュ値を計算する必要が生じるためである．本論文では，BF を利用し，ハッシュ値の計算のみでなく，素数による MOD 演算を併用することで，これまでと同レベルの安全性を保持したうえで検索速度を向上させる手法を提案する．

本論文の構成を示す．2 章でシステム概要や安全基準を述べ，関連研究を 3 章で記述する．4 章で提案手法を述べ，5 章で安全性について議論する．6 章において評価結果を記し，7 章で考察する．最後に 8 章でまとめる．

2. 背景

2.1 システム概要

ユーザは，自分が保有する複数のデータを第三者が管理するクラウドに登録する．各データにはキーワードが付与されており，クラウド上でキーワード検索を行うことができる．データは，たとえばドキュメントであり，全文検索を行う場合は，すべての単語がキーワードとなる．また，動画や音声の場合は，自然言語で表現されたメタデータがキーワードとなりうる．ユーザは，クラウドに保存している全データのキーワードを手元の端末で管理することはしない．また，ユーザは随時新しいデータを追加できるものとし，クラウドがあらかじめ全キーワードの集合やその統計情報を保有しているような状況は想定しない．

また，ユーザ側およびクラウド側にはそれぞれ情報管理エージェントが存在しており，本論文で提案するプロトコルに従って動作する．

ユーザエージェントは，ユーザのローカルマシン上に存在するエージェントであり，データの追加・修正・削除・検索に関する要求をユーザから受け取る．これらの要求を受け取ったユーザエージェントは，4 章で説明するように必要な情報を生成し，クラウドエージェントに生成した情報を提供する．また，検索時には，クラウドエージェントから検索結果の情報を受け取り，復号化等の処理をした後，ユーザに提供する．

クラウドエージェントは，クラウド上に存在するエージェントであり，データの追加・修正・削除・検索に関する要求をユーザエージェントから受け取る．これらの要求を受け取ったクラウドエージェントは，4 章で説明するように，クラウド上のデータを加工したり，検索結果をユーザエージェントに返したりする．

2.2 目標とする安全性

キーワードやデータの中身を知られないだけでなく，単語の出現頻度や，データ間の類似性についても，統計的な分析がなされることを防ぐことを目標とする．具体的には IND2-CKA [4] と呼ばれる安全性を目標とする．これは以下を要求する．多くの既存研究が，明示的または非明示的に，IND2-CKA の安全性に基づいて，クラウドコンピューティング上の暗号化されたデータ検索に取り組んでいる [3], [4], [14]．クラウド上に暗号化されたデータおよび検索用の索引を格納するモデルにおいて，IND2-CKA は以下のことを要求する [4]．

- 暗号化されたデータおよび索引からは，元のデータについて何の情報も得られない．ただし，データサイズを除く．
- 検索クエリはユーザのみが生成できる．
- 暗号化されたデータと検索クエリからは，その検索結果だけが明らかになる．

この条件を満たすものを IND2-CKA であると定義し，本研究ではこの実現を目指す．

2.3 Bloom Filter（BF）

本論文では，Bloom Filter（BF）[1] という，集合を表現するデータ構造を利用する．BF で表現された集合に対しては，ある要素が含まれているかどうかの判定を行うことができる．このとき，ある一定の確率で擬陽性を許容することで，BF の作成に必要なビット数を削減できる．この特徴を活かし，効率的な分散データ共有 [9], [10] に広く利用されている．また，この擬陽性を持つという特徴を積極的に利用して，プライバシ保護の目的 [6], [7] にも利用されている．

BF の作成手順は以下のとおりである．まず，すべての値が 0 に設定されている m ビットのベクトル B を用意する．ベクトル B の i 番目の要素を $B[i]$ と表現する．また，k 個の独立したハッシュ関数 $h_1, \ldots, h_k$ を用意する．各ハッシュ関数は $0, 1, \ldots, m-1$ の値を返す．

要素数が n である，ある集合 $W = \{w_1, \ldots, w_n\}$ の BF を作成する際は，各要素 $w \in W$ に対して $h_1(w), \ldots, h_k(w)$ を計算し，$B[h_1(w)], \ldots, B[h_k(w)]$ にそれぞれ 1 をセットする．このとき，同一のビットが複数回 1 にセットされることがあるが，2 回目以降は無視する．集合 W から作成

© 2015 Information Processing Society of Japan

されたBFを$B(W)$と表現する．

ある要素w'がWの要素であるかどうかのテストは，$B[h_1(w')],\ldots,B[h_k(w')]$の値を確認することで行われる．すべての値が1である場合は，要素w'はBFの作成元となった集合Wに含まれている可能性がある．そうでない場合，w'は集合Wには必ず含まれていない．

BFでは，偽陰性は発生しないが偽陽性が発生する可能性がある．つまり，ある要素w'に対して$B[h_1(w')],\ldots,B[h_k(w')]$の値がすべて1であったとしても，$w'$は$W$の要素でない可能性もある．逆に，1つでも0があった場合は，w'は必ずWの要素ではない．BFのビット長をm，BFの作成元となった集合の大きさをn，BFを作成する際のハッシュ関数の数をk，自然対数をeとすると，擬陽性が発生する確率（False Positive Rate: FPR）は次の式で表される[2]．

$$\mathfrak{R} \approx (1-e^{-kn/m})^k \tag{1}$$

3. 関連研究

3.1 BFを用いた手法

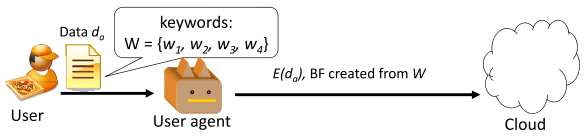
BFを用いて本課題に取り組む研究がさかんに行われている．以下では，共通BF方式，非共通BF方式，非共通BF方式の改良手法についてそれぞれ述べる．なお共通BF方式はIND2-CKAを満たさない．共通BF方式，非共通BF方式および提案手法の主な流れは**図1**に示すとおり，同じである．

以下では，データdに対し秘密鍵sで暗号化したものを$E_s(d)$と表記する．

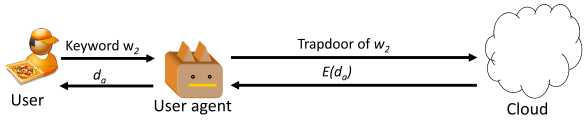
3.1.1 共通BF方式

単純に，データに付与されているキーワード集合をもとにBFを作成し，それを索引とする方式が考えられる．この手法を**共通BF方式**と呼ぶ．具体的には以下のとおりである．

ユーザが，キーワードw_1，w_2，w_3，w_4を含むデータd_aをクラウド上に保管したいとする．ユーザエージェントは，各キーワードに対してハッシュ関数hおよび秘密鍵sを使って，ハッシュ値$h(w_1,s)$，$h(w_2,s)$，$h(w_3,s)$，$h(w_4,s)$を計算する．この集合をW_Eとおく．次にユーザエージェントは，W_EからBloom Filter $B(W_E)$を作成する．ユーザエージェントはクラウドに，暗号化したデータ$E_s(d_a)$および索引$B(W_E)$を送信し，クラウドはこれらの情報を保管する．



(a) Indexing data



(b) Searching data

図1 共通BF方式，非共通BF方式，提案手法の主な流れ
Fig. 1 Outline of shared BF, non-shared BF and proposal.

ユーザが，キーワードw_2を含むデータをクラウドから入手したいとする．ユーザエージェントは，クラウドにトラップドア$h(w_2,s)$を送信する．これをを受け取ったクラウドは，当該ユーザから提供されたBF群すべてに対し，要素$h(w_2,s)$を含むかどうかのテストを行う．このテストを行うためには，要素$h(w_2,s)$からk個のハッシュ値を作成する必要があるが，このk個のハッシュ値は，当該ユーザの全BFに対して共通に利用することができる．テストの結果，要素$h(w_2,s)$を含む可能性のあると判断されたデータを抽出し，抽出されたデータ集合をユーザエージェントに返す．

共通BF方式を採用した場合，データd_aとデータd_bが同じキーワードを持っていたとしても，クラウド管理者はそうだという確信を持つことはできない．なぜなら，異なるキーワードを持っていても同じ箇所のビットが偶然立つ場合もあるからである．しかし同じ箇所のビットが立っているということは，そうでない場合と比べて，同じキーワードを保有している確率は高い．また，BF間において，共通に立っているビットがない場合は，共通のキーワードを絶対に保有していないということが分かる．したがって，IND2-CKAが満たされていない．

3.1.2 非共通BF方式

暗号化されたキーワードおよびデータIDを入力としてハッシュ値を計算し，それをもとにBFを作成する手法が提案されている[4], [12], [14]．この手法を**非共通BF方式**と呼ぶ．具体的には以下のとおりである．

ユーザが，キーワードw_1，w_2，w_3，w_4を含む，IDがaであるデータd_aをクラウド上に保管したいとする．ユーザエージェントは，集合$W_F=\{\{h(w_1,s),a\},\{h(w_2,s),a\},\{h(w_3,s),a\},\{h(w_4,s),a\}\}$を計算し，$\{h(w_i,s),a\}$を1つの要素と見なして，$W_F$からBloom Filter $B(W_F)$を作成する．

ユーザエージェントはクラウドに，暗号化したデータ$E_s(d_a)$，索引$B(W_F)$およびデータID aを送信し，クラウドはこれらの情報を保管する．

ユーザが，キーワードw_2を含むデータをクラウドから入手したい場合は，クラウドにトラップドア$h(w_2,s)$を送信する．この情報を受け取ったクラウドは，当該ユーザから提供されたBF群すべてに対し，要素$h(w_2,s)$を含むかどうかのテストを行う．データIDがaであるデータに紐付

© 2015 Information Processing Society of Japan

けられている BF に対してテストを行う際は，$\{h(w_2, s), a\}$ が要素に含まれているかどうかをテストし，データ ID が b であるデータに紐付けられている BF に対してテストを行う際は，$\{h(w_2, s), b\}$ が要素に含まれているかどうかをテストする．したがって，ユーザが登録した各データに対し，最大 k 個のハッシュ値をそれぞれ計算する必要がある．テストの結果，要素 $h(w_2, s)$ を含む可能性のあると判断されたデータを抽出し，抽出されたデータ集合をユーザエージェントに返す．

非共通 BF 方式では，データ ID も用いてハッシュ値を計算しており，データ ID が異なれば同じキーワードを保有していたとしても異なるビットが立つ（もちろん，偶然同じビットが立つ場合もある）．したがって，クラウド管理者は各データの BF を見ても，データ間の関係やキーワードの出現頻度を推測することができなくなり，IND2-CKA を満たす．しかし，ユーザがあるキーワード w' で検索を行いたい場合，各データに対して，最大 k 個のハッシュ値を計算する必要がある．ユーザ i が登録したドキュメント総数を N_i とすると，最大で $N_i \times k$ 個のハッシュ値を計算する必要が生じるため，検索性能が大幅に悪化する．

3.1.3 非共通 BF 方式の改良

多くの研究がこの非共通 BF 方式の改良版を提案して検索速度の向上を目指している．渡辺ら [15] や金子ら [13] はデータのキーワード集合の統計的な特徴をあらかじめ把握可能である前提の下で，検索速度の向上を図っている．したがって，あらかじめクラウド上に保管するデータ集合が確定している場合は有効であるが，本論文が想定するように，ユーザが随時自由にデータを追加できる状況には対応していない．

Goh [4] や山本ら [14] はユーザ側に各データのすべての検索キーワードを管理させ，データの修正や削除があまり発生しないという前提で，クラウド側で木構造を構築して検索速度の向上を図っている．クラウド上に保管するデータ数が多い場合や，各データのキーワード数が多い場合，これらの情報をユーザ側で管理する必要があるため，クラウドに保管するメリットが大きく損なわれてしまう．また，データの削除が発生した場合は，そのたびに木構造を再構築する必要が生じる．なお，文献 [14] と本提案手法は単純に併用可能であり，組み合わせることでさらにキーワード検索を高速に行うことができる．具体的には，BF の作成については本提案手法を用い，クラウド側でそれを文献 [14] が提案する木構造に基づいて配置すればよい．データ数やキーワード数が少なく，データの削除があまり起こらない状況では，このように文献 [14] と本提案手法を組み合わせて用いることができる．

Watanabe ら [12] は，数値を持つデータを対象にし，範囲検索を行うことのできる手法を提案している．本研究では，キーワード検索のみを対象とし，数値検索への応用はスコープ外とする．

3.2 その他の方式

あらかじめデータのキーワードとなりうる集合を定義し，その各キーワードに対して暗号化された索引を作成し，索引からデータ ID への紐付けにおいて，IND2-CKA を満たすアルゴリズムが提案されている [3], [5]．キーワードの集合をあらかじめ定義できればこれらの手法は有効である．しかし，定義されていないキーワードを含むデータをクラウドに追加する場合，その事実がクラウド側に漏れてしまうため，この状況を考慮する場合は IND2-CKA を満たすことができない．

クラウド上で索引を作成せずに，IND2-CKA を満たす手法もある [8], [11]．彼らの手法は，索引を作成しないためにクラウドに余分なデータ保存量を必要としないという利点がある．しかし，検索時に全データの全キーワードの数だけ暗号化処理を行う必要があるため，検索時間がかかるという問題がある．

4. 提案手法

同じキーワードを保有していても，データによって異なる箇所にビットが立つよう，BF を作成する必要がある．本論文では，キーワードのハッシュ値に対し，データ ID で MOD をとることによってこれを実現する．しかし，単純に MOD をとると，異なるデータから作成された BF から，ビットが立っている位置を推測できてしまうという問題がある．たとえば，あるキーワード w のハッシュ値が 3884 であったとする．データ ID が 15 である場合も 30 である場合も，$3884 \pmod{15} = 3884 \pmod{30} = 14$ となってしまう．これは，30 が 15 の倍数であるためであり，$x \pmod{30} < 15$ となる任意の自然数について，$x \pmod{15} = x \pmod{30}$ となる．このような問題を避けるよう，アルゴリズムを構築する必要がある．

4.1 事前準備

ユーザエージェントは暗号に用いる秘密鍵 s を保有している．

まず，以下のように BF のパラメータ，データに与える ID の集合の範囲，k 個のハッシュ関数をクラウド上で用意し，これらの値をユーザエージェントと共有する．また，BF 作成用の k 個のハッシュ関数とは別のハッシュ関数を 1 つ用意する．このハッシュ関数を h と表し，BF 作成用の k 個のハッシュ関数を $h_1, \ldots, h_k$ と表す．

(1) 各データに付与することが可能なキーワード数の最大値 u を決定する．
(2) BF の長さ m および利用するハッシュ値の数 k を決定する．
(3) データ ID がとりうる範囲として I_{min} および I_{max} を

© 2015 Information Processing Society of Japan

決定する．これは，$m < I_{min} < I_{max}$ の関係を満たすようにする．ユーザごとに登録可能なデータ数は，最大で I_{min} 以上 I_{max} 以下の範囲内にある素数の個数に限定されるため，I_{max} は十分大きな値に設定しておく．

(4) k 個のハッシュ関数 $h_1, \ldots, h_k$ を用意する．ここで，各ハッシュ関数が出力するハッシュ値のビット数を b とおくと，$2^b > I_{max}$ を満たすようにする．

なお，BF のビット長が m であるから，BF のどのビットを立てるかについての計算結果は，0 から $m-1$ までのいずれかの値をとる必要がある．通常，BF を作成する際には，m より十分大きい値（たとえば $m = 1{,}000$ のとき，2^{160}）を返しうるハッシュ関数を用いてハッシュ値を計算し，その値に対して m で MOD をとる．提案手法においては，ハッシュ値に対してまずデータ ID で MOD をとり，さらに m で MOD をとる操作を行うため，上述のように $m < I_{min}$ という制約を設けている．

上記のうち，BF の長さ m およびハッシュ値の数 k に決定方法ついては Broder らの論文 [2] が参考になるが，7 章においても考察する．

$I_{max} - I_{min}$ の範囲に存在する素数の個数は以下のように近似的に求められる．$\pi(x)$ を，x 以下の素数の個数を返す関数であるとする．素数定理により，

$$\pi(x) \sim x / \ln x \tag{2}$$

である．たとえば，$I_{max} = 2^{48}$，$I_{min} = 2^{32}$ であるとき，$\pi(I_{max}) - \pi(I_{min}) \approx 8.46 \times 10^{12}$ となり，十分な数の素数があることが確認できる．この値は，各ユーザにおける最大のデータ数であり，クラウド上に保管可能なデータ数の上限ではないことに注意いただきたい．

なお，ある与えられた値より大きい最小の素数を求めるには，フリーソフトウェアである Maxima[*1]等を利用することもできる．

4.2 データの追加

ユーザエージェントは追加したいデータ d に対し，クラウドエージェントからデータ d 用の ID（これまで利用した ID より大きい次の素数）を受け取る．ここで，データ d の ID を $D(d)$ とおく．

次に，データ d を秘密鍵 s で暗号化する．また，d に含まれる各キーワード $\{w_1, \ldots, w_n\}$ をそれぞれ秘密鍵 s と組み合わせてハッシュ値を計算し，集合 $W_d = h(w_1, s), \ldots, h(w_n, s)$ を作成する．

また，m ビットのすべてが 0 であるビット列を用意する．各要素 $h(w, s) \in W_d$ に対し，以下の式で Λ_j $(j = 1, \ldots, k)$ を求める．

$$\Lambda_j = h_j(h(w, s))|2^b \tag{3}$$

ここで，| はビット和を表す．ビット和をとる理由は，Λ_j が必ず I_{max} より大きくなることを保証するためである．

各 Λ_j の値はデータ ID に依存しない．データ ID ごとに異なる値となるように，以下のようにデータ ID で MOD をとる．さらに，BF の長さ m の範囲に収まるように m でも MOD をとる．

$$\Xi_{D(d),j} = \Lambda_j \pmod{D(d)} \pmod{m} \tag{4}$$

各キーワードについて式 (4) を計算し，BF の該当ビットを 1 にセットする．

さらに，キーワード数を隠蔽するため，事前に設定した最大キーワード数を u とおくと，ランダムに生成された意味のない文字列を $u - |W_d|$ 個追加する．この追加は，意味のない文字列に対し，式 (3) および式 (4) の作業を行うことでも実現可能であるし，0 から $m-1$ までの値を一様分布の確率に基づいて返す関数を $k \cdot (u - |W_d|)$ 回呼び出して BF の該当するビット部分を 1 にすることでも実現できる．

ユーザエージェントのアルゴリズムを Algorithm 1 に示す．ここで関数 Ran は，引数として自然数 m を受け取り，0 から $m-1$ までの値を一様分布の確率に基づいて返す関数である．

Algorithm 1 Creating a BF of document d

Input: Data d, set of keywords W
Output: BF of d

```
    /*Prepares data ID*/
 1: D(d) ⇐ id of d, received from the Cloud
    /*Creates a BF*/
 2: B ⇐ m-bit array initially all set to 0.
 3: for w ∈ W do
 4:     for j = 1, ..., k do
 5:         Λ_j ⇐ h_j(h(w, s))|2^b
 6:         Ξ_{D(d),j} ⇐ Λ_j (mod D(d)) (mod m)
 7:         B[Ξ_{D(d),j}] ⇐ 1
 8:     end for
 9: end for
10: for l = 1, ..., u − |W| do
11:     for j = 1, ..., k do
12:         B[Ran(m)] ⇐ 1
13:     end for
14: end for
15: return B
```

4.3 データの削除，修正

データの削除については，単純に該当するデータおよび BF を削除することで対応できる．また，データやキーワードの修正を行う場合は，データの削除および追加を行うことで対応する．

[*1] http://maxima.sourceforge.net/

© 2015 Information Processing Society of Japan

4.4 クラウドにおけるデータ検索

ユーザがあるキーワード w で検索を行いたい場合，ユーザエージェントはトラップドア $h(w,s)$ を生成してクラウドエージェントへ送信する．

クラウドエージェントは式 (3) を使って，$h(w,s)$ に対して各 Λ_j（$j=1,\ldots,k$）を計算する．クラウド上に保管されている当該ユーザのデータ ID の集合を A とおく．各データ ID $a \in A$ に対し，$\Xi_{a,j}$ を計算し，各 BF の当該ビットを確認する．すべて 1 であれば，レスポンスとして返す集合に，ID が a であるデータを追加する．クラウドエージェントのアルゴリズムを Algorithm 2 に示す．ここで，$BF(d)$ はデータ d の BF を表している．

Algorithm 2 で得られた集合には，BF の擬陽性の特性により誤ったデータが一定の確率で含まれている．この誤ったデータを処理するためユーザエージェントは，クラウドエージェントから得た集合を復号化してユーザにそれらを提示する際に，これらのデータを除外する．これは復号化したデータに対してキーワード検索を行うことで可能である．

偽陽性の発生確率とクラウド上でのデータ保存量について，適切にトレードオフをとるための考察は 7 章で述べる．

Algorithm 2 Searching data

Input: Trapdoor $h(w,s)$, a set of all encrypted documents D_E and their BFs
Output: Encrypted documents that potentially contain w

```
Creates an empty set S
for j = 1, ..., k do
    Λ_j = h_j(h(w,s))|2^b
end for
for d ∈ D_E do
    flag ⇐ true
    a ⇐ ID of d
    for j = 1, ..., k do
        Ξ_{a,j} ⇐ Λ_j (mod a) (mod m)
        if BF(d)[Ξ_{a,j}] ≠ 1 then
            flag ⇐ false
            break
        end if
    end for
    if flag == true then
        S ⇐ S ∪ {d}
    end if
end for
return S
```

5. 安全性についての議論

5.1 概要

非共通 BF 方式において，BF を作成する際に利用する k 個のハッシュ関数を $q_1,\ldots,q_k$ とおく．非共通 BF 方式では，データ ID が $D(d)$，キーワードが w であるものについて，BF を作成する際の j 番目のハッシュ値の計算を $q_j(h(w,s),D(d)) \pmod m$ として行っている．ここで，このハッシュ値のとりうる値の範囲は 0 から $m-1$ である．攻撃者が，データ ID が $D(d')$，キーワードが w' であるもの（$d' \neq d$ and/or $w' \neq w$）について，$q_j(h(w',s),D(d'))$ の値を知っていたとしても，$q_j(h(w,s),D(d)) \pmod m$ の値の推測精度に影響を与えない（$1/m$ を超えない）．つまり，キーワードもしくはデータ ID のいずれかが異なれば，索引間に何の関係性もなくなるため，索引からは何の情報も得られない．また，データは暗号化されていているため，クラウド側はその中身を知ることはできない．したがって，IND2-CKA が満たされている [4]．

本論文では，データ ID が $D(d)$，キーワードが w であるものについて，BF を作成する際の j 番目のハッシュ値の計算を $h_j(h(w,s))|2^b \pmod{D(d)} \pmod m$ として行っている．攻撃者が，データ ID が $D(d')$，キーワードが w' であるもの（$D(d') \neq D(d)$ and/or $w' \neq w$）について，$h_j(h(w',s))|2^b \pmod{D(d')} \pmod m$ の値を知っていたとしても，$h_j(h(w,s))|2^b \pmod{D(d)} \pmod m$ の値の推測精度が $1/m$ を超えないとき，IND2-CKA が満たされる．

$\Lambda_j = h_j(h(w,s))|2^b$ とおき，以下では，任意の 2 つの素数のデータ ID $D(d)$，$D(d')$ について，$\Lambda_j \pmod{D(d')} \pmod m$ の値を攻撃者が知っている（ただし Λ_j や $\Lambda_j \pmod{D(d')}$ の値は知らない）場合においても，$\Lambda_j \pmod{D(d)} \pmod m$ の値の推測精度が「無視できるほど小さい値」を除いて $1/m$ を超えないことを示す．

なお，ある BF においてある a 番目のビットに対応するハッシュ値の候補は，ハッシュ空間の 2^b 個のうち，$2^b/m$ 個に限定される．したがって，b の値が小さいとき，また，BF のビット長 m の値が大きいときは，安全性が低下する．具体的には，この限定されたハッシュ値候補を使い，他 BF に対して検索をかけ，同じ単語を持つ可能性の高いデータ，あるいは可能性の低いデータを推測する攻撃が考えられる．この問題は提案手法に限定したものではなく，非共通 BF 方式においても同じように発生する．非共通 BF 方式やその改良手法を提案している既存手法でもこの問題は議論されていないが，本論文では 5.4 節において議論する．

5.2 アプローチ

攻撃者が $\Lambda_j \pmod{D(d')} \pmod m$ の値を知っているときに，$\Lambda_j \pmod{D(d)} \pmod m$ の値を推測できる精度の上限値を求める．

$\Lambda_j = h_j(h(w,s))|2^b$ であるから，Λ_j のとりうる値は，2^b から $2^{b+1}-1$ までである．$i \in [2^b : 2^{b+1}-1]$ の各 i に対して，$i \pmod{D(d)} \pmod m$ の値と $i \pmod{D(d')} \pmod m$ の値を計算する．ここで，ある整数 α, β を考える．すべての $i \in [2^b : 2^{b+1}-1]$ について $i \pmod{D(d)} \pmod m$ と $i \pmod{D(d')} \pmod m$ を計算し，その計算した結果の組合せが (β,α) となる個数を C_{max} とおく．つ

© 2015 Information Processing Society of Japan

まり，以下の計算を行う．

$$C_{max} = \sum_{i\in[2^b:2^{b+1}-1]} M$$
$$\text{where} \quad M = \begin{cases} 1 & \begin{pmatrix} i \pmod{D(d)} \pmod m = \beta \ \wedge \\ i \pmod{D(d')} \pmod m = \alpha \end{pmatrix} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

また，すべての $i \in [2^b : 2^{b+1}-1]$ について $i \pmod{D(d')} \pmod m$ を計算し，その値が α となる個数を C_{sum} とおく．つまり，以下の計算を行う．

$$C_{sum} = \sum_{i\in[2^b:2^{b+1}-1]} M$$
$$\text{where} \quad M = \begin{cases} 1 & (i \pmod{D(d')} \pmod m = \alpha) \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

攻撃者は，$\Lambda_j \pmod{D(d')} \pmod m$ の値が α であると知っている場合，C_{max}/C_{sum} の精度で，前者 $\Lambda_j \pmod{D(d)} \pmod m$ の値が β であると推測することができる．

たとえば，$D(d)$ と $D(d')$ が互いに素ではなく，$D(d) = 2 \times D(d')$ の関係があるときの例を説明する．仮に，$b = 5$，$D(d) = 14$，$D(d') = 7$，$m = 5$ とする．すべての $i \in [2^5 : 2^6 - 1]$ について，$i \pmod{14} \pmod 5$ と $i \pmod 7 \pmod 5$ を計算する．$i = 2^5, 2^5+1, \ldots$ のとき，それぞれ $4, 0, 1, 0, \ldots$ と $4, 0, 1, 2, \ldots$ のように計算される．ここでたとえば $\alpha = \beta = 0$ とおく．$i \in [2^5 : 2^6 - 1]$ の範囲において，$i \pmod 7 \pmod 5$ の値が α となる i の数は 10 個ある．また，$i \in [2^5 : 2^6 - 1]$ の範囲において，$i \pmod 7 \pmod 5$ の値が α となりかつ $i \pmod{14} \pmod 5$ の値が β となる i の数は 5 個ある．したがってこの場合，$C_{max}/C_{sum} = 0.5$ となる．つまり攻撃者は，$\Lambda_j \pmod{D(d')} \pmod m$ の値が 0 だと知っている場合，$\Lambda_j \pmod{D(d)} \pmod m$ の値を 50%の確率で 0 だと推測することが可能となってしまう．

本来は，BF のビット数が m であるため，$\Lambda_j \pmod{D(d)} \pmod m$ の値を推測できる確率は $1/m$ となるはずであるが，$D(d)$ と $D(d')$ が互いに素でないこの例の場合は，推測確率が大幅に増加してしまっている．

以下では，この C_{max}/C_{sum} の上限値を求める．本論文では「上限値」という言葉を，実際にその値になりうるかどうかにかかわらず，この値を上回ることはない，という意味で利用する．「下限値」も同様である．

5.3 議論の詳細

5.3.1 Λ_j の分析

Λ_j=$h_j(h(w,s))|2^b$ である．またハッシュ関数 h_j が出力するビット数は b ビットである．このビット列と 2^b とのビット和をとるということは，ハッシュ関数 h_j が出力する b ビットに対して，最上位ビットに単純に “1” を追加することを意味する．

h_j が，あらゆる入力値 $h(w,s)$ に対して，値域（0 から $2^b - 1$）に一様に分布するようなランダムの応答を返すとき，$h_j(h(w,s))|2^b$ は，あらゆる入力値 $h(w,s)$ に対して，値域（2^b から $2^{b+1} - 1$）に一様に分布するようなランダムの応答を返す．

したがって，$h_j(h(w,s))|2^b$ については，攻撃者が $w' \neq w$ となる $h_j(h(w',s))|2^b$ の値を何らかの手段で推測できたとしても，$h_j(h(w,s))|2^b$ の推測精度は向上しない．

5.3.2 $\Lambda_j \pmod{D(d)} \pmod m$ の分析

一般に，ある整数 x を用意し，$i \in [x : x + D(d) - 1]$ の各 i について，$i \pmod m$ を計算する場合，そのとりうる値は 0 から $m-1$ までの m 種類ある．$i \in [x : x + D(d) - 1]$ の各 i について，$i \pmod m$ の値を計算し，0 から $m-1$ までの各値が何度出現するかをカウントし，その結果を $F_{D(d),m,s}$ に格納する（$s = 0, \ldots, m-1$）．つまり，以下のように計算する．

$$F_{D(d),m,s} = \sum_{i\in[x:x+D(d)-1]} M$$
$$\text{where} \quad M = \begin{cases} 1 & (i \pmod m = s) \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

このとき，与えられた $D(d)$，m において，F の上限値は

$$F_{D(d),m,s} \leq \left\lceil \frac{D(d)}{m} \right\rceil \tag{8}$$

であり，下限値は

$$F_{D(d),m,s} \geq \left\lfloor \frac{D(d)}{m} \right\rfloor \tag{9}$$

である．

たとえば $x = 0, D(d) = 5, m = 2$ としたとき，$F_{5,2,s} \leq 3$，$F_{5,2,s} \geq 2$ となる．実際に，$i \in [0 : 4]$ の各 i について，$i \pmod 2$ を計算すると，$i \pmod 2$ の値が 0 となるのは 3 通り，この値が 1 となるのは 2 通りであるので，上限値ならびに下限値とも条件を満たしている．

次に，ある整数を用意し，$i \in [x : x + D(d) \times D(d') - 1]$ の各 i について，$i \pmod{D(d)} \pmod m$ および $i \pmod{D(d')} \pmod m$ の値を計算し，各値のペアが何度出現するかをカウントし，その結果を $G_{D(d),D(d'),m,s,s'}$ に格納する（$s, s' = 0, \ldots, m-1$）．つまり，以下のように計算する．

© 2015 Information Processing Society of Japan

$$G_{D(d),D(d'),m,s,s'} = \sum_{i\in[x:x+D(d)\times D(d')-1]} M_{s,s',i,m}$$

where $M_{s,s',i,m}$

$$= \begin{cases} 1 & \begin{pmatrix} i \pmod{D(d)} \pmod m = s \ \wedge \\ i \pmod{D(d')} \pmod m = s' \end{pmatrix} \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

このとき，与えられた $D(d)$，$D(d')$，m において，$D(d)$，$D(d')$ が互いに素であるとき，$G_{D(d),D(d'),m,s,s'}$ の上限値は，

$$G_{D(d),D(d'),m,s,s'} \le F_{D(d),m,s} \times F_{D(d'),m,s'} \le \left\lceil \frac{D(d)}{m} \right\rceil \times \left\lceil \frac{D(d')}{m} \right\rceil \quad (11)$$

である．なぜなら，$i \in [x : x + D(d) \times D(d') - 1]$ の各 i について，$\{i \pmod{D(d)}, i \pmod{D(d')}\}$ の値の組合せを計算すると，各値の組合せは $\{0, 0\}$ から $\{D(d)-1, D(d')-1\}$ まで全通り 1 度ずつ出現するためである．

5.3.3 C_{max}/C_{sum} の上限値

前項の $F_{D(d),m,s}$ は，ある x に対して x から $x+D(d)-1$ の範囲，$G_{D(d),D(d'),m,s,s'}$ は x から $x + D(d) \times D(d') - 1$ の範囲を対象としている．対象範囲が 2^b から $2^{b+1} - 1$ である C_{sum} の上限値および C_{max} の下限値に拡張すると，以下のように表すことができる．

$$C_{sum} \ge F_{D(d),m,s} \times \left\lfloor \frac{2^{b+1} - 1 - 2^b - 1}{D(d)} \right\rfloor \quad (12)$$

$$C_{max} \le G_{D(d),D(d'),m,s,s'} \times \left\lceil \frac{2^{b+1} - 1 - 2^b - 1}{D(d) \times D(d')} \right\rceil \quad (13)$$

式 (12) および式 (13) より，C_{max}/C_{sum} の上限値は以下のように求まる．

$$\frac{C_{max}}{C_{sum}} \le \frac{\lceil D(d)/m \rceil \lceil D(d')/m \rceil \lceil (2^b - 2)/(D(d)D(d')) \rceil}{\lfloor D(d)/m \rfloor \lfloor (2^b - 2)/D(d) \rfloor} \quad (14)$$

5.3.4 C_{max}/C_{sum} の数値分析

式 (14) から $1/m$ を引いた値がほとんど無視できるほど小さい値を除いて 0 であることを示す．

たとえば $I_{min} = 2^{32}$，$b = 160$ のとき，m の値が 1,000 であるときは式 (14) から $1/m$ を引いた値は約 4.0×10^{-10} となる．さらに，$I_{min} = 2^{64}$ とすると，式 (14) から $1/m$ を引いた値は，m の値が 1,000 であるときは，約 7.5×10^{-20} となる．

m の値を変えると式 (14) から $1/m$ を引いた値は変動する．しかし，$I_{min} = 2^{32}$，$b = 160$ のとき，m の値が 1 から 10,000 までにおける最大値は 4.7×10^{-10} であり，十分無視できるほど小さいことが分かる．さらに，b や I_{min} の値を大きくすることで式 (14) から $1/m$ を引いた値を限りなく 0 に近づけることが可能となる．

したがって，$\Lambda_j \pmod{d'} \pmod m$ の値を攻撃者が知っている場合においても，b や I_{min} の値を十分大きく設定すると，$\Lambda_j \pmod d \pmod m$ の値の推測精度は，このようなごく小さい値を除いて $1/m$ を超えないことが分かる．

5.4 限定されたハッシュ値候補を使った攻撃

あるデータ d の ID を $D(d)$ とする．このデータの BF において，ある a 番目のビットが 1 であるとする．このとき，$\Lambda_j \pmod{D(d)} \pmod m = a$ という計算がなされたことになる．ここで，Λ_j のとりうる値の候補の集合を $T_{d,a}$ とおくと，以下の式で表される．

$$T_{d,a} = \{t | 2^b \le t \le 2^{b+1}\text{–}1 \ \&\ t \pmod{D(d)} \pmod m = a\} \quad (15)$$

また，この集合 $T_{d,a}$ の要素数は以下の式で表すことができる．

$$|T_{d,a}| \approx \left(\frac{2^b}{D(d)}\right) \times \left(\frac{D(d)}{m}\right) = \frac{2^b}{m} \quad (16)$$

また，データ d の BF において，1 であるすべてのビットについて $T_{d,a}$ を計算し，その和をとったものを $T_{d,all}$ とおく．つまり，当該 BF において 1 であるビットの集合を A_d とおくと，

$$T_{d,all} = \bigcup_{a\in A_d} T_{d,a} \quad (17)$$

である．

この集合 $T_{d,all}$ を用いて，その他の BF に対して網羅的に検索をかけ，同じ単語を持つ可能性の高いデータや，同じ単語を持つ可能性がないデータを推測する攻撃を行うことが考えられる．以下で，この両者について検討する．

5.4.1 共通の単語が存在しないと判明する可能性の検討

集合 $T_{d,all}$ のすべての要素 t について，$t \pmod{D(d')} \pmod m$ を計算し，データ d' の BF における当該ビットが 1 となるものが 1 つもない場合，データ d に含まれている単語は，データ d' には 1 つも含まれていないということが確定する．

データ d' の BF において，1 であるビット数を c とおく．また，$|T_{d,all}|$ のとりうる最小値は，すべての $T_{d,a}$ が同じ集合であるときであり，このときの値は式 (16) で表される値，つまり，およそ $2^b/m$ となる．したがって，データ d および d' について，共通の単語が存在しない場合に，それが判明する確率を P とおくと，

$$P \le \left(1 - \frac{c}{m}\right)^{2^b/m} \le \left(1 - \frac{1}{m}\right)^{2^b/m} \quad (18)$$

と表すことができる．

たとえば，$m = 1{,}000$，$b = 30$ のとき，$P \le (1 - 1/1000)^{2^{30}/1000} \approx 2.8 \times 10^{-467}$ であり，この確率は限りなく 0 に近いことが分かる．b の値を大きくすると，さらに 0 に近い値となる．

© 2015 Information Processing Society of Japan

5.4.2 共通の単語が存在する可能性が高いと判明する可能性の検討

仮にデータ d と d' が共通の単語を保有している場合，集合 $T_{d,all}$ の各要素 t について $t \pmod{D(d')} \pmod{m}$ を計算すると，少なくとも 1 つ以上，データ d' の BF における当該ビットが 1 となっている．

データ d と d' が共通の単語を保有していない場合において，データ d' の BF における該当するビットが 1 であるものが 1 つでもある確率を Q とおくと，

$$Q \geq 1 - \left(1 - \frac{c}{m}\right)^{2^b/m} \geq 1 - \left(1 - \frac{1}{m}\right)^{2^b/m} \qquad (19)$$

とおくことができる．

Q の値がごく小さいときに，集合 $T_{d,all}$ の各要素 t について $t \pmod{D(d')} \pmod{m}$ を計算し，少なくとも 1 つ以上，データ d' の BF における当該ビットが 1 となった場合は，これが偶然発生したと考えるには確率が小さすぎると考え，データ d と d' は共通の単語を保有していると判断することができる．しかし，Q の値がほぼ 1 の場合，偶然このような状況が発生することが十分想定されるため，データ d と d' は共通の単語を保有しているとの判断ができなくなる．

たとえば，$m = 1{,}000$，$b = 30$ のとき，$Q \geq 1 - (1 - 1/1000)^{2^{30}/1000}$ となるが，この右辺は 0.99... のように小数点以下，9 が 450 桁以上続く値となり，限りなく 1 に近い．b の値を大きくすると，さらに 1 に近付く．

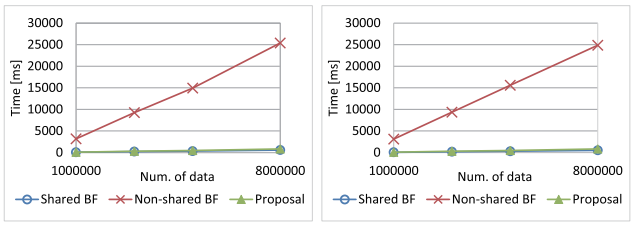
6. 評価

本章では，共通 BF 方式，非共通 BF 方式と提案手法について，キーワードによる検索速度を評価する．前述したとおり，共通 BF 方式は検索が高速であるが，IND2-CKA を満たさない．IND2-CKA を満たす提案手法が，IND2-CKA を満たさない共通 BF 方式と同程度の検索速度を実現できることを示す．

評価は，Intel Xeon CPU E5-2687W v2 @ 3.40 GHz 3.40 GHz および 128 GB の RAM を搭載したワークステーションで行った．

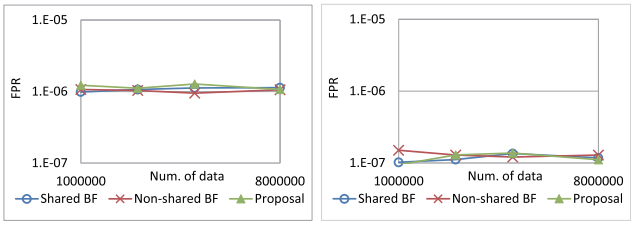
株式会社ドワンゴが提供しているニコニコ動画コメント等データを使い，各動画に付与されているタグをキーワードとして利用した．利用した動画数は 7,933,174，総タグ数は 43,635,584 であり，異なるタグ数は 5,158,893 であった．

事前準備として，各手法でワークステーション上に全データの索引を追加した．評価は，ユーザがあるキーワードで検索した際に該当する動画 ID の集合を得るまでに要した時間および偽陽性の発生割合を計測して行った．デフォルトのパラメータ設定として，$I_{min} = 2^{32}$，$I_{max} = 2^{64}$，$k = 20$，$b = 160$ とした．また，各動画に付与されているキーワード数はそれぞれ異なっているが，設定可能な最大値を $u = 12$ とした．



(a) # of keywords is 1. (b) # of keywords is 2.

図 2 検索に要した時間とデータ数との関係
Fig. 2 Searching time vs. number of data.



(a) # of keywords is 1. (b) # of keywords is 2.

図 3 FPR とデータ数との関係
Fig. 3 FPR vs. number of data.

また，k および BF を作成する集合の要素数 u の値が与えられると，BF のビット長 m の最適なパラメータ値が決定する．BF における最適なパラメータ設定の方法については 7 章に示す．同時に，想定される FPR の値も決まる．たとえば u=12 である場合，$k = 10$，15，20，25 のとき，m の値はそれぞれ 174，261，347，434 となる．

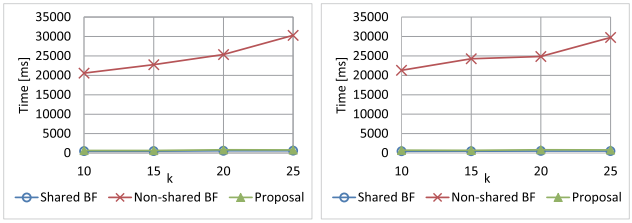
動画に付与されているタグをランダムに選択して検索キーワードを作成し，この検索キーワードを使って検索を行った．この作業を 500 回繰り返し，その平均値をとった．

データ数を変えて実験を行った結果を**図 2** および**図 3** に示す．

図 2 (a) は 1 つのキーワードで検索した検索時間の結果を，図 2 (b) は 2 つのキーワードによる AND 検索を行った検索時間の結果を表している．どの手法においても，検索にかかる時間はデータ数に対して線形に増加していることが分かる．非共通 BF 方式は，データ数が 100 万程度の場合でも検索に約 3.1 秒かかっている．2 キーワードによる検索ですべてのデータを利用した場合は約 30.2 秒を要しているが，提案手法では 0.7 秒であった．2 キーワードによる検索では，1 キーワード目がヒットした動画に対してのみチェックを行っているので，1 キーワードにかかるオーバヘッドよりも 2 キーワード目にかかるオーバヘッドのほうが大幅に小さい．したがって，1 キーワードによる検索時間も 2 キーワードによる検索時間もほとんど変わらなかった．

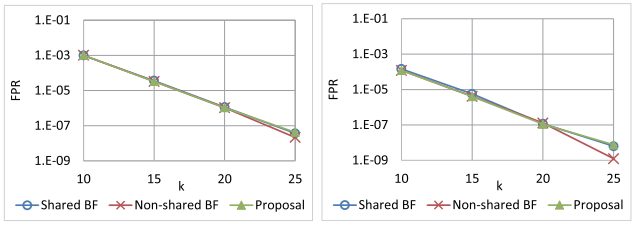
図 3 (a) は 1 つのキーワードで検索した FPR の結果を，

© 2015 Information Processing Society of Japan



(a) # of keywords is 1. (b) # of keywords is 2.

図 4 検索に要した時間と k との関係
Fig. 4 Searching time vs. k.



(a) # of keywords is 1. (b) # of keywords is 2.

図 5 FPR と k との関係
Fig. 5 FPR vs. k.

図 3 (b) は 2 つのキーワードによる AND 検索を行った FPR の結果を表している．非共通 BF 方式，共通方式，提案手法の FPR は，データ数に依存せずほとんど同じ結果になった．これは，作成方法に違いはあるが，いずれも同じパラメータ（k および m）を使って BF を作成しているためである．たとえば k=20 のとき，最適な m の値は式 (21) より 347，想定される FPR は式 (1) より約 10^{-6} であり，図 3 (a) の実験結果とほぼ一致している．2 キーワードによる AND 検索では，各キーワードによる検索の本来のヒット率や，1 番目と 2 番目のキーワードの共起率によって，FPR は変動する．全データが，両方のキーワードのいずれも含んでいない場合は，$k = 20$ のとき FPR は平均的には $(10^{-6})^2$ となる．そうでない場合，FPR は平均的には $(10^{-6})^2$ から 10^{-3} の間の値となる．

次に，k の値を変えて実験を行った結果を**図 4** および**図 5** に示す．

図 4 (a) は 1 つのキーワードで検索した検索時間の結果を，図 4 (b) は 2 つのキーワードによる AND 検索を行った検索時間の結果を表している．どの手法においても，k の値が増加するにつれて検索にかかる時間が増加している．これは，BF に対する要素の存在チェックは，各 BF に対して最大 k 回行われるためである．

図 5 (a) は 1 つのキーワードで検索した FPR の結果を，図 5 (b) は 2 つのキーワードによる AND 検索を行った FPR の結果を表している．いずれの手法においても，k の値が増加するほど，小さい FPR が実現できていることが分かる．

7. 考察

BF を作成する際には，パラメータとして BF のビット長 m およびハッシュ関数の数 k を決定する必要がある．これらは主に，偽陽性が発生する確率および BF を保存するクラウドの保存コストのトレードオフから決定される．

偽陽性が発生する確率の目標値を T とおくと，最適な k および m の値は以下のとおりとなる．ここでは議論を簡単にするために，T はある自然数 r を使って $T = (1/2)^r$ と表されるものとする．また，データに付与される最大キーワード数を u とする．

$$k = -\frac{\ln T}{\ln 2} \tag{20}$$

$$m = \frac{k \cdot u}{\ln 2} \tag{21}$$

1 つのデータあたり，索引として m ビットの容量を必要とすることになる．これがクラウド上において許容されるのであれば，これらの値を使う．もし許容できない場合は，偽陽性が発生する確率の目標値を緩和したり，各データに付与できる最大のキーワード数を削減したりする必要がある．

本論文で提案した手法は，共通 BF 方式および非共通 BF 方式と比べると，索引に必要な容量や FPR に差はなく，トレードオフについての考え方は既存研究と同じである．

8. おわりに

本論文では，ユーザが第三者のクラウドにデータを安全に保管し，検索を行うシナリオにおいて，あらかじめキーワード集合が既知でない場合でも利用でき，また，ユーザ側に検索キーワードを管理させることなく，従来よりも大幅に検索速度を高めることのできる手法を提案した．

また，本論文では，ごく小さい値を無視することで IND2-CKA を満たしていると考えているが，将来課題として，この値も考慮することができる指標を用いて，各パラメータの設定値を決めるための考え方を整理する必要がある．

謝辞 本研究は JSPS 科研費 24300005, 26330081, 26870201 の助成を受けたものです．

参考文献

[1] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors, *Comm. ACM*, Vol.13, No.7, pp.422–426 (1970).
[2] Broder, A. and Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey, *Internet Mathematics*, Vol.1, No.4, pp.485–509 (2004).
[3] Curtmola, R., Garay, J., Kamara, S. and Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions, *Proc. ACM CCS*, pp.79–88 (2006).
[4] Goh, E.-J.: Secure indexes, Technical Report, IACR ePrint Cryptography Archive, Report 2003/216 (2003).

© 2015 Information Processing Society of Japan

[5] Kamara, S., Papamanthou, C. and Roeder, T.: Dynamic searchable symmetric encryption, *Proc. ACM CCS*, pp.965–976 (2012).
[6] Kerschbaum, F.: Outsourced private set intersection using homomorphic encryption, *Proc. ACM ASIACCS*, pp.85–94 (2012).
[7] Nagy, M., De Cristofaro, E., Dmitrienko, A., Asokan, N. and Sadeghi, A.-R.: Do I know you?: Efficient and privacy-preserving common friend-finder protocols and applications, *Proc. ACM ACSAC*, pp.159–168 (2013).
[8] Popa, R.A., Redfield, C.M.S., Zeldovich, N. and Balakrishnan, H.: CryptDB: processing queries on an encrypted database, *Comm. ACM*, Vol.55, No.9, pp.103–111 (2012).
[9] Reynolds, P. and Vahdat, A.: Efficient peer-to-peer keyword searching, *Proc. ACM/IFIP/USENIX Middleware*, pp.21–40 (2003).
[10] Sei, Y. and Ohsuga, A.: False Event Detection for Mobile Sinks in Wireless Sensor Networks, *Proc. European Intelligence and Security Informatics Conference* (*EISIC*), pp.52–59 (2013).
[11] Wagner, D. and Perrig, A.: Practical techniques for searches on encrypted data, *Proc. IEEE S&P*, pp.44–55 (2000).
[12] Watanabe, C. and Arai, Y.: Privacy-Preserving Queries for a DAS Model Using Encrypted Bloom Filter, *Proc. International Conference on Database Systems for Advanced Applications*, Vol.5463, pp.491–495 (2009).
[13] 金子静花，渡辺知恵美，柿澤美穂，天笠俊之：暗号化データベースにおける多属性索引に対する統計攻撃モデルと攪乱戦略，日本データベース学会論文誌，Vol.12, No.1, pp.109–114 (2013).
[14] 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).
[15] 渡辺知恵美，新井裕子，天笠俊之：ブルームフィルタを用いたプライバシ保護検索における攻撃モデルとデータ撹乱法の一検討，日本データベース学会論文誌，Vol.8, No.1, pp.113–118 (2009).

清 雄一 （正会員）

1981 年生．2009 年東京大学大学院情報理工学系研究科博士後期課程修了．同年（株）三菱総合研究所入社．同社情報技術研究センター，金融ソリューション本部等に所属．2013 年より電気通信大学助教，現在に至る．分散コンピューティング，セキュリティ，プライバシ保護技術等の研究に従事．電子情報通信学会，IEEE Computer Society 各会員．

竹之内 隆夫 （正会員）

2003 年電気通信大学電気通信学部情報工学科卒業．2005 年同大学大学院情報システム学研究科博士前期課程修了．2013 年同大学院同研究科博士後期課程修了．博士（工学）．2005 年日本電気（株）入社．現在，クラウドシステム研究所主任．主としてパーソナル情報の利活用におけるプライバシ保護の研究に従事．電子情報通信学会会員．

大須賀 昭彦 （正会員）

1958 年生．1981 年上智大学理工学部数学科卒業．同年（株）東芝入社．同社研究開発センター，ソフトウェア技術センター等に所属．1985～1989 年（財）新世代コンピュータ技術開発機構（ICOT）出向．2007 年より電気通信大学大学院情報システム学研究科教授．2012 年より国立情報学研究所客員教授兼任．工学博士（早稲田大学）．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．IEEE Computer Society Japan Chapter Chair，人工知能学会理事，日本ソフトウェア科学会理事を歴任．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE Computer Society 各会員．

© 2015 Information Processing Society of Japan