Semi-ShuffledBF:ブルームフィルタを用いた安全かつより高速なプライバシ保護検索手法の提案

金子 静花† 渡辺 知恵美† 天笠 俊之‡

†お茶の水女子大学 理学部情報科学科 〒112-8610 東京都文京区大塚 2-1-1
‡筑波大学 大学院システム情報工学研究科 〒305-8573 茨城県つくば市天王台 1-1-1
E-mail: †{g0720515, chiemi}@is.ocha.ac.jp, ‡amagasa@cs.tsukuba.ac.jp

あらまし DaaS (Database as a Service) において，ユーザはインターネット上の第三者が管理するデータベースのサービスをネットワーク経由で利用することができる．このような環境では，ユーザがデータ管理者から機密情報を守ることが困難となる．この問題に対し，我々は先行研究において「ブルームフィルタを用いたスキーマ情報を隠蔽するプライバシ保護検索手法」を提案した．この手法では，各タプルに対して問合せ用のブルームフィルタを生成し，タプル毎にキーを用いてブルームフィルタのビット列をシャッフルする (ShuffledBF)．これにより，ビットパターンの漏えいを防ぐことが可能となる．その反面，問合せの際，各タプル毎にハッシュ関数を適用してシャッフルしたビット列を復元する必要があり，タプル数に比例した処理時間がかかってしまうという問題があった． 一方，ブルームフィルタのシャッフルを行わない (Non-ShuffledBF) 場合，ビットパターンの推測が可能となりセキュリティ上問題がある． そこで我々は，問合せの第 1 段階の絞り込みに Non-ShuffledBF を用い，第 2 段階の絞り込みに ShuffledBF を用いるハイブリッドな手法 Semi-ShuffledBF を提案する．

キーワード データベースアウトソーシング，クラウドコンピューティング，Database as a Service，ブルームフィルタ，プライバシ保護検索

1. はじめに

現在，データベース製品の管理運用を外部のデータベース管理者に委託する Database as a Service (DaaS) サービスが注目を集めている．クラウド環境による DaaS サービスには現在 S3, EC2 , SimpleDB[1], Azure[2], Google Apps Engine[3] などがあり，データベースサーバを常時運用することが難しい個人や小規模な企業などによって利用されている．

データベースシステムの管理を第 3 者に委託する環境では，委託業者であるデータベース管理者に機密情報を閲覧されたくないというユーザの要求が生じる．この問題に対する解決策として，データを暗号化した状態でデータベースに保存し，暗号化したまま問合せを施すプライバシ保護検索手法があり，これまで多くの研究が提案されてきた[1][3][4][6][8][10]．プライバシ保護検索手法における一般的な処理の流れを図 1 に示す．まず，データをタプル毎に暗号化してサーバに保存する．また，各タプルに検索用の索引（図 1 中①）を付与する．この索引はその値から元データが推測できないものであり，サーバにおける問合せはこの索引を用いて行われる．従来この索引は各タプルの属性毎に用意されており，属性のデータ型や問合せの演算に合わせて様々な索引が提案されてきた．

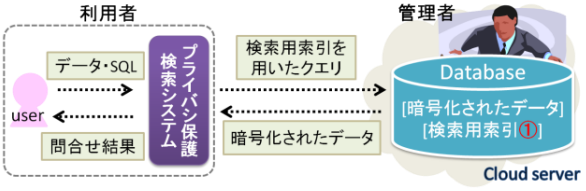

図 1 : プライバシ保護検索システム

我々は先行研究[2][13][14]において，ブルームフィルタを用いたプライバシ保護手法 (ShuffledBF) を提案してきた．ShuffledBF では，他の手法とは異なり属性毎に索引を構築しないため，検索用索引やクエリを手がかりとしたスキーマ情報や分布情報への攻撃を防ぐことができる．しかし，ShuffledBF は高い安全性を有している一方処理速度が遅いという問題があった．

そこで本研究では，安全性は高くないが処理速度の速い Non-ShuffledBF に着目する．これを ShuffledBF と組み合わせることによって，両者の特徴を生かした処理が期待される．これを Semi-ShuffledBF と呼ぶ．性能測定では，スキーマ情報や検索条件等を隠蔽した上で単一サーバ環境にて問合せ処理時間を調査し，手法の有効性を検証する．

[1] Amazon 社
[2] Windows 社
[3] Google 社

本稿の構成は以下の通りである．2 節で先行研究である ShuffledBF，3 節で今回提案する Semi-ShuffledBF について述べる．4 節で性能測定の結果を示し考察を行う．そして 5 節で関連研究，6 節でまとめと今後の課題について述べる．

2. ShuffledBF

ShuffledBF はブルームフィルタを用いたプライバシ保護検索手法である．ShuffledBF は単一のリレーションに対する selection-projection 演算が可能であり，selection では文字列の完全一致，部分一致，数値属性の比較演算が可能である．また，プライバシ保護の安全性が高く，サーバに保存するデータや発行されるクエリから元データのスキーマ情報や値の分布などを隠蔽することができる．

2．1 項で検索索引の生成，2．2 項で問合せ，2．3 項で数値属性の変換方法について述べる．

2.1 検索索引の生成

本項では，テーブルの変換方法，タプルの暗号化，問合せの変換方法について述べる．本手法では検索用の索引にブルームフィルタを用いる．ブルームフィルタとは，集合にある要素が含まれるかどうかを高速に判定するための索引であり，空間効率がよく検索が高速，OR 演算が可能，偽陽性（false positive）を持つといった特徴がある．テーブルの変換例を図 2 に示す．

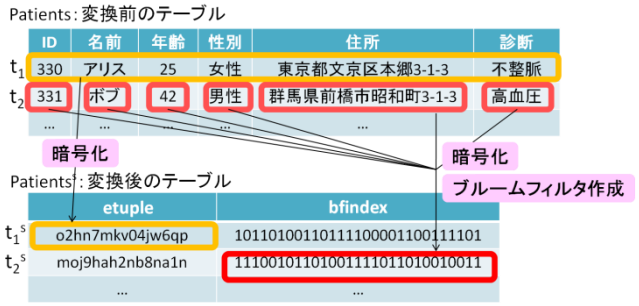

図 2 ： テーブル変換例

テーブル Patients は ID，名前，年齢，性別，住所，診断の属性で構成されている．サーバには Patients に対応するテーブル Patients^s が用意される．このテーブルには etuple と bfindex という二つの属性しかない．属性 etuple には，タプルを暗号化した値が格納される．属性 bfindex はタプルに関する検索用索引である．ShuffledBF の検索用索引である bfindex は元テーブルのスキーマにかかわらず各タプルにつき一つしか構築されないため，サーバ上のテーブル Patients^s から変換前のテーブル Patients がどのような属性を持つかを推測することは困難である．それにより各属性の値の分布を分かりにくくなり，統計情報への攻撃を防ぐことができる．

タプル t_1 からブルームフィルタ索引（ShuffledBF）を生成する流れを以下の図 3 に示す．

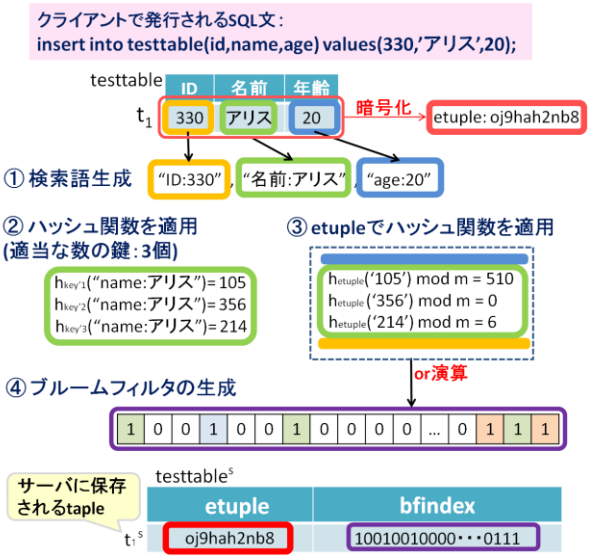

図 3 ： ブルームフィルタ索引の生成

bfindex は，ブルームフィルタによる索引である．これは，タプルの属性名と属性値を元に以下のように構築される．例えば，図 3 にあるタプル t_1 の属性“名前”の値“アリス”に対応する語は“名前：アリス”となる（図 3 中①）．このようにして作られた各語に対して，複数のハッシュ関数を適用する．図 3 中②は語“名前：アリス”に対して三つのハッシュ関数を適用した例である．ハッシュ関数には HMAC（かぎ付きハッシュによるメッセージ認証関数）を用い，必要なハッシュ関数の数だけ鍵を用いる．図 3 中②では key1，key2，key3 の三つの鍵を使っている．次にこれらのハッシュ値に対して etuple を鍵として再び HMAC を適用する．これにより，複数のタプルが同じ値を持っていた場合も 2 回目のハッシュ関数適用により異なる場所にビットが立ち，攻撃者がビットパターンから元データの特徴を推測することができなくなる．なお etuple によるハッシュ関数の適用を行わず，一度目のハッシュ関数の結果を用いて生成するブルームフィルタを我々は Non-ShuffledBF（NSBF）と呼ぶことにする．

2.2 問合せ

次に問合せの変換について述べる．問合せの変換例を図 4 に示す．図 4 の上部の SQL はクライアントで入力された SQL 文，下部はサーバに発行される SQL 文である．サーバに発行される問合せ文は，各タプルの属性に対する検索条件やテキスト検索が索引

bfindex に対する条件に置き換えられている．そのため，データベース管理者はどの属性値にどのような検索条件を指定したか読み取ることができない．

次に，サーバでの問合せ処理の手順を図 5 に示す．

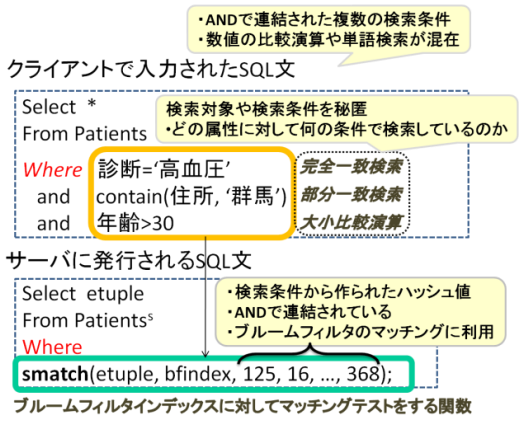

図 4 ： 問合せ変換例

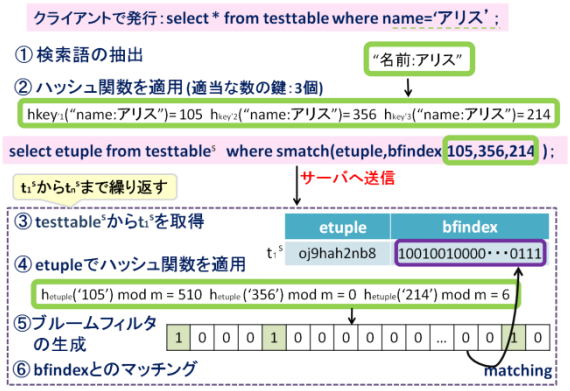

図 5 ： 問合せ実行例

まずクライアントで検索条件文から検索用の語を生成し(図 5 中①),1 回目のハッシュ関数を適用し，図 5 中②に示す SQL 文によりサーバに送る．サーバ側ではタプル毎に SQL 文の Where 節に書かれている smatch 関数が実行され,各タプルが問合せ条件に合致するかチェックされる.smatch 関数ではクライアントで生成されたハッシュ値(図 5 中では 105,356,214)に対して etuple を鍵にしてハッシュ関数を適用する（図 5 中④）．その値でブルームフィルタへのマッチングを行う（図 5 中⑤）．このようにして，テーブルのスキーマ情報や検索条件に数や演算の種類を隠蔽することができる．

2.3 数値属性の変換方法

ブルームフィルタを用いて数値の大小比較をすることはできないので，ブルームフィルタ索引を数値に適用するためにはデータ中の数値を語に変換する必要がある．基本的には数値属性のドメインを複数のバケットに分割し，該当するバケットの名前を属性名と合わせて語とする．

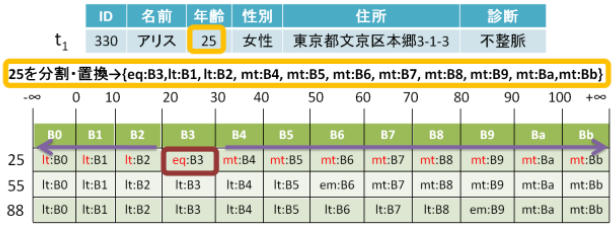

図6 ： 数値属性のバケットによる表現

図 6 は 25，55，88 とそれぞれに対する語の集合を現したものである．全てのバケット B＝B1，…，Bb に対して,バケットの上限が値 v より小さい場合 「<属性名>:lt:<バケット名>」,バケットの下限が値 v より大きい場合「<属性名>:mt:<バケット名>」,それ以外の場合は「<属性名>:eq:<バケット名>」という語を追加する．これを用いて値の大小を比較する場合，例えば 75 より大きな値を調べたい場合は,75 が含まれるバケット(B8)の左隣(B7)に注目し,「<属性名>:lt:B7」という語を持つタプルを探せば良い．逆に 35 より小さい値を探す場合には，右隣のバケット（B5）に注目し，「<属性名>:mt:B5」という語を持つタプルを探す．このようにして，数値の比較演算を文字列のマッチングと同様に扱う．

3. Semi-ShuffledBF

ShuffledBF では，サーバ上にあるブルームフィルタから元データの推測が困難であるため安全性は高い．しかしながら，問合せの際に各タプルに対してハッシュ関数を適用しなければならず，処理時間が長くなるという問題がある．そこで我々は，ShuffledBF と，文字列に変換関数を適用しないブルームフィルタ（Non-ShuffledBF）を組み合わせることで安全性を損なわず高速にプライバシ保護検索ができる Semi-ShuffledBF を提案する．

3．1 項で基本的な考え方，3．2 項でデータの挿入方法，3．3 項で問合せの方法について述べる．

3.1 基本的な考え方

Semi-ShuffledBF の基本的なアイデアは，ハッシュ関数を適用するタプル数を絞り込むため，ShuffledBF に加えて,Non-ShuffledBF による索引を付与するというものである.問合せ時には,まず Non-ShuffledBF 索引による絞り込みの後 ShuffledBF 索引による絞り込みを行うことで,ShuffledBF 索引によるハッシュ関数の適用回数を減らすことが期待される．Non-ShuffledBF 索引では以下のハッシュ関数を適用する．

$$h'_i(x) = \left\lceil \left| g_j\left(h_i(x) \bmod \lceil m/l \rceil \right) \right| \right\rceil \bmod m \quad \cdots \quad (1)$$

ここで m はブルームフィルタのビット長，l は 2 以上の整数をとるパラメタとし，関数 g_j は元テーブル $R(A_1, \cdots, A_n)$ における属性 A_j 毎に設定する．ShuffledBF ではハッシュ値に対してブルームフィルタのビット長である m の剰余を求めていたのに対し，Non-ShuffledBF では $\lceil m/l \rceil$ の剰余を求めている．l の値を大きくすることで Non-ShuffledBF による擬陽性をあえて高くし，ブルームフィルタによる元データの推測が難しくしている．関数 g_j は，属性 A_j ごとにビットの立つ位置を定めるための関数で，単一のブルームフィルタ内でのビットを立てる位置の重複を防ぐ．また，Non-ShuffledBF と ShuffledBF を別々に用意するのではなく，これらを合成し一つのブルームフィルタとすることによって Non-ShuffledBF における元データの推測をより難しくすることができる．合成により擬陽性は高くなる可能性はあるが，これはブルームフィルタのビット長 m を少し長くすることによって調整が可能であると考えられる．

3.2 データの挿入方法

Semi-ShuffluedBF のデータの挿入方法は以下の 4 段階に分けられる．

（1）ShuffledBF の生成 （2．1 項図 3）
（2）Non-ShuffledBF の生成 （図 7 中 1～3）
（3）ShuffledBF と Non-ShuffledBF とを OR 演算 （図 7 中4）
（4） 結果を bfindex に保存

以下に Semi-ShuffledBF の生成例を示す．

m = 512， l = 16 とし， 属性 A_1：ID，A_2：name，A_3：age に対して A_1：$g_1(x)$ = $2x+1$，A_2：$g_2(x)$ = $-2x+9$，A_3：$g_3(x)$ = $3x+2$ を用意する．

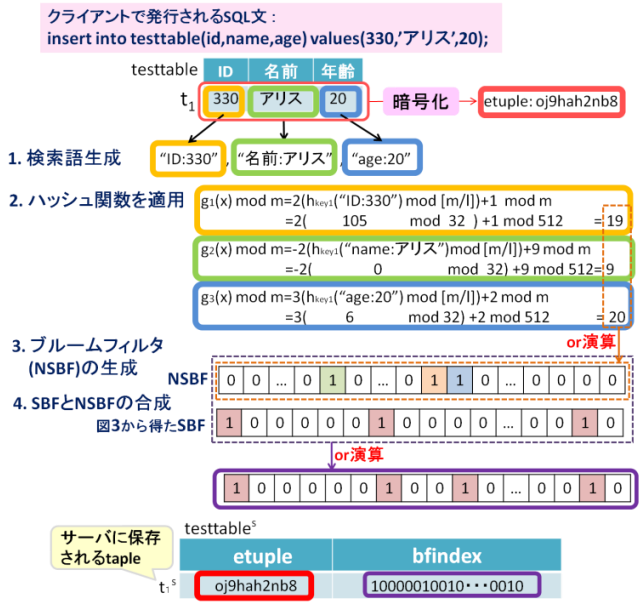

図 7 ：Semi-ShuffledBFの生成例

まず ShuffledBF と同様に検索語を生成し，式（1）を適用する．この例では l = 16 としているため，ハッシュ値に対して 32（ = 512÷16 ）の剰余を求めている．図 7 中の例では，検索語”ID：330”のハッシュ値 h_{key1}（“ID ： 330”）= 105 に対し 32 の剰余を求め $g_1(x)$ = 2x+1 を適用した結果 19 という値を得ている．関数 $g_1(x)$ により，属性 ID の値に対するブルームフィルタ上のビット位置は $\lceil 2x+1 \rceil$ 上に配置されることになる．こうして生成された Non-ShuffledBF と ShuffledBF とを OR 演算した結果を Semi-ShuffledBF として bfindex に保存する．

3．3 問合せの方法

Semi-ShuffledBF の問合せの方法は以下の 2 段階で行われる．

（1）Non-ShuffledBF で検索
（2）ShuffledBF で検索

まず Non-ShuffledBF を各タプルに適用し，問合せ条件に該当すると判断したもののみ ShuffledBF を適用する．

Semi-ShuffledBF の問合せ手法を示す．

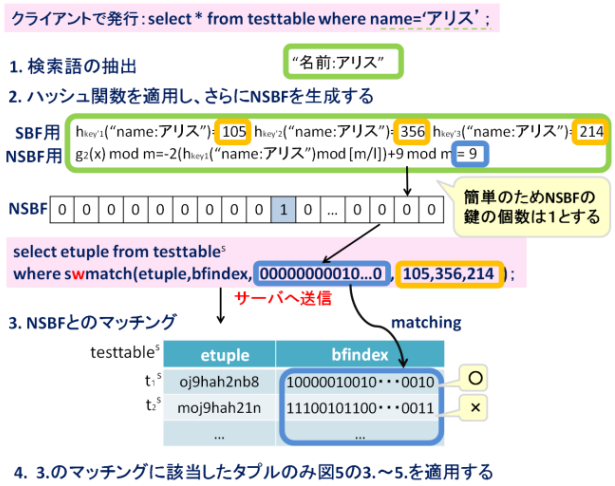

図8 ： 問合せ実行例

まずクライアントで検索条件文から検索用の語を生成し，Non-ShuffledBF 用ハッシュ関数，ShuffledBF 用ハッシュ関数を各々適用しサーバに送る．サーバ側ではまず Non-ShuffledBF 用ハッシュ関数から得られた値（NSBF）でマッチングを行い，そこから得られたものにだけ ShuffledBF 用ハッシュ関数から得られたブルームフィルタ索引タプル毎に etuple を鍵にしてハッシュ関数を適用し，その値（SBF）でブルームフィルタへのマッチングを行う．

このように Non-ShuffledBF でのマッチング結果だけにハッシュ関数を適用することで，サーバ側でハッシュ関数を適応する回数を減らすことができる．さ

らに，テーブルのスキーマ情報，検索条件，演算の種類の隠蔽をしつつ，処理速度も向上させることができる．

4. 性能評価

提案手法のための予備実験として，先行研究で構築されたプライバシ保護検索システムのプログラムの処理効率を改善,DBMS として機能するよう機能を拡張し性能測定を行った．

4．1 実験環境

実験環境として，Linuxサーバ（CPU：Intel（R）Xeon（R）2.00GHz　メモリ：8GB）を用い，またデータベースサーバには PostgresSQL を利用して性能測定を行った.実験データは 100,000 タプルの人工データを用い，ブルームフィルタの長さ m：128byte，関数 g_j：一次関数，分割数 I：10 に設定した．

本実験ではサーバ上での問合せ処理時間のみを測定した．プライバシ保護検索では，サーバでの検索で該当したタプルはクライアントにダウンロードされた後復号化され，復号化されたデータに対して再び問合せを行った上で結果を得ることができる．実際は，クライアントへのデータのダウンロード時間や復号にかかる時間が非常に大きくなることがあるが，提案手法の範囲外であるため，今回の実験では考慮に入れていない．

4．2 実験結果

ShuffledBF，Non-ShuffledBF，Semi-ShuffledBF の実験サーバでの性能測定結果は以下の通りである．

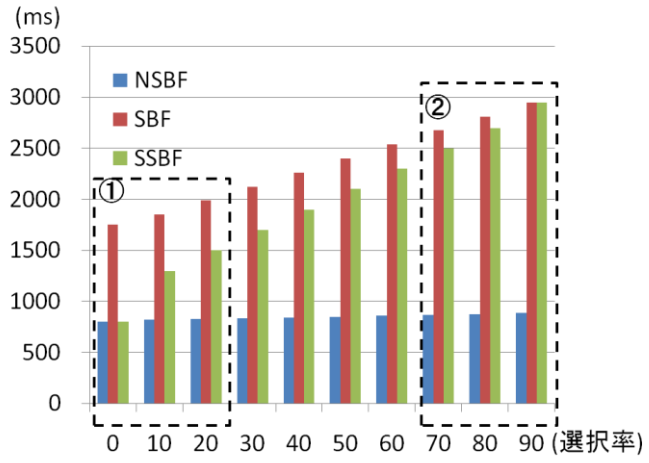

図10 ： 性能測定結果

選択率が低いケース（図 10 中①）では，Semi-ShuffledBF を用いることで処理効率を改善できた．

しかしながら，選択率が高いケース（図 10 中②）では Semi-ShuffledBF の処理効率はあまり改善率が高くないことがわかった．

4．3 考察

実験において ShuffledBF と比較して本提案手法が大幅な改善に至っていない理由は 2 点考えられる．

（1）Non-ShuffledBF の処理がそれほど速くない

Non-ShuffledBF では基本的にビット演算しか行わないため処理時間がそれほどかからないことを期待していたが，実際は 700 ミリ秒近くかかっている．これは全タプルに関してチェックをしなければいけないため,テーブル走査が必要になることが原因として考えられる．

（2）正解であるタプルに対して処理数が変わらない

本提案手法は問合せの条件に当てはまらないタプルに対して処理を軽減しているが，選択率が高い場合はその効果はほとんど得られないために前節のような結果になっていると言える．

上記 2 点の問題に対してそれぞれ以下の改善案が考えられる．

(1)Semi-shuffledBF にビットマップ索引を付与する

Non-ShuffledBF による一次検索を高速にするため，ビットマップ索引に対してビットマップ索引を付与することが考えられる．これにより，全タプルへのファイルスキャンをせずにビットが立つ位置への直接的なアクセスができるためディスク I/O のコストが軽減されることが期待できる．

（2）ShuffledBF による検索を行わない

選択率が高い場合，ほとんどのタプルに対して ShuffledBF による検索を行ってしまう．そこで Non-ShuffledBF による検索を行ったあとに選択率を計算し，選択率が高い場合には ShuffledBF による検索を行わずに該当するタプルを正解とする方法が考えられる．これによって検索の擬陽性は高まるが，4．1節に述べたようなクライアント側での工夫を行うことによって対応できると考えられる．

5．関連研究

アウトソーシングデータベースに対するプライバシ保護の研究は，Hacigumus らによる研究[6]を先駆けとして数多く行われている．Hacigumus らは検索用の索引をサーバ上の DB に格納し，ユーザが発行する問合せをサーバ上の索引に対する問合せと，クライアント側で行う問合せに分割し実行する問合せ実行プラン生成の提案を行った．検索用の索引は属性毎に用意され，属性のデータ型や問合せ条件で用いられる演算によって索引の生成方法を使い分けている．索引の生成

法では元の値が同じであると索引の値も一致するため，その値の分散から元の値が推測される可能性があった．これに対して Hore らは分散による値の推測を困難にするバケット分割法を提案している[8]．Agrawal ら[1]は,順序関係を保存した数値属性の変換方法を提案している．変換した値の分散が元の値の分散と異なるように変換することによって元の値の推測を防ぐ．属性の比較演算や結合などが可能である．ただし，順序関係が保存されているため一部のデータが漏洩した場合に他の値がなし崩しに判明するという問題があり，Lee ら[9]，Hasan ら[7]などにより改良手法が対案されている．また Mykletun ら[5],Ge ら[11]による準同型性を持つ暗号化手法を利用した集約演算や k-近傍検索[15]なども提案されている．これらの研究は本研究と同様に安全性及び性能についての問題が存在する．セル毎に索引を作成するため，サーバに多くのデータがおかれた場合に索引の値の傾向を分析して元の値が推測される可能性がある．またタプル毎に条件を確認する場合は索引が使用できないという問題も起こる．また索引を作成した場合でもその索引から元の値が推測される可能性もある．我々の提案手法では索引をタプルにつき１つにまとめるため，セル毎に索引を生成する手法と比べて推測の可能性は低い．また今回提案した Semi-shuffledBF ではブルームフィルタのビットマップ索引を適用することにより性能向上が期待でき，一部タプル毎にシャッフルされたビットが含まれているため索引によって元の値が推測される可能性も低いと考えられる．

6. まとめと今後の課題

本稿では先行研究での提案手法（ShuffledBF）に文字列に変換関数を適用せずに生成されたブルームフィルタである Non-ShuffledBF を組み合わせることで安全性を損なわず高速にプライバシ保護検索ができる Semi-ShuffledBF を提案した．

今後は Semi-ShuffledBF の高速化や性能と安全性に関する指標を定めること．また、実際に Windows SQL Azure などの DaaS サービス環境上での性能測定を行っていきたい．

謝辞

本研究の一部は科学研究費補助金若手研究（B）（課題番号：21700099）によるものである．

参 考 文 献

[1] Agrawal R., Kiernan J., Srikant R. and Xu Y.: Order preserving encryption for numerical data, Proceedings of the 2004 SIGMOD International Conference, pp.563 - 574 (2004).

[2] 新井裕子，渡辺知恵美：データベースアウトソーシングにおけるプライバシ保護に考慮した範囲検索法，電子情報通信学会 第 19 回データ工学ワークショップ，C1-1（2008）.

[3] Bellovin S. and Cheswick W. ：Privacy-enhanced searches using encrypted bloom filters” （2004） .

[4] Boneh D, Crescenzo G.D., Ostrovsky R. and Persiano G,Public Key Encryption with Keyword Search,Proceedings of EUROCRYPT ’04, vol.3027 LNCS, pp.506-522 （2004）.

[5] E. Mykletun, G. Tsudik：Aggregation queries in the database-as-a-service model. IFIP WG 11.3 on Data and Application Security （2006） .

[6] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra.：“Executing SQL over Encrypted Data in the Database-Service-Provider Model , ”Proceeding of the ACM SIGMOD International Conference on Management of Data, pp. 216-227, （2002） .

[7] Hasan Kadhem, Toshiyuki Amagasa, and Hiroyuki Kitagawa : ″A Secure and Efficient Order Preserving Encryption Scheme for Relational Databases,″ Int’l Conf. on Knowledge Management and Information Sharing (KMIS 2010), Valencia, Spain, October 25-28 (2010) .

[8] Hore B, Mehrotra S. and Tsudik G. : A privacy-preserving index for range queries, Proceedings of the 30th International Conference on Very Large Data Bases, pp.720-731 (2004) .

[9] S. Lee, T. Paek, D. Lee, T. Nam and S. Kim：Chaotic Order Preserving Encryption for Efficient and Secure Queries on Databases, IEICE Transactions on Information and Systems E92.D (11) , 2207-2217 (2009) .

[10]Ting Yu and Shushil Jajodia：Secure Data Management in Decentralized Systems, Springer-Verlag NewYork Inc, p.462 (2006) .

[11]Tingjian Ge, Stanley B. Zdonik：Answering Aggregation Queries in a Secure System Model. ,Proceedings of VLDB 2007, pp.519-530 （2007） .

[12]W.K.Wong, D.W.Cheung, B.Kao and N. Mamoulis：Secure kNN computation on encrypted databases, Proceedings of the 35th VLDB Conference, pp.139-152 (2009) .

[13]渡辺知恵美， 新井裕子：DAS におけるスキーマ情報と複合的な検索条件を隠ぺいしたプライバシ保護検索手法，情報処理学会研究会報告 ,No.2008-DBS-146, Vol.2008, No.88, pp.163-168 (2008) .

[14]Watanabe C. and Arai Y.：Privacy-Preserving Queries for a DAS model using Two-Phase Encrypted Bloomfilter, Proc. of International Conference on Database Systems for Advanced Applications （2009） .