# 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　(指導教員：渡辺知恵美)

## 1　はじめに

DaaS サービスでは，ユーザはインターネット上の外部の第三者が管理するデータベースの機能をネットワーク経由でサービスとして利用することができる．

このような環境でユーザがデータ管理者から機密情報を守るための手段として，プライバシ保護検索という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手法がこれまで多く研究されており，我々もまた先行研究 [1] に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．

先行研究では，各 taple に対して問い合わせ用のブルームフィルタを生成し，taple 毎のキーを用いてシャッフルする (ShuffledBF) ことにより，ブルームフィルタのビットパターンの漏えいを防いでいたが，この手法では，問い合わせの際，各 taple でハッシュ関数を適応するので，安全であるが時間がかかってしまう．逆に，ブルームフィルタのシャッフルを行わない (Non-ShuffledBF) と，ビットパターンの推測が可能である．

そこで，問い合わせの際 1 段階の絞り込みに Non-ShuffledBF をもちい，2 段階の絞り込みに ShuffledBF を用いる手法 (Semi-ShuffledBF) を提案する．

## 2　ShuffledBF

ShuffledBF(SBF) はブルームフィルタを用いることで属性毎ではなくタプル毎に索引を構成したプライバシ保護検索手法である．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検索するための索引で，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．

### 2.1　検索索引の生成

先行研究 [1] にて提案した手法におけるテーブル変換とブルームフィルタ索引の生成例を図 1 に示す．

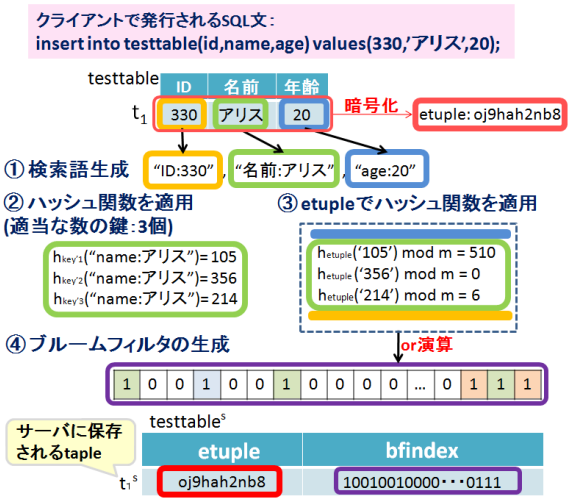


図 1: 変換前後のテーブルとブルームフィルタ索引生成

**♦テーブル変換**

クライアント側のテーブル *testtable* に対して，サーバ側には etuple と bfindex という 2 つの属性で構成されているテーブル $testtable^s$ が用意される．etuple には，タプル全体を暗号化した値が格納され，bfindex はタプルに関する検索用索引である．

**♦ブルームフィルタ索引の生成**

bfindex を生成するための語は属性名と属性値からなっており (①)，まずそれらに対して複数のハッシュ関数を適用する (②)．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．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 (③)．

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っている．

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 NonShuffledBF(NSBF) と呼ぶことにする．

### 2.2　問合せ

問合せ変換とサーバでの問合せ処理を図 2 に示す．

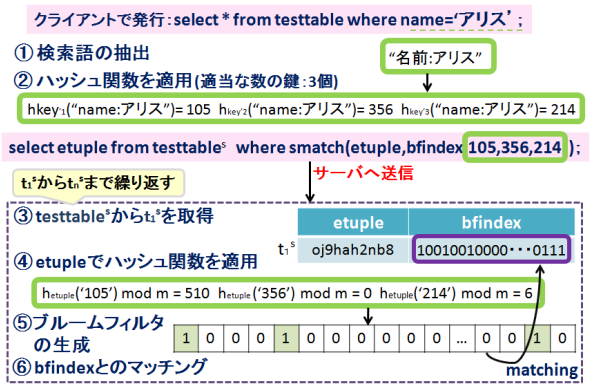


図 2: 変換前後の SQL と問合せ処理手順

**♦問合せ変換**

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が索引 bfindex に対する条件に置き換えられているため，データベース管理者は何の属性値を用いて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．

**♦問合せ処理**

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 (①)，1 回目のハッシュ関数を適用し (②) た SQL 文をサーバに送る．サーバ側ではタプル毎に 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に対して etuple を鍵にしてハッシュ関数を適用し (④)，その値でブルームフィルタへのマッチングを行う (⑤)．

## 3 Semi-ShuffledBF

### 3.1 基本的な考え方

Semi-ShuffledBF では ShuffledBF による索引に加えて，NonShuffledBF による索引を付与することで ShuffledBF 索引によるハッシュ適用回数を減らし処理速度を向上させる．

NonShuffledBF 索引では以下のハッシュ関数を適用する．

$h_i'(x) = \lceil |g_j(h_i(x) \bmod \lceil m/l \rceil)| \rceil \bmod m \cdots (1)$

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，NonShuffledBF 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 NonShuffledBF 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，NonShuffledBF 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 NonShuffledBF 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 3 段階に分けられる．

(1)NonShuffledBF の生成 (図 3：1〜3)

(2)ShuffledBF の生成 (図 1)

(3)ShuffledBF と NonShuffledBF とを合成 (図 3：4)

した結果を Semi-ShuffledBF として bfindex に保存

以下図 3 に Semi-ShuffledBF の生成例を示す．

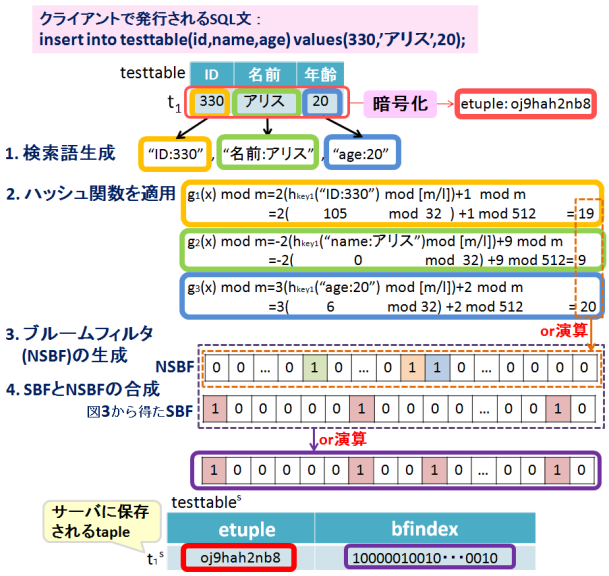


図 3: Semi-ShuffledBF の生成例

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

まず ShuffledBF と同様に検索語を生成し，式 (1) を適用し，ブルームフィルタを生成する．次に ShuffledBF 生成 (2．1 項参照)．そして NonShuffledBF と ShuffledBF とを合成した結果を 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，

(1)NonShuffledBF で検索

し，問合せ条件に該当すると判断したもののみ

(2)ShuffledBF で検索

する，という 2 段階で行われる．

以下図 4 に Semi-ShuffledBF の問合せ手法を示す．

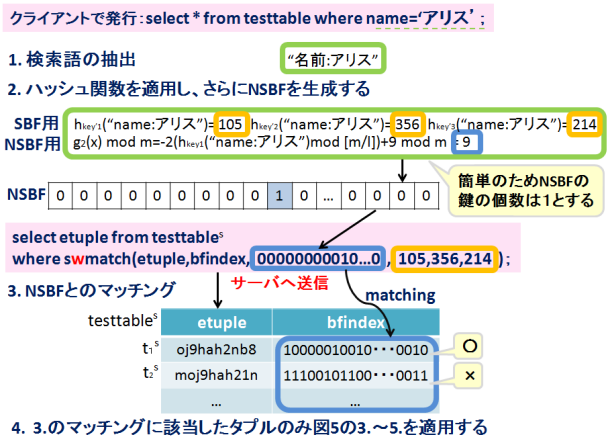


図 4: 問合せ実行例

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，NonShuffledBF 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 NonShuffledBF 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．

2 段階の絞り込みにすることでサーバ側でハッシュ関数の適応回数を減らすことができ，すべてのタプルにテーブルのスキーマ情報や検索条件に数や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 まとめと今後の課題

本稿では先行研究の提案手法 (ShuffledBF) に変換関数を適用せずに生成されたブルームフィルタ (NonShuffledBF) 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．

今後は性能測定を実施し，Semi-ShuffledBF における更なる安全性と性能のトレードオフを見極めて行きたい．

## 参考文献

[1] Watanabe C. and Arai Y.: "Privacy-Preserving Queries for a DAS model using Two-Phase Encrypted Bloomfilter," *International Conference on Database Systems for Advanced Applications*, pp.491-495 (2009)