索引を用いた複数演算に対応した安全で効率的な秘匿検索手法

秋山　賢人† 　渡辺知恵美†† 　北川　博之††

† 筑波大学情報学群情報科学類　〒 305–8573 つくば市天王台 1–1–1
†† 筑波大学システム情報系　〒 305–8573 つくば市天王台 1–1–1
E-mail: †k_akiyama@kde.cs.tsukuba.ac.jp, ††{chiemi,kitagawa}@cs.tsukuba.ac.jp

あらまし　近年普及しつつある DBaaS ではクラウドサーバ管理者によるデータ，クエリの漏洩を防ぐためにプライバシ保護が必要である．先行研究である篠塚らの手法では，データとクエリのプライバシを守りつつ高速な検索処理を可能にする OSIT-bs フレームワークを提案している．OSIT-bs は検索用の索引をサーバとクライアント間で分散管理し，クライアントがサーバとの通信により索引を探索する手法であり，１つのクエリにつき１つの索引を探索して完全一致検索と範囲検索を行うことができる．しかし，この手法では１つのクエリで複数の索引探索が必要になった場合に，各索引での探索結果、つまり中間結果をクライアント上で明らかにしなければならずプライバシ要求が満たされないという問題があった．そこで本研究では，複数の索引探索を必要とするクエリのうち連言節で結ばれた二つの条件を含むクエリ文を対象とし，中間結果をクライアントに明かさずに問合せを行う手法を提案する．本手法では Wong らの手法にある鍵更新演算を利用し，２つの索引探索結果である暗号化されたレコード ID 集合の積集合を暗号化したままサーバ上で行い，その結果をサーバに知らせることなくクライアントに渡す．これによりクライアントに中間結果を与えることなく複数条件による検索を処理することができる．

キーワード　プライバシ保護，秘密計算，Database as a Service

1. はじめに

近年，Database as a Service（DBaaS）を利用したデータの検索サービスが普及しつつある．DBaaS とはネットワークを経由してデータベースの機能を提供するクラウドサービスである．DBaaS を利用したサービスの提供例としては，Amazon RDS [1]，Oracle Enterprise Manager 12c [2]，Google Cloud Bigtable [3] などがある．DBaaS を用いる利点は，DBaaS の利用者がサーバの購入や DBMS の構築を行うことなく DBaaS のサービスプロバイダにサーバの稼働及びデータベース管理を委託することで利用者側の負担の軽減を行うことができる点，データへの高速なアクセスが保障される点，データ量に応じて柔軟にデータベースの拡張を行える点などが挙げられる．

DBaaS は手間のかかるタスクを委託できる一方で，機密データやクエリの漏洩が起こりうるという問題がある．機密データやクエリの漏洩を防ぐためには，下記にあるデータとクエリに関するプライバシの要求を満たさなければならない．

データプライバシに関する要求
・データ所有者はサーバ管理者に対し，保存するデータの中身を知られたくない
・データ所有者はクライアントに対し，必要以上の情報を与えたくない

クエリプライバシに関する要求
・クライアントはデータ所有者及びサーバ管理者に対し，自身が発行するクエリの履歴を知られたくない

このような要求に対応するため，近年は暗号化データベースシステムに関する研究が盛んに行われている．暗号化データベースシステムではデータを暗号化した状態で DBaaS にて管理し，検索可能暗号スキームを用いることによってデータを暗号化したまま問合せ処理を行う．暗号化データベースに関する研究として，CryptDB [4]，SDB [5] などが提案されている．

篠塚らは [6] において，機密性を守りつつ高速な検索処理を可能にする検索可能暗号フレームワークである OSIT-bs（Oblivious Secure Index Traversal-binary search）フレームワークを提案した．

このフレームワークでは索引を暗号化したままクライアントとサーバに分割して配置し，クライアント自身が索引を探索して該当するレコード集合を求めるところが特徴である．索引はサーバとクライアントに分割されて配置されており，紛失通信を使って探索を行う．これによりサーバ及びクライアントが完全な索引の情報を知らないまま探索を実現することができる．また索引の探索に秘匿計算を用いることで，データや索引を秘匿したまま問合せができるほかクエリの内容自体も秘匿することが可能であり，データ所有者及びクライアントの個人情報を秘匿できる．

しかしこの研究では，クライアントは一つのクエリを評価するために一つの索引を探索することを想定している．つまり「SELECT * FROM R WHERE age > 40」のような一つの属性を検索条件とした範囲検索または完全一致検索にのみ対応していた．そのため「SELECT * FROM R WHERE age > 40 AND salary < 200,000」というように連言節を含んだ複数の条件を処理する場合，「SELECT * FROM R WHERE age > 40」「SELECT * FROM R WHERE salary < 200,000」という別々の条件をクライアントで評価し，それぞれの結果集合の

論理積を求めることになる．これはクライアントにとって負荷が大きくなるだけでなく，問合せの途中結果がクライアントに漏洩するという安全性の問題も存在する．

そこで，本研究では先行研究をもとに，連言節で接続された複数の条件を含む問合せに対し，問合せの中間結果をクライアントに知られずに求める手法を提案する．本研究を実現するために，Wong らの手法 [5] で導入されている鍵更新演算を利用し，属性ごとに異なる鍵で暗号化されたレコード ID を比較してクライアントに中間結果を与えることなく複数条件による検索を行う．

本稿の構成は以下のようになっている．まず，2 節で関連研究について述べ，3 節で本研究に必要な前提知識について述べる．次に，4 節で先行研究についての紹介を行う．続いて，5 節で本研究の提案手法について述べ，6 節で提案手法を評価する実験について述べる．そして，7 節でまとめと今後の課題について述べる．

2. 関 連 研 究

DBaaS 環境における暗号化データベースシステムは，Hacigumus [7] らによって提案されたのを先駆けにこれまで数多くの研究が取り組まれている．Hacigumus らはサーバ上で一部検索可能な安全性の低いデータをおき，サーバで一部を検索しクライアントで精査するという仕組みを提案しその問合せ処理モデルを定義した．Hore [8] らは，Hacigumus [7] らの手法をベースに統計による値の推測を困難にするサーバ側データの生成法を提案した．また，Agrawal [9] は数値属性の順序関係を保存できる検索用索引の生成方法 (OPE) を提案した．この手法では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指摘されており，Lee ら [10] や Hasan ら [11] などにより改良手法が提案されている．また Mykletun ら [12] や Ge ら [13] によって準同型暗号を利用した集約演算を可能にするサーバ側でのデータ暗号化手法や，k-近傍探索への対応なども提案されている．2011 年にはこれらの提案手法を組合せて実装した CryptDB というオープンソースパッケージが Popa ら [4] により公開され，暗号化データベースシステムは大きな注目を集めた．CryptDB では，RND，DET，Paillier 暗号 [15] などの異なる暗号化スキームを用いて機密データを幾重にも暗号化することで，機密データの漏洩を防ぐという手法がとられている．また，各暗号化スキームごとに異なるクエリの処理を行うことでキーワード検索，結合演算，大小比較などの様々な演算をサポートしている．また Stephen らにより CryptDB の検索処理最適化機構が改良された Monomi [14] も発表された．

これらの暗号化データベースシステムで用いられている手法は，特に範囲検索や結合処理を行う際に OPE を使うことが多かったが，上記の指摘にもある通り現在この手法は安全性の問題が指摘されている．そのため近年では準同型暗号を用いた検索手法や検索可能暗号，紛失通信等を利用した手法が提案されつつある．準同型暗号は Paillier 暗号 [15] などのように加法準同型性を持つものと ElGamal [16] 暗号のように乗法準同型性を持つものがある．また Gentry らにより整数上完全準同型暗号が提案 [17] されたが，この手法はビットごとの演算が必要でありデータベースに用いるにはまだ多くの改良が必要とされる [18]．また，PEKS [19] は ID ベース暗号を元にした暗号スキームであり，キーワード検索が可能である．

これらの手法は非常に安全性は高いが暗号化データベースとして大量のレコードに対して検索を行う場合には十分なパフォーマンスを満たすことができない．これらの暗号化スキームを用いた場合，レコード一つ一つに対して検索条件に該当するか否かをチェックする必要があり，データの量に対して検索時間がかかってしまうからである．これに対し，Hu らの手法はデータベース索引を用いた安全な検索フレームワーク OIT [20] を提案した．この手法では索引をクライアントが所有しクライアント自身が索引を探索することで安全性を確保しているが，クライアントに対するプライバシ要件が侵害されると，クライアントに大きな負荷がかかることから，篠塚らはサーバとクライアントで索引を分散配置した OSIT フレームワーク [6] を提案している．

また，暗号化データベースにおいては，関係代数のようにクエリを構成する各演算の間に相互運用性（interoperability）を持たせることも課題である．SDB [5] では相互運用性のある加法準同型性をもった暗号スキームを提案し，それを用いた暗号化データベースシステムを実装している．この手法により，サーバ側で中間結果を一度クライアントに転送して処理をしたり，サーバ側で一部のデータを復号することなく，暗号化したまま演算の出力結果を次の演算の入力とすることができる．

3. 基 本 事 項

本節では本研究を説明する上で必要な知識について述べる．

3.1 準同型暗号

準同型暗号とは，2 つの暗号文が与えられた時に平文や秘密鍵の値を知らなくても，暗号化したまま平文の加算や乗算を行える準同型性を持つ暗号のことである．準同型暗号の暗号化関数を $E(\cdot)$，2 つの平文を m_1，m_2 とした時に以下の式 (1) が成り立つ．ただし，$\oplus$ や $\otimes$ は加法演算子や乗法演算子のような演算子とする．

$$E(m_1 \oplus m_2) = E(m_1) \otimes E(m_2). \tag{1}$$

準同型暗号には RSA 暗号，ElGamal 暗号，Paillier 暗号 [15] などがあり，本稿では Paillier 暗号を使用している．Paillier 暗号の暗号化関数を $E(\cdot)$，平文を m，公開鍵を $pk = (n, g)$，r を n 以下のランダムな値とすると Paillier 暗号の暗号化は以下の式 2 で表すことができる．

$$E(m) = g^m \cdot r^n mod\ n^2. \tag{2}$$

また，Paillier 暗号は加法準同型性を持つ．以下の式 3 のように 2 つの平文 m_1，m_2 の暗号文から $m_1 + m_2$ の暗号文を計算することができる．

$$
\begin{aligned}
&E(m_1) \cdot E(m_2) \\
&\quad = (g^{m_1} \cdot {r_1}^n mod\ n^2) \cdot (g^{m_2} \cdot {r_2}^n mod\ n^2) \\
&\quad = g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n mod\ n^2 \\
&\quad = E(m_1 + m_2).
\end{aligned} \tag{3}
$$

3.2 GT-SCOT プロトコル

GT-SCOT プロトコル [21] は大小比較を行う紛失通信プロトコルであり，お互いの持つ値を知ることなく受信者のみが比較結果を知ることができるという特徴を持つ．受信者 X と送信者 Y がそれぞれ x，y という値を持っている時に大小比較を行う手順を説明する．まず，X は x を Paillier 暗号で暗号化した値 $E(x)$，$x < y$ を示す値 s_0，$x > y$ を示す値 s_1 を Y に送る．次に，Y で $E(x)$，y，s_0，s_1 を用いて $E(s_0)$ または $E(s_1)$ が得られる計算を行う．そして得られた結果を X に送信すると，X は結果を復号して s_0 または s_1 を得ることにより，x と y の大小関係を知ることができる．Y では暗号化された x の値しか知ることができず，比較結果も暗号化された状態で出てくるため大小関係も知ることができない．X も大小比較の結果のみしか知ることができない．

4. 先行研究

篠塚らはデータとクエリ双方のプライバシを保護した高速な秘匿検索フレームワークである OSIT-bs [6] を提案している．このフレームワークは，データ所有者がデータを暗号化した状態で DBaaS のサービスプロバイダに管理を委託し，クライアントはデータ所有者から検索の権限をもらって問合せを行うという状況を想定している．まず，4.1 小節で問題定義について，4.2 小節で OSIT-bs の概要について述べる．そして，4.3 小節で先行研究の問題点について述べる．

4.1 問題定義

先行研究では,「SELECT * FROM R WHERE age $>$ 40」というような一つの属性を検索条件とした範囲検索または完全一致検索を安全に行うことを想定している．安全に検索を行うために，満たすべきプライバシ要件は以下の通りである．

- サーバ管理者は暗号化されたデータの内容及びクエリ履歴から元の値を推測できない
- クライアントは発行した問合せに対する結果以外の情報を得ることができない

4.2 OSIT-bs フレームワークの概要

この手法の主要な特徴はデータ所有者がデータとともに検索用のデータベース索引を作成し、暗号化した索引をサーバとクライアントで分割して管理するところにある．索引には属性値でソートされた配列を使用する．サーバは索引のノードを暗号化した状態で所有しているが，ノード間の兄弟関係や親子関係は所有せず索引の構造を把握することはできない．また，データ所有者から検索の権限を与えられたクライアントは，索引のノード間関係に関する情報をデータ所有者から与えられる．各ノードに関してはサーバ上のアドレスのみ知らされており，ノード自体を取得することはできない．このようにサーバ・クライアント双方が完全な索引の情報を知らないまま，本フレームワークが提案する手法で索引を探索することができる．

以下に索引の探索方法について述べる．OSIT-bs においてサービスプロバイダは索引とデータを保管するデータサーバと大小比較を行う計算サーバの 2 台のサーバを提供する．索引を探索するにはノードの値とクエリ値の大小比較を繰り返し行う必要がある．索引の探索には二分探索を使用するが，毎回の検索で最初に選ばれる索引が中間値であり，以降の探索による索引構造の漏洩を防ぐために探索範囲の中間値にランダム値 s を加えて摂動化したものを使用している．ここではサーバ上のあるノードの値 k とクライアントで発行されたクエリ値 q の比較を行う手順について述べる．なおサーバ上のノードには Paillier 暗号 $E(\cdot)$ にて暗号化された値 $E(k)$ が保存されており，そのノードのサーバ上のアドレスを $h(i)$，索引のサイズを n とする．索引の探索手順を図 1 及び Algorithm 1 に示す．探索が開始されると，クライアントは n に初期値である索引のノードサイズ N を代入する (Step.0)．まず，クライアントは r を Paillier 暗号で暗号化した $E(r)$，データサーバ上のアドレス $h(i)$ を求めてデータサーバに送る (Step.1)．次に，クライアントはクエリ値 q にランダム値 r を加えた q' を求めて計算サーバに送り，$h(i)$ にあるノードの値 $E(k)$ に $E(r)$ を乗算して $E(k) \cdot E(r) = E(k + r)$ を計算サーバに送るよう要求する (Step.2)．続いて，計算サーバは $E(k')$ と q' の大小比較をしてその結果をクライアントに返す (Step.3)．クライアントは計算サーバから受け取った結果を復号し (Step.4)，n を更新する (Step.5)．なお，大小比較は $E(k')$ の値を暗号化したまま比較結果を求め，その結果もサーバに知られないようにしなければならない．そのため比較演算では GT-SCOT プロトコル [21] を利用している．以上の (Step.1) から (Step.5) を繰り返して索引を探索する．

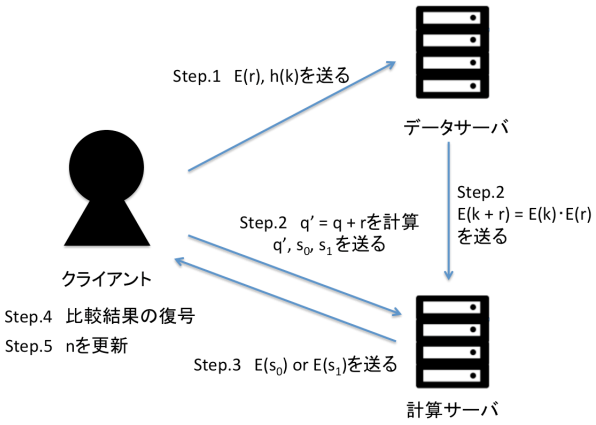

図 1 索引の探索

なお，データサーバで保存されているリーフノードにはノードに該当するレコードのアドレスリストが暗号化された形で保存されている．クライアントは索引をリーフノードまでたどった後，該当するレコードのアドレスリストを入手し，サーバ上のデータからレコードを取得する．

Algorithm 1 先行研究における索引の探索

Procedure:

1: $n \leftarrow N$ // Step.0
2: **while** the search is not terminated **do**
3: client sends $h(n/2+s)$ and $E(r)$ to data server // Step.1
4: client sends q', s_0, s_1 and data server sends $E(k+r)$ to calculation server // Step.2
5: calculation server performs GT-SCOT$(E(k'), q', s_0, s_1)$ // Step.3
6: client decrypts the result // Step.4
7: client updates n // Step.5
8: **end while**

4.3 先行研究の問題点

OSIT-bs は 1 回の問合せにつき 1 つの索引を探索することを想定している．またレコードに複数の属性が含まれている場合は、属性ごとに異なるキーで索引を作成する．この手法の問題は複数の索引を探索する必要がある問合せを安全に評価することができないことである．例えば「SELECT * FROM R WHERE age $>$ 40 AND salary $<$ 200,000」というように連言節で接続された複数の条件式を評価する場合，OSIT-bs ではクライアントが「age」と「salary」それぞれの索引に対して探索を行い，探索で得たアドレスリストを復号してサーバからレコードを入手する．クライアントは各結果のレコードから両方の条件を満たすレコードを最終結果として得る．この場合クライアントは各々の条件を満たす結果を知ることができ,「クライアントは発行した問合せの結果以上の情報を得ることができない」というプライバシ要件を満たすことができない．

5. 提 案 手 法

提案手法では 4.3 小節で課題とされていた連言節で接続された複数の条件を持つ問合せに対し，クライアントに中間結果を明かさずに問合せを評価する手法を提案する．まず，5.1 小節で提案手法における問題定義について述べる．続く 5.2 小節で提案手法の概要について述べる．

5.1 問 題 定 義

提案手法では「SELECT * FROM R WHERE age $>$ 40 AND salary $<$ 200,000 」というような連言節で接続された複数の条件を持つ問合せを想定している．問合せを安全に行うために満たすべきプライバシ要件は 4.1 節で述べた以下である．

(1) サーバ管理者は暗号化されたデータの内容及びクエリ履歴から元の値を推測できない．

(2) クライアントは発行した問合せに対する結果以外の情報を得ることができない．

特に，本研究では複数の条件を持つ問合せに対しても（2）を満たす，すなわちクライアントに中間結果を与えないということを目標としている．

5.2 提案手法の概要

本研究では連言節で接続された複数の条件を持つ問合せをクライアントに中間結果を明かさずに求めることを目標としており，目標を達成するために Wong らの手法 [5] にある鍵更新演算を利用している．提案手法の概要を説明する上で，表 1 のリレーションを用いて，条件 A「age $>$ 40」 AND 条件 B「salary $<$ 200,000」を例に考える．

まず先行研究の索引の探索アルゴリズムを利用し，各条件に対応する索引を探索する．この例の場合，前提としてレコード集合に対して属性 age の値による索引と属性 salary に対する索引がデータ所有者によってあらかじめ作成され，先行研究のフレームワークに従ってサーバとクライアントに分散配置されていることを想定する．なお，索引のエントリ暗号化の際は，属性ごとに異なる鍵を使用する．ここでは属性 age の索引に対して鍵 k_a を属性 salary の索引に対して鍵 k_b を用いたとして説明を進める．

先行研究の索引の探索手順 (Step.1) から (Step.5) を繰り返した後，クライアントが結果のレコード ID を入手する手順を図 2 に示す．条件 A，B を満たすレコード ID へのアドレスを $h_A(k_{A1}), ..., h_B(k_{B1}), ...$ とする．また，ここでは条件 A を満たすレコード ID の集合を $R_A = \{E_{k_a}(a_1), ..., E_{k_a}(a_n)\}$, 条件 B を満たすレコード ID の集合を $R_B = \{E_{k_b}(b_1), ..., E_{k_b}(b_m)\}$ とする．ここで用いている暗号プロトコルは Wong らのスキーム [5] を適用している．このスキームは加法準同型性を持つほか，暗号化したまま鍵を更新する関数が定義されている．Wong らのスキームと鍵を更新する関数については付録を参照．まず，クライアントは $h_A(k_{A1}), ...$，$h_B(k_{B1}), ...$ をデータサーバに送り (Step.a)，計算サーバに暗号化されたレコード ID の集合 R_A, R_B を送るよう要求する (Step.b)．次に，クライアントは鍵更新に必要な値 p，q を計算して計算サーバに送り，計算サーバに対し，鍵更新とレコード ID の論理積演算を要求する (Step.c)．そして，計算サーバはクライアントに演算結果を送り (Step.d)，クライアントは結果を復号してレコード ID を入手する (Step.e)．

続いて，クライアントがレコード ID を入手してから結果レコードを得るまでの手順を図 3 に示す．まず，クライアントは受け取った結果を復号して得たレコード ID をデータサーバに送る (Step.f)．次に，データサーバはクライアントに該当する暗号化されたレコードを送る (Step.g)．クライアントは各属性の鍵を利用してレコードの復号を行い，結果のレコードを入手する (Step.h)．

ここからは，サーバでの具体的な論理積の計算方法について説明する．なお，鍵 k_a から k_c への鍵更新関数をここでは $ChangeKey(E_{k_a}(a), k_a, k_c)$ とする．

提案手法のアルゴリズムを Algorithm 2 に示す．出力結果を $R_C = \{E_{k_c}(s_1), ..., E_{k_c}(s_n)\}$ とすると，R_c は R_A の要素数と同数である．レコード ID a_i が R_B に含まれていた場合 s_i は a_i となり，含まれない場合はレコード ID のドメイン外の値となる．R_C はクライアントで復号し，レコード ID のドメイン内の値のリストが A and B を満たすレコードのリストとなる．

Algorithm 2 の 3 行目の S_i を求める式では，まず A_i と B_j の鍵を共通の鍵 k_c に変更した後 A_i と B_j の差を求めている．ID が一致すれば 0 となり，そうでなければ 0 以外の値となる．これを B_0 から B_m の値に対して求めて総積を求めると，A_i の

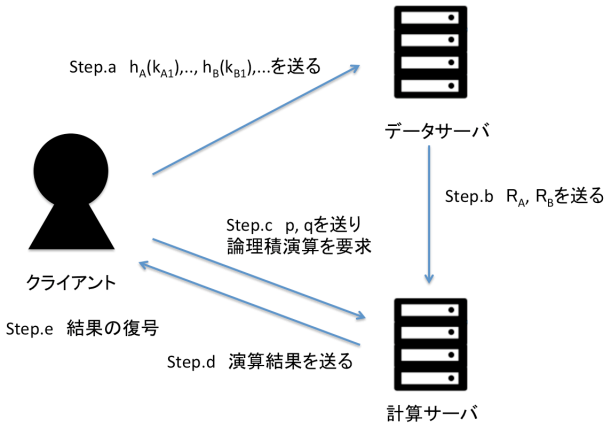

図 2　結果のレコード ID 取得

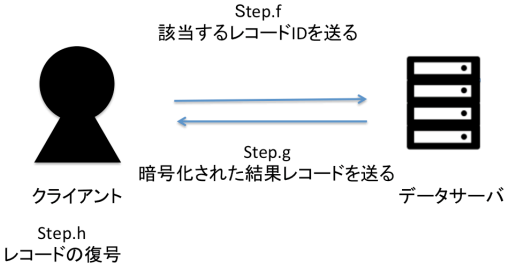

図 3　結果のレコード取得

Algorithm 2 Logical Product (R_A, R_B)

Input: $R_A = \{A_1, ..., A_n\} = \{E_{k_a}(a_1), ..., E_{k_a}(a_n)\}$,
$R_B = \{B_1, ..., B_m\} = \{E_{k_b}(b_1), ..., E_{k_b}(b_m)\}$

Output: $R_C = \{S_1, ..., S_n\} = \{E_{k_c}(s_1), ..., E_{k_c}(s_n)\}$

1: **for** each $A_i \in R_A$ **do**
2: r_i はレコード ID のドメインを超えるランダム値で鍵 k_r で暗号化されている
3: $S_i = E_{k_r}(r_i) \times \prod_{j=0}^{m}(ChangeKey(A_i, k_c) - ChangeKey(B_j, k_c)) + A_i$
4: **end for**

レコード ID が R_B に含まれていた場合のみ値は 0 となり，S_i は A_i となる．これらの計算は暗号化したまま全て行うことができるのでサーバに比較結果を明かすことはない．

提案手法を用いて，表 1 のようなリレーションに上記例の問合せを行い正しくレコードが検索できることを示す．まず，索引の探索により $R_A = \{E_{k_a}(1), E_{k_a}(3)\}$, $R_B = \{E_{k_b}(2), E_{k_b}(3)\}$ が求められる．次に，求めた R_A，R_B を入力として Algorithm 2 を適用すると以下のような計算が行われる．

$$\begin{aligned} S_0 &= E_{k_r}(r_0) \cdot (E_{k_c}(1) - E_{k_c}(2)) \cdot (E_{k_c}(1) - E_{k_c}(3)) \\ &\quad + E_{k_c}(1) \\ &= E_{k'}(2r_0 + 1) \end{aligned}$$

$$\begin{aligned} S_1 &= E_{k_r}(r_1) \cdot (E_{k_c}(3) - E_{k_c}(2)) \cdot (E_{k_c}(3) - E_{k_c}(3)) \\ &\quad + E_{k_c}(3) \\ &= E_{k'}(3) \end{aligned}$$

ここで k' は計算後の暗号鍵である．計算により得られた $R_C = \{S_0, S_1\}$ をクライアントは復号し，3 つ目のレコードが条件に該当するとわかる．

表 1　リレーション R

ID	age	salary
1	45	300,000
2	20	100,000
3	42	180,000

6. 実　　験

本節では，提案手法の性能評価のための実験について述べる．

6.1 実験環境

本実験には，Mac OSX，Intel Core i7 @ 3GHz CPU，16GB RAM で構成された PC を使用した．データセットに属性「A」と「B」からなるレコードを 100，1,000，10,000，100,000 件作成して用いた．また，各属性値は 0 から 1000 の整数値を取る．

6.2 実験評価

本稿では，提案手法の性能を評価するために，SDB [5] と比較を行った．SDB は提案手法と同様に，サーバで複数演算を安全に行うことができる．また，提案手法と SDB では以下の 2 点が異なる．

- **問合せ処理時の索引の利用:**　SDB はテーブル内のレコードに対して問合せ条件との参照を行う．一方で提案手法は索引の探索により条件に該当するレコードを求める
- **条件該当レコードの開示：** SDB では演算そのものは値を暗号化したまま行うことができるが，選択演算や結合演算，論理積等の演算において条件を満たすレコードの集合はサーバに明らかになるという性質を持つ．提案手法はこれらの結果がサーバ上で明らかになることはない．

1 点目の相違点によって，本提案手法は選択演算の際には SDB と比べて理論的には検索速度は優れていると想定される．ただし 2 点目の相違点によって，本提案手法のみサーバ上で結果の漏洩をしないという安全性を確保しているため，計算コストとしては $O(n^2)$ を費やすこととなり，検索速度としては劣ることが予想される．

実験は「SELECT * FROM R WHERE A < 10 AND B > 990」という条件での問合せに対し，各データ数における平均クエリ応答時間の比較を行った．この問合せに該当するレコード数は A,B ともに全レコード数の 1%程度になっている．各データ数における平均クエリ応答時間を図 4 及び表 2 から表 4 に示す．結果は各条件に該当するレコード ID を求める計算時間，両方の条件を満たすレコード ID を求める計算時間，結果のレコードを復号する計算時間に分けて示してある．図 4 から提案手法では [5] と比べて，レコード数が 100 件の時は実行時間のほとんどを索引の探索にかけている．提案手法の場合は探

索回数は少ないが，１回の探索でサーバとクライアントのやりとりが必要となり時間がかかるためレコード数が少ない場合は索引は有効でないことがわかる．図 4 が示すようにレコード数が 1,000，10,000 の場合は SDB よりも実行時間が短くなっている．特に，レコード数 10,000 の時は索引を利用していることで各条件に該当するレコードを高速に求めることができていることが示せた．図 4 からレコード数 100,000 の時も索引による効率的な処理が行われていることがわかる．また，平均クエリ応答時間も SDB よりも 20%ほどの増加に抑えることができている．レコード数の増加により実行時間の大部分が論理積演算にかかってしまったことが実行時間増加の原因であることが示せた．

本実験から，提案手法は索引を用いているためレコード数が多く，問合せの各条件に該当するレコード数が少ない場合に効率的な問合せ処理を行うことができることが示せた．また，各条件に該当するレコード数が増加すると，索引を用いて削減できた実行時間以上に論理積演算にかかる時間の影響で SDB よりも遅くなるということが示せた．論理積演算の計算コストを削減することが今後の課題となると言える．

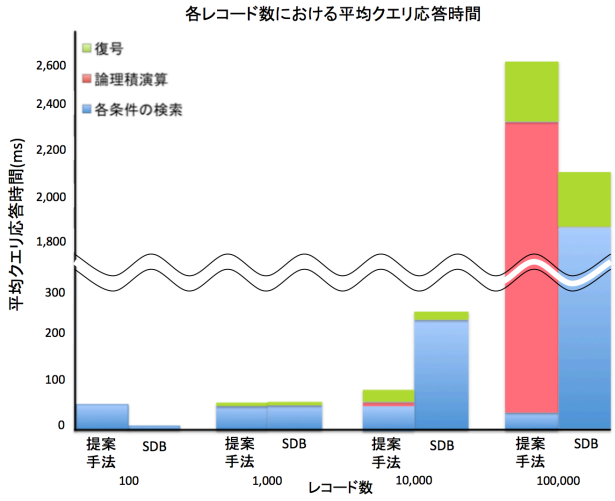

図 4　各レコード数における平均クエリ応答時間

表 2　各データ数における各条件を満たすレコードの探索時間 (ms)

データ数	提案手法	SDB
100	52	8
1,000	49	53
10,000	46	228
100,000	36	1,855

表 3　各データ数における論理積演算の時間 (ms)

データ数	提案手法	SDB
100	0	0
1,000	0	0
10,000	6	0
100,000	2,290	0

表 4　各データ数における結果レコードの復号時間 (ms)

データ数	提案手法	SDB
100	0	0
1,000	5	6
10,000	38	29
100,000	286	271

7. おわりに

本研究では，連言節で接続された複数の条件を含む問合せに対し，中間結果をクライアントに明かさずに求める手法を提案した．また，実験結果から各条件に該当するレコード数が増加すると論理積演算の回数が膨大になり，論理積演算が実行時間の大部分を占めていることが示された．しかし，索引を用いているためデータ数が多く各条件を満たすレコード数が少ない場合は効率的な処理を行うことができた．

今後の課題としては，安全性を追求したことにより実行に時間がかかったことを踏まえて安全性を考慮しつつ計算時間を減少させることを目標とし，比較演算の改善を検討する必要がある．改善方法としては，各条件に該当するレコード全ての組合せについて計算を行わずに，各該当レコード集合を分割した小さなグループに対し比較演算を行う手法が有効であると考えている．

文　献

[1] Amazon RDS. https://aws.amazon.com/jp/rds

[2] Oracle Enterprise Manager 12c. http://www.oracle.com/technetwork/jp/oem/enterprise-manager/overview/index.html

[3] Cloud Bigtable - Google Cloud Platform. https://cloud.google.com/bigtable/?hl=ja

[4] R. A. Popa et al. Cryptdb: processing queries on an encrypted database. CACM, 2012.

[5] W. K. Wong et al. Secure Query Processing with Data Interoperability in a Cloud Database Environment. Proceedings of the 2014 ACM SIGMOD International Conference on Management of data, pp. 1395-1406 (2014)

[6] 篠塚 千愛, 渡辺 知恵美, 北川 博之. DaaS 環境におけるデータとクエリ双方のプライバシ保護を実現する効率的な秘匿検索. DEIM Forum 2015 G2-6 (2014).

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

[8] B. Hore, S. Mehrotra, and G. Tsudik. A Privacy-Preserving Index for Range Queries. In Proceedings of the 30th International Conference on Very Large Data Bases, 2004

[9] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Orderpreserving encryption for numeric data. in Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD '04), pp. 563-574, 2004.

[10] S. Lee, T. Peak, 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)

[11] Kadhem H. Amagasa T. and Hiroyuki K. A Secure and Efficient Order Preserving Encryption Scheme for Relational Databases. In Int'l Conf. on Knowledge Management and

Information Sharing (KMIS2010), Valencia (2010)
[12] E. Mykletun, G. Tsudik. Aggregation Queries in the Database-As-a-Service Model. In IFIP WG 11.3 on Data and Application Security (2006)
[13] T. Ge, S. B. Zdonik. Answering Aggregation Queries in a Secure System Model. In Proceedings of VLDB 2007, pp. 519-530 (2007)
[14] S. Tu et al. Processing Analytical Queries over Encrypted Data. In PVLDB, 2013
[15] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. in Proceedings of the International Conference on the Theory and Application of Cryptogravarphic Techniques Prague (EUROCRYPT '99), pp. 223-238, 1999.
[16] T. Elgamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985, pp. 469-472
[17] C. Gentry. Fully homomorvarphic encryption using ideal lattices. In STOC, 2009.
[18] C. Gentry et al. Fully homomorvarphic encryption with polylog overhead. In EUROCRYPT, 2012.
[19] D. Boneh, G. D. Crescenzo, R. Ostrovsky and G. Persiano. Public key encryption with keyword search. In EUROCRYPT, 2004.
[20] H. Hu et al. Private search on key-value stores with hierarchical indexes. in Proceedings of IEEE 30st International Conference on Data Engineering (ICDE 2014), pp. 628-639, 2014.
[21] I.F. Blake and V. Kolesnikov. Strong Conditional Oblivious Transfer and Computing on Intervals. in Proceedings of the 10th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2004), pp. 515-529, 2004.

付　　録

ここでは Wong らの暗号化スキーム及び鍵更新式について述べる．まず，暗号化のためにクライアントは 2 つの秘密の数 g, n を準備する．ここで g と n は互いに素，n は 2 つの大きなランダムな素数 ρ_1，ρ_2 の積である．また，$\varphi(n)$ を以下の式 A·1 で定める．

$$\varphi(n) = (\rho_1 - 1)(\rho_2 - 1). \tag{A·1}$$

以降で鍵の生成，暗号化，復号，乗算，鍵更新式の式を示す．

- **暗号化鍵の生成**

各カラムごとに定められたカラムキー $ck = \langle m, x\rangle$ から暗号化及び復号鍵を生成する．ただし，m, x は $0 < m, x < n$ の整数とする．以下に暗号化鍵の生成式 A·2 を示す．

$$v_k = mg^{rx \bmod \varphi(n)} \bmod n. \tag{A·2}$$

- **暗号化**

v を平文，v_e を暗号化した値とする．以下に暗号化の式 A·2 を示す．

$$v_e = v{v_k}^{-1} \bmod n. \tag{A·3}$$

- **復号**

以下に復号の式 A·2 を示す．

$$v = v_e v_k \bmod n. \tag{A·4}$$

- **乗算**

A，B の暗号化値をそれぞれ a_e，b_e とし，カラムキーを $ck_A = \langle m_A, x_A\rangle$，$ck_B = \langle m_B, x_B\rangle$ とする．a_e と b_e の乗算方法について述べる．クライアントは乗算後の鍵を式 A·5 で計算する．

$$\begin{aligned} m_C &= m_A m_B \bmod n. \\ x_C &= x_A + x_B \bmod \varphi(n). \end{aligned} \tag{A·5}$$

乗算結果を c_e とし，サーバは式 A·6 で暗号化された値を書き換える．

$$c_e = a_e b_e \bmod n. \tag{A·6}$$

c_e が正しく復号できることを式 A·7 で示す．なお，簡単のため mod は省略している．

$$\begin{aligned} c = c_e c_k &= a_e b_e c_k \\ &= a_e b_e m_C g^{rx_C} \\ &= a_e b_e m_A m_B g^{r(x_A + x_B)} \\ &= a_e a_k b_e b_k \\ &= ab \end{aligned} \tag{A·7}$$

- **鍵更新式**

以下で鍵の更新演算について述べる．まず，データ管理者がデータを預ける際にカラム内の値がすべて 1 のカラム S を暗号化してサーバに保存する．

S のカラムキーを $ck_S = \langle m_s, x_s\rangle$，カラム A の暗号化値を a_e，カラムキーを $ck_A = \langle m_A, x_A\rangle$，更新後のカラムを C，暗号化値を c_e，カラムキーを $ck_C = \langle m_C, x_C\rangle$ とする．クライアントは以下の式 A·8 で p，q の値を計算してサーバに送る．

$$\begin{aligned} p &= x_s^{-1}(x_C - x_A) mod\ \varphi(n) \\ q &= m_A {m_S}^p {m_C}^{-1} mod\ n \end{aligned} \tag{A·8}$$

サーバはクライアントから p，q を受け取り以下の式 A·9 で c_e 計算する．

$$c_e = q a_e {s_e}^p \tag{A·9}$$

以下の式 A·10 で c_e を ck_C で復号できることを示す．以下では mod を省略する．

$$\begin{aligned} c = c_e m_C g^{rx_C} &= m_A {m_S}^p {m_c}^{-1} a_e {s_e}^p m_C g r x_c \\ &= m_A m_S a_e ((m_S g^{rx_S})^{-1}) {s_e}^p m_C g r x_c \\ &= m_A a_e (g^{rx_S {x_S}^{-1}(x_C - x_A)})^{-1} g r x_c \\ &= a_e (m_A g^{rx_A}) = a \end{aligned} \tag{A·10}$$

以上より暗号化値を変更することなく，鍵の更新を行えることが示せた．