情報処理学会論文誌　データベース　Vol.12　No.2　1–11 (Apr. 2019)

暗号化データベースにおける構造とデータを分離した索引を用いた安全かつ高速な検索手法

渡辺 知恵美[1,a)]　秋山 賢人[2,†1,b)]　天笠 俊之[3,c)]　北川 博之[3,d)]

受付日 2018年9月10日, 採録日 2018年12月29日

概要：機密情報が含まれたデータや資産価値の高いデータをクラウド環境で管理する場合，管理者からもデータの中身を秘匿するため暗号化データベースシステムを用いるケースは近年一般的となってきた．ただし，大規模なデータを扱う場合，検索速度が問題となることがある．検索高速化のためには索引を使うのが一般的であるが索引の構造や検索時の探索パターンにより元データを推測される恐れがある．そのため，本論文ではサーバ上で索引の構造を秘匿した索引による検索手法を提案する．クライアントが索引構造の一部を持ち索引を探索することで高速かつ安全な検索を実現する．

キーワード：暗号化データベース，プライバシ保護，検索可能暗号

Secure and Fast Query Scheme for Encrypted Database Using Shuffled Index

Chiemi Watanabe[1,a)]　Kento Akiyama[2,†1,b)]　Toshiyuki Amagasa[3,c)]　Hiroyuki Kitagawa[3,d)]

Received: September 10, 2018, Accepted: December 29, 2018

Abstract: Due to the rapid proliferation of cloud computing services in diverse applications, such as database as a service (DBaaS), and encrypted database systems (EDBSs) have been gaining much attentions as a way to construct secure databases in DBaaS. However, most of the existing works suffer from poor performance when dealing with large data. Some works proposed index-based query processing schemes, but they have a privacy problem that the order of attribute values may be revealed from the index structure on the server. To this problem, we propose a novel secure index-based query processing scheme where the order of attribute values is not disclosed. In the scheme, the index is maintained in such a way that the structural information regarding the index and the values (or index entries) are maintained separately, and only the latter is stored in a cloud server. When searching, a client uses the structural information (without entries) to traverse the index by cooperating with cloud servers, thereby securing the order among the index entries. We prove that, in our scheme, the order among the index entries would not be disclosed even though the data and the query log are disclosed. In addition, our experimental results show that the proposed scheme significantly outperforms existing state-of-the-art schemes.

Keywords: encrypted database, privacy preserving technique, searchable encryption

1. はじめに

クラウドコンピューティングの発達により，DBaaS（Database as a Service）が広く提供されている．DBaaSとはネットワークを介してデータベースの機能を提供する

[1] 筑波大学図書館情報メディア系
Faculty of Library, Information and Media Science, University of Tsukuba, Tsukuba, Ibaraki 305–8550, Japan
[2] 筑波大学システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[3] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[†1] 現在，セコム株式会社
Presently with SECOM Co., Ltd.

[a)] chiemi@slis.tsukuba.ac.jp
[b)] kentoa@kde.cs.tsukuba.ac.jp
[c)] amagasa@cs.tsukuba.ac.jp
[d)] kitagawa@cs.tsukuba.ac.jp

サービスであり，データ所有者がサーバ管理者の管理するクラウドデータベースにデータの管理を委託することができる．代表的な DBaaS としては，Amazon RDS [2]，Google Cloud Bigtable [7]，Oracle Enterprise Manager 12c [19] などがあり，データベース管理を委託してユーザの負担軽減が可能であるほか，データ量に応じてデータベースの柔軟な拡張を行うことができるという利点がある．DBaaS を利用したデータ提供サービスでは，データ所有者がサービスプロバイダが管理するサーバにデータを保管し，クライアントに保存したデータへのアクセス権を提供する．クライアントはサービスプロバイダが管理するサーバに対して問合せを行うことができる．しかし，データ所有者が機密データを預ける場合には，機密データやクエリの漏洩によるプライバシの侵害が重大な問題となりうる．機密データやクエリに関するプライバシを保護するために，以下のようなデータやクエリに関するプライバシの要求がある．

これらの要求を満たすために，近年では暗号化データベースシステムが提案されている．暗号化データベースシステムではデータの機密性を保証するために，データ所有者が事前にデータを暗号化してクラウドサーバに保存する．そして，クライアントはクエリを検索可能暗号で暗号化することでデータとクエリを暗号化したまま安全に検索を行う．暗号化データベースシステムに関する研究としては CryptDB [21]，Monomi [22]，SDB [23] などがある．しかし，暗号化データベースにおける検索処理は時間計算量が大きく，データへの高速なアクセスの保障を損ねてしまうため，データ提供への応用には適さない．

本研究では，データとクエリに関するプライバシを保護しつつ暗号化索引を利用した効率的な検索を可能にする検索可能暗号フレームワークを提案する．本フレームワークは，検索用索引を暗号化してクライアントとサーバで分散管理し，クライアントがサーバ上の索引を探索することで効率的な検索を実現する [24], [25], [26], [28]．通常の索引はノード間の順序関係により値の大小関係が推測できる問題がある．これに対して OIT [15] はクライアントが暗号化された索引を所有し，その探索をサーバで行う手法を提案しているが，索引をすべてクライアントが所有し索引探索時に探索ノードをサーバに送信するなどクライアントの負荷およびサーバとの通信コストが高いという問題がある．本フレームワークにおける索引はサーバに格納されているが，ノード間の順序関係を保持していないため構造から内容が推測されにくい．また，クエリに対してもノイズを付与することで，索引へのアクセス頻度をもとに値が推測されないことを保証する．本索引を用いて実現可能なクエリは，属性の選択演算で，完全一致と範囲検索，文字列属性の部分一致である．

本論文の構成は以下のとおりである．2 章において本フレームワークが対象とする問題について明確にし，3 章において関連研究について述べる．本フレームワークの提案を 4 章，5 章で提案内容に関するプライバシを分析する．6 章で評価結果を示し，7 章でまとめと今後の課題を述べる．

2. 問題の定義

2.1 システムモデル

本研究で想定する暗号化データベースのシステムモデルについて述べる．本フレームワークの利用者は，データ所有者（Data Owner：DO），サービスプロバイダ（Service Provider：SP），クライアント（Client：CL）の 3 種類に分けられる．想定するシステムモデルにおける利用者の関係とデータや問合せの流れを**図 1** に示す．

データ所有者は自身の持つ機密データをサービスプロバイダが管理するサーバにアップロードし管理を委託する．データ所有者にとって機密データは重要な資産であるため，サービスプロバイダにも内容を秘匿するためサーバへアップロードする前に機密データを暗号化する．サービスプロバイダはデータ所有者に委託された機密データを管理するとともに，クエリ実行のための強力な計算リソースを提供する．

クライアントはクエリを発行し，サーバからデータを取得する．クエリの内容からクライアントのプライバシを保護するため，クエリそのものも暗号化しサービスプロバイダに内容を推測させないようにする．

また本研究では，データは一括でデータ所有者からサービスプロバイダへ保存され，データはその後の更新を想定しないものとする．そのため，データベースおよび索引の更新は最初に 1 度行うのみであり，その後データ所有者およびクライアントからの更新は原則として行わない．ただしデータ所有者が鍵を更新してデータベースおよび索引を一から作り直して再アップロードする際はその限りではない．

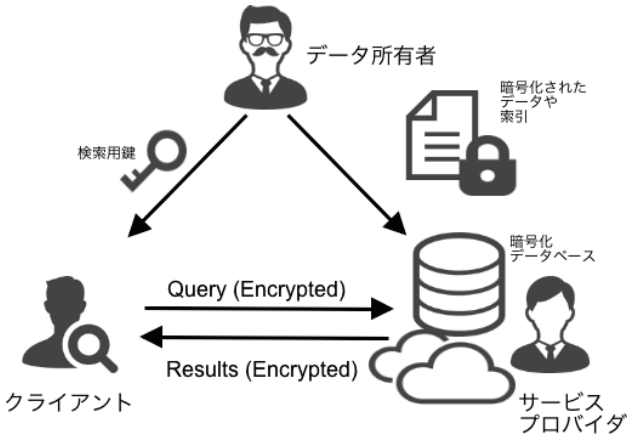


図 1　システムモデル
Fig. 1　System model.

2.2　攻撃者モデル

本フレームワークで想定する攻撃者モデルについて述べる．まず，クライアントとサービスプロバイダは semi-honest であり結託しないと仮定する．また，攻撃者としてサービスプロバイダを想定し，サービスプロバイダはサーバに保存されたデータ，索引，クエリログからレコードに含まれる属性値の順序関係を推定しようとすることを考える．攻撃者が属性値の順序関係を入手した場合，選択平文攻撃（Chosen Plaintext Attack）により属性値の推定が可能である．このことから，攻撃者に索引のエントリの順序を推定されることは望ましくない．本研究では，以下の定義を満たすことで高い安全性を提供する．

定義 1．索引の要素に関する識別不可能性

N を索引の要素数，$E = \{e_1, \ldots, e_N\}$ を要素集合，$E' = \{e'_1, \ldots, e'_N\}$ を暗号化された要素集合とする．ただし，E' と E' に含まれる要素の順序は対応せず，e'_i に対応する E の要素を $f(e'_i)$ とする．攻撃者が e'_i の要素に対応する E の要素を $s(e'_i)$ と推測したとき，この推測が正解する確率が以下の式を満たすとき索引 E' は安全である．

$$\forall i, P(s(e'_i) = f(e'_i)) \leq \frac{1}{N} \tag{1}$$

つまり，安全な索引は E は推測される確率が N 個の要素をランダムに選択したときの確率と同等かそれ以下となることを保証する．

3.　関連研究

暗号化データベースシステムに関する研究は Hacigümüs ら [13] に提案されて以降，数多く取り組まれている．Hacigümüs らはクライアントとサーバの問合せ処理モデルを定義した．その後，Hore ら [14] は Hacigümüs ら [13] の手法を基に，統計による値の推測を困難にするサーバ側のデータ生成法を提案した．暗号化データベースに利用される暗号化手法として，Agrawal ら [1] は数値属性の順序関係を保存可能な順序保存暗号（OPE）を提案し，Lee ら [16] や Hasan ら [12] などにより OPE の改良手法が提案されている．また，Mykletun ら [18] や Ge ら [9] 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-近傍探索なども提案されている．Popa ら [21] は，これらの手法を組合せて暗号化データベースシステム CryptDB を提案した．CryptDB は，RND，DET，Paillier 暗号 [20] などの異なる暗号化手法を用いて機密データを幾重にも暗号化することで機密性を保証しており，暗号化手法ごとに異なるクエリの処理を行うことにより，キーワード検索，結合演算，大小比較など様々な演算をサポートしている．なお，CryptDB はオープンソースパケージとして公開されている．Stephen らは，CryptDB の検索処理最適化機構を改良した Monomi [22] を提案している．

近年では，準同型暗号や検索可能暗号を利用した検索手法が注目を集めている．準同型暗号とは暗号化したまま暗号文どうしの加算や乗算を行うことができるという性質を持つ暗号である．加算を行える暗号は加法準同型暗号，乗算を行える暗号は乗法準同型暗号，加算と乗算の両方を行える暗号は完全準同型暗号と呼ばれている．加法準同型暗号には，Paillier 暗号 [20]，乗法準同型暗号には ElGamal [8] 暗号などがある．完全準同型暗号には，Gentry らが提案した整数上完全準同型暗号 [10] がある．しかし，Gentry らの手法で計算時にビットごとの演算が必要となるため計算コストが非常に高いことが知られている [11]．Boneh ら [6] は，任意回の加算と数回の乗算が可能な Somewhat 準同型暗号と呼ばれる暗号方式を利用した共通集合演算を提案している．また，Ma ら [17] は準同型暗号である DBGN 暗号（deterministic BGN）と Bloom Filter を利用して結合演算を提案している．検索可能暗号にはキーワード検索が可能なキーワード検索公開鍵暗号（PEKS）[5] が Boneh らによって提案されている．

暗号化データベースシステムにおいて，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．Wong ら [23] は相互運用性のある加法準同型暗号を提案し，SDB と呼ばれる暗号化データベースシステムを提案した．SDB では演算の中間結果をクライアントに転送することなく，サーバ上で暗号化したまま演算の出力結果を次の演算の入力にすることが可能である．

これまでに述べた暗号化データベースシステムは高い安全性を保証する一方で，大規模データに対して検索を行う場合には良いパフォーマンスを示さない．データ数が大きい場合，これらの手法では各レコードが検索条件に一致するかを確認しなければならないため，検索に時間がかかるという問題があった．

この問題に対し，Hu ら [15] 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（Oblivious Index Traversal）を提案した．OIT では，クライアントが所有する索引を暗号化した状態でサーバに持ち込み，紛失通信技術を使って大小比較を行い探索をすることで安全性を確保している．しかし，サーバとクライアント間の通信量が多く，クライアントが索引を所有するため大きな負荷がかかるという問題がある．

4.　提案手法

我々は，暗号化索引を利用して安全かつ効率的に検索を行う暗号化データベースシステムを提案する．提案手法はクライアントとサーバが索引を分散管理し索引探索の際にはクライアントがサーバ上にある索引を探索するという特徴を持ち，索引探索のたびに通信をする必要があるものの探索回数を $O(\log(N))$ に抑えることができる．またサーバが索引やアクセスされたノードのログから索引の構造を推

測できないという安全性（定義 1）を保証する．

対象とする演算は選択演算であり，データ型は問わず完全一致と大小比較演算に対応する．また索引に接尾辞配列を用いることで部分文字列検索にも対応できる [27]．

以降，データ格納スキーム，検索アルゴリズムについて述べる．

4.1　データ格納

データ所有者がサーバへデータを格納するプロセスを**図 2** に示す．データ所有者は 1 つ以上の属性を持つリレーショナルテーブルを持つとし，テーブルをもとに暗号化索引を構築する．作成する配列は属性値をキーとするハッシュテーブルで，ハッシュ値として，属性値をソートしたときの並び順に基づいた値を用いる．

以下索引の手順を述べる．データ所有者は索引を作る属性を決め，テーブルに含まれるすべての属性値 v に対してそれぞれその値を持つレコード ID リスト $r(v)$ を生成する．テーブル内のレコード数を N，属性値の異なり数を N'（$\leq N$），属性値リストを $v_1, \ldots, v_{N'}$ とすると，$I = (\langle v_1, r(v_1)\rangle, \ldots, \langle v_{N'}, r(v_{N'})\rangle)$ が生成される．また各レコードはレコード全体を暗号化し，レコード ID で参照できる形にする（図 2 右上）．続いて，データ所有者は属性値 v で I を昇順ソートする（図 2 左下）．この配列に対し，それぞれの $\langle v, r(v)\rangle$ にソートされた配列上での位置を示すアドレス $a_1, \ldots, a_{N'}$ に対して鍵付きハッシュ関数を適用した値 $h(a_i) = HMAC_{seed}(a_i)$ を追加する．また，v と $r(v)$ を Paillier 暗号などの加法準同型暗号 $E(\cdot)$ で暗号化する．最終的に，索引の構造としては三項組 $\langle h(a_i), E(v_i), E(r(v_i))\rangle$ のリストとなる（図 2 右下）．データ所有者はリストの順序をシャッフルしたうえで，暗号化されたレコードリストとともにサーバにアップロードする．

作成されたリストはシャッフルによって索引の順序関係がバラバラの状態でサーバに保管される．一方 a_i に対して適用されたハッシュ関数の $seed$ を持つ場合，配列上のアドレス $addr$ に対してハッシュ関数 $HMAC_{seed}(a_i)$ を適用することによって配列の a_i 番目に保存された属性値とレコード ID リストの組 $\langle E(v_i), E(r(v_i))\rangle$ を取得することができる．

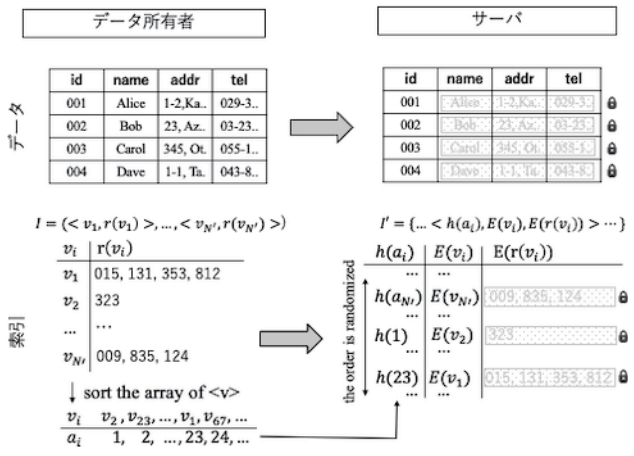


図 2　データ所有者からサーバへのデータ格納プロセス

Fig. 2　Data stroing process to the server from data owner.

4.2　検索スキーム

サーバに保管された三項組 $\langle h(a_i), E(v_i), E(r(v_i))\rangle$ の集合は属性値によってソートされた配列と見なすことができる．データ所有者は鍵付きハッシュ関数の $seed$ および索引の復号鍵 sk をクライアントに渡すことで，クライアントは属性値でソートされた配列の a_i 番目の属性値とレコードリストを取得することができる．

我々の提案する検索するスキームではこの性質を利用して，サーバ上にある暗号化された配列をクライアントが m 分探索をする．しかし，m 分探索をそのまま適用すると，サーバに索引の順序関係が分かってしまう．たとえば，エントリ数を N とし，$m = 2$ の場合を考えると，クライアントはクエリ q と key を比較するため最初の探索で $\frac{N}{2}$ 番目のエントリ $e_{\frac{N}{2}}$ にアクセスする．2 分探索をしているので，サーバはアクセスしたエントリについて索引上の位置が $\frac{N}{2}$ 番目であることを推測することができる．2 回目の探索では，クライアントは $\frac{N}{4}$ 番目のエントリ $e_{\frac{N}{4}}$ または $\frac{3N}{4}$ 番目のエントリ $e_{\frac{3N}{4}}$ にアクセスする．すると，サーバは $\frac{N}{4}$ 番目または $\frac{3N}{4}$ 番目のエントリであることを推測することができる．Boldyreva ら [4] はサービスプロバイダがエントリの順序関係を知っている場合，索引のエントリが暗号化されていたとしても選択平文攻撃に脆弱であると述べている．

この問題に対処するために，探索位置の摂動を行う．初回はランダムな位置を k 個（ただし $k > m-1$）以上選んで探索を行い，2 回目以降は絞り込まれた範囲の中で $m-1$ 個の位置を選択するとともに探索範囲外にランダムに $k-m+1$ 個の位置を追加する．これにより探索パターンによる推測を防ぐ．アルゴリズム 1 に暗号化索引の探索方法を示し，**図 3** に $m = 3$ の場合の 2 回目以降の探索プロセスを示す．

まず，クライアントは検索結果用の変数 i_q を null，探索領域 (l, u) を $(1, N)$ に初期化し，$E(q)$ をサーバに送る（Algorithm 1，1–3 行目，図 3 (1)）．初回の探索では，クライアントは配列の位置を k 個ランダムに選び（$\{i_n\}_{n=1}^{k}$），対応するハッシュ値集合 $H = \{h(i_n)\}_{n=1}^{k}$ を計算してサーバに送る（4–5 行目）．

サーバは送られたハッシュ値 $h \in H$ と一致するアドレスを持つ三項組 $\langle h(a_i), E(v_j), E(r(v_j))\rangle$ から暗号化された属性値 $E(v_j)$ を取り出し，$E(q)$ と $E(v_j)$（$1 \leq j \leq k$）の大小関係を暗号化したまま計算し，クライアントに結果を送る（6–14 行目）．

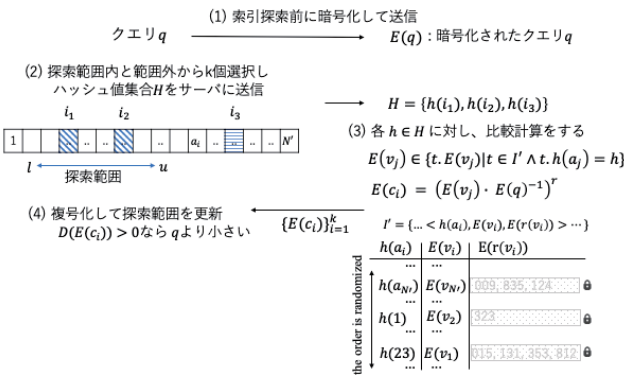


図 3　索引探索のプロセス

Fig. 3　Process of index traverse.

Algorithm 1 traverseIndex

Input: N, k, m, q, *type*: "=", "<" or ">"
Ensure: i_q: an index of record which matches query value q

1: $i_q \leftarrow null$
2: $(l, u) \leftarrow (1, N)$
3: Client sends $E(q)$ to the server.
4: Client chooses $\{i_n\}_{n=1}^{k}$ from (l, u) randomly.
5: Client sends $H = \{h(i_n)\}_{n=1}^{k}$ to the server.
6: **procedure at the server**
7: $List_e \leftarrow \{\}$
8: **for** $h_j \in H$ **do**
9: Server obtains $E(v_j)$ corresponding to h_j.
10: $E(c) = (E(v_j) \cdot E(q)^{-1})^r$
11: $List_e.add(E(c))$
12: **end for**
13: Server sends $List_e$ to the client.
14: **end procedure**
15: Client decrypts $\{E(c_i)\}_{i=1}^{k} \in List_e$.
16: Client updates (l, u) based on $c_1, \ldots, c_k$.
17: **while** $l < (u - l)$ **do**
18: Client calculates $\{i_n\}_{n=1}^{m-1}$ which evenly divides (l, u) into m regions.
19: Client chooses $i'_1, \ldots, i'_{k-m+1}$ from outside of (l, u) randomly.
20: Client sends $H = \{h(i_1), \ldots, h(i_{m-1}),$
21: $h(i'_1), \ldots, h(i'_{k-m+1})\}$ to the server.
22: **procedure at the server**
23: $List_e \leftarrow \{\}$
24: **for** $h_j \in H$ **do**
25: Server obtains $E(v_j)$ corresponding to h_h.
26: $E(c) = (E(v_j) \cdot E(q)^{-1})^r$
27: $List_e.add(E(c))$
28: **end for**
29: Server sends $List_e$ to the client.
30: **end procedure**
31: Client decrypts $\{E(c_i)\}_{i=1}^{m-1} \in List_e$.
32: Client updates (l, u) based on $c_1, \ldots, c_{m-1}$.
33: **end while**
34: **if** $l \neq N$ or $u \neq 1$ **then**
35: **if** *type* is "≤" **then**
36: $i_q \leftarrow u$
37: **else if** *type* is ">" **then**
38: $i_q \leftarrow l$
39: **end if**
40: **end if**

Algorithm 2 Oblivious Secure Index Traversal

Input: N, $h(\cdot)$, min, max
Ensure: $Result = Q(min, max)$: a set of records corresponding to query result

1: $Result \leftarrow \{\}$
2: $i_{min} \leftarrow traverseIndex(N, min, \text{"}\leq\text{"})$
3: $i_{max} \leftarrow traverseIndex(N, max, \text{">"})$
4: **for** $i = i_{min}$ to i_{max} **do**
5: Client obtains $E(r_i)$ corresponding to $h(i)$ from the server.
6: $r_i \leftarrow D(E(r_i))$
7: $Result.add(r_i)$
8: **end for**

クエリ値と key 値の大小比較演算は Paillier 暗号 [20] の性質に基づき計算される．$E(c)$ を大小比較結果，$E(v)$ を暗号化した key 値，$E(q)$ を暗号化したクエリ値，r をランダムな正の整数とすると，クエリ値と key 値の大小比較は式 (2) で計算できる．

$$\begin{aligned} E(c) &= (E(v) \cdot E(q)^{-1})^r \\ &= E(r(v - q)) \end{aligned} \tag{2}$$

サーバで式 (2) を計算することで q や v をサーバに明かさずに計算でき，クライアントで復号することでクライアントのみが q と v の大小比較結果を知ることができる．その後，クライアントは比較結果の復号を行い，探索領域 (l, u) を更新する（15–16 行目）．

2 回目以降の探索では，クライアントは (l, u) を m 分割する $m - 1$ 個のエントリを選択し，$k - m + 1$ 個のエントリを (l, u) 以外の領域から選択する（Algorithm 1，18–19 行目，図 3 (2)）．そして，クライアントは選択するエントリに対応したハッシュ値を計算しサーバに送り，サーバで $E(q)$ と $E(v_j)$ の大小比較を行う（図 3 (3)）．クライアントは結果を復号する（Algorithm 1，20–30 行目，図 3 (4)）．なお，クライアントはダミーのエントリを知っているので，$c_1, \ldots, c_{m-1}$ のみを復号して (l, u) を更新する（31–32 行目）．

最後に，クライアントは (l, u) と $type$ に基づき i_q を得る（34–40 行目）．このようにして，提案手法は整数属性に対して安全で効率的な範囲検索と完全一致検索を行うことができる．アルゴリズム 2 は全体のクエリ処理の手順である．クライアントは暗号化索引の暗号化を行う関数 $E(\cdot)$ と復号を行う関数 $D(\cdot)$ をあらかじめデータ所有者から受け取っておく．クライアントが $[min, max)$ を指定し索引を探索したとき，クライアントは $min \leq A$ を満たす最小の A の値のアドレス i_{min} と $A < max$ を満たす最大の A の値のアドレス i_{max} を見つける．そして，クライアントは i_{min}，i_{max} に基づき，暗号化されたクエリ結果 $E(r_i)$ を復号する．

5. 安全性と効率性を両立するためのパラメタ決定

検索手法では配列へのアクセスパターンから配列上の位置を推測されないために毎回 k（$> m$）個の位置を指定しクエリ値との比較を行っている．本節ではパラメタ m と k に関する，安全性と効率性を両立するパラメタの設定について述べる．5.1 節では本提案手法で安全性が保証されるための k の条件を求めるとともに定義 1 で述べた安全性が保証されていることを証明する．5.2 節では検索時間を考慮したパラメタの決定について述べる．

5.1 安全性を満たす m と k の関係

まず前提として，本手法は以下の 2 点を保証していることを示す．1) 攻撃者（サービスプロバイダ）は，索引探索の間，入力と出力から何の情報も得ることができない，2) 攻撃者は，クラウドサーバに保存されている索引と暗号化されたレコードそのものから何の情報も得ることができない．これらの要件は準同型暗号の性質により満たされるので，安全性の証明は省略する．

本節ではこの前提に基づき，攻撃者は索引のエントリの順序に関して，定義 1 を満たすためのパラメタ k の条件を求める．

索引は属性値 v の値で昇順ソートされた配列である．ソートされた配列の要素を $e_1, e_2, \ldots, e_N$ とすると，配列 E は以下のように表せる：

$$\begin{aligned} &E = (e_1, e_2, \ldots, e_N) \\ &\quad \exists i \exists j, i < j \Rightarrow e_i < e_j \end{aligned} \tag{3}$$

データ所有者は $e_1, e_2, \ldots, e_N$ を暗号化し，要素をシャッフルした集合 E' を作成する：

$$E' = \{e'_1, e'_2, \ldots, e'_N\} \tag{4}$$

e'_i（$i = 1, 2, \ldots, N$）は E の暗号化された要素を表し，E' の要素の順序は E のエントリの順序とは無関係である．

$AccessLog_r$ を r ラウンド目にクライアントによってアクセスされた要素の集合とすると，$AccessLog_r$ は以下の式 (5) で表される：

$$AccessLog_r = \{e'_{q1}, e'_{q2}, \ldots, e'_{qk} | e'_{qi} \in E'\} \tag{5}$$

ただし，e'_{qi}（$i = 1, 2, \ldots, k$）はアクセスされた要素を表す．また，攻撃者が推定する e_i に対応する E' 上の位置を $s(e_i)$ とし，実際に e_i に対応する E' 上の位置を $f(e_i)$ とする．

1 回目の探索においてクライアントは k 個の要素にランダムにアクセスする．参照された要素はすべてどの位置になるか同様に確からしいため $e'_{q1}, e'_{q2}, \ldots, e'_{qk} \in AccessLog_1$ に関して，$P(s(e'_{qi}) = f(e'_{qi}))$ は $\frac{1}{N}$ となる．

r（$r \geq 2$）回目の探索では，(l_r, u_r) の範囲から m 分探索で $m-1$ 個のエントリを選ぶ．簡単のため，まず探索範囲を攻撃者が知っていると仮定して m 分探索をした場合の攻撃者の正答率を求め，本提案手法における攻撃者の正答率を求める．

探索範囲を知っている攻撃者は r ラウンド目に選択されるのは $(l_r + \lceil \frac{xN'}{m} \rceil)$ 番目の要素（$1 \leq x \leq m-1$）であることを知っている．したがって，すべての $e'_{qi} \in AccessLog_r$ に対する要素を e_j と推測して正解する確率 $P(s(e'_{qi}) = e_j \wedge s(e'_{qi}) = f(e'_{qi}))$ は以下の式 (6) で表される：

$$\begin{cases} \frac{1}{m-1} & \text{if } j = l_r + \lceil \frac{xN}{m} \rceil \, (\exists x, 1 \leq x \leq m-1) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

これにより正答率が $\frac{1}{N}$ となる場合があるため E' は安全ではない．

一方で本提案手法の場合 1 回目の探索ですべてのエントリがランダムに選択されるため，攻撃者は 2 回目以降の探索の探索領域 (l_r, u_r) を知ることができない．そのため，考えうるすべての探索領域の中で $m-1$ 分割された位置が推定候補となり，攻撃者の持つ知識においてどの探索領域になるかは同様に確からしいと考えると，$AccessLog_2$ の各要素に対し，e'_{qi} に対応する E の要素が e_j である確率 $P(s(e'_{qi}) = e_j \wedge s(e'_{qi}) = f(e'_{qi}))$ は以下の式 (7) で表すことができる．

$$\begin{aligned} &P(s(e'_{qi}) = e_j \wedge s(e'_{qi}) = f(e'_{qi})) \\ &\quad = \frac{1}{N-m+2} \sum_{len=m-1}^{N} \frac{1}{N-len+1} \\ &\quad \sum_{st=1}^{N-len+1} Q(st, len, j) \end{aligned} \tag{7}$$

len $(= u_r - l_r + 1)$ は探索領域の領域長を，st $(st = l_r)$ は開始位置を表す．

$Q(st, len, j)$ は攻撃者が探索範囲を $(st, st + len)$ と知っているうえでの推定 $s(e'_{qi}) = e_j$ が正解する確率であり，式 (7) と同様に以下の式 (8) で表される：

$$\begin{cases} \frac{1}{m-1} & j = \frac{len}{m} x + st \; (1 \leq x \leq m-1) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

加えて，クライアントは $k-m+1$ 個のダミーを含んだ k 個の要素を選択する．ダミーの要素を導入することで，式 (7) は以下の式 (9) に変更される：

$$\begin{aligned} &P(s(e'_{qi}) = e_j \wedge s(e'_{qi}) = f(e'_{qi})) \\ &\quad = \frac{m-1}{k} \frac{1}{N-m+2} \sum_{len=m-1}^{N} \frac{1}{N-len+1} \\ &\quad \sum_{st=1}^{N-len+1} Q(st, len, j) \end{aligned} \tag{9}$$

式 (9) にある $\sum_{st=1}^{N-len+1} Q(st, len, j)$ は開始長が 1 から $m-1$ に動く間に $(m-1)$ 回 $P(f(e'_{qi}) = e_j \wedge st \leq e_j \leq st + len) = \frac{1}{m-1}$ となったとき，最大となる．

また，式 (8)，式 (9) に基づき，$P(f(e'_{qi}) = e_j)$ の上限値を以下のように表せる：

$$\begin{aligned} &\forall e'_j, P(s(e'_{qi}) = e_j \wedge s(e'_{qi}) = f(e'_{qi})) \\ &\leq \frac{1}{N-m-2} \sum_{len=m-1}^{N} \frac{1}{N-len+1} \cdot 1 \\ &\leq \frac{1}{N-m+2} \int_{1}^{N-m+2} \frac{1}{x} dx \\ &= \frac{\log(N-m+2)}{N-m+2} \end{aligned} \tag{10}$$

クライアントは式 (11) を満たすような k を決定する．

$$\frac{N(m-1)\log(N-m+2)}{N-m+2} \leq k \tag{11}$$

したがって，式 (11) に基づき適切な k を選ぶことで，すべての $e'_j \in AccessLog_2$ において $P(s(e'_{qi}) = e_j \wedge s(e'_{qi}) = f(e'_{qi})) < \frac{1}{N}$ 以下となる．$r > 2$ に関しては正答する確率は $r = 2$ 以下となるため，上記の条件を満たす k を満たすことで安全な索引 E' が保証される．

5.2　検索時間見積りに基づくパラメタ設定

本提案手法における探索で適切な m を決定するための検索時間見積りについて述べる．まず，t_{query}，t_{comm}，t_{comp}，t_{dec} をそれぞれ，クエリ値を暗号化してサーバに送る時間，1 往復の通信時間，暗号化された属性値 $E(v)$ とクエリ $E(q)$ のサーバ上での計算時間，1 個の暗号化された比較結果の復号に必要な時間とする．1 回目の探索および 2 回目以降の 1 回分の探索時間は以下のように見積もることができる．

$$\begin{aligned} T_{first}(m,k) &= k(t_{comm} + t_{comp} + t_{dec}) \\ T_{otherwise}(m,k) &= k(t_{comm} + t_{comp}) + (m-1)t_{dec} \end{aligned}$$

ここでは説明を簡単にするために 1 回目の探索で均等に要素を選択して 2 回目以降の探索領域長を固定して見積もることとする．属性の異なり値が N であるとき探索数は $\left\lceil \frac{\log(N)}{\log(m)} \right\rceil$ となり，1 回の検索にかかる時間の見積り値は

$$\begin{aligned} T_{total} = T_{query} &+ T_{first}(m,k) \\ &+ \left\lceil \frac{\log(N)}{\log(m)} \right\rceil T_{otherwise}(m,k) \end{aligned}$$

となる．

6.　評価

本提案手法の検索速度の評価として，以下の 2 つの実験を行った．

- 実験 1：実行環境における適切なパラメタの設定と検索時間見積りの精度
- 実験 2：既存手法との検索時間の比較

また，安全性に関しても定量的に評価するため以下の実験を行った

- 実験 3：本提案手法における各要素のアクセス頻度
- 実験 4：アクセスログにおける各要素のアクセス共起度

6.1　実験環境

本実験には，クライアントに Mac OS X，Intel Core i7 @ 3 GHz CPU，16 GB RAM で構成された PC を，サーバに Ubuntu，Intel Core i7-2600 @ 3.40 GHz CPU，16 GB RAM で構成された PC を使用した．データセットは属性「A」，「B」からなり，データ数は 100，1,000 と 10,000 から 100,000 まで 10,000 ごと増やした 12 種類の人工データを作成した．各属性値は 0 から 1,000 までの整数値をとる．索引の要素数 N については，データ数 100 では 96，1,000 では 637，10,000 から 100,000 では 1,000 となっている．クエリは「SELECT * FROM dataset WHERE A < 10」とし，レコードの選択率は 1%程度とする．提案手法と SDB における各データの最大サイズについては**表 1** に示す．また，クライアントとサーバの間の通信速度は高速な環境（3.5 Mbps）と低速な環境（32 Kbps）の 2 種類で評価する．

6.2　実験 1：検索時間見積りの精度とパラメタ設定

実験 1 では実測値が定義した見積り時間に従っているかどうかを検証する．まず，3.5 Mbps と 32 Kbps の場合で事前に計測した値を**表 2** に示す．転送速度が低速な場合と高速な場合を設定したが，1 度に送るデータ量が非常に小さいため転送速度には大きな違いが見られなかった．また

表 1　提案手法と SDB における各データサイズの最大値
Table 1　Maximum data size of proposed method and SDB.

データ名	データサイズ
クエリ	128 bit
ハッシュ値	256 bit
比較結果	128 bit
鍵更新のパラメータ	126 bit

表 2　検索時間見積りに用いた変数と計測値
Table 2　Parameters for query time estimation and their measured values.

変数	計測値（ms）（3.5 Mbps）	計測値（ms）（32 Kbps）
t_{query}	8.79	9.49
t_{comm}	0.50	0.45
t_{comp}	0.05	0.04
t_{dec}	0.17	0.15

表 3　$m = 2$ のときの属性値数にともなうパラメタ値 k の変化
Table 3　Transition of parameter k for number of attributes in case of $m = 2$.

属性値数	1,000	10,000	100,000	1,000,000	10,000,000
k の値	7	10	12	14	17

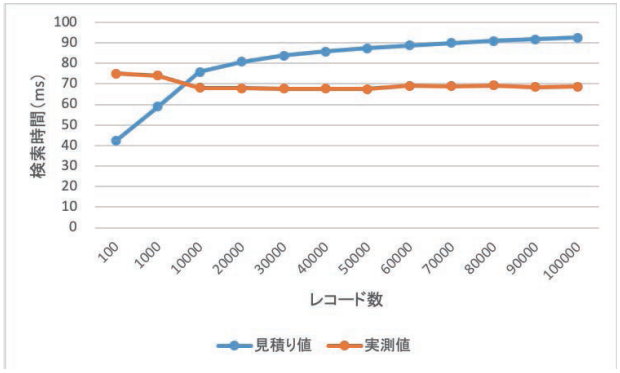


図 4　通信速度 3.5 Mbps での実行時間および見積り時間
Fig. 4　Execution time and estimation in case of 3.5 Mbps.

式 (11) の式を大まかにとらえると，$k \simeq m\log(N)$ と考えることができる．データ数が 10,000〜100,000 程度だと k は m の約 10 倍程度となる．また検索速度見積り式を大まかにとらえると $\frac{k\log(N)}{\log(m)}$ と考えることができ $k = 10m$ とすると $\frac{10m}{\log(m)} * \log(N)$ となることから，適切な m の値は 2 となることが分かる．k の値も $\log(N)$ で抑えられるため**表 3** に示すように 10 から 20 の間となり，通信回数も同様に 10〜20 回程度と見積もることができる．

次にパラメタ $m = 2$，$k = 7$ としたときの**図 4** に通信速度 3.5 Mbps での各レコード数での実行時間と見積り実行時間を示す．見積り値より実測値のほうが早くなっている理由は，見積り値の N が属性値数を用いている一方で実測値ではレコード数を用いていることが理由である．10,000 以上では属性値の異なり数が頭打ちになったためレコード数にかかわらず実行時間が一定となった．またデータ数が少ないときに見積りと比較して実行時間が高くなっているのは，データ数に対して k の値が大きくダミーによる負担が大きいことも原因として考えられる．しかしながらレコード数にかかわりなく約 65〜75 ms による検索を保証することができている．また見積り値も 10,000 件以上の場合は実測値の上限を見積もることができているといえる．

6.3　実験 2：既存手法との比較

本提案手法の検索速度を評価するため，以下の 2 種類の範囲検索が可能な暗号化検索スキームと検索速度を比較した．

(1) SDB [23]：索引を利用しない暗号化データベースシステムであり，データの相互運用性（interoperability）を持つ加法準同型暗号を提案している．秘密分散手法を利用し，非常に軽量な計算で選択演算を行うことができる．各レコードに対して条件に合致するか確認するため検索時間はレコード数に比例するが，通常の DBMS と同様 1 度のクエリに対してサーバとのやりとりは 1 度のみである．

(2) OIT [15]：大小比較が可能な紛失通信プロトコル GT-SCOT [3] を利用し，本提案手法と同様にクライアントが索引を探索する．B^+-tree や R^+-tree の探索も可能である．本手法と異なる点はクライアントが索引をすべて所有し，大小比較演算のためにサーバに 2 つの比較データを送って比較結果を受け取る手法をとっている．レコード数に対して探索回数は $\log N$ で抑えられるが，データ送受信量および GT-SCOT による計算に時間がかかるという問題がある．

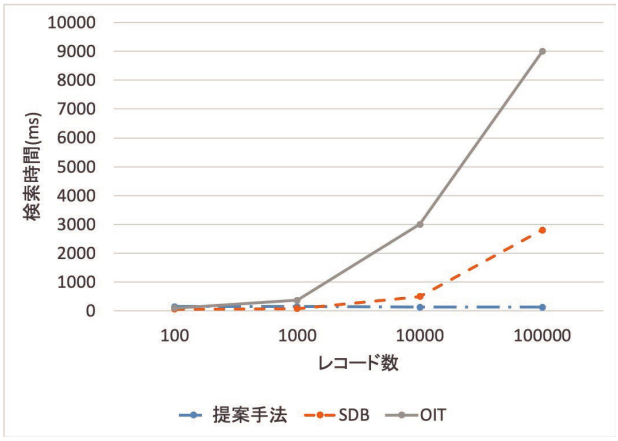


図 5　既存手法との実行時間の比較
Fig. 5　Processing time of proposed method compared to existing method.

比較対象としては，サーバ上に保存されたデータの安全性に関して提案手法と同等の安全性を保障する手法を対象としている．たとえば CryptDB [21] や Monomi [22] は範囲検索に順序保存暗号 [1] を用いており，安全性に問題があることがすでに知られている．同等の安全性を保障するものとして，サーバ上のレコード 1 つ 1 つに対し検索条件を満たすかどうかをチェックする手法のうちで演算が非常に軽量である SDB と，本提案手法と同様に索引を用いる手法として OIT を選択した．

図 5 に提案手法および SDB，OIT との検索時間を示す．既存手法はデータ量に従って検索時間が比例する形で増加する．特に OIT は本提案手法と同様に探索回数を $\log N$ で抑えられるものの，比較演算する配列の要素に対する紛失通信演算で非常に多くの検索時間をとられている．一方提案手法は，索引の要素はサーバに保管されており，クライアントからはハッシュ化されたアドレスのみを送ればよいため通信量が少なく，サーバ上の計算も非常に軽量であるため検索時間を大きく削減できていることが分かる．

6.4　実験 3：暗号化索引の要素のアクセス頻度

提案手法における各要素のアクセス頻度を検証する．本

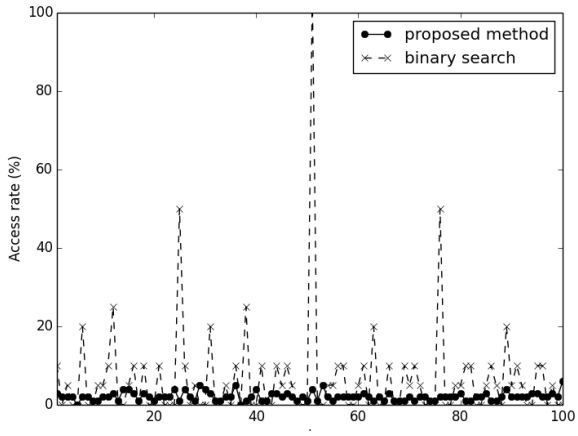


図 6　暗号化索引の要素のアクセス頻度
Fig. 6　Access frequencies to the elements of encrypted index.

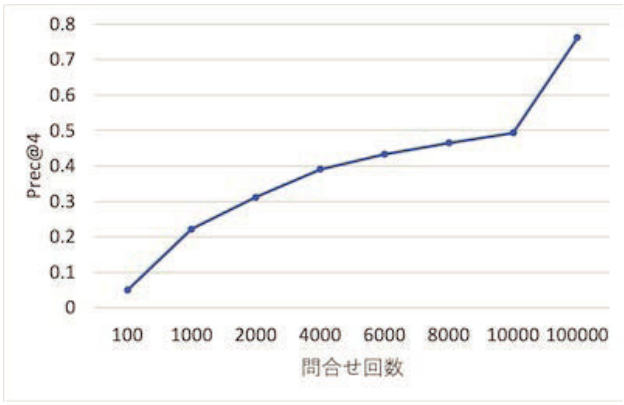


図 7　暗号化索引の要素のアクセス共起度
Fig. 7　Access co-occurrence of elements of encrypted index.

実験では，$m = 2$，$k = 10$ とした場合の各要素のアクセス確率について，提案手法と通常の二分探索を比較する．各要素のアクセス確率は，索引サイズ $N = 100$ のデータセットに対して，様々なクエリごとに 50 回ずつ検索を行い，そのときのアクセス回数から算出した．比較結果を**図 6** に示す．通常の二分探索では頻繁にアクセスする要素が集中しているため容易に暗号化前の配列上の位置が特定されることが分かる一方，提案手法ではどの要素もほぼ均等にアクセスされているため，アクセス頻度のヒストグラムからは暗号化前の配列状の位置を得るのが難しいことが分かる．

6.5　実験 4：暗号化索引の要素のアクセス共起度

提案手法において 1 回の問合せでアクセスされる要素の共起度から各要素の近接性が推測される恐れがある．特に探索範囲を絞り込む終盤の数ラウンドは共起の可能性が高まる．本実験では，$m = 2$，$k = 10$，属性値の異なり数を 1,000 としたときに各要素で最も共起する要素が近接する可能性について検証した．各要素に対して共起度の高い 4 要素を求め，それらがその要素と 4-近傍に含まれる割合を求めた．その結果を**図 7** に示す．横軸は問合せの回数を表し，縦軸は要素の周辺 4 要素に対する適合率を表す．図を見て分かるように，10,000 回近くなると適合率が 0.5 に近くなり，共起度から特定される可能性が高まることが分かる．これを防ぐ方法としては，問合せ 10,000 回程度を目処に，定期的にデータおよび索引を入れ替えたり，ノイズとしての擬似問合せを発行したりするなどが考えられる．

7.　まとめと今後の課題

暗号化データベースにおける安全で高速な検索を実現するため，順序関係を秘匿し各要素を暗号化した配列を索引とした検索フレームワークを提案した．本手法はクライアントが配列の位置を把握しており，クライアント自身が索引を探索するという手法をとる．1 回のクエリで 10 回程度のインタラクションが必要となるが，1 回の通信データ量が少ないため，サーバで各レコードに対して条件を満たすかどうかチェックするタイプの検索と比較して大幅に検索時間が短縮された．また，探索位置の摂動を行うことによって，アクセスログによる索引の要素の推測を困難にすることを示したが，問合せ回数が増えると要素のアクセス共起度によって推測の可能性が高まるため定期的な更新が必要であることも分かった．

本提案手法では，単純な選択演算のみを対象としているため，SQL をカバーできる他の演算に対しても演算方法を検討する必要がある．また，現時点では索引はすべてメモリ上に保管されていることを想定しているが，異なり数が大きな場合はディスクからの呼び出しも検討する必要がある．順序関係をサービスプロバイダに知らせずにディスク IO を抑える手法を検討する必要がある．また，本手法ではデータベースや索引の更新を想定していない．これを解決するためには更新の差分が攻撃者にとってヒントになるという問題を解決する必要がある．今後これらの問題についても取り組んでいきたい．

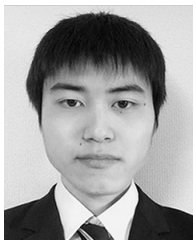
参考文献

[1] Agrawal, R., Kiernan, J., Srikant, R. and Xu, Y.: Order-preserving encryption for numeric data, *Proc. ACM SIGMOD International Conference on Management of Data*, pp.563–574 (2004).
[2] Amazon RDS, available from ⟨https://aws.amazon.com/jp/rds⟩.
[3] Blake, I.F. and Kolesnikov, V.: Strong Conditional Oblivious Transfer and Computing on Intervals, *Proc. ASIACRYPT*, pp.515–529 (2004).
[4] Boldyreva, A., Chenette, N., Lee, Y. and O'Neill, A.: Order-preserving symmetric encryption, *Advances in Cryptology - EUROCRYPT 2009, Proc. 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp.224–241 (2009).
[5] Boneh, D., Crescenzo, G.D., Ostrovsky, R. and Persiano, G.: Public key encryption with keyword search, *Advances in Cryptology - EUROCRYPT 2004, Proc. International Conference on the Theory and Applications*

of Cryptographic Techniques, pp.506–522 (2004).
[6] Boneh, D., Gentry, C., Halevi, S., Wang, F. and Wu, D.J.: Private database queries using somewhat homomorphic encryption, *Applied Cryptography and Network Security - Proc. 11th International Conference*, *ACNS 2013*, pp.102–118 (2013).
[7] Cloud Bigtable: Google Cloud Platform, available from ⟨https://cloud.google.com/bigtable/?hl=ja⟩.
[8] El Gamal, T: A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Information Theory*, Vol.31, No.4, pp.469–472 (1985).
[9] Ge, T. and Zdonik, S.B.: Answering aggregation queries in a secure system model, *Proc. 33rd International Conference on Very Large Data Bases*, pp.519–530 (2007).
[10] Gentry, C.: Fully homomorphic encryption using ideal lattices, *Proc. 41st Annual ACM Symposium on Theory of Computing*, pp.169–178 (2009).
[11] Gentry, C., Halevi, S. and Smart, N.P.: Fully Homomorphic Encryption with Polylog Overhead, *Proc. EUROCRYPT 2012*, pp.465–482 (2012).
[12] Kadhem, H., Amagasa, T. and Kitagawa, H.: A secure and efficient order preserving encryption scheme for relational databases, *KMIS 2010 - Proc. International Conference on Knowledge Management and Information Sharing*, pp.25–35 (2010).
[13] Hacigümüs, H., Iyer, B.R., Li, C. and Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model, *Proc. 2002 ACM SIGMOD International Conference on Management of Data*, pp.216–227 (2002).
[14] Hore, B., Mehrotra, S. and Tsudik, G.: A privacy-preserving index for range queries, *Proc. 30th International Conference on Very Large Data Bases*, pp.720–731 (2004).
[15] Hu, H., Xu, J., Xu, X., Pei, K., Choi, B. and Zhou, S.: Private search on key-value stores with hierarchical indexes, *IEEE 30th International Conference on Data Engineering*, *ICDE 2014*, pp.628–639 (2014).
[16] Lee, S., Park, T.-J., Lee, D., Nam, T. and Kim, S.: Chaotic order preserving encryption for efficient and secure queries on databases, *IEICE Trans.*, Vol.
[17] Ma, S., Yang, B. and Zhang, M.: PPGJ: A privacy-preserving general join for outsourced encrypted database, *Security and Communication Networks*, Vol.7, No.8, pp.1232–1244 (2014).
[18] Mykletun, E. and Tsudik, G.: Aggregation queries in the database-as-a-service model, *Data and Applications Security XX*, *Proc. 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pp.89–103 (2006).
[19] Oracle Enterprise Manager 12c, available from ⟨http://www.oracle.com/technetwork/jp/oem/enterprise-manager/overview/index.html⟩.
[20] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes, *Advances in Cryptology - EUROCRYPT '99*, *Proc. International Conference on the Theory and Application of Cryptographic Techniques*, pp.223–238, (1999).
[21] 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).
[22] Tu, S., Kaashoek, M.F., Madden, S. and Zeldovich, N.: Processing analytical queries over encrypted data, *PVLDB*, Vol.6, No.5, pp.289–300 (2013).
[23] Wong, W.K., Kao, B., Cheung, D.W.-L., Li, R. and Yiu, S.-M.: Secure query processing with data interoperability in a cloud database environment, *International Conference on Management of Data*, *SIGMOD 2014*, pp.1395–1406 (2014).
[24] 秋山賢人，渡辺知恵美，北川博之：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，第 8 回データ工学と情報マネジメントに関するフォーラム（DEIM 2016），F8-2 (2016).
[25] 秋山賢人，渡辺知恵美，北川博之：秘匿検索フレームワーク OSIT における最適なクエリプラン選択法，第 9 回データ工学と情報マネジメントに関するフォーラム（DEIM 2017），H6-5 (2017).
[26] 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，第 7 回データ工学と情報マネジメントに関するフォーラム（DEIM 2015），G2-6 (2015).
[27] 篠塚千愛，渡辺知恵美，北川博之：クラウド環境における暗号化索引を用いた文字列属性の部分一致検索手法，コンピュータセキュリティシンポジウム 2015 論文集，pp.979–986 (2015).
[28] 篠塚千愛，渡辺知恵美，北川博之：秘密計算による秘匿検索フレームワーク OSIT-bs における検索コストと最適化，第 8 回データ工学と情報マネジメントに関するフォーラム（DEIM 2016），F5-6 (2016).
[29] 篠塚千愛，渡辺知恵美，北川博之：暗号化データベースにおけるデータの秘匿性を保証した検索手法，DEIM Forum 2017, H6-4 (2017).

渡辺 知恵美　（正会員）

2003 年お茶の水女子大学大学院博士後期課程修了．博士（理学）．同年奈良女子大学助教，2005 年お茶の水女子大学講師，2013 年筑波大学助教，2017 年産業技術大学院大学准教授を経て 2018 年より筑波大学図書館情報メディア系准教授．データベースにおけるプライバシ保護検索技術の研究に従事．日本データベース学会，ACM 各会員．



秋山 賢人

2016 年筑波大学情報学群情報科学類卒業．2018 年同大学大学院システム情報工学研究科博士前期課程修了．在学中は，データベースにおけるプライバシを保護した検索技術の研究に従事．現在，セコム株式会社技術開発本部でソフトウェア開発に従事．日本データベース学会会員．

天笠 俊之　（正会員）

1999 年群馬大学大学院工学研究科修了．博士（工学）．奈良先端科学技術大学院大学情報科学研究科助手，筑波大学大学院システム情報工学研究科講師，同計算科学研究センター/システム情報系准教授を経て，2017 年より筑波大学計算科学研究センター教授．データベース，データマイニング等の研究に従事．日本データベース学会，ACM 各会員．電子情報通信学会，IEEE 各シニア会員．

北川 博之　（正会員）

1978 年東京大学理学部物理学科卒業．1980 年同大学大学院理学系研究科修了．日本電気（株）勤務の後，筑波大学電子・情報工学系講師，同助教授を経て，現在，筑波大学計算科学研究センター教授．理学博士（東京大学）．データベース，情報統合，データマイニング，情報検索等の研究に従事．電子情報通信学会フェロー，日本データベース学会監事，ACM，IEEE，日本ソフトウェア科学会各会員．本会フェロー．

（担当編集委員　山口 実靖）