DEIM Forum 2021 G33-1

Volume-Hiding暗号マルチマップにおけるデータ量秘匿を可能にする挿入の実現

石原　詢大†　　渡辺知恵美††　　天笠　俊之†††

† 筑波大学システム情報工学研究群情報理工学位プログラム　〒305-8577 茨城県つくば市天王台１丁目
†† 筑波技術大学 産業技術学部 産業情報学科　〒305-8520 茨城県つくば市天久保４丁目３－１５
††† 筑波大学計算科学研究センター　〒305-8577 茨城県つくば市天王台１丁目１－１
E-mail: †ishihara@kde.cs.tsukuba.ac.jp, ††chiemi@a.tsukuba-tech.ac.jp, †††amagasa@cs.tsukuba.ac.jp

あらまし　暗号化データベースにおける新たな脅威としてボリュームの漏洩と呼ばれるものが Kellaris ら [3] や Grubbs ら [4] によって報告された．ボリュームとは，マルチマップのように一つのキーに対して複数のデータが紐づけられているようなデータ構造において，キーに紐づけられているデータの個数のことであり，データ構造を暗号化しただけでは，キーのボリュームのヒストグラムが知られてしまい，ボリュームの分布から最頻値などの情報を知られてしまう可能性がある．Sarvar ら [5] の手法では，差分プライバシー [2] を満たすようにそれぞれのキーのボリュームにノイズを加えることでボリュームの秘匿を実現している．しかし，Sarvar ら [5] の手法は，データの更新には適していない．そこで本研究では，ボリュームを秘匿しつつ，データの更新が行えるデータベースを実現することが目的である．基本的なアイディアとしては，局所差分プライバシー を用い，データ挿入時にノイズを加えることによって，サーバー側にボリュームを秘匿したままデータの挿入を達成する．また，ORAM [16] を適用することで，データの挿入箇所の秘匿を実現する．

キーワード　暗号化データベース，プライバシー保護，セキュリティ, Encrypted Database, Security and Privacy

1　背　　景

サーバー管理の負担の軽減，導入や運用にかかるコストの削減，データの一元管理などのメリットからクラウドデータベースサービスが普及している．このように，他の人や組織（第三者）が管理しているサーバーにデータを保存することが増えた．しかし，第三者のサーバーを利用している以上完全に信頼できるわけではない．もしかすると，管理者の中にはデータの中身を見ている人がいるかもしれない．そのような場合，個人情報の漏洩やプライバシー侵害などの問題につながってしまう．そこで，一般的には，データ所有者（クライアント）はデータを暗号文にしてからクラウドデータベースサービスなどの第三者のサーバーに預けるといったことを行う．これにより第三者のサーバーにデータの中身を見られないまま（暗号文のまま）データの管理をすることができる．これを暗号化データベースと呼ぶ. [15]

しかし，データを暗号化しただけでは安全ではないことが知られている [7]．例えば，アクセスパターンの漏洩による脅威が知られている．アクセスパターンとはデータへのアクセスのログのことであり，統計的推論を通じて，平文（もとのデータ）に関する大量の機密情報を漏らす可能性があることが報告されている．Isram ら [7] は，暗号化された電子メールリポジトリへのアクセスを監視することにより，検索クエリの 80%を推測できることを実証した．

また，暗号化データベースにおける新たな脅威としてボリュームの漏洩が Kellaris ら [3] や Grubbs ら [4] によって報告された．マルチマップと呼ばれるマップや連想配列を一般化したデータ構造がある．マルチマップでは，キーに対して複数の値が紐付けることができる（つまり，キーの重複が許される）．マルチマップは，例えば，著者をキーにして同じ著者の本を格納することができる．ボリュームとは，マルチマップのようなデータ構造において，キーに紐づけられているデータの個数のことである．このような場合，データ構造を暗号化しただけでは，キーのボリュームのヒストグラムが知られてしまい，ボリュームの分布から最頻値などの情報を知られてしまう可能性がある．Sarvar ら [5] の手法では，差分プライバシー [2] を満たすようにそれぞれのキーのボリュームにノイズを加えることでボリュームの秘匿を実現している．しかし，Sarvar ら [5] の手法では，ボリューム へのノイズ付加はクライアント側が全てのデータに対して DP を保証するように行う必要がある. そのため， データを更新する際は，一度全てのデータをクライアントに集め，再度ノイズを加える必要があり，データの挿入に適していない．

そこで，本研究では，検索されたキーのボリュームの秘匿に加え，

- 更新操作後もボリュームの秘匿が維持されること
- データの更新をした場合に，どのキーに対して更新操作が行われたのかを秘匿すること

が目的である．

現段階では，更新のうち挿入のみを想定している．また，挿入の際の同時実行制御は考えない．基本的なアイディアとしては，局所差分プライバシーを用い，データ挿入時にノイズを

加えることによって，サーバー側にボリュームを秘匿したままデータの挿入を達成する．局所差分プライバシーとは，データに対しノイズを加えてプライバシー保護を実現するための安全性指標である（2.2 節を参照）．また，局所差分プラバシーを適用しただけでは，更新箇所が知られてしまう可能性があるので，ORAM [11] という手法を適用する．その際に先行研究である Sarvar ら [5] が検索高速化のために使っている Cackoo hashing を拡張した ORAM を適用する [11].

2 事 前 知 識

2.1 差分プライバシー

統計量開示のプライバシー保護のアプローチとして広く用いられているものの一つに Dwork [2] が提案した差分プライバシー（Differential Privacy, DP）がある．これは，公開された統計量からはデータベース中の 1 レコードのみ異なる 2 つのデータベース（隣接データベースという）について確率的にしか区別できないことを保証する安全性の指標である．

直感的な理解としては，”あるデータが含まれていても含まれていなくても出力（統計量）に差分がない”ことを保証する．つまり，”そのデータがあってもなくても出力が変わらなければ，そのデータのプライバシーは含まれていない”という考えに基づいている．

差分プライバシーでは，統計量にノイズを加えることによってプライバシーを保護する．では，どのようにノイズを加えれば差分プライバシーが保証されるのであろうか．ϵ-差分プライバシーでは，次のように定義されている．

$\forall D, D'$ は互いに 1 レコードのみ異なる任意のデータベースとする．また，q はクエリの結果を出力し，m はノイズを加えるメカニズムとする．Y は出力値の集合とし，$S \subseteq Y$ はその部分集合とする．このとき，

$$\frac{Pr(m(q(D)) \in S)}{Pr(m(q(D')) \in S)} = exp(\epsilon) \quad (\epsilon > 0) \tag{1}$$

を満たすならば，メカニズム m は ϵ-差分プライバシーを保証するという．ここで，ϵ はプライバシーバジェットと呼ばれるパラメータであり，小さいほど強いプライバシー保護を実現する．

2.2 局所差分プライバシー

通常の差分プライバシーでは，データ収集者がデータ提供者の生データを収集し，統計量の計算後，ノイズを加えることでプライバシーを保護している．一方で，局所差分プライバシー（Local Differential Privacy, LDP）[8] は，統計量の計算をする前に，データにノイズを加えることによってプライバシーを保護する．これは，データ提供者がデータ収集者にデータを渡す前にプライバシーが保護されているので通常の差分プライバシーよりも安全性の高いプライバシー保護を実現する．

LDP では，データ提供者がランダム化されたデータを送信することで，データ収集者は提供者の真のデータを知ることができない．しかし，収集者はメカニズムを通じて収集したデータの統計量は知ることができる．そのため，たとえ収集者に悪意があったとしても提供者のプライバシーを保護したまま統計量のみを公開できる．

2.3 ORAM

Oblivious Random Access Machine (ORAM) は，暗号データに対してアクセスが行われる度に，その暗号データの格納位置を変化させることにより，アクセスパターンを秘匿する技術である．サーバーの管理者や外部の攻撃者は，どのデータがアクセスされたかということや検索頻度，さらにデータ間の関係も知ることはできない．ORAM の具体例として，E.Stefanov ら [16] による path ORAM という手法について説明する．path ORAM では，サーバに預けた木構造を変化させることでアクセスパターンを秘匿する．用いられるデータ構造は，二分木（多分木でもよいが今回は簡単のため二分木で説明をする），スタッシュ，Position Map の 3 つであり，二分木はサーバの記憶領域に置かれ，スタッシュと Position Map はクライアントの記憶領域に置かれる．まず，それぞれのデータ構造について説明する．

2.3.1 二 分 木

深さ $L+1$，葉の枚数 2^L の二分木を用意する．ノード数を N とすると，$L = \lceil log_2 N \rceil - 1$ である．二分木の根を，レベル 0 とし，葉をレベル L とする．各ノードは Z 個のブロックを保持する．実際の格納データが Z よりも少ない場合は，ダミーデータがブロックに格納される．葉には，左から順に 0 から $2^L - 1$ までの ID がつけられ，根から葉 x までの経路上に含まれるノードを P(x) と表記する．経路上のノードが保持するブロックの集合をパスと呼ぶ．path ORAM では，一回のアクセスにおいて，1 つのパスへの読み出しと書き戻しを伴う．パスに含まれるブロック数は (L+1)Z である．

2.3.2 スタッシュ

スタッシュは，二分木から読み出され，複合されたデータを保持する．path ORAM のアクセスに先立って，スタッシュにはパス 1 つ分すなわち (L+1)Z 個の空きが必要となる．また，書き戻せなかったブロックはスタッシュに残される．もしスタッシュに残されたブロックが多く，空きブロックが不足してしまうと，読み出しの際にスタッシュに格納しきれなくなるおそれが生じる．この状況を破綻（Failure）と呼ぶ．スタッシュのサイズ（ブロック数）は破綻の可能性を考慮して定める必要がある．具体的には，パラメータ λ を定め，破綻をきたす確率 $2^{-\lambda}$ がを下回るようにスタッシュサイズを定める．

2.3.3 Position Map

実データの格納されるブロックがそれぞれどのパス上に属するかを表す表である．具体的には，ブロックが属するパスに対応する葉の ID を保持している．

path ORAM のアクセスの手順は，以下の 5 つの手順から成る．ただし，外部記憶領域へのアクセス時にはデータの暗号・複合を行う

（1） スタッシュの探索：スタッシュを探索し，対象ブロックがあれば，そのブロックにアクセスして終了する．この場合，外部記憶領域へのアクセスは生じない．

（2） Position Map の読み出しと更新：対象ブロックが属する葉の ID を取得し，0 から $2^L - 1$ までのランダムな番号に置き換える．取得した葉の ID を x とおく．

（3） パスの読み出し：$P(x)$ に属するノードの全てのブロックを読み出し，ダミーでないブロックをスタッシュに移動する．

（4） 対象ブロックへのアクセス：スタッシュ に移動された対象ブロックにアクセスする．

（5） パスの書き戻し：$P(x)$ に属する頂点に対し，葉から順に以下の処理を繰り返す．まず，スタッシュ 内のブロッ クのうち，そのブロックの属するパスが対象の頂点を含むものを抽出する．もし Z 個以上のブロックが抽出されたら，そのうち Z 個を選び，対象の頂点に書き戻す．そうでなければ，抽出されたブロックを対象の頂点に書き戻し，不足分にはダミーブロックを書き込む．

パスの書き戻し処理は，直感的には，読み出したブロックをなるべく葉に近い位置に書き戻す，とも表現できる．以上の手順の繰り返しを外部から観察するともはやランダムなパスへの読み書きを繰り返しているようにしか見えない．これにより，アクセスパターンを秘匿できる．

path ORAM のアクセスの動作例を図 1 を用いながら説明する．葉のレベルを $L = 2$，各ノードのブロック数を $Z = 2$ とする．実データのブロックを A, B, C, ... , G とし，他はダミーブロックとする．実データのうち，A と B は根ノード，C は葉 0，D は葉 1 に置かれており，ID はそれぞれ 3,1,0,1 が割り当てられている．ブロック C に対するアクセスが実行されたとし，このうち該当ブロックへのアクセスが完了したとする．このときの状態を図 2 に示す．ブロック C は スタッシュ には存在しないので Position Map を読み出し，葉の ID として 0 を得る．Position Map の該当エントリはランダムな値 (ここでは 2) で更新される．ブロック C はグレーで示す $P(0)$ 上のどこかにあるので，このパスに含まれるブロックを読み出す．その結果，ブロック A，B，C が スタッシュ に移動され，ブロック C へのアクセスが可能になる．つづいて，パスの書き戻しを完了したときの状態を図 3 に示す．まず，葉 0 を含むパスをもつ，すなわち対応 する葉の ID が 0 であるブロックを スタッシュから探索する．そのようなブロックは存在しないので，葉 0 には 2 つのダ ミーブロックが書き込まれる．次に，レベル 1 の左の頂点は $P(0)$ と $P(1)$ に含まれるので，対応する葉の ID が 0 または 1 であるブロックを探索する．これには B が該当するので，この頂点には B とダミーブロックが書き込まれる．最後に，根は全てのパスに含まれるので，根には任意のブ ロックを書き込める．したがって，残る A と C は根に書 き戻される．今回は全てのブロックが書き戻されたが，最終的に書き戻せないブロックがあれば，それらはスタッシュに残される．

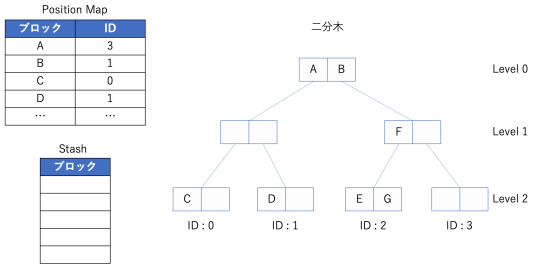

図 1　path ORAM のアクセス動作例 (a)

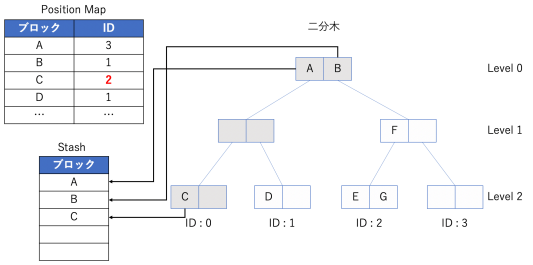

図 2　path ORAM のアクセス動作例 (b)

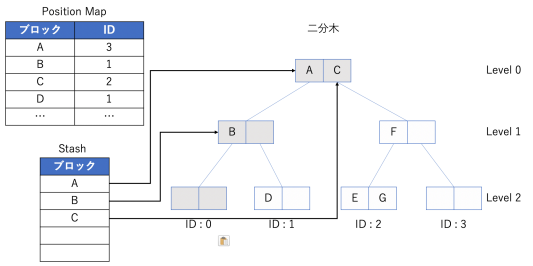

図 3　path ORAM のアクセス動作例 (c)

2.4　Cuckoo Hashing

Cuckoo Hashing は Pagh と Rodler [17] によって提案されたハッシュテーブルで，ハッシュ値の衝突を回避することと検索においては必ず定数時間で済むことが保証されることを目的として作られたハッシュテーブルである．cuckoo hashing は，構築と検索の 2 つのアルゴリズムから成る．まず，構築について説明する．Cuckoo hashing では，2 つのハッシュ関数 h_1,h_2 と 2 つのテーブル T_1,T_2 を用いる．ハッシュテーブルにデータ x を挿入することを考える．まず，T_1 上の $h_1(x)$ の位置が空いているかどうかを調べる．空いている場合は，データ x をそこに挿入し操作を完了する．もし，すでにそこに別のデータ y が挿入されている場合は，データ y を追い出してデータ x を T_1 上の $h_1(x)$ に挿入する．追い出されたデータ y については，T_2 上の $h_2(y)$ が空いているかどうかを調べ，空いている場合はそこい挿入して操作を完了する．もし，すでにその場所にも別のデータ z が挿入されていた場合は，同じように操作を繰り返す．このように,「テーブル上のハッシュ値の位置が空いているかどうかを調べ，先約がいれば，その値を追い出して，もう一方のテーブル上のハッシュ値の位置に挿入する」という繰り返しの操作をこの論文では，Cuckoo 操作と呼ぶことにする．

このようにして作られたハッシュテーブルでは，データ x は $T_1[h_1(x)]$ または $T_2[h_2(x)]$ のどちらかに必ず入っていることが保証される．よって検索に関しては，定数時間がで済ませることが可能となる．構築の際に，運が悪いと Cuckoo 操作に非常に長い時間がかかる，または，終わらない可能性がある．そのため，Kirsch, Mitzenmacher, Wieder の 3 人はスタッシュを用いた Cuckoo Hashing [18] という手法を提案した．この手法では，まずスタッシュの容量の大きさを予め決めておく．全体の要素の数を n とし，追い出し操作を $\log n$ 以上繰り返してもテーブルに入らない値がある場合は，スタッシュに格納するという手法である．また，スタッシュのサイズを s としたときに，スタッシュに格納される要素数が s を超える確率は $O(n^{-s})$ になるとされている．この証明については, [18] の Theorem2.1 を参照していただきたい．

3 先 行 研 究

信頼できない第三者のサーバーにクライアントが構築したデータベースの管理を委託するための暗号化スキームとして Structured encryption(STE) が Chase と Kamara [12] によって提案された．STE では，データ所有者は，委託したデータベースを暗号化したまま操作できるためサーバーにデータの情報を明かさずに委託することができる．STE のひとつに暗号化マルチマップ (encrypted Multi Map, EMM) [13] がある．EMM は，キーに複数のシーケンシャルな値が紐づいているデータ構造の暗号化スキームである．EMM では暗号化したままキーに紐づいている全ての値の読み出しが可能である．

しかし，上記のような暗号化スキームだけでは，アクセスパターンやボリュームが漏洩しており，機密情報を推定されることが Isram ら [7] の研究によって指摘された．また，EMM への新たな脅威として，クエリされたキーのデータ量（ボリューム）の漏洩による危険性が Kellaris ら [3] や Grubbs ら [4] によって報告された．

そこで，Kamara と Moataz [14] や Sarvar ら [5] はボリュームを秘匿した EMM の手法を提案している．本研究では，特に Sarvar ら [5] の手法を基盤としている．

3.1 Sarvar らの研究

ボリュームを秘匿するナイーブな手法としては，全てのキーのボリュームを一番ボリュームの大きいキーに合わせるようにそれぞれのキーにダミーデータを挿入することでボリュームを秘匿することが可能である．しかし，この方法では，ボリュームが小さいキーを検索する際に毎回最大のボリュームをクライアントに送信する必要があるので通信容量のオーバヘッドも大きい．図 4 は，ナイーブな手法のイメージである．

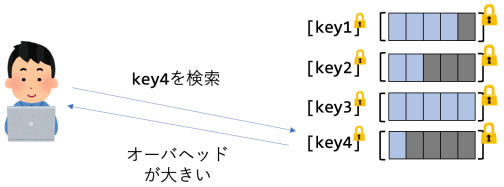

図 4　Volume leakage に対する Naive な手法

そこで，Sarvar ら [5] の研究では，DP を保証するようにノイズを加えることで効率の良い通信容量を実現する手法を提案している．以下で，Saevar ら [5] の手法の大まかな説明をする．

通信容量を軽減するために，全てのキーのボリュームの大きさを最大値に合わせるのではなく，それぞれのボリュームにノイズを加えることで，ボリュームが小さいキーの場合における通信容量の軽減をしている．まず，データをサーバーに送信する前にそれぞれのキーのボリュームに対してノイズを加える処理を行う．ボリュームは，それぞれのキーのヒストグラム（統計量）とみなすことができる．そのため，ノイズを加えるときは，上記で説明した DP が満たされるようにすることで，安全性が高くなるようにノイズを加えられる．図 5 は，Sarvar ら [5] のイメージである．

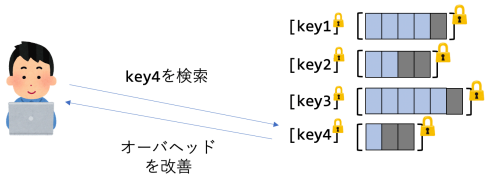

図 5　Sarvar らの手法

また，Sarvar ら [5] の手法では，検索速度の高速化とサーバー側のストレージ容量の削減のために Cuckoo Hashing が使われている．検索速度については 2.4 節で説明した通り定数時間で行える．ここでは，どのようにサーバー側のストレージ容量を削減しているのかを説明する．まず，サーバー側とクライアント側で使用されるデータ構造を説明する．

3.1.1 サーバー側

Data Table: 2 つの配列からなる Cuckoo Hashing テーブル である．このテーブルには暗号化されたキーと値が格納される．また，空いている場所にはダミーデータが格納される．

Count Table: このテーブルには DP が保証されたそれぞれのキーのボリュームが暗号化されて格納される．データを読み込む際は，クライアント側は，まずこの Count Table から真のボリュームを読み出す．その後，読み出したボリュームを基に，Data Table を探索する．

3.1.2 クライアント側

Stash（スタッシュ）: Data Table に格納しきれなかったキーと値を格納する．

3.1.3 ストレージ容量の削減

次に，どのようにサーバー側のストレージ容量の削減を実現しているかを説明する．ナイーブな手法では，キーごとに最大ボリュームの大きさのブロックが用意されており，それぞれのブロックの空いている箇所にはダミーデータを格納する．検索の際には，ブロックごと読み込むことでボリュームの秘匿を実現する．そのため，ナイーブな手法では，キーごとに最大ボリュームの大きさのブロックが必要なためサーバー側のストレージ容量の効率が悪い．具体的には，キーの数を $n = |K|$ とし，最大ボリュームを m としたとき，$O(nm)$ のストレージ容量となる．一方で，Sarvar ら [5] の手法では，キーに紐づく値にはそれぞれインデックスがつけられており，値ごとに異なるハッシュ値になるようにキーとインデックスを組み合わせて Cuckoo Hashing のハッシュ値が生成される．これにより，キーに紐づく値は 2 つのテーブル上のそれぞれ違う位置に格納される．検索する際は，Count Table から検索するキーのボリュームを読み込む．その後，読み込んだボリューム分のデータを Data Table から検索する．このとき，ボリュームには DP が保証されるようにノイズが加わっており，真のボリュームよりも大きく設定されている．キーに紐づく値のインデックスに加え，ノイズ分の偽のインデックスを検索する．つまり，真の値が格納されているハッシュ値とは関係ないハッシュ値も検索することによって，ボリュームの秘匿を実現している．このとき，サーバー側のストレージ容量は Data Table と Cuckoo Hashing に用いる配列 2 つの大きさと Count Table の大きさになる．全体のデータ量を d としたときに，Data Table の配列の大きさは，$(d + \alpha)$ とする．ここで，$\alpha > 0$ である．また Count Table の大きさは，キーの数 $n = |K|$ となるので，全体のストレージ容量は $O(d + n)$ となる．

図 6 は，Sarvar らの手法における検索のプロセスを表している．図 6 では，key4 を検索する場合を例としている．クライアントは key4 を検索する際にまず，Count Table から key4 のボリュームを読み込む．次に，取得したボリュームををもとに Data Table を検索する．検索する際は，ハッシュ関数 h を用いて $h(key4+0)$, $h(key4+1)$, $h(key4+2)$ のように取得したボリューム分のハッシュ値を生成する．ここで，key4 が格納されている位置はハッシュ値 $h(key4+0)$ のみであり，残りのハッシュ値 $h(key4+1)$, $h(key4+2)$ は key4 とは関係ないダミーのデータを検索する．このようにダミーの検索結果を混ぜることでボリュームの秘匿を行う．

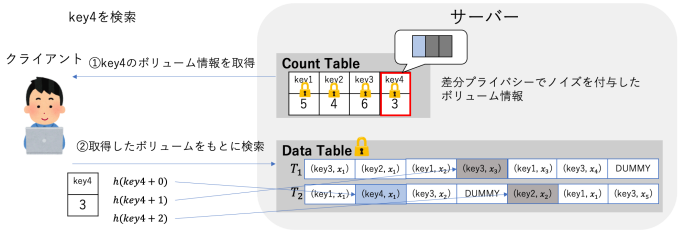

図 6　Sarvar らの手法における検索のプロセス

3.2 先行研究の問題点

Sarvar ら [5] の手法では，ノイズを加えるときに，全てのデータがクライアントの手元にある必要がある．そのため，新たなデータを挿入する場合は，一度クライアントの手元に全てのデータを戻し，再度 DP を満たすようにボリュームにノイズを加える処理を行う必要がある．

4 提 案 手 法

4.1 本研究で取り組む課題

研究の前提として，データの更新については挿入のみを考える．また，同時実行制御は考えない．本研究目的を達成するための課題は以下の通りである．

- DP を保証しながらデータを挿入すること
- どのキーに対してデータを挿入したのかを知られないようにすること

4.2 概　　要

ここでは，本研究の提案手法の概要を説明したい．Sarvar らの手法 [5] では，DP（差分プライバシー ）を保証するようにボリュームにノイズを加えることによってボリュームの秘匿を実現している．そのため，データの挿入を可能にするには，挿入後も DP が保証されている必要がある．そこで，我々はデータを挿入するときに LDP（局所差分プライバシー）を用いることを考えている．つまり，データの挿入時に嘘の挿入情報を混ぜることによってボリュームの秘匿を実現する．しかし，データ挿入を観察された場合に，どの箇所に対してデータが挿入されたのかが知られてしまい，キーが推測される可能性がある．そこで，我々は，ORAM を用いてデータの格納位置を変更することによってキーの推測をさせないようにすることを考えている．

4.3 データ構造

まず，サーバー側とクライアント側で使用するデータ構造とそれぞれの役割について説明する．

4.3.1 サーバー側

Data Table: 2 つの配列からなる Cuckoo Hashing Table である．このテーブルには暗号化されたキーと値が格納される．また，空いている場所にはダミーデータが格納される．

Count Table: このテーブルには DP が保証されたキーのボリュームが暗号化されて格納される．Count Table の役割は，Sarvar ら [5] の Count Table と同じである．データを読み込む際は，クライアント側は，まずこの Count Table から真のボリュームを読み出す．その後，読み出したボリュームを基に，Data Table を探索する．一方で，データを更新する際は，後ほど説明する LDP を用いた手法によってボリュームにノイズを加えたものに値を更新する．

4.3.2 クライアント側

Copy Table: このテーブルは，Cuckoo Hashing テーブルであり，データを更新する際に，サーバーにある Data Table

のコピーを受け取る．クライアント側はこのコピーされた Data Table を用いて，後に説明するデータの挿入と Cuckoo 操作のシミュレーションを行う．そして，データの挿入後は破棄される．

Update List: このリストは，データを更新する際に使用される．クライアント側で行ったシミュレーションの結果，変更される箇所がリストアップされたリストである．サーバー側はこのリストに従い，Data Table の更新を行う．

Stash（スタッシュ）： スタッシュでは，Data Table に格納しきれなかったデータを挿入する．

4.4 局所差分プライバシーを用いたデータの挿入

Sarvar ら [5] の研究では，全てのキーにに対して，DP（差分プライバシー）が保たれるようにノイズを加えていたが，データを更新する際に，LDP を保証するようにそれぞれのキーのボリュームを更新することで，ボリュームの秘匿が達成できるという考えに基づいている．具体的には，Randomize Response [6] を適用する．キーの集合を K，としたときに以下の式 (2) に従って $key \in K$ を選び次の操作を行う．摂動確率 $p \in [0,1]$ において

$$key = \begin{cases} key_{true} & with\ probability\ p \\ key_{dummy} & with\ probability\ (1-p) \end{cases} \tag{2}$$

ここで，key_{true} は真のデータを挿入するキー，key_{dummy} はボリュームにノイズを加える K から一様ランダムに選んだキーである．提案手法では，確率 p で真のデータが挿入されるキーが選ばれるまで，Randomized Response を繰り返し，得られたすべてのキーについて Count Table のボリュームを更新する．

つまり，関係ないキーのボリュームをインクリメントすることで，どのキーに対してデータが挿入されたのかを知られないようにする．また，Randomized Respoce は DP を保証することが知られているので，挿入後もボリュームの秘匿を維持することが可能となる図 7 は，key_4 に対する挿入操作の秘匿を表している．ここでは，key_4 の真のデータの挿入に対して，Count Table の key_1 と key_2 のボリュームを Randomized Response によってインクリメントしている．

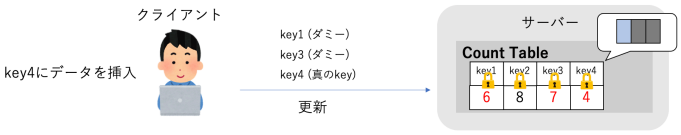

図 7　LDP を用いたデータの挿入例

4.5 Cuckoo 操作による挿入箇所の秘匿

LDP を用いて挿入操作を行うだけでは安全ではないと考えられる．例えば，図 7 において，その後も運悪く $key4$ への挿入が続いてしまった場合，$key4$ への挿入が漏洩してしまうことが考えられる．そのため，同じキーへの挿入が続いてもどのキーに対して挿入が行われたのかを知られないようにする必要がある．これは，挿入箇所の秘匿となる．挿入箇所の秘匿には，2.3 節で紹介した ORAM [9] を 2.4 節で説明した Cuckoo 操作を適用して実装する．Cuckoo Hashing では，テーブル上のハッシュ値が衝突をしてしまった場合，Cuckoo 操作を行うことで，衝突を回避する．ここでは，この Cuckoo 操作を挿入するデータの格納だけでなく，ORAM の機能として用いる．具体的には，テーブルに格納されているデータをランダムに 1 つ選び，データの格納位置を強制的にもう一方のテーブルのハッシュ値の位置に入れ替え，Cuckoo 操作を行う．これにより，Cuckoo 操作の影響を受ける位置は，挿入されたデータによるものだけでなく，ランダムに選ばれたデータによるものも含まれるため，データの挿入箇所の秘匿ができる．以下のデータ挿入のプロセスにおいて具体的に説明する．

4.6 挿入時のプロセス

ここでは，新しくデータが挿入されるときのプロセスを説明する．

（1） クライアント側で，真のデータが挿入されるキーが選ばれるまで，Randomized Response を繰り返す．

（2） (1) で得られたキーのボリュームをサーバーの Count Table から取得し，複合する．

（3） (1) で得られたキーのボリュームをすべてインクリメントし，暗号化してからサーバーの Count Table に戻す．

（4） サーバーから Data Table を取得する（Copy Table に保存する）．

（5） 真のデータを挿入する．挿入する際に，Copy Table を用いて Cuckoo 操作をシミュレーションし，変更箇所を Update List に追加する．また，Cuckoo 操作を $\log n$ 回行って，Table に格納しきれなかったデータはスタッシュに格納する．

（6） Copy Table から格納されているデータをランダムに選び，データの格納場所を強制的にもう一方のテーブルのハッシュ値の位置に入れ替え，Cuckoo 操作をシミュレーションする．Cuckoo 操作の回数は $[1,5]$ を毎回ランダムに選び，操作を行う．(5) と同様に，変更箇所は Update List に追加し，Table に格納しきれなかったデータはスタッシュに格納する．

（7） Update List をサーバーに送り，Copy Table を破棄する．

（8） サーバーは，Update List を受け取り，Data Table の変更箇所を更新する．

5 実　　験

実験では，4.6 節の (4) から (8) におけるプロセスの処理時間を測定した．本実験では，サーバーとクライアントはすべてローカルに置いて実験を行っている．また，暗号処理は行っていない．実験環境は，プロセッサ 3.1 GHz デュアルコア Intel Core i7, メモリ 16 GB 1867 MHz DDR3 である．言語は Java11 を使用している．

図 8, 図 9, 図 10 は Cuckoo Hashing のハッシュテーブルサイズを 100000 に設定して，10 万個の挿入操作をしたときのプ

ロセス (4), プロセス (5)(6), プロセス (7)(8) の処理時間をグラフにしたものである．図 9 から Cuckoo 操作にかかる時間は最大でも 2.5(ms) であり，現実的な時間で処理が行えることがわかる．

図 11 は，ハッシュテーブルサイズを 100000, 50000, 33333 にした時の 10 万個のレコードを挿入した時の平均処理時間を比較したものである．この比較は，ハッシュテーブルサイズが小さいほど，Cuckoo 操作のシミュレーションにかかる時間が増加することを懸念したものであるが，表??からシミュレーションにかかる時間はさほど変化しないことがわかった．また，シミュレーション時間よりも Data Table をコピーする時間のほうがハッシュテーブルのサイズに比例して減少するため全体の処理時間としては小さくなることがわかる．また，図 12 は，10 万行の人工データを Data Table のサイズが 10 万（人工データと同じ）場合，その半分のサイズの場合，1/3 のサイズの場合，それぞれのときに逐次挿入したときの最大データ量である．実験結果から，更新箇所のデータ量はテーブルサイズに反比例することがわかる．これは，テーブルサイズが小さい方が，ハッシュ値の衝突が起こりやすくなるため，更新箇所が多くなるためだと考えらる．しかし，データ量の場合も Data Table を丸ごとダウンロードするデータ量の方が圧倒的に大きいので，テーブルサイズが小さい方が全体的なデータ量は小さくなっている．

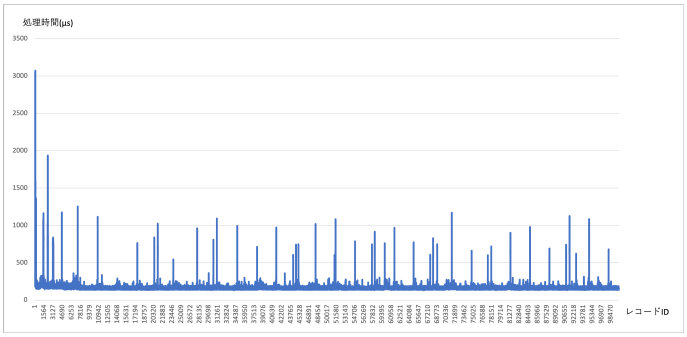

図 8　プロセス (4) Data Table のコピーにかかる時間

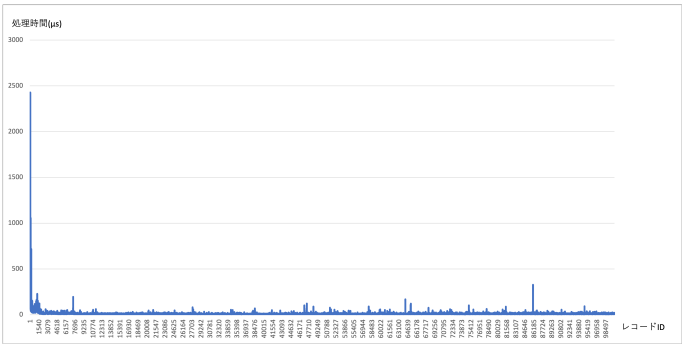

図 9　プロセス (5)(6) Cuckoo 操作のシミュレーションにかかる時間

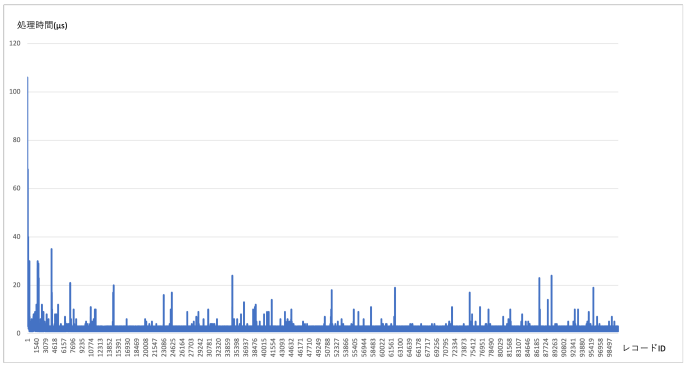

図 10　プロセス (7)(8) Data Table のアップデートにかかる時間

N = 100000 : データの数

処理内容	Data Table サイズ		
	N	N/2	N/3
コピー	165.68	81.29	**56.48**
シミュレーション	**57.43**	70.92	82.94
更新	2.22	2.30	2.30
合計	225.33	154.51	141.72

単位（μs）

図 11　10 万個の挿入操作におけるハッシュテーブルサイズ 100000, 50000, 33333 の場合のそれぞれの平均処理時間 (μ s) 間

N = 100000 : データの数

処理内容	Data Tableのサイズ		
	N	N/2	N/3
ダウンロードするデータ量	1000000	500000	333330
更新箇所のデータ量	**918**	1734	2700
合計	1000918	501734	336033

単位（byte）

図 12　10 万個の挿入操作におけるハッシュテーブルサイズ 100000, 50000, 33333 の場合のそれぞれの最大データ量 (byte)

6　まとめと今後の課題

データ挿入の際に局所差分プライバシーを適用することで，ボリュームの秘匿を維持し，Cuckoo 操作を強制的に行うことで挿入箇所を秘匿することにより，情報漏洩を最小限に抑えたデータ挿入の手法を提案した．

今後の課題として，データ通信量の削減と同時実行制御がある．今回の手法では，データ挿入時に一度サーバー側の Data Table を全てコピーして受け取り，クライアント側で Cuckoo 操作のシミュレーションを行っているため，データの通信量が非常に大きい．そのため，Data Table 全体をコピーするのではなく，Cuckoo 操作の影響を受ける部分のみをコピーして受け取ることが必要である．また，今回の手法では，複数のクライアントから同時にデータ挿入が行われることは考えていないため，今後は同時実行制御も考慮したデータ挿入を実現したいと考えている．

謝　　辞

本研究はJSPS科研費 JP19K11978の助成を受けたものです。

文　　献

[1] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More Robust Hash- ing: Cuckoo Hashing with a Stash. SIAM J. Comput. 39. 2009

[2] C. Dwork: Differential privacy; Proc. 33rd Intl. Conf. Automata, Languages and Programming - Volume Part II, Vol. 4052 of LNCS, pp. 1–12 (2006)

[3] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O ' Neill. Generic Attacks on Secure Outsourced Databases. In ACM CCS 16. 2016.

[4] Paul Grubbs, Marie-Sarah Lacharite, Brice Minaud, and Kenneth G. Paterson. Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018.

[5] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, Moti Yung. Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing.CCS '19: ACM SIGSAC Conference on Computer and Communications Security. November 2019.

[6] C.Dwork and A.Roth. The Algorithmic Foundations of Differential Privacy. Foundations and Trends in Theoretical Computer Science Vol. 9, 2014

[7] M. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In Network and Distributed System Security Symposium (NDSS), 2012.

[8] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? SIAM Journal on Computing, Vol.40, No.3, pp.793–826, 2011.

[9] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In STOC, 1987.

[10] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. J. Algorithms 51.2004

[11] Michael T. Goodrich, Michael Mitzenmacher. Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. Proceedings of the 38th international conference on Automata, languages and programming - Volume Part II. July 2010.

[12] Melissa Chase and Seny Kamara. 2010. Structured encryption and controlled disclosure. In EUROCRYPT ' 10. Springer, 577–594.

[13] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky.2011.Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security (2011).

[14] Seny Kamara and Tarik Moataz. 2019. Computationally Volume-Hiding Struc- tured Encryption. In EUROCRYPT 2019, Yuval Ishai and Vincent Rijmen (Eds.). 183–213.

[15] Hakan Hacıgümüş, Bala Iyer, Chen Li and Sharad Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. June 2002.

[16] E.Stefanov, M.Dijk, E.Shi, C.Fletcher, L.Ren, X.Yu, and S.Devadas. Path ORAM: An Ex- tremely Simple Oblivious RAM Protocol. In CCS, pp. 299-310, 2013.

[17] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. J. Algorithms 51.2004

[18] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More Robust Hash- ing: Cuckoo Hashing with a Stash. SIAM J. Comput. 39. 2009