DEIM2022 J43-3

信頼できる実行環境 TEE を用いた暗号マルチマップにおけるデータ量秘匿可能な挿入法

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　マルチマップの各キーに紐づくデータの件数がからデータが推測される攻撃に対し，差分プライバシを用いたデータ量秘匿手法が提案されているが，既存手法では更新が想定されていなかった．そこで我々はデータ量を秘匿したままデータ更新を可能にする手法について研究を進めている．本論文では，信頼できる実行環境 (TEE) を用いることで挿入時の撹乱処理をサーバー側で安全に行える手法を提案する．提案手法におけるデータ量と挿入箇所の秘匿に対する安全性を論理的に示し、実験で挿入の処理時間やノイズの量を検証することで，想定するユースケースにおいて大きな負担にならない性能で実行できることを示した．

キーワード　暗号化マルチマップ, プライバシ保護, セキュアデータベース, TEE, Intel SGX, Cuckoo hashing, 準同型暗号, Volume-hiding, 差分プライバシ

1　はじめに

暗号化データベースの新たな脅威として，**データ量の漏洩**と呼ばれるものが Kellaris ら [1] や Grubbs ら [2] によって報告された．データ量の漏洩とは、攻撃者がクエリ結果のサイズ（データ量）を取得し，その結果，データベースや問い合わせに関連する情報が明らかになるというセキュリティ上の脅威である．

この脅威に対し，Sarvar ら [3] は，データ量を秘匿した暗号化マルチマップへの検索手法を提案した．しかし，彼らの手法では，データベースが構築された後，更新がなされないことを前提とした手法であるため，ユースケースの範囲が狭い．

そこで，我々は Sarvar らの手法を基に暗号化マルチマップにデータ量を秘匿したままデータ更新を可能にする手法について研究を進めている．我々の先行研究 [4] では，局所差分プライバシと Cuckoo hashing を用いたデータ挿入法を提案した．しかし，その手法では挿入場所の撹乱をサーバープロバイダに秘匿するために，一度全てのデータをクライアントにコピーし，クライアント側で撹乱処理を行う必要がある．そのため，大規模なデータベースには適さない手法であった．また，データ量を秘匿するため，データ挿入時に局所差分プライバシを用いてノイズが付与されるため，ノイズ量が膨大になるという問題があった．

本論文では，これらの問題に対し，**TEE (Trusted Execution Environment)** [5] と呼ばれるプロセッサが提供する信頼できる実行環境を用いることで，撹乱処理をサーバー側で行い，局所差分プライバシを用いずにデータ量を秘匿する手法を提案する．

本論文では，提案手法におけるデータ量と挿入箇所の秘匿に対する安全性を論理的に示し，実験で挿入の処理時間やノイズの量を検証することで，想定するユースケースにおいて大きな負担にならない性能で実行できることを示した．

本研究における貢献は以下の通りである．

- 暗号化マルチマップに対して，差分プライバシを維持したまま，データ挿入を可能とする．提案手法では，**TEE** を利用し，検索時に差分プライバシを保証するようにノイズを計算・付与することでデータ量の秘匿を行う．TEE とは，CPU の拡張機能の一つで，OS やハイパーバイザーなどのシステムから隔離された実行環境を構築する機能である．検索時のノイズ計算をサーバープロバイダから秘匿するために用いる．
- TEE を利用し，サーバープロバイダに秘匿したまま，サーバー上で挿入箇所の秘匿処理を可能とする．我々の先行研究 [4] において **Cuckoo hashing** [6] の特性を用いた挿入箇所の秘匿法を提案した．しかし，秘匿操作をサーバープロバイダから観察されないようにするため，クライアント上で処理を行う必要があった．そこで，提案手法では，TEE を利用する．TEE を用いることで，クライアントに頼ることなく，秘匿操作を処理することができる．しかし，TEE が提供できる信頼領域のサイズは非常に小さく，大きなデータを扱うことができない．そこで，テーブルを小さなブロックに分け，それぞれのブロックに対して独立に秘匿処理を行う手法を提案する．
- 提案手法の安全性を分析する．本論文では，加法準同型暗号を用いたデータ件数の更新や検索時のノイズ付与の安全性，また，Cuckoo hasing と TEE を利用した挿入箇所秘匿の安全性を理論的に示し評価した．
- TEE の一つである **Intel SGX** を使用した一連の操作の実験を行い，提案手法の実現性を示した．実験では，提案手

法，我々の先行研究の手法，非セキュア手法それぞれについて，挿入の実行時間を比較した．また，差分プライバシと局所差分プライバシという異なるプライバシ機構のもとでデータ量を秘匿する際に必要なノイズを比較した．さらに，検索の実行時間について Sarvar らの手法と比較した．実験から，我々の手法が想定されるユースケースにおいて有用であることを示した．

本章以降の構成は以下の通りである．第 2 章では，本研究のためのいくつかの事前知識を簡単に説明する．第 3 章では，関連研究についての紹介をする．第 4 章では，データ量を秘匿した暗号化マルチマップへの挿入法の提案をし，その安全性を示す．第 5 章では，提案手法における挿入と検索の実験結果を示し，実用性を示す．最後に第 6 章で，本研究のまとめを記し，今後の課題について言及する．

2 事 前 知 識

本章では，本研究で利用する技術やプライバシ指標についての説明をする．

2.1 差分プライバシ (DP)

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 [7] が提案した差分プライバシー（以降 DP と表記）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を保護する．では，どのようにノイズを加えれば DP 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 (LDP)

通常の DP 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を保護している．一方で，局所差分プライバシ（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 Cuckoo hashing

cuckoo hashing [6] とは，2 つの配列と 2 種類のハッシュ関数を用いたハッシュテーブルである．Cuckoo hashing の特徴は，片方の配列でハッシュ値が衝突した場合，既に格納されている値をもう一方の配列に移し，新しい値を挿入することで，衝突を回避することである．本論文では，この cuckoo hashing の特徴を**托卵操作**と呼ぶ．図 1 は，2 つのハッシュ関数（h_1 と h_2）と 2 つの配列（T_1 と T_2）を使用し，値 x を挿入する様子を示している．

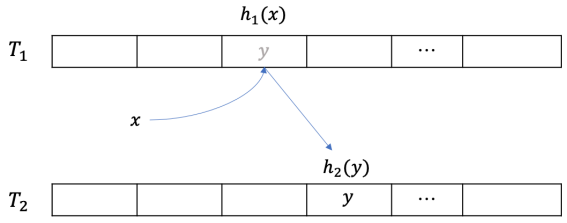


図 1: Cuckoo hashing におけるハッシュ値の衝突回避の動作

まず，T_1 の $h_1(x)$ の位置が空であるかどうかをチェックする．空であれば，x を T_1 上の $h_1(x)$ に挿入して処理を完了し，そうでなければ，既存の値 y を追い出して，代わりに値 x を挿入する．追い出された y は，T_2 上の $h_2(y)$ が空いているかをチェックして，空いていれば T_2 の $h_2(y)$ に y を挿入し，そうでなければ，同様の操作を再帰的に処理する．このようにハッシュテーブルを作成することで，値 x は必ず $T_1[h_1(x)]$ または $T_2[h_2(x)]$ のどちらかに入っていることが保証されるため，$O(1)$ で検索可能となる．

一方で，托卵操作は，処理が完了するまでに非常に時間がかかる場合や運が悪ければ終わらない可能性もある．そこで，Kirsch，Mitzenmacher，Wieder の 3 人によって，**stash-based cuckoo hashing** [9] 提案された．stash-based cuckoo hashing は，予め決められた容量のスタッシュと閾値を用意し，托卵操作を閾値以上繰り返してもテーブルに挿入できない値は，スタッシュに格納することで上記の問題に対処する手法である．

2.4 TEE (Trusted Execution Environment)

最近のプロセッサは，安全な計算への要求の高まりに対応するため，TEE (Trusted Execution Environment) [5] をサポートしている．TEE はプロセッサ内の独立した実行環境で，

内部のコードとデータを保護し，機密性と完全性を保証する．具体的には，第 6 世代（またはそれ以上）の Intel 社の CPU には、**Intel SGX** [10] と呼ばれる TEE をサポートしている．

SGX は，**エンクレーブ**と呼ばれるシステムの他の部分から隔離されたメモリ領域を作成できる．これにより，機密データを第三者が管理する OS や様々なアプリケーション・システムレベル攻撃から保護したままプログラムを実行することができる．

しかし，既存の SGX が提供するエンクレーブは最大で 96MB のみであるため，大きなデータを扱うことができない．また，SGX には，エンクレーブ内の処理は速いのに対して、エンクレーブ内とエンクレーブ外を跨る関数の呼び出しは，通常の関数呼び出しと比較して非常に約 2 倍から 2000 倍遅いという欠点がある [11,12]．さらに，エンクレーブ内とエンクレーブ外を跨ぐ関数呼び出しの際の引数のサイズは，一度に最大で 8MB しかサポートされていない．

3 関 連 研 究

3.1 Sarvar らの研究

Sarvar ら [3] が提案した暗号化マルチマップへの検索手法には，2 つの特徴がある．1 つ目は，差分プライバシ（DP）を利用した通信量の削減であり，2 つ目は，Cuckoo hashing を利用したストレージコストの削減である．彼らの手法における，暗号化マルチマップの構築と検索プロセスについて具体的に説明する．

まず，構築プロセスを紹介する．彼らの手法の暗号化マルチマップでは，それぞれのキーのデータ量（要素数）を管理する Count Table とデータを管理する Cuckoo Hash Table から成る．Count Table には，それぞれのキーのデータ量（要素数）に対して差分プライバシを適用したノイズを付与した値が暗号化された状態で格納する．Cuckoo Hash Table には，cuckoo hashing を利用して構築されるハッシュテーブルであり，$\{Enc(key), Enc(value)\}$ のようにキーバリューデータが暗号化されて格納される．ここで，重複するキーの要素について，ハッシュ値が重ならないように，$key+1, key+2, \ldots$ のように，キーとインデックスを組み合わせてハッシュ値を生成することで，マルチマップを実現する．また，全てのデータを格納後，テーブルの空いている箇所には暗号化したダミーのキーバリューデータ $Enc(dummy\ key), Enc(dummy\ value)$ が格納される．これにより，それぞれのキーにダミーの要素を追加する必要がなく，ストレージコストが削減される．

次に，検索プロセスを紹介する．検索は 2 ラウンドの通信によって行われる．まず 1 ラウンド目の通信によって，クライアントは検索するキーのデータ件数を Count Table から取得し，複合化する．次に，クライアントは取得したデータ量と検索キーからそれぞれの要素のハッシュ値を生成し，Cuckoo Hash Table からそれらの要素を取得する．これにより，通信量が (実際の要素数 + ノイズ分の要素数) × 2 に抑えることができる．ここで，2 倍となる理由は，cuckoo hashing は検索の際，2 つの配列から要素を取得するからである．図 2 は，Sarvar らの暗号化マルチマップの構造と key4 の検索を例とした検索プロセスを示している．

3.2 先 行 研 究

我々の先行研究 [4] では，Sarvar らの手法をもとにクライアントと連携した暗号化マルチマップに対するデータ量を秘匿したデータ挿入法を提案した．データ挿入後もデータ量を秘匿するためには，挿入後もデータ量に付与されているノイズが差分プライバシを保証している必要がある．さらに，データ量の秘匿だけでなく，挿入箇所の秘匿も必要となる．これは，挿入前後のテーブルの変化から新たに追加したデータがどこに格納されたかを知られてしまうからである．そこで我々は，局所差分プライバシ (LDP) [8] と加法準同型暗号を用いて，データ挿入後のデータ量を秘匿し，Cuckoo hashing の特性である托卵操作を用いて挿入箇所の秘匿を実現した．ここで，局所差分プライバシ (LDP) とは，データ提供者が自身のデータをデータ収集者に送る際に，プライバシを保護できるように拡張された差分プライバシである．しかし，挿入箇所の秘匿操作は（サーバープロバイダを含む）攻撃者から秘匿する必要があるため，クライアント側で実行する．

データ挿入後のデータ量を秘匿するには，挿入後もデータ量に付与されているノイズが差分プライバシを保証している必要がある．我々は局所差分プライバシのメカニズムの 1 つである **Randomized Response** を挿入時に用いることで，この問題に対処した．具体的には，挿入時に以下の式 2 に従ってキーを選択する．

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

ここで，key_{true} は実際にデータ挿入を行うキーであり，key_{dummy} はダミーの挿入を行うキーである．挿入時には，確率 p で実際に挿入するキーが選ばれるまで Randomized Response を繰り返し，それまでに生成れた全てのキーをサーバーに送る．生成されたキーはデータ挿入には利用せず，Count Table の更新にのみ用いる．これにより，更新後の Count Table は差分プライバシを維持することができる．Count Table の更新は，加法準同型暗号を用いて，暗号文のまま処理することで，サーバー上で更新することが可能となる．

挿入箇所の秘匿をするには，挿入後の Cuckoo Hash Table が複数箇所同時に更新される必要がある．2.3 章で，説明したように，Cuckoo hashing にはハッシュ値の衝突回避をするために托卵操作が行われる．Cuckoo Hash Table はデータの格納場所が秘匿されるように，空いている箇所にダミーデータが格納されている．そのため，データ挿入をすると必ず挿入箇所と連動して複数の箇所の更新が起こる．托卵操作の回数は，ユーザー指定のパラメータとして与えらた閾値以上繰り返す．

しかし，托卵操作による更新箇所はハッシュ値の衝突によるデータの遷移であるため，ランダムな複数箇所の更新とはならない．そこで，Cuckoo Hash Table 上のランダムに選んだ位置にダミーのキーバリューデータを挿入し，托卵操作を発生させ

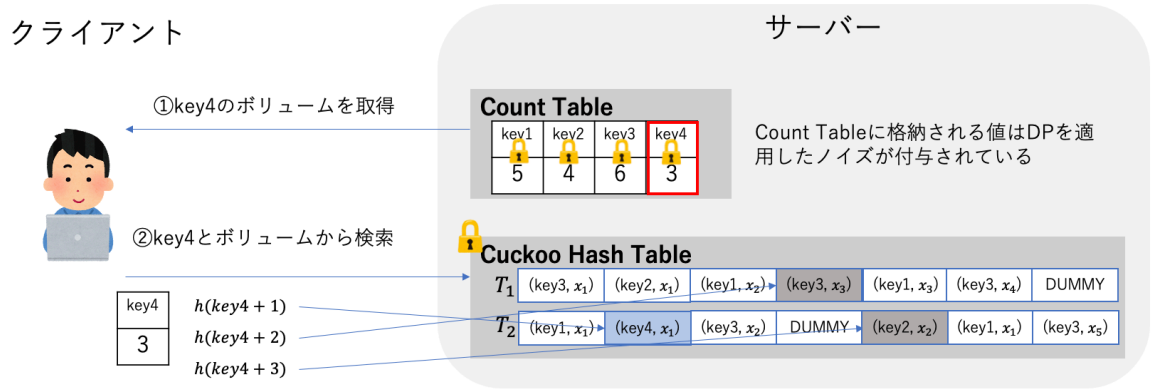


図 2: Sarvar らの手法における暗号化マルチマップの構造と検索プロセス

る．これにより，更新後の Cuckoo Hash Table の状態を見ても，どの箇所の更新がデータ挿入と連動して発生したのかを区別できなくする．図 3 は，托卵操作による複数箇所の更新処理の様子を表している．

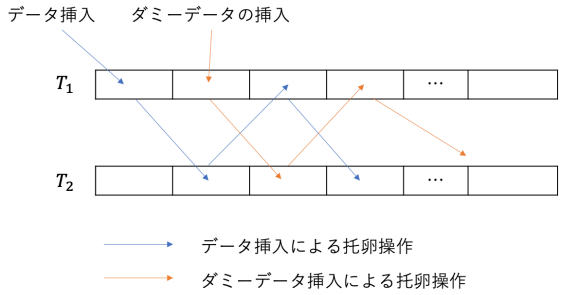


図 3: 托卵操作による複数箇所の更新の様子

一方で，我々の先行研究では，以下の 2 つの問題がある．

- サーバー管理者から挿入箇所の秘匿の操作を守るために，クライアントで托卵操作を行う必要があるため，Cuckoo Hash Table 全体をクライアントに転送する必要がある．これは，マルチマップのサイズが大きい場合に，ネットワークトラフィックのコストが高くなり，処理に時間がかかることや処理性能がクライアントの性能に大きく依存することで，リソース不足により，クライアントが処理を完了できない可能性がある．
- 局所差分プライバシによるノイズ付与によって，Sarvar らの手法と比較してノイズ量が大幅に増加することにより，検索時の通信量と処理時間への影響が大きくなる．

4 提案手法

本論文では，我々の先行研究 [4] の手法における問題点に対処するため，局所差分プライバシを用いずにノイズ付与を行い，挿入位置の摂動をクライアントに依存しない暗号化マルチマップへのデータ挿入法を提案する．

提案手法では，クライアントに依存しない代わりに，2.4 節で紹介した TEE を用いる．TEE が提供するエンクレーブを用いることで，内部攻撃者から処理を秘匿しつつ、サーバ側でノイズの付与と Cuckoo Hash Table の更新の処理を実行することができる．

4.1 節において，TEE を用いて局所差分プライバシを利用せずにデータ量を秘匿する手法を提案し，4.2 節において，TEE を用いてサーバー上で托卵操作を行う手法を提案する．その後，提案手法における挿入プロセスを 4.3 節で説明し，4.4 節において，検索プロセスの説明をする．最後に，4.5 節において，本提案手法の安全性の分析をする．

4.1 LDP を用いないデータ量の秘匿法

我々の先行研究 [4] では，局所差分プライバシのメカニズムである Randomized Response により，ノイズを付与することでデータ挿入後のデータ量の秘匿を達成した．しかし，データ挿入の度にノイズが付与されるため，Sarvar ら [3] の手法と比較して大幅にノイズ量が増大するという問題があった．そこで，本節では，局所差分プライバシを利用せずにデータ挿入後のデータ量を秘匿する手法を提案する．

本手法では，データ挿入時に，Count Table にノイズの付与を行わない．Count Table は常にノイズが付与されていないデータ件数（つまり真のデータ件数）が格納される．そして，検索の都度，差分プライバシを保証するようにノイズを計算し，付与してから検索を行うことでデータ挿入後のデータ量の秘匿を実現する．

図 4 は，データ挿入時の Count Table の更新の様子を示し，図 5 は，データ検索時のノイズ付与の様子を示している．

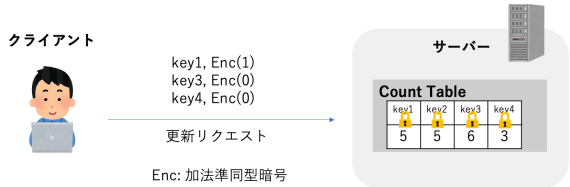


図 4: データ挿入時の Count Table の更新の様子

図 4 のように，1 件のデータを挿入する際には，クライアントは挿入するキー（図 4 では，*key1*）とランダムな複数のキー（図 4 では，*key3, key4*）の更新リクエストを送る．これらのリ

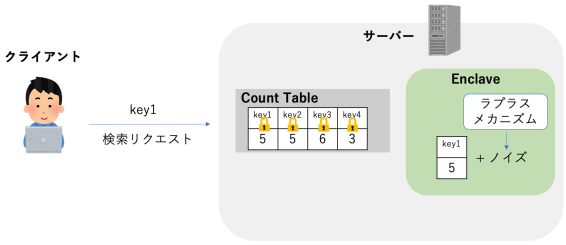


図 5: データ検索時のノイズ付与の様子

クエストは，加法準同型性暗号によって暗号化されるため，暗号文のままカウントが更新される．このとき，クライアントから送られる更新リクエストのうち，実際に挿入が行われるキー（図 4 では，*key1*）のみ 1 が暗号化されており，その他のキー（図 4 では，*key3, key4*）は 0 が暗号化されている．そのため，Count Table のカウントは実際に挿入が行われるキーのみインクリメントされ，その他のキーのデータ件数は変化しない．しかし，暗号文は変化するため，サーバー管理者からはどのキーのデータ件数がインクリメントされたのかを判別することができない．

次に，検索時にノイズを付与する手法について説明をする．図 5 のように，検索リクエストを受け取るとエンクレーブに検索キーの暗号化されたカウントを Count Table から読み込みエンクレーブ内で複合化を行う．そして，エンクレーブ内でノイズ量を計算し，データ件数に付与する．このとき，差分プライバシを保証するように 2.1 節で説明したラプラスメカニズムを用いてノイズの計算を行う．これにより，データ挿入後も差分プライバシを保証するようにノイズが計算されるため，データ量を秘匿することが可能となる．

4.2 TEE を用いた挿入箇所の秘匿

先行研究 [4] では，クライアントに Cuckoo Hash Table を転送し，クライアント上で托卵操作を行うことで，サーバー管理者から秘匿した．本手法では，クライアントの代わりに TEE が提供するエンクレーブに Cuckoo Hash Table を置き，エンクレーブ内で托卵操作をすることでサーバー管理者から秘匿する．しかし，2.4 で述べたように，TEE が提供するエンクレーブにはメモリ領域の制限がある（Intel SGX の場合 96MB）．そこで，Cuckoo Hash Table を 96MB 以下の小さなブロックに分け，それぞれのブロックで独立にデータ挿入を行う．また，TEE には，エンクレーブ内の処理は速いのに対して，エンクレーブ外からエンクレーブ内の関数を呼ぶ（またはその逆）の動作は通常の関数呼び出しに比べて非常に遅い（Intel SGX の場合，約 2 倍から 2000 倍遅い [11,12]）という特徴がある．そして，Intel SGX では，エンクレーブ内とエンクレーブ外を跨ぐ関数呼び出しの際の引数のサイズは，一度に最大で 8MB しかサポートされていない．そのため，ブロックサイズを 96MB にしてしまうと，ブロック全てをエンクレーブ内に渡すために，12 回に分けて渡す必要がある．これは，非常に時間がかかるため，本手法では，1 ブロックのサイズを 8MB 以下に設定し，一度の呼び出しでエンクレーブ間のデータの受け渡しを完了できるようにしている．

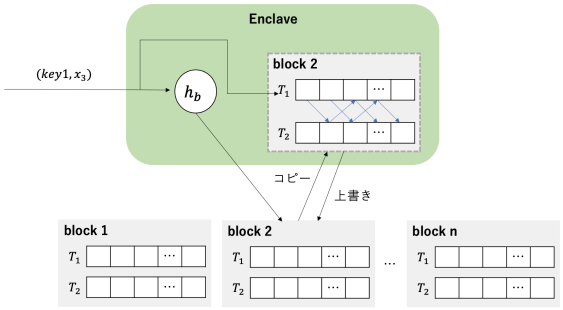


図 6: ブロックへの振り分け方とブロックへのデータ挿入の様子

図 6 は，ブロックへの振り分け方とブロック内のデータ挿入の様子を示している．図 6 のように，挿入するデータは，まず，エンクレーブ内のハッシュ関数 h_b によって生成されたハッシュ値によってブロックへの振り分けを行う．次に，振り分けられたブロックをエンクレーブ内にコピーし，データ挿入を行う。エンクレーブ内では，ブロック全体を複合化するのではなく，托卵操作によって更新される箇所のみがその都度複合化される．データ挿入後，エンクレーブ外のブロックは更新されたブロックに上書きされ、托卵操作によりブロックから追い出されたデータは，サーバー上のスタッシュに格納される．

4.3 挿入プロセス

暗号化マルチマップへのデータ挿入の全体のプロセスを説明する．図 7 は，*key*1 への挿入を例にした挿入プロセスを示している．

まず，クライアントは，次の手順で挿入リクエストを生成する．

（1） ランダムな複数のキーを選択する

（2） 準同型性暗号 (HE と表記) で挿入キー ($key1$) は 1 を暗号化 $HE(1)$ し，ランダムに選ばれたダミーのキー ($key3, key4$) は 0 を暗号化 $HE(0)$ する

（3） 挿入データを暗号化する $Enc(key1, x_6)$(Enc は公開鍵暗号)

次に，サーバーは，クライアントからの挿入リクエストを受け取り，4.1 節で説明したように Count Table の更新を行う．その後，更新したデータ件数と挿入キー ($key1$) からハッシュ値を生成し， 4.2 節で説明したようにブロックの割り当てと Cuckoo Hash Table への挿入を行う．最後に，托卵操作の結果，Cuckoo Hash Table から追い出されたデータをサーバー上のスタッシュに格納する．図 7 では，スペースの都合上，スタッシュは省略されている．以上が，挿入のプロセスとなる．

4.4 検索プロセス

提案手法における検索は次の手順で行われる．まず，クライアントは暗号化した検索キー ($Enc(key)$) を検索リクエストとして送る．サーバーは，検索リクエストを受け取ると，エンクレーブ内で複合化を行い，検索キーのデータ件数を Count

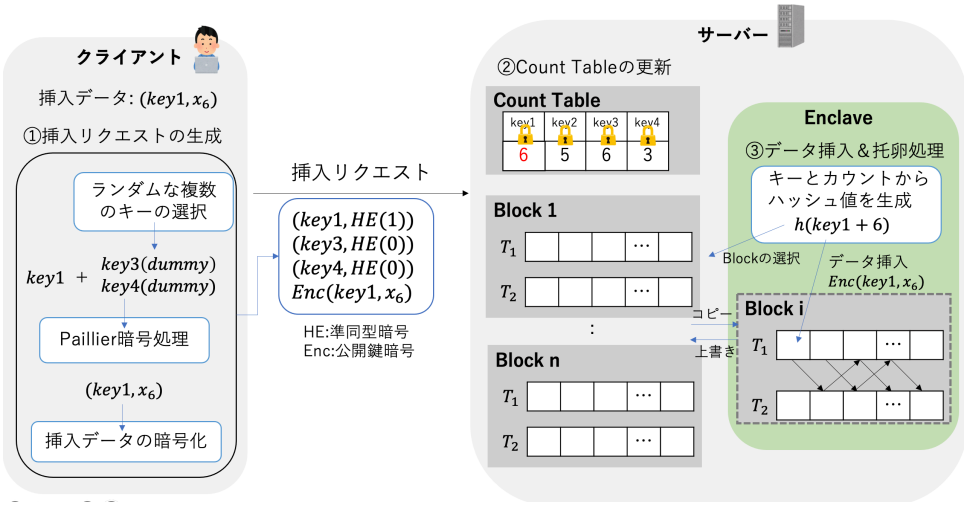


図 7: key1 への挿入を例とした挿入プロセス

Table から読み取る．次に， 4.1 節で説明したように，ノイズを計算し，Count Table から得たデータ件数に付与する．その後，ノイズを含むデータ件数分の (ブロック番号, Cuckoo Hash Table, スタッシュそれぞれの) ハッシュ値を生成し，エンクレーブ外で検索を行う．最後に，得られた結果をクライアントに送る．以上が，検索のプロセスとなる．

4.5 安全性証明

ここでは，提案手法の安全性について説明する．まず，提案手法で想定するプライバシモデルについて説明する．サーバープロバイダは，semi-honest であるとする．semi-honest とは，確立されたプロトコルに従うが，入手したデータ情報を最大限に盗もうとするモデルである．また，(サーバープロバイダを含む) 敵対者は，エンクレーブ以外のシステムはすべて監視することができるとする．そして，敵対者は，データに対する背景知識を持っていないとする．

上記のプライバシモデルをもとに，まず，Count Table の更新について安全性を分析する．Count Table の更新は加法準同型暗号によって暗号文のまま行われる．また，ランダムな複数のキーと一緒に更新される．そのため、攻撃者はデータ挿入によって値が変更されたことは知ることができるが，どの値が変更されたかを知ることはできない．ランダムに選ぶキーの個数を k とすると，攻撃者は，$1/(k+1)$ の確率でしか判別することができない．

次に，検索時のデータ量の安全性について分析を行う．提案手法では，検索の度にノイズを計算し，付与を行う．そのため，同じキーに対する検索であったとしても毎回データ量が変化することになる．これにより，攻撃者は，前回の検索と比較して，データ量が変化していても，それがデータ挿入によって引き起こされたものなのか，ノイズ量の違いによるものなのかを判別することが不可能となる．差分プライバシを用いたノイズ付与の安全性については，Sarvar ら [3] によって安全性が証明されている．

次に，托卵操作による挿入箇所秘匿の安全性を説明する．挿入箇所が一箇所だけの場合，更新箇所を観察することにより，挿入箇所が漏洩してしまう．我々の手法では， 1 回のデータ挿入に対して，少なくとも k+1 個（k¿=1）以上の箇所が更新される．ここで，k はユーザーが設定するパラメータで，データ挿入とは別に，ダミーデータを挿入することで，強制的に托卵操作を起こさせる回数である．そのため，サーバー管理者は，実データが挿入された箇所を最悪の場合でも 1/(k+1) の確率でしか断定することができない．実験では、1 ブロックの最大容量は 20000 レコードほどであったが，サーバープロバイダには，データがどのブロックに振り分けられたかを知られてしまう恐れがある．そこで，割り振られるブロックについても撹乱することで，安全性を高めることを可能としている．具体的には，実際に割り振られるブロックに加えてダミーのブロックへの操作を追加することによって，どのブロックに挿入されたのかを秘匿する．

5 評価実験

我々は，提案手法におけるデータ挿入・検索において以下の 2 つのことを評価することを目的とした実験を行った．1 つ目は，TEE を用いた処理による実行時間への影響の評価，2 つ目は，検索時に差分プライバシを保証するようにノイズを付加することによる通信量と検索時間への影響の評価である．

実行環境は以下の通りである．

- **サーバー:** Microsoft Azure, Intel(R) Xeon(R) E-2288G CPU 3.70GHz, 4 GB RAM, Ubuntu 18.04.5 LTS
- **クライアント:** MacBook Pro, dual-core Intel Core i7 3.1 GHz , 16 GB 1867 MHz DDR3, macOS (ver. 11.5.2)

実験に用いたデータセットは，人工データ（キーの数:1000, データ数:1,000,000）と実世界データ（キーの数: 5305, データ数: 1067371）を使用した．実世界データは，UCI Online Retail II [13] を利用した．

暗号化処理には，Paillier 暗号と RSA 暗号を使用する．

また，実験では，提案手法を **DP-TEE**，我々の先行研究の手法を **LDP-Client**，ノンセキュアの手法を **Non-secure**,

Sarvar ら [3] の手法を **Sarvar** と表記する．ここで，我々の先行研究の手法 **LDP-Client** とは，3.2 節で紹介した手法であり，ノンセキュアの手法 **Non-secure** とは，TEE，ノイズ付与，暗号（Paillier,RSA）が使用されていない手法である．

5.1 データ挿入の評価

DP-TEE，LDP-Client，Non-secure のそれぞれの手法に対してデータ挿入の実行結果を比較した．

図 8 は，一様分布に従う人工データ，正規分布に従う人工データ，実世界のデータが格納された **DP-TEE, LDP-Client, Non-secure** に対してクライアントが 1000 件のデータを挿入したときの一件のデータの挿入にかかった平均実行時間を示しており，図 9 は，平均通信量を示している． 実行時間におい

図 8: 1000 件のデータ挿入における **DP-TEE, LDP-Client, Non-secure** の平均実行時間

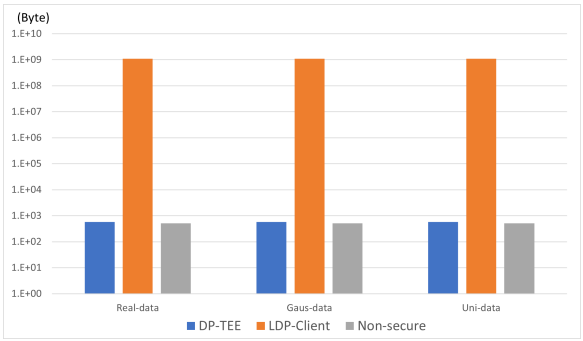


図 9: 1000 件のデータ挿入における **DP-TEE, LDP-Client, Non-secure** の平均通信量

ては，マルチマップの性質からデータの分布によらず挿入にかかる時間は一定である．また，提案手法の **DP-TEE** は，我々の先行研究の手法 **LDP-Client** と比較して，560 倍以上の高速化を達成している．さらに，ネットワークコストにおいては 100 万倍以上の通信量の削減が達成されている．

一方で，**DP-TEE** は **Non-secure** と比較して，約 100 倍ほど処理に時間がかかる．表 1 は，**DP-TEE, Non-secure** それぞれに対して，先ほどと同様に 1000 件のデータ挿入を行なったときの Count Table の更新と Cuckoo Hash Table の更新それぞれにかかった平均処理時間を示している．Count Table の更新には，加法準同型暗号である Paillier 暗号の加算処理が使われているため，その影響が大きいと考えられる．Cuckoo

表 1: Count Table と Cuckoo Hash Table の更新における平均処理時間

処理内容	***DP-TEE***	***Non-secure***
Count Table の更新時間 (ms)	2.21×10^{-2}	7.93×10^{-7}
Cuckoo Hash Table の更新時間 (ms)	1.28×10^{1}	2.06×10^{-6}

(a) 実世界データセットに対するデータ挿入における処理時間

処理内容	***DP-TEE***	***Non-secure***
Count Table の更新時間 (ms)	1.40×10^{-2}	2.01×10^{-6}
Cuckoo Hash Table の更新時間 (ms)	1.28×10^{1}	6.00×10^{-6}

(b) 正規分布のデータセットに対するデータ挿入における処理時間

処理内容	***DP-TEE***	***Non-secure***
Count Table の更新時間 (ms)	2.60×10^{-2}	2.00×10^{-6}
Cuckoo Hash Table の更新時間 (ms)	1.27×10^{1}	6.50×10^{-6}

(c) 一様分布のデータセットに対するデータ挿入における処理時間

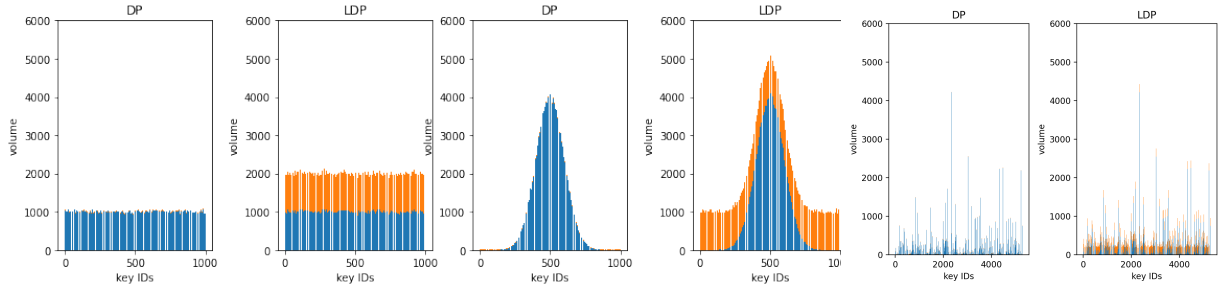
Hash Table の更新には，TEE による影響が大きいと考えられる．2.4 節で説明したように，TEE には信頼領域と非信頼領域間のデータの受け渡しにかかる処理コストが大きいという問題点 [11,12] がある．

5.2 DP と LDP のノイズ量の比較

一様分布に従う人工データ，正規分布に従う人工データ，実世界のデータそれぞれについて，全てのデータを挿入後に DP を用いてノイズを付加した場合とデータ挿入時に LDP を用いてノイズを付加した場合の結果を図 10 に示す．実験結果では，DP に対して LDP のノイズ量は非常に大きくなっている．人工データの場合は，DP に比べてそれぞれの key に対して約 1000 件ずつのノイズが付与されている．実世界のデータの場合は，DP に比べてそれぞれの key に対して約 200 件ずつのノイズが付与されている．これは，(ノイズの量) = (データセットの件数)/(キーの数)，となるためである。**LDP-Client** では Count Table のカウントにのみノイズが付与されるため，Cuckoo Hash Table にノイズが付与されるわけではない．そのため，このノイズの量は空間計算量には影響しない．しかしながら検索時にはカウントテーブルの個数によって検索が行われるため，このノイズ量は検索時間に影響を及ぼすと考えられる．そこで，5.3 節では，このノイズ量の差が検索処理にどの程度の影響を及ぼすを評価する．

5.3 検索の評価

DP-TEE, LDP-Client, Sarvar, Non-secure の手法に対して検索実験を行なった．図 11 は，一様分布に従う人工データ，正規分布に従う人工データ，実世界のデータが格納されたそれぞれの手法に対してクライアントが全てのキーに対して検索をしたときの平均実行時間を示し，図 12 は，平均通信量を示している． 実験より，**DP-TEE** は，**LDP-Client** と比較して，全てのデータセットにおいて 1.5 倍以上の通信量の削減がされており，それに伴う高速化を達成している．また，**DP-TEE** は，**Sarvar** と同等の通信量と処理時間を達成することができ



(a) 一様分布のデータセット　(b) 正規分布のデータセット　(c) 実世界のデータセット

図 10: DP と LDP のノイズ量の比較グラフ

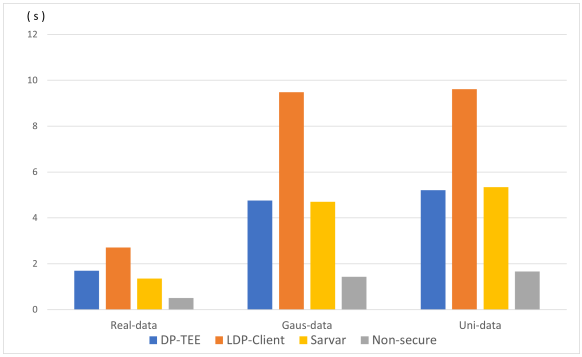


図 11: 全てのキーに対して検索を実行したときの平均実行時間

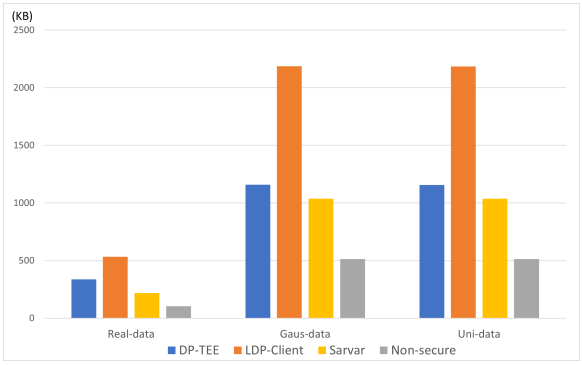


図 12: 全てのキーに対して検索を実行したときの平均通信量

ている．

6　まとめと今後の課題

本論文では，TEE を用いて，クライアントの負担を軽減し，ノイズ量を削減するデータ挿入法を提案した．本手法では，局所差分プライバシによる過剰なノイズの付与に対して，TEE が提供する信頼領域内で検索時に差分プライバシを保証するようにノイズ量を計算することによってノイズ量を削減した．また，TEE の技術的な課題であった限られた小さなメモリ空間を解決するために，テーブルをブロックに分けてそれぞれのブロックで独立に処理を行なっている．

実験から，1 回のデータ挿入に約 113ms かかることがわかった．これは，先行研究 [4] と比較すると約 560 倍以上の高速化を達成している．また，キーのデータ量と更新箇所を秘匿したままのデータ挿入が 113ms に抑えられていることは非常に実用性が高いと考えている．検索においては，先行研究と比較して全てのデータセットに対して 1.5 倍以上の通信量の削減とそれに伴う高速化を達成した．また，Sarvar らの手法と同程度の時間に抑えられている．

以上のことから，安全性と実用性を兼ね備えたデータ量秘匿を可能とする暗号化マルチマップへのデータ挿入を実現することができた．

今後の課題として検討しているのは，以下の 2 点である．

- データ検索において，挿入の前後に同じキーで検索を行うことによる脅威への対策をしたい．データ挿入の前後で同じキーに対して検索が行われた場合，Cuckoo Hash Table 上の更新された位置が新たに参照されることで，高確率で挿入データがどこに挿入されたのかが判別できてしまうという脅威がある．この脅威への対策として，挿入されたばかりのデータについては TEE の信頼領域内でキャッシュとして保持しておき，挿入からすぐに検索された場合は，キャッシュ上のデータを返すという手法を考えている．
- 余っているエンクレーブ領域の有効的な活用したい．本手法では Intel SGX の引数のサイズの制限からエンクレーブの領域は 8MB しか使われていない．そこで，残りの領域をキャッシュとして用いることを考えている．実世界のワークロードでは，キーへのアクセスが大きく偏るケースが多い．そこで，頻繁にアクセスされるキーについては，cuckoo hash table に格納せずに，エンクレーブでキャッシュをしておくことで，挿入と検索の高速化を狙いたい．

謝　　辞

この成果は，SKY(株) の委託業務 (CPI03114) の結果得られたものです．また、本研究は JSPS 科研費 JP19K11978 の助成を受けたものです。

文　　献

[1] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, p. 1329–1340, New York, NY, USA, 2016. Association for Computing Machinery.

[2] 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 '18, p. 315–331, New York, NY, USA, 2018. Association for Computing Machinery.

[3] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, p. 79–93, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Shunta Ishihara, Chiemi Watanabe, and Toshiyuki Amagasa. Supporting insertion in encrypted multi-maps with volume hiding. In *IEEE International Conference on Smart Computing, SMARTCOMP 2021, Irvine, CA, USA, August 23-27, 2021*, pp. 264–269. IEEE, 2021.

[5] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. pp. 20–38, 05 2019.

[6] R. Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, Vol. 51, pp. 122–144, 2004.

[7] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pp. 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[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] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, Vol. 39, No. 4, p. 1543–1561, December 2009.

[10] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086.

[11] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[12] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 689–703, Savannah, GA, November 2016. USENIX Association.

[13] Daqing Chen. Online Retail II. UCI Machine Learning Repository, 2019.