Paillier暗号を用いたデータベース演算実装方式における性能解析に関する検討

内藤　華† 中野美由紀†† 小口　正人†

† お茶の水女子大学理学部情報科学科　〒 112–8610 東京都文京区大塚 2-1-1
†† 津田塾大学学芸学部情報科学科　〒 187–8577 東京都小平市津田町 2-1-1
E-mail: †hana-n@ogl.is.ocha.ac.jp, oguchi@is.ocha.ac.jp, ††miyuki@tsuda.ac.jp

あらまし　企業の機密情報や顧客の個人情報などさまざまな電子データが，外部のクラウドサービスへ委託，処理されている．情報漏洩や改ざんを防ぐためにデータを暗号化する必要があるが，従来の暗号化方式だとデータを処理する際に復号しなければならず秘匿性が損なわれてしまう．ここで，準同型暗号という暗号方式を用いることにより暗号文のまま処理を行うことが可能になるが，扱える演算の制限が少ないほどデータサイズや処理コストが大きくなることが知られている．本稿では，加算のみ演算可能な加法準同型暗号の一種である Paillier 暗号を用いて 7 種類のデータベース演算プリミティブを実装し，その性能評価を行なった．演算を行うサーバと暗号化のための鍵の管理を行うサーバの 2 種類を用いることにより，乗算も行うことが可能となった．また，演算の手順の簡素化を図り，既存手法とは異なる手法での実装も行った．

キーワード　準同型暗号，プライバシ，問い合わせ処理

1 はじめに

近年，個人情報に関与するような機密情報を含むさまざまな電子データが，外部のクラウドサービスに委託，処理されている．第三者のサーバ上でデータをそのままの状態で扱うと情報漏洩や改ざんのリスクがあるため，データの秘匿性を確保するために暗号化が必要である．従来の暗号化手法の場合，データを暗号化してからクラウドに送信したとしても，利用する際に復号が必要となるため秘匿性が損なわれてしまう．一方で，準同型暗号という暗号方式を用いると，データを暗号化したまま演算を行うことが可能になり，これによりデータの中身がデータ所有者以外の目に触れることなく処理結果を取り出すことができる．

準同型暗号には，加算または乗算のどちらかのみ計算可能な部分準同型暗号（以下 PHE: Partial Homomorphic Encryption），回数に制限があるが加算も乗算も行うことのできる somewhat 準同型暗号，加算と乗算の演算回数の制限を変更可能な Leveled 準同型暗号（以下 Leveled HE: Leveled Homomorphic Encryption），演算回数に制限のない完全準同型暗号（以下 FHE: Fully Homomorphic Encryption）などが存在する．しかしながら，行える演算の種類や回数の制限が少ない方式ほど，有用である反面，処理負荷が非常に大きくなることが知られている．

本研究で用いる Paillier 暗号 [1] は，Paillier によって提案された公開鍵暗号方式で，加法準同型性を持つ PHE である．したがって，Leveled HE や FHE と比べてデータサイズが小さく低いコストでの演算が可能であるといった利点がある．さらに，Chowdhury らの Cryptϵ [2] では，Paillier 暗号を用いて暗号文同士の加算に加えて乗算も行う手法が提案されており，豊富な種類の演算が可能である．

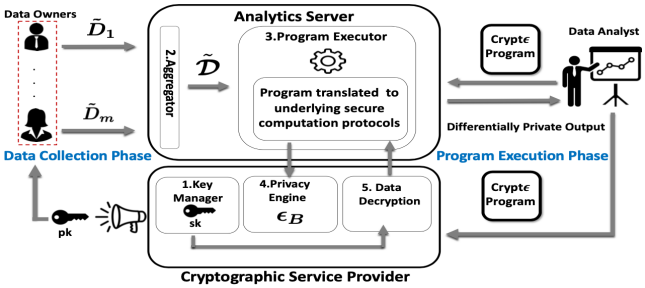


図 1: Cryptϵ システム構成 [2]

本稿では，Cryptϵ を基に 7 種類のデータベース演算プリミティブを実装し，演算の手法や処理速度について考察する．さらに，Cryptϵ とは異なる手法での演算方法についての提案を行う．

2 関連研究

本節では，図 1 の Cryptϵ のシステム構成や処理の手順について説明する．

2.1 Cryptϵ 概要

Cryptϵ は，暗号化されたデータに対して 2 種類のサーバを用いてプログラムを実行することで，差分プライバシ [3,4] を満たした演算結果を出力するシステムである．データ解析者や処理を行う 2 つのサーバにはデータ所有者の個人情報を知られることなく，実行することが可能である．

2.2 処理の手順

暗号データを収集し演算を実行する Analytics Server（以下

AS），と，鍵の生成・管理やプライバシコストの管理，暗号データの復号を行う Cryptographic Service Provider（以下 CSP）の 2 つのサーバを中心に以下の手順で動作する．

（1） CSP はデータ所有者から伝えられたプライバシコストの見積もり（以下 ϵ^B）を記録する．また，Paillier 暗号方式を用いて公開鍵と秘密鍵の鍵ペアを生成する．

（2） 各データ所有者は，CSP が生成した公開鍵を利用して自分のデータを暗号化し AS に送信する．ここで，暗号化に Linearly Homomorphic Encryption（以下 LHE）[5] を用いると暗号文同士の加算，暗号文と平文の乗算が可能であるが，LHE を発展させた Labeled Homomorphic Encryption（以下 labHE）[6] を用いることで暗号文同士の乗算も可能となる．

（3） AS はそれらのデータを集めて暗号化データベース $\tilde{\mathcal{D}}$ を作成し，これに対してデータ解析者から送られた Cryptϵ プログラムを実行する．Project や Filter などのデータベース演算プリミティブを実行した結果にノイズを付与したうえで，CSP に送信する [7]．これにより，CSP は個人のプライバシが保護された結果のみ知ることができる．

（4） CSP は，演算によって発生したプライバシコストが ϵ^B を超えていないことを確認できると，保管しておいた秘密鍵で復号し結果を出力する．

3 実験概要

本節では，2 節の Cryptϵ を基に実装したプログラムについて説明する．

3.1 処理の手順

図 2 に示すように，2 台の PC 上にそれぞれ AS と CSP を作成し通信することで演算を行う．暗号化は，GitHub 上で公開されている Paillier 暗号のライブラリ [8] を用いて labHE で行なった．実験で使用したマシンの性能は表 1 の通りである．演算処理前と処理後の暗号データは，Google Cloud Storage [9] に保存することで AS と CSP 間でのデータの共有を行えるようにした．以下の手順で処理を行う．

（1） CSP が平文のデータをデータベースから取得し暗号化したうえでクラウド上に保存する．

（2） AS がクラウドから暗号データを読み込む．Project や Filter などのデータベース演算プリミティブを暗号化状態で実行した結果を，暗号文のまま再びクラウド上に保存する．

（3） CSP がクラウドから演算結果を読み込み，保管しておいた秘密鍵で復号して平文の結果を取得する．

なお，本稿では AS 上でのデータベース演算プリミティブに焦点を置いて実装を行うため，差分プライバシのためのノイズ付与とそれに伴って発生するプライバシコストの計算は行わない．したがって，Cryptϵ ではどちらのサーバにも個人情報が見えないが，本実験では CSP はデータ収集時と演算結果の復号後に平文にアクセスできてしまう．乗算を行う際に AS と CSP がデータのやり取りを 1 回行う必要があるが，ここでは秘匿化によって AS のみならず CSP にも正しい情報を知ることはできない．

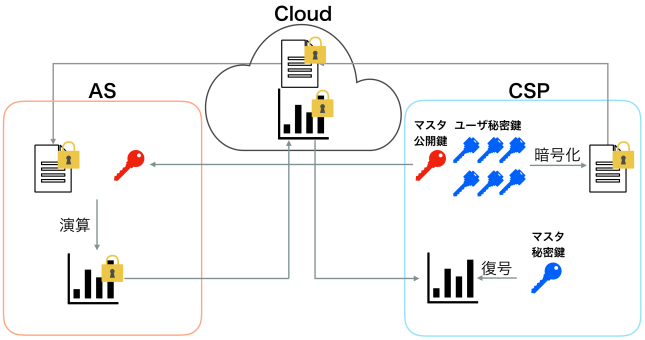


図 2: システム構成

表 1: 実験環境

	AS	CSP
CPU	Intel Core i5 1.6 GHz	Intel Core i5 1.6 GHz
OS	macOS 13.1	macOS 13.1
メモリ	16 GB	8 GB

表 2: テーブルを one-hot-encoding で表現した例

···	Age_{22}	···	Age_{59}	···	Gender_{Female}	Gender_{Male}	···
···	1	···	0	···	1	0	···
···	0	···	1	···	0	1	···

3.2 扱うデータ

Cryptϵ と同様，Adult データセット [10] を用いる．⟨Age, Gender, NativeCountry, Race⟩ のカラムを抽出し，レコード数 10000 件のテーブルを作成した．表 2 に示すように one-hot-encoding という 0 と 1 のダミー変数でデータを表現したうえで labHE で暗号化を行い，$\tilde{\boldsymbol{T}}$ とした．one-hot-encoding では，例えば $\langle 22, Female\rangle$ というレコードは，$\langle[\underbrace{0,\ldots,0}_{21},1,\underbrace{0,\ldots,0}_{78}],[1,0]\rangle$ のように表現する．また，$\tilde{\boldsymbol{T}}$ に加えて，各レコードが演算に必要であるか否かを 0 と 1 で表すベクトル $\boldsymbol{B}$ を作成し，すべての要素を 1 で初期化する．これは，複数の演算プリミティブを連続して利用する際に有用である．例えば Filter の条件を満たさないレコードを 0 にすると，それ以降の演算には用いられないようにすることができる．

3.3 暗号化手法

labHE による暗号化手法を示す．記号の一覧は表 3 にまとめた．

Setup(1^λ)：セキュリティパラメータ λ に基づいてマスタ公開鍵（以下 mpk）とマスタ秘密鍵（以下 msk）の鍵ペアを作成する．

KeyGen(mpk)：ランダムにユーザごとのシード σ_i を生成し，mpk で暗号化することでユーザ秘密鍵（以下 usk_i）を作成する．

labEnc(mpk, usk_i, τ, m)：ユーザごとのラベル τ と σ_i を基に，擬似ランダム関数によって b を求める．$a = m - b$，$\beta = Enc_{mpk}(b)$ とすると，$C = (a, \beta)$ が m の暗号文となる．

labDec(msk, c)：$C = (a, \beta)$ とすると，$m = a + Dec_{msk}(\beta)$ として復号できる．

表 3: 記号一覧

記号	意味
m	平文．$m \in \mathcal{M}$.
c	暗号文．$c \in \mathcal{C}$.
C	平文 m を labHE で暗号化した結果．$C = (a, \beta) \in \mathcal{M} \times \mathcal{C}$.
pk	公開鍵.
sk	秘密鍵.
$Enc_{pk}(m)$	pk を用いて LHE で m を暗号化.
$Dec_{sk}(c)$	sk を用いて LHE で c を復号.
$c_1 \oplus c_2$	暗号文 c_1，c_2 の加算.

Add(C_1, C_2)：平文と暗号文を加算する場合と，暗号文同士を加算する場合の 2 通りが考えられる．C_1 が平文，C_2 が暗号文の場合，$C_1 = a_1$，$C_2 = (a_2, \beta)$ とする．このとき，$C' = (a_1 + a_2, \beta)$. C_1，C_2 ともに暗号文の場合，$C_1 = (a_1, \beta_1)$，$C_2 = (a_2, \beta_2)$ とする．このとき，$C' = (a_1 + a_2, \beta_1 \oplus \beta_2)$.

cMult(c, C)：$C = (a, \beta)$ の場合，$C' = (a \cdot c, \beta \cdot c)$．$C = \alpha$ の場合，$C' = c \cdot \alpha$.

Mult(C_1, C_2)：$C_1 = (a_1, \beta_1)$，$C_2 = (a_2, \beta_2)$ とすると，$C' = Enc_{mpk}(a_1 \cdot a_2) \oplus cMult(a_1, \beta_2) \oplus cMult(a_2, \beta_1)$.

genLabMult(C_1, C_2)：*genLabMult* を用いると，複数回乗算を行うことが可能になる．なお，AS と CSP の間でデータのやりとりが必要となる．$C_1 = (a_1, \beta_1)$，$C_2 = (a_2, \beta_2)$ とする．

AS：乱数 r を生成し，$e' = Mult(c_1, c_2) \oplus Enc_{mpk}(r)$ と β_1，β_2 を CSP へ送信する．

CSP：$e'' = Dec_{msk}(e') + Dec_{msk}(\beta_1) \cdot Dec_{msk}(\beta_2)$ を計算する．ランダムなシード σ' とユーザごとのラベル τ' を基に，擬似ランダム関数によって b' を求める．$\bar{a} = e'' - b'$，$\beta' = Enc_{mpk}(b')$ として，$\bar{e} = (\bar{a}, \beta')$ を AS に送信する．

AS：$a' = \bar{a} - r$ とすると，$e = (a', \beta')$ が C_1 と C_2 の積となる．

3.4 演 算 手 法

3.3 項の暗号化手法を用いてデータベース演算プリミティブを実装する．

（1） **CrossProduct**

テーブル $\tilde{\boldsymbol{T}}$ に含まれる属性 A_i と A_j の直積をとり新たな属性 A' で置き換え $\tilde{\boldsymbol{T}}'$ を出力する．A_i, A_j の重複しない値の個数をそれぞれ s_i, s_j とすると，新たに $s = s_i \cdot s_j$ 個の値が生成される．したがって，one-hot-encoding で表すと，A_i の s_i 列と A_j の s_j 列が取り除かれ A' の s 列が追加される．

（2） **Project**

$\tilde{\boldsymbol{T}}$ から指定した属性以外のカラムを取り除き $\tilde{\boldsymbol{T}}'$ を出力する．

（3） **Filter**

指定した条件を満たすレコードに対応するビットを 1 としたベクトル $\boldsymbol{B}'$ を出力する．抽出する値を one-hot-encoding で表したベクトルを $\boldsymbol{V}$ とし，$\tilde{\boldsymbol{T}}$ のレコードごとに $\boldsymbol{V}$ との内積を求める．求めた内積と $\boldsymbol{B}$ をビットごとに乗算した結果 $\boldsymbol{B}'$ を出力する．図 3 に，Age が 30 以上 35 以下であるレコードを Filter する方法を示す．レコード 1 とレコード 2 はそれぞれ，Age が 30 と 33 であるため，$\boldsymbol{V}$ との内積が 1 となり，$\boldsymbol{B}$ との積も 1 であれば抽出される．

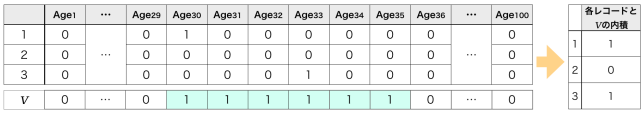


図 3: Filter 演算方法

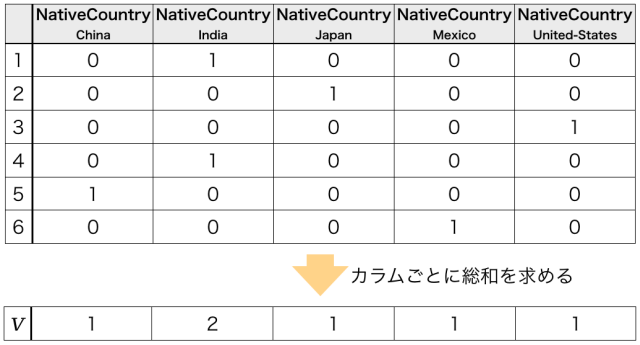


図 4: GroupByCount 演算方法

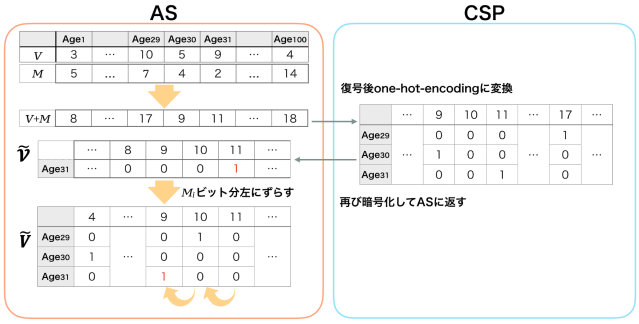


図 5: GroupByCountEncoded 演算方法

（4） **Count**

$\boldsymbol{B}$ の要素をすべて足し合わせた結果 c を出力する．

（5） **GroupByCount**

ある属性について値ごとの要素数を求める．$\tilde{\boldsymbol{T}}$ のカラムごとにすべての要素を足し合わせることでそれぞれの値の個数を求め，ベクトル $\boldsymbol{V}$ を出力する．図 4 に，属性 NativeCountry について GroupByCount を行う方法を示す．

（6） **GroupByCountEncoded**

GroupByCount の結果 $\boldsymbol{V}$ を入力として受け取り，これを one-hot-encoding で表した $\tilde{\boldsymbol{V}}$ を出力する．暗号文のままでは one-hot-encoding に変換できないため，図 5 に示すように CSP にデータを送信し復号する必要がある．まず，ランダムな整数の配列 M を作成して $\mathcal{V} = \boldsymbol{V} + M$ とし，$\mathcal{V}$ を CSP へ送信する．CSP は $\mathcal{V}$ を復号したものを one-hot-encoding に変換し，再び暗号化したうえで AS へ送り返す（$\tilde{\mathcal{V}}$）．$\tilde{V}_{i,j} = \tilde{\mathcal{V}}_{i,j+M_i} (i = 1, 2, \ldots, s,\ j = 1, 2, \ldots, |\boldsymbol{T}|)$ とすることで，M_i ビット分ずつ左にずれるので元の $\boldsymbol{V}$ を one-hot-encoding で表現することができる．

（7） **CountDistinct**

属性 A について重複しない値の個数を求める．CountDistinct

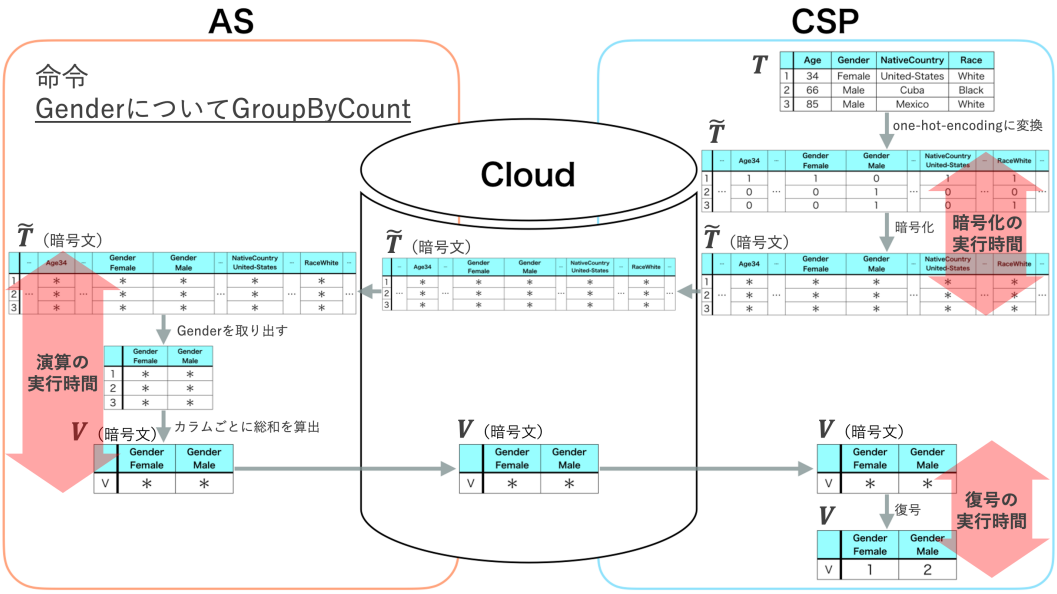


図 6: 処理の流れ

については，Cryptϵ と異なる手法で演算を行うことにより簡素化を図った．Cryptϵ では，ほかの演算手法とは異なり CSP が生成した Garbled circuit [11, 12] を用いて演算を行う．提案手法では，Garbled circuit は用いず，AS と CSP がデータのやり取りを 1 回のみ行う．属性 A について図 5 の GroupByCountEncoded と同様の処理をを行い $\tilde{\boldsymbol{V}}$ を受け取る．このとき，$\tilde{\boldsymbol{T}}$ にひとつも含まれていない値は全てのビットが 0 となっている．したがって，$\tilde{\boldsymbol{V}}$ に含まれる 1 の個数が CountDistinct の出力となる．

4　基本性能評価結果

本節では，データサイズや 3.4 項で述べた演算プリミティブの実行時間について考察する．図 6 は，例として属性 Gender について GroupByCount を実行する際の暗号化から演算，復号までの流れを示している．図中の赤い矢印は，実行時間として計測した部分を表している．暗号化の実行時間はすでに one-hot-encoding に変換されている $\tilde{\boldsymbol{T}}$ の暗号化に要した時間を，演算の実行時間は AS が取得した $\tilde{\boldsymbol{T}}$ から演算結果を算出するのに要した時間を，復号の実行時間は取得した演算結果を復号するのに要した時間を示している．他の演算についても，同様に計測を行った．

4.1　データサイズ

レコード数 1000 件，10000 件，20000 件のときの，元データ，暗号化前の one-hot-encoding で表現したデータ，暗号データのデータサイズは図 7 のとおりである．クラウド上に pickle ファイルとして保存したときのデータサイズを示している．いずれのデータも，レコード件数に比例してサイズが増大している．データを one-hot-encoding に変換するとカラム数が増えるため，元のデータよりもサイズが平均して 29 倍程度の大きさとなっている．暗号データは one-hot-encoding の 6 倍程度にとどまった．

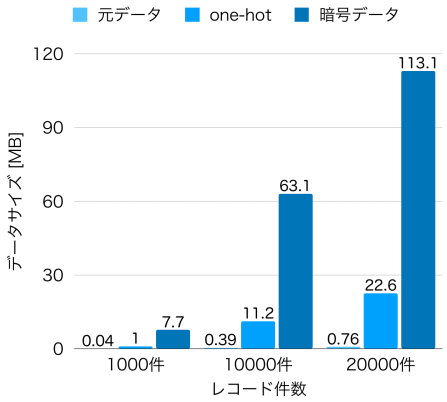


図 7: データの種類とサイズの関係

4.2　実行時間

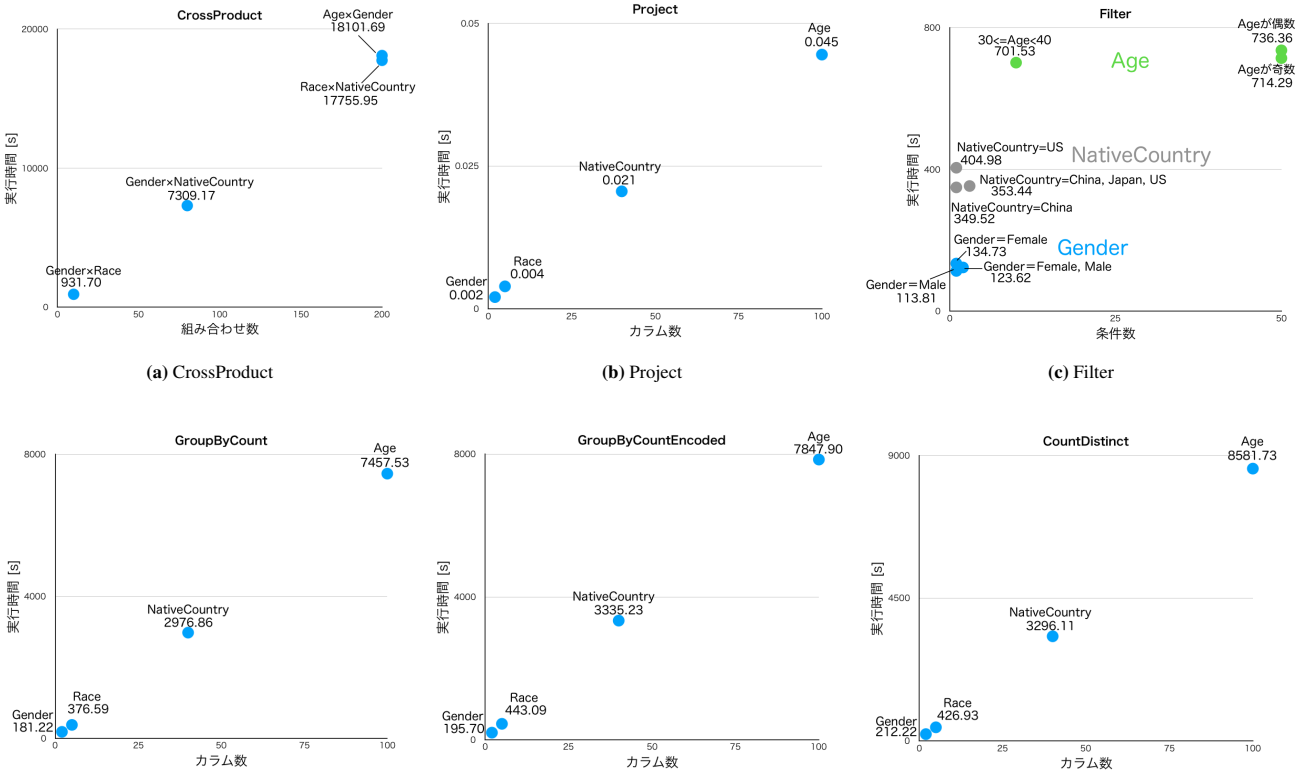
147 ビットで表現したレコード 10000 件の場合，データの暗号化には平均して 850 秒程度の時間を要した．一方で，同じレコード数の暗号データ $\tilde{\boldsymbol{T}}$ の復号には 300 秒程度かかった．以下，図 8 に示した各演算の実行時間について考察する．

（1） **CrossProduct**

属性の組み合わせは全部で 6 通りあり，2 つの属性のカラム数の積 s が小さい方から 4 通りの組み合わせについて実行時間を計測した．2 つの属性の直積を求めるためにレコード 1 件につき *genLabMult* を s 回行う必要があり，ほかの演算と比べて処理時間が非常に長くなった．2 つの属性に含まれる値の組み合わせ総数と実行時間は，図 8(a) の通りほぼ比例している．このデータセットで最も時間のかかる組み合わせは NativeCountry と Age で，値の組み合わせ総数が $40 \times 100 = 4000$ であることから 100 時間以上かかると予想できる．

（2） **Project**

それぞれの属性を one-hot-encoding で表現したときのカラム数と 100 回 Project を実行した平均の処理時間の関係は図 8(b) の通りである．Add，cMult， Mult などの演算は行わないため，



(a) CrossProduct **(b)** Project **(c)** Filter

(d) GroupByCount **(e)** GroupByCountEncoded **(f)** CountDistinct

図 8: 実行時間計測結果

ほかの演算と比べて処理時間が非常に短くなった．それぞれの属性を one-hot-encoding で表現したときのカラム数と実行時間はほぼ比例している．

（3） **Filter**

Age，Gender，NativeCountry の 3 種類の属性について Filter を実行した．結果は図 8(c) のとおりである．緑色の点は Age，灰色の点は NativeCountry，青色の点は Gender に関する条件で Filter を行った結果を表している．ここで条件数とは，その条件によって抽出される値の数を表す．例えば，“ $30 \leqq \mathrm{Age} < 40$ ”を条件とする場合は 10，“ Age が奇数 ”を条件とする場合は 1 から 100 に含まれる奇数は 50 個なので 50，“ Age が偶数 ”を条件とする場合も 50 となる．どの属性においても，抽出する条件の数と実行時間は関係していない．一方で，3 つの属性を比較すると，one-hot-encoding で表したときのカラムが 100 列の Age や 40 列の NativeCountry では，カラムが 2 列の Gender と比べて長い時間を要していることが読み取れる．

（4） **Count**

100 回 Count を実行した結果，処理時間は平均で 0.029 秒となった．$\boldsymbol{B}$ の要素の加算のみであるため，非常に短時間で処理が完了した．

（5） **GroupByCount**

それぞれの属性を one-hot-encoding で表現したときのカラム数と GroupByCount の処理時間の関係は図 8(d) の通りである．カラム数の増加に伴って処理時間も増加していることが読み取れる．

（6） **GroupByCountEncoded**

それぞれの属性を one-hot-encoding で表現したときのカラム数と GroupByCountEncoded の処理時間の関係は図 8(e) のとおりである．GroupByCount と同様，それぞれの属性を one-hot-encoding で表現したときのカラム数に比例して処理時間が増加した．また，GroupByCount の実行時間との差より，全体の 1 割程度が one-hot-encoding への変換にかかる処理時間であると分かる．

（7） **CountDistinct**

それぞれの属性を one-hot-encoding で表わしたときのカラム数と実行時間の関係は図 8(f) の通りである．他の演算と同様，それぞれの属性を one-hot-encoding で表現したときのカラム数に比例して処理時間が増加した．また，GroupByCount の結果 $\boldsymbol{V}$ を利用して GroupByCountEncoded と同様の処理を行なってから算出するため，GroupByCountEncoded と同程度の時間を要した．

5 まとめと今後の課題

加法準同型性を持つ Paillier 暗号を用いてデータベース演算プリミティブを実装，性能評価を行った．Paillier 暗号は PHE の一種であるが，暗号化方式として labHE を用いることにより，加算のみならず乗算の演算も可能となった．CrossProduct のように複数の属性の直積を用いる演算や，GroupByCount のように乗算を多用するような演算では，処理時間が増大した．一方で，Count では $\boldsymbol{B}$ を利用することで非常に短い時間での処理が可能であった．また，データの暗号化にも比較的長い時間を要

しており，改善の余地があると感じた．今後の課題として，直積演算や本実験で扱っていない演算の実装，データの暗号化の効率的な手法などを検討していきたい．

文　献

[1] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques, EUROCRYPT ’99, pp. 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[2] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha, “Cryptϵ: Crypto-Assisted Differential Privacy on Untrusted Servers,” Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 603–619, 2020

[3] A. Agarwal, M. Herlihy, S. Kamara, and T. Moataz, “Encrypted Databases for Differential Privacy,” 2018.

[4] N. Li, M. Lyu, D. Su, and W. Yang, “Differential Privacy: From Theory to Practice,” Morgan and Claypool, 2016.

[5] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon, “Privacypreserving ridge regression with only linearly-homomorphic encryption,” In B. Preneel and F. Vercauteren, editors, Applied Cryptography and Network Security, pp. 243–261, Cham, 2018. Springer International Publishing.

[6]] M. Barbosa, D. Catalano, and D. Fiore. Labeled homomorphic encryption - scalable and privacy-preserving processing of outsourced data. In ESORICS, 2017.

[7] C. Dwork and A. Roth, “The Algorithmic Foundations of Differential Privacy,” Found. Trends Theor. Comput. Sci., Vol.9, pp. 211–407, August 2014.

[8] CSIRO’s Data61, Python Paillier Library, 2013. https://github.com/data61/python-paillier.

[9] Google cloud platform. https://cloud.google.com

[10] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.

[11] A. Yao, “Protocols for Secure Computation,” Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, Chicago, pp. 160–164, November 1982.

[12] Y. Lindell and B. Pinkas, “A Proof of Security of Yao’s Protocol for Two-Party Computation,” Journal of Cryptology, Vol.22, No.2, pp. 161–188, April 2009.