Vol.2015-ARC-215 No.19
Vol.2015-OS-133 No.19
2015/5/27

暗号化データベースシステムにおける効率的な集約処理の評価

堀尾 健太郎[1,a)]　川島 英之[2,b)]　建部 修見[2,c)]

概要：暗号化データベースシステムの先駆的研究に CryptDB がある．CryptDB は暗号化による性能劣化への対処をしておらず，大規模データでの運用が困難である．Paillier 暗号の特性を用いた効率的な総和計算手法が MONOMI で提案されているが，その性能はバックエンド DBMS の機構により阻害されている可能性がある．本研究では，暗号化データベースシステムのバックエンド DBMS をクリーンステートデザインにより開発する．これにより MONOMI の総和計算法を適切に評価する．MONOMI 方式は TPC-H Q1 について CryptDB 方式よりも 17 倍程度効率的であることを示した．一方，本研究はその性能差が 2547 倍程度まで広がる事を示す．

1. はじめに

クラウドでは管理者がデータを盗み見したり，あるいはセキュリティの弱点を突いた攻撃によりデータを奪われる疑念を拭い去ることができない．そのような問題に立ち向かうため，セキュアデータベース技術が研究開発されてきた [1], [2], [3], [4]．セキュアデータベース技術とは RDBMS を含むデータベース全般をセキュアにする技術の総称である．この中にはプライバシ保護技術，権限管理，秘匿化技術などが含まれる．この中で近年脚光を浴びている技術に暗号化データベース技術がある．これはデータベース中のデータを暗号化しておき，それらのデータを復号することなく問合せ処理を行って，その結果をクライアント側で復号する技術である．

暗号化データベースシステムの先駆的研究に CryptDB[1] がある．CryptDB ではデータ暗号化に起因する性能劣化への対応は殆ど考えられていない．従って CryptDB は大規模データに対しては実用に堪えないものとなっている可能性がある．特に集約演算については総和計算のサーバサイドにおける高性能化が未解決課題として残されている．

CryptDB は RDBMS が管理する全てのデータを 4 種類の暗号化方式を用いて秘匿化しつつ管理する．プロキシサーバはユーザから発行された SQL 問合せ及び RDBMS から返す結果を仲介し，RDBMS はデータを暗号化したまま SQL 問合せを処理する．CryptDB を用いれば，RDBMS サーバ上には暗号化されたデータのみが置かれ，返す結果も暗号化されている．従って RDBMS サーバのデータが不法に閲覧されたり，不正な手段で RDBMS に直接 SQL 問合せが発行されようとも，データの内容を類推することは暗号が解かれない限り不可能になる．

CryptDB ではデータ暗号化に伴ってデータ量が増大し I/O コストとデータ処理コストが増大する．暗号化によって最も重大な影響を受けるリレーショナル演算子は総和演算子である．これを実現するには準同型性暗号が必要になるが，それは計算コストが平文と比べて著しく高いことで知られている．Google 検索を完全準同型性暗号を用いた処理のコストは平文の 1 兆倍程度になる [5]．2.2 節で示すように CryptDB が用いている加法準同型性暗号による加算演算の処理コストは，平文と比較してデータサイズが 64 倍となり，計算時間は約 2500 倍となる．

暗号化データベースシステムを効率化する最新研究に MONOMI[3] がある．MONOMI は平文稠密格納法 (DPE)[6] と DSM ページレイアウトを用いて総和演算の効率化を達成しており，7 つの総和計算を行う TPC-H Q1 を CryptDB よりも 17 倍程度効率化する．一方，MONOMI は PostgresSQL をバックエンドに用いており，データパイプラインやバッファプールアクセス等のクエリプロセスが本来の性能改善を阻害している可能性がある．また，MONOMI では TPC-H を用いてマクロ的評価のみを行っ

[1] 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 筑波大学 システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba
[a)] horio@hpcs.cs.tsukuba.ac.jp
[b)] kawashima@cs.tsukuba.ac.jp
[c)] tatebe@cs.tsukuba.ac.jp

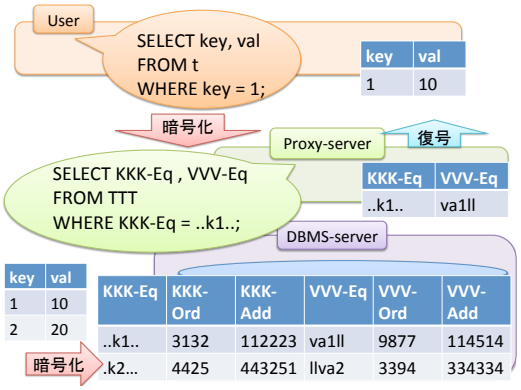

図 1　CryptDB の概要

ているため，総和演算に関するミクロ的評価が不明である．即ち，平文稠密格納法と DSM ページレイアウトによる性能向上の限界が従来研究では明らかにされていない．そこで本研究では MONOMI の方式をスクラッチ実装した暗号化データベースシステム上で評価する．これにより性能阻害要因を排除し，同手法の性能限界を調査する．

本論文は次のように構成されている．2 節では本研究の先行研究について述べる．3 節では効率化手法の評価に用いる DBMS の設計と実装について述べ．4 節ではそれを用いた実験について述べる，5 節で関連研究について述べ，6 節で本論文の結論と今後の課題について述べる．

2. 先行研究

2.1 CryptDB

暗号化データベースシステムの草分け的存在である CryptDB[1] では RDBMS は平文を扱うことはなく，全て暗号文上で問合せを処理する．そのため DBMS サーバへの攻撃等の脅威に対しても情報を秘匿することが可能である．

CryptDB はユーザから発行された問合せ及び DBMS サーバから返ってきた結果を仲介するプロキシサーバ，DBMS が稼働する DBMS サーバの 2 つのサーバにより構成される．ユーザから問合せが発行されると，プロキシサーバは問合せを受け取り，変数の値，テーブル名を適切に暗号化する．又，必要に応じて暗号化レベルを下げる問合せを発行して DBMS サーバに送る．DBMS サーバ上では平文が露わになることなく問合せが処理され，プロキシサーバに対して暗号化された結果を返す．プロキシサーバは結果を復号し，アプリケーションサーバに返す (**図 1**)．

DBMS 上には暗号 Onion と呼ばれる多重に暗号化された暗号文が保存される．暗号 Onion には等号チェックのための Onion（Eq-Onion），大小関係チェックのための Onion（Ord-Onion），加算演算のための Onion（Add-Onion），文章から単語を検索するための Onion（Search-Onion）の 4 種があり，変数の種類に応じて必要な Onion が生成される．例えば int 等の数値型のデータに対しては，Eq-Onion，Ord-Onion，Add-Onion が生成される．また，問合せ処理の際には問合せに応じて適切な Onion が参照される，例えば WHERE 句で等号チェックを行なう問合せ（e.g. SELECT val FROM table WHERE key = 1）では Eq-Onion が参照される．

2.1.1 準同型性暗号

準同型性暗号は準同型性を有する暗号方式である．準同型性とは，複数の対象に対して特定の数学的構造の類似性を表す概念である．例えば平文 m_1，m_2 に対して，準同型性暗号 Enc を用いて暗号化した暗号文 $Enc(m_1)$ と $Enc(m_2)$ が与えられた時，平文や秘密鍵を用いることなく $Enc(m_1 \circ m_2)$ を計算することができる．ここで $\circ$ は $+$ や $\times$ のような二項演算子である．

RSA 暗号，ElGamal 暗号など整数論をベースとしている多くの公開鍵暗号方式が準同型性を有することが知られている．CryptDB では，Add-Onion の実装に加法準同型性をもった Paillier 暗号 [7] を用いており，総和計算に利用している．Pailler 暗号は加算のみが可能であるが，加算と乗算が可能である完全準同型暗号と比較して軽量である．

2.2 暗号化に伴う性能劣化

暗号化データベースシステムでは元データを暗号化してデータを保存し，暗号文上で問合せ処理を行なうため，平文を扱う場合と比べてデータサイズが増大したり，演算コストが増加することにより性能劣化する．

まず，暗号化に伴う特有のコストとして，鍵生成と暗号化・復号のコストが挙げられる．準同型性暗号では暗号化するための鍵の生成にコストがかかる．安全性を確保するため大きな素数のペアを生成する必要がある．また，暗号化及び復号には長い桁数の数に対するべき乗，剰余等の演算が必要になる．例えば Paillier 暗号では安全性のために最低でも 1024 bit の平文長・鍵長が必要とされており 10 進数で 309 桁となる．任意精度整数の演算が必要となり，これはプロセッサのレジスタの大きさに合わせられている数値型の演算よりも性能が劣るため，暗号化及び復号は遅い処理となる．

次に，暗号化により，加算演算のコストの増加がある．本研究の予備実験として，Pailler 暗号による性能劣化を調査した．実験を行ったマシンのスペックは**表 2** に示す．暗号化した状態で複数回の加算演算を行い，平文での同回数の加算演算の処理時間と比較した．**図 2** にその結果を示す．横軸が加算回数，縦軸が処理時間 (μsec) となっている．ある 2 つの暗号文から平文同士の和を求めたい場合，暗号文同士の乗算が必要になる．鍵長を 1024 bit と仮定すると暗号文は 2048 bit であり，10 進数で 617 桁同士の乗算となる．平文での加算と比較しておよそ 2500 倍の計算時間となった．さらに，Paillier 暗号では安全性の確保の

Vol.2015-ARC-215 No.19
Vol.2015-OS-133 No.19
2015/5/27

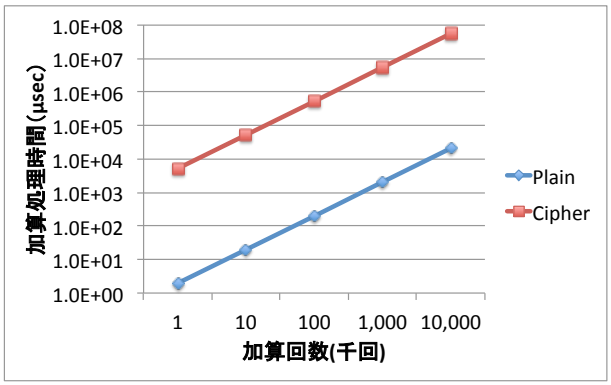

図 2　Paillier 暗号の性能劣化に関する予備実験

ために長い鍵長と平文長が必要になる．それは少なくとも 1024 bit 長が必要とされる．この場合，暗号文長は 2048 bit となる．1024 bit 鍵の場合で 1 つの int 型 (32 bit) の値を暗号化することを考えた場合，データサイズは 64 倍となる．これらの性能劣化は総和計算に影響を与える．

2.3　平文の稠密な格納による集約演算子の効率化

Pailler 暗号で用いられる平文は 1024 bit の長い桁数の数値であるが，int 型のような短い桁数の数値は複数並べて結合することで 1 つの長い桁数の数値とみなすことができる．すなわち 1 つの平文ブロックの中に複数の値を稠密に格納 (**ブロック化**) することが可能である．この，1 つの平文ブロックに複数の値を稠密に格納し暗号化する方式 [6] を本論文では Densely Packing Encryption(DPE) と呼ぶ．また，これ以降平文の値を複数まとめて結合したものを**ブロック**，ブロックの中の一つ一つの値のことを**スライス**と呼ぶ．例として，int 型の値 $a_1, ..., a_4$ を結合したブロック A，$b_1, ..., b_4$ を結合したブロック B を考える．暗号化関数 $Enc()$，復号関数 $Dec()$ として，$S = Enc(A) \times Enc(B)$ とすると，$Dec(S)$ から取り出せる，先頭のスライス s_1 は $s_1 = a_1 + b_1$ となる．鍵長 1024 bit の Paillier 暗号を用いる場合，平文ブロックは 1024 bit であるため int 型 (32 bit) の値を最大 32 個格納可能であり，一度の暗号文同士の乗算により最大 32 個のスライスの加算を同時に処理することが可能である．

各スライスにおいて加算による桁あふれは起こらないと仮定し，選択演算を含まないような総和を想定する場合，次の方法で総和を求めることが可能である．

$i \times j$ 個の値の総和を求めるとする．k 個の値 $v_{ij}(j = 1, ..., k)$ を結合し，ブロック毎に暗号化した n 個の暗号文 $c_i(i = 1, ..., n)$ を用意する．すべての暗号文を掛けあわせた結果を c とすると，c を復号して得られる平文 S は k 個の値を結合したものとなっている．平文 S を k 個の値 $(S = s_1 \circ s_2 \circ ... \circ s_k)$ に分割し，足し合わせることで総和が得られる．

例えば 1 つの暗号文に 4 つの値を含めて，1 から 16 まで

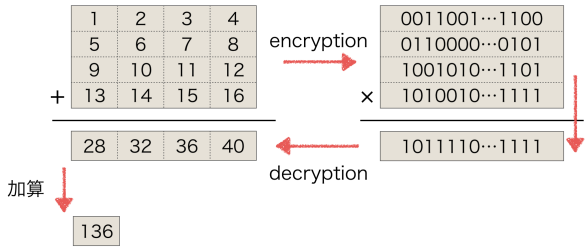

図 3　DPE を用いた総和計算の概要

での総和を求めようとする場合，**図 3** のように計算をおこなう．4 つの値ごとに暗号化し 4 つの暗号文が生成される．4 つの暗号文を総乗し，復号すると各スライスが，それぞれ該当するスライスの和を含む平文が得られる．スライスごとの総和をすべて足し合わせることで全体の総和が得られる．この方法を用いると，1 つのブロックに含む値の数を k としたとき，値を一つ一つ暗号化したものと比べて暗号文同士の乗算の回数が約 $1/k$ となるため高速化が期待できる．また，暗号文の数が約 $1/k$ となり，記憶容量の節約にもつながる．

2.3.1　選択演算を含む総和計算法

選択演算を含む総和を求める場合，各ブロックにはすべての値が含まれているため，ブロック単位での乗算を行なうとブロック内の一部の値を総和から除くことはできない．各スライス毎に正しい部分和を求めて，正しい部分和だけを足し合わせることにより選択演算を含む総和を求める．k 個の値 $v_{ij}(j = 1, ..., k)$ を結合し，ブロック毎に暗号化した n 個の暗号文 $c_i(i = 1, ..., n)$ を用意する．n 行 k 列の重み付け行列を用意し，v_{ij} を総和に含める場合は $(i, j) = 1$，含めない場合は $(i, j) = 0$ とする．$\prod_{i=1}^{n} c_i^{w_{ij}}$ で j 番目のスライスの正しい部分和を含む暗号文が得られる．暗号文を復号して得られた S を k 個の値に分割すると，j 番目の値が正しい部分和となっている．k 個の部分和を導出し，足し合わせることで 選択演算を含む総和が得られる．

選択演算を含む総和を求める場合は単純な総和と比べて乗算回数と復号回数が増加する．暗号文同士の乗算回数に関して，1 つのブロックに含む値の数を k，すべての値の数を m，総和に含む値の数を p とすると，単純な総和を求める場合には m/k 回の乗算が必要である．選択演算を含む総和を求める場合 p 回の乗算となり，値を一つづつ暗号化した場合の乗算回数と同等になる．復号回数に関して，単純な総和を求める場合には復号回数は 1 回であるが，選択演算を含む総和を求める場合には k 回となる．

2.3.2　グルーピング処理を含む総和計算法

グルーピング演算子についても，この方式で処理可能である．GROUP BY 句で指定される key が取る値ごとに重み付け行列を生成し，選択演算を含む総和と同じ要領で計

表 1　Packing 方式の検討に用いるテーブル

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_3	d_3
a_4	b_4	c_4	d_4
a_5	b_5	c_5	d_5
a_6	b_6	c_6	d_6

算する．GROUP BY 句で指定される key が 1 から m までのいずれかの値をとるとすると，$key = 1$ の重み付け行列，$key = 2$ の重み付け行列，というように m 個の重み付け行列を生成する．すべての重み付け行列を生成したら，key のすべての場合について選択演算を含む総和を求める．1 つの平文ブロックに含む値の数を k 個とすると，key が取る値ごとに k 回の復号が必要となるため CryptDB の方式と比較して復号回数が k 倍となる．

2.4　DPE の RDBMS への適用法

2 次元であるテーブルのデータを DPE で暗号化する場合，1 つの行をデータをブロックにまとめて暗号化する**行指向暗号化方式**と 1 つの列のデータをブロックにまとめて格納する**列指向暗号化方式**がある [3]．**表 1** のような 6 行 4 列のテーブルを暗号化する場合を例として 2 方式について説明する．この節において $\circ$ はビット列の結合を表す．

行指向格納方式は行ごとにデータをブロックに格納して暗号化する．例の場合，1 つ目のブロックは $m_1 = a_1 \circ b_1 \circ c_1 \circ d_1$，2 つ目のブロックは $m_2 = a_2 \circ b_2 \circ c_2 \circ d_2$ というように 6 つのブロックに値を格納し暗号化する．1 つの行に含まれるすべての値を 1 つのブロックに格納できない場合は複数のブロックに分けて格納する．1 つの暗号文の中には複数の列の値が保持されるため，一度の暗号文同士の乗算により複数の列の集約が同時に計算される．選択演算を含む総和を求めたい場合は，単純に条件から外れる行の暗号文を無視して総乗を行えば良い．

列指向格納方式は列ごとにデータをブロックに格納して暗号化する．例の場合，1 つ目のブロックは $m_1 = a_1 \circ a_2 \circ a_3 \circ a_4 \circ a_5 \circ a_6$ になる．例えば平文ブロックのサイズが 1024 bit で int の値を扱う場合，列の先頭から 32 個づつ 1 つのブロックに格納し暗号化する．選択演算を含む総和については，条件に従って重み付け行列を作成し，それを用いて 2.3 節で示した方法で求めることができる．

DPE を列指向格納方式で用い，Add-Onion のみ DSM[8] で保存する方法を本論文では **MONOMI 方式**と呼ぶ．それに対して，Add-Onion をナイーブに値を暗号化し，すべての Onion のデータを NSM で保存する方式を **CryptDB 方式**と呼ぶ．

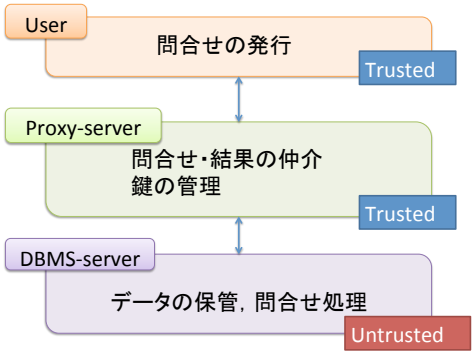

図 4　脅威モデル

3.　設計と実装

3.1　設計

提案システムは CryptDB をベースとして，プロキシサーバ，DBMS サーバの 2 つのマシンで構成する．

本研究では脅威モデルを定義し，システムが達成するセキュリティ目標を設定する．モデルを**図 4** に示す．trusted としている部分においては，システムに対する攻撃や盗聴などの脅威は発生しないとする．untrusted としている部分においては，すべてのデータに対するアクセス権限を持つサーバ管理者や攻撃者がデータの盗み見などを試みるとする．また，攻撃者はアプリケーションが発行した SQL 問合せの書き換えや DBMS が返す結果の改ざんなどは行わないことを想定する．提案システムでは，先に述べた脅威に対して DBMS に保存されるデータの機密性を保持することを目標とする．

DBMS サーバは次の機能を持つ．DBMS サーバは攻撃され得る状況を仮定するため，平文を一切扱う事なく**データの保存**と**問合せ処理**を行なう．暗号化されたデータはサイズが増大するため通常の DBMS の数値型では保持できない場合がある．加えて準同型性暗号の場合，元の平文同士の加算を実現するために暗号文同士で乗算が行われる．そのため DBMS サーバは暗号文の適切な取り扱いのために専用の型を持ち，それに対する演算をユーザ定義関数 (UDF) で定義しておく必要がある．

プロキシサーバは安全であると仮定し，アプリケーションが発行する問合せの暗号化と，DBMS が返す結果の復号を行なうため**暗号化・復号鍵の管理**と**問合せの仲介**の機能を持つ．MONOMI 方式では選択演算を含む総和を求める際，複数の暗号文タプルが返され，部分和を足しあわせて総和を求める場合がある．**部分和の併合**は平文の状態で行なうため，安全なプロキシサーバ上で行う必要がある．

Vol.2015-ARC-215 No.19
Vol.2015-OS-133 No.19
2015/5/27

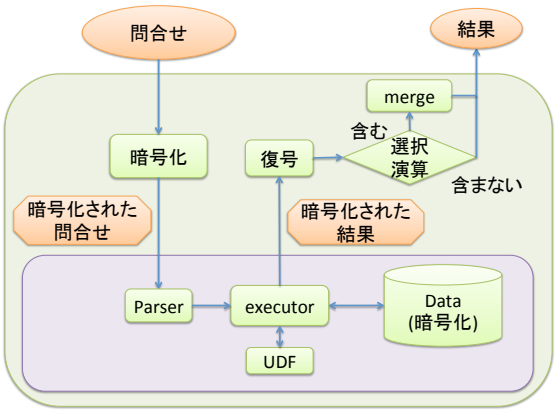

図 5　プロトタイプ DBMS の構成

3.2　実装

この節では実験に用いるプロトタイプ DBMS の実装について述べる．今回は単純化のためプロキシサーバが持つ復号などの機能も DBMS に搭載した．

3.2.1　プロトタイプ DBMS

プロトタイプとなる DBMS および暗号化システムを C++言語にて実装した．システムのモジュール構成を図 5 に示す．今回は単純化のためプロキシサーバと DBMS サーバを合体し，単一プログラムで問合せのパース，ファイル読み出し，問合せ処理，返り値の復号を行う．コードは yacc/lex によって生成されたパーサのコードを除き，全て合わせて 2245 行であった．

暗号化について，1 つの列に対してデータの種類に応じて複数の Onion を生成する．プロトタイプでは Eq-Onion と Add-Onion のみ実装した．Onion 毎に 1 つの列をつくり，行ごとにまとめてファイルに書き込む．すなわち，1 つの行には同一の平文に対応する複数の Onion が含まれる．データを MONOMI 方式で保存するため，基本的には 1 テーブル 1 ファイルで保存するが，Add-Onion のみ別のファイルに保存する．

問合せに応じて必要なファイルのみ読み込み，問合せの処理を行う．また，Onion に応じてデータの保持の方法を変える．Add-Onion 以外のデータはタプルリストの形（リレーション）でデータを保持する．選択演算で特定の行を取り除く場合はタプルを Free() する．同時に，暗号文に含まれる値の数 k，暗号文の数 n として k 行 n 列の重み付け行列 (bitmap) を生成する．Add-Onion のデータは 1 つの長いバイト列として保持する．総和計算の際には暗号文の長さごとに切り取って暗号文型の変数に格納し，計算を行う．問合せに選択演算が含まれ，bitmap が生成された場合には 2.3 節で述べた方法に従って計算を行う．

暗号文は通常の数値型には格納できないサイズとなる場合がある．また，Paillier 暗号では暗号文同士の乗算によって元の平文の加算が行われるなど，目的とする処理と実際に行われる処理に差異がある．そこでそのような暗号文の取り扱いのために以下のデータ型を定義する．

Algorithm 1 総和の併合

Input: 暗号文 $C_i(i = 1..k)$, 復号鍵 K
Output: 結果 ret
1: $ret = 0$
2: **for** C_j について $(j = 1, ..., k)$ **do**
3: $tmp = dec(K, C_j)$
4: $ret = ret + tmp[j]$ {C_j を復号した tmp は k 個の値を含む配列であり，j 番目に求める部分和が入っている．}
5: **end for**

・ **ENC_DET**

Eq-Onion のための暗号文を格納する型である．暗号化手法としては，CryptDB に準じて Blowfish[9] を用いた．int 型の平文は 64 bit ブロック暗号である Blowfish 暗号で暗号化するため，64 bit 整数の暗号文が格納される．

暗号化関数は OpenSSL ライブラリを利用した．値の等号チェックを可能にするために，共通の IV を用いて CBC モードで暗号化し，同一の平文から同一の暗号文を出力するようにした．

・ **ENC_ADD**

Add-Onion のための暗号文を格納する型である．暗号化手法としては，CryptDB に準じて Paillier 暗号を用いた．鍵長によって暗号文の長さが異なるため，任意精度整数の値が格納される．この型に対する加算の問合せでは実際には乗算が行われる．

暗号文同士の乗算，暗号化，復号などの関数は既存の実装 [10] を利用する．任意精度整数の取り扱いに関しては The GNU Multiple Precision Arithmetic Library(GMP)[11] を利用する．

選択演算を含む総和の問合せの場合には複数の暗号文が結果として返される．プロトタイプでは併合のための関数を用意した．実際にはこの関数は安全なプロキシサーバ上で動作する．この関数は復号鍵と複数の暗号文を引数として正しい平文の結果を返すため，algorithm 1 に示す処理機能を持つ．

4.　実験

この節では本研究で行った実験について述べる．3 節で述べたプロトタイプ DBMS において MONOMI 方式での総和計算を評価する．

4.1　実験環境及び比較する手法

実験を行ったマシンのスペックを表 2 に示す．又，実験で比較した手法について表 3 に示す．はじめにデータの格納方法について，RDBMS において一般的に用いられ

Vol.2015-ARC-215 No.19
Vol.2015-OS-133 No.19
2015/5/27

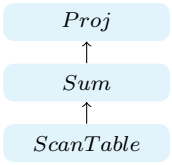

図 6　Q1 の演算木

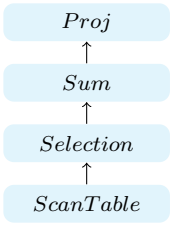

図 7　Q2 の演算木

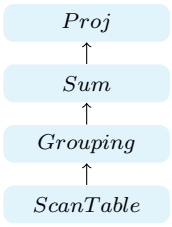

図 8　Q3 の演算木

表 2　実験環境

OS	Mac OS X 10.9.5
プロセッサ	Intel Core i7
プロセッサ速度	2.3 GHz
コア数	4
メモリ	16 GB 1600 MHz DDR3

Listing 1　実験に用いた問合せ

```
Q1. SELECT sum(val) FROM table;
Q2. SELECT sum(val) FROM table WHERE key > n;
Q3. SELECT sum(val) FROM table GROUP BY key;
```

ている NSM と MONOMI 方式を比較した．MONOMI 方式では Add-Onion のデータのみを別ファイルで保管する．row が NSM，p-col が MONOMI 方式を表す．次に暗号化に関して，CryptDB で用いている暗号化手法と DPE の性能比較した．また，平文でも実験を行い，性能劣化の度合いを比較した．plain は平文，encrypt は CryptDB の方式，DPE は効率化手法をそれぞれ表す．

効率化手法の評価のために，listing1 に示す問合せを用いて実験を行った．それぞれの演算木は**図 6**，**図 7**，**図 8** にそれぞれ示す．問合せで参照するテーブルの構造，変数の型などについては後で述べる．

4.2　総和

本節では総和計算の性能に関する実験の結果を示す．実験 1 及び実験 2 では，テーブルは**表 4** の構成のものを用いた．平文を扱う実験の際には plain，暗号文を扱う実験の際には encrypted のテーブル構造を用いた．

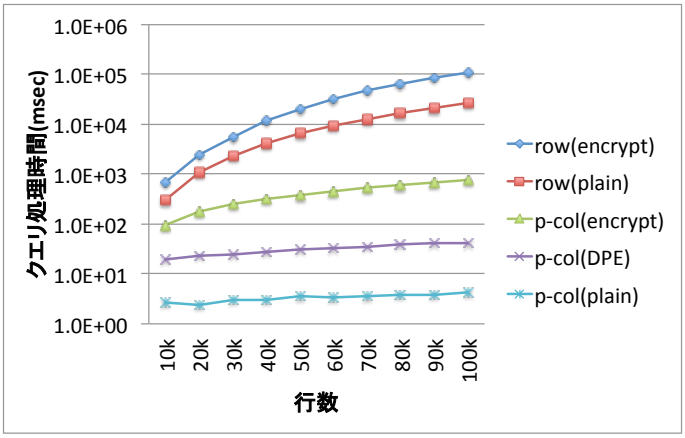

図 9　既存方式との比較

4.2.1　実験 1: 単純な総和計算

row (plain)，row (encrypt)，p-col (plain)，p-col (encrypt)，p-col (DPE) の 5 つにおいて，listing 1 の Q1 の問合せを実行した．暗号鍵は 1024 bit，p-col(DPE) において 1 つのブロックには 32 個の値を格納した．Add-Onion の暗号文は 2048 bit となる．テーブル構造は表 4 に示すものを用い，行数は 10240 行から 102400 行の 10 通りのデータで実験を行った．**図 9** に結果を示す．縦軸が問合せのパースから結果を返すまでにかかった合計時間 (msec)，横軸がテーブルの行数となっている片対数グラフである．row(encrypt) が row(plain) に対して最大で約 4 倍の遅延増加であったのに対し，p-col(DPE) は row(plain) に対して最大で約 628 倍高速となった．従って p-col(DPE) は row(encrypt) に対して 2547 倍程度の効率的である．一方，MONOMI の結果では p-col(DPE) は row(encrypt) に比べて，TPC-H Q1 が高々 17 倍程度効率的である報告がされている．TPC-H Q1 は 7 つの総和計算を含む一方，本実験における総和計算数は 1 である．この大差が生じた原因の 1 つにバックエンド DBMS が考えられる．MONOMI は PostgreSQL を用いる一方，本研究ではスクラッチから DBMS を構築している．PostgreSQL におけるクエリプロセスの複雑性が性能向上を阻害している可能性がある．我々が得たこの結果は DBMS アーキテクチャが暗号化データ処理の性能に大きく影響を与えることを示唆する．

4.3　選択

この節では選択演算を含む総和の性能に関する実験の結果を示す．

4.3.1　実験 2: 異なる選択率における総和計算問合せ処理時間の比較

選択率を変更して実験した．テーブルは**表 5** の構造を用

表 3　各手法の名称

名称	手法
row (plain)	NSM で平文を扱う．
row (encrypt)	NSM で暗号文を扱う．CryptDB の方式．
p-col (plain)	MONOMI の方式で平文を扱う．
p-col (encrypt)	MONOMI の方式で暗号文を扱う．1 つの値を 1 つの暗号文に含める．
p-col (DPE)	MONOMI の方式で暗号文を扱う．複数の値を 1 つの暗号文に含める．

表 4　実験 1，2 での table の内容

タイプ	カラム名	型	内容
plain	key	unsinged int	自然数 (32 bit)
	val	unsigned int	自然数 (32 bit)
encrypted	key_det	ENC_DET(uint64_t)	自然数を暗号化した暗号文 (64 bit)
	val_det	ENC_DET(uint64_t)	自然数を暗号化した暗号文 (64 bit)
	val_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (任意長)

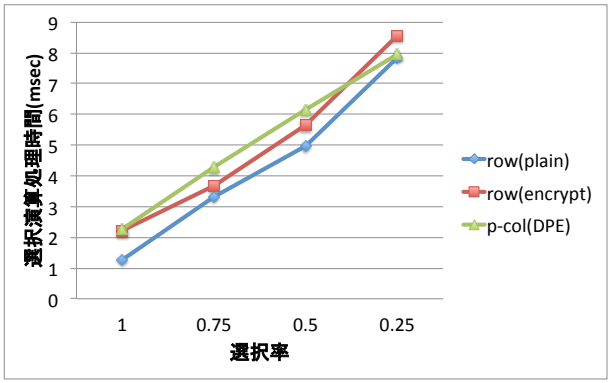

図 10　異なる選択率における選択処理時間の比較

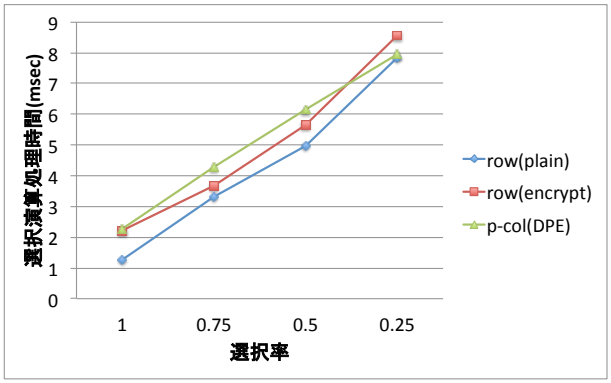

図 11　異なる選択率における総和計算処理時間の比較

い，データの行数は 51200 とした．はじめに選択処理に関する結果を示す．問合せは選択率毎に listing 2 に示すものを利用した．

図 10 のグラフは，異なる選択率における選択演算の処理時間を表している．縦軸が処理時間 (msec)，横軸が選択率となっている．ここでは有意な差は見られなかった．p-col(DPE) では row(encrypt) と共通の選択演算処理に加えて，2.3.2 節で述べたように総和計算のための bitmap を生成しているため，そのためのコストが現れるかと予測していたが誤差の範囲内に収まる結果になった．

次に**図 11** に総和計算処理に関する結果を示す．縦軸が

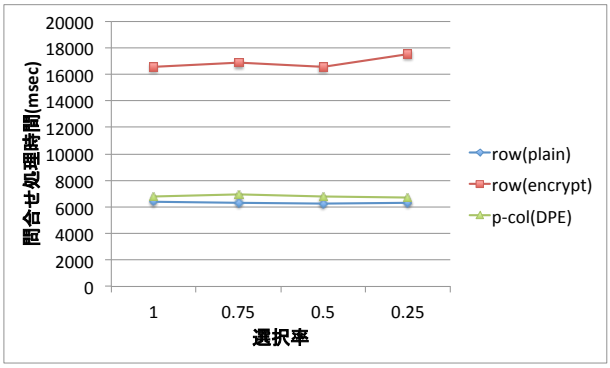

図 12　異なる選択率における総和問合せ処理時間の比較

表 6　実験 2 の I/O と復号に関する結果

	row(plain)	row(encrypt)	p-col(DPE)
initTable	6115.1 msec	18413.5 msec	6234.2 msec
initCol	-	-	3.5 msec
decryption	-	11.1 msec	175.4 msec

処理時間 (msec)，横軸が選択率となっている．選択率が低くなるほど，必要な加算回数が減少するため，処理時間は短くなる．p-col(DPE) は選択演算を含む総和で処理が複雑化するが，加算回数は row(encrypt) と同一である．

図 12 に選択演算を含む総和計算問合せの処理時間を比較したグラフを示す．縦軸が問い合わせ処理時間 (msec)，横軸が選択率となっている．row(plain) と p-col(DPE) を比較すると最大で 1.09 倍程度の遅延増加となった．

表 6 に I/O と復号処理に関する結果を示す．initTable はファイルを読み込みタプルリストを生成する関数，initCol は総和を求める列のファイルを読み込み配列にデータを格納する関数である．row(encrypt) では initTable によって Eq-Onion と Add-Onion のデータを読みだしており，p-col(DPE) では initTable で Eq-Onion，initCol で Add-Onion のデータを読み出している．I/O コストに関しては，読み込むファイルのサイズから row(plain)，p-col(DPE)，row(encrypt) の順に低くなった．今回の実験

表 5　実験 2 での table の内容

タイプ	カラム名	型	内容
plain	key	unsinged int	自然数 (32 bit)
	val	unsigned int	自然数 (32 bit)
encrypted	key	unsigned int	自然数 (32 bit) 範囲指定問合せ対応のため
	val_det	ENC_DET(uint64_t)	自然数を暗号化した暗号文 (64 bit)
	val_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)

Listing 2　実験に用いた問合せ

```
selectivity 1:    SELECT sum(val) from table where key > 0;
selectivity 0.75: SELECT sum(val) from table where key > 12800;
selectivity 0.5:  SELECT sum(val) from table where key > 25600;
selectivity 0.25: SELECT sum(val) from table where key > 38400;
```

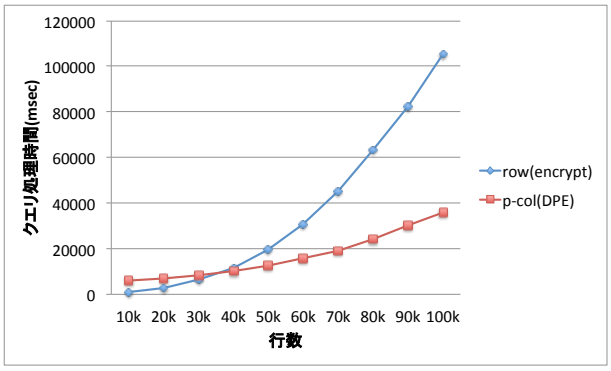

図 13　グルーピング処理を含む総和計算処理時間の比較

では p-col(DPE) は DPE により Add-Onion のデータサイズが平文と同一になっている一方で，row(encrypt) は暗号文長 (2048*bit*)× 行数 (51200) と大きくなるため他の 2 方式と比べて I/O コストが高くなった．復号に関しては，p-col(DPE) は複数の暗号文を返すため，部分和の併合処理のために複数回復号処理をせねばならず，row(plain) よりもコストがかかる．しかしながら，復号コストは鍵長に対応して固定である．そのため，同一の鍵長ではテーブルの行数が多いほどに p-col(DPE) の方が高速に処理可能であると考えられる．

4.3.2　実験 3: グルーピング処理を含む総和計算問合せ処理時間の比較

Listing 1 の Q3 の問合せにおいて，row(encrypt) と p-col(DPE) で比較実験を行った．テーブルは実験 1，2 と同様に表 4 の構成のものを用いた．行数は 10240 行から 102400 行までの 10 通りで行った．また，それぞれの場合において全体の 32 の 1 ごとに同一のキーを設定し，全部で 32 個のグループとなるようにした．例えば 10240 行のデータの場合，1 行目から 319 行目までが $key = 0$, 320 行目から 639 行目までが $key = 1$，というように設定した．結果を図 **13** に示す．

図 13 のグラフは縦軸が復号時間を含む問合せ処理時間，横軸が行数となっている．グルーピング処理を含む総和計算の問合せの場合，row(encrypt) と p-col(DPE) で暗号文同士の乗算の回数は同一である．p-col(DPE) では group by が指定する key がテーブル内で取りうる値すべてに対して bitmap を生成し，2.3 節の方法を行なう必要がある．それに加えて algorithm 1 に示す部分和の併合処理を行なう必要がある．行数を n，key が取りうる値の数を g，平文ブロックに含む値の数 k とした時，結果を求めるのに必要な乗算の回数及び復号の回数を表 **7** に示す．

表 7　結果の導出に必要な計算回数

	row(encrypt)	p-col(DPE)
復号回数	g	$k \times g$
乗算回数	$n - g$	$n - g$

key が取りうる値の数と平文ブロックに含む値の数に依存して p-col(DPE) では復号回数が増加するため，テーブルの行数が少ない場合は row(encrypt) のほうが高速に問合せを処理する結果となった．一方，テーブルの行数が多くなると読み込むデータサイズが小さい p-col(DPE) のほうが有利となり，row(encrypt) は指数的に処理時間が増加した．

5.　関連研究

本研究は効率的かつセキュアなデータ処理基盤を構築するものであるので，関連研究としては暗号化データベースシステム，検索可能暗号，秘密分散に関するものが挙げられる．暗号化データベースシステムの研究としては CryptDB[1], [2] ならびに MONOMI[3] が挙げられる．CryptDB は暗号化データベースシステムの先駆けであり，MONOMI はその高性能化システムである．MONOMI においては総和計算を効率化するために DPE を採用し，さらに DPE のページレイアウトとして DSM を用いている．これらの研究はいずれも一般的な DBMS を用いて性能評価を行っている．一方，本研究ではスクラッチから暗号化データベースシステムを実装し，総和計算に関して CryptDB 方式と MONOMI 方式を評価している．TPC-H Q1 につ

いて MONOMI 方式は CryptDB 方式よりも 17 倍程度の性能向上を示しているが，総和計算のみの単純な場合においてはその性能差が 2547 倍程度まで広がるという興味深い事実が得られた．

検索可能暗号としては PEKS[12]，ORAM[13], [14], [15], [16] 等が知られている．PEKS は暗号文を復号することなくキーワード検索出来る暗号技術である．アクセス制御が可能である ID ベース暗号と組み合わせた ID ベース検索可能暗号 [17] も考案されている．ORAM は暗号化したデータを自由に検索することができ，暗号データに対して演算が行われる度にデータの格納位置を変化させることでデータの秘匿に加えてデータのアクセスパターンの秘匿も可能にしている．秘密分散に関しては，マルチパーティ計算を用いた方法 [18], [19] が挙げられる．

6. 結論と今後の課題

本研究は，Paillier 暗号における平文ブロックの稠密化による暗号文データ容量及び総和計算コスト削減手法について，クリーンスレート設計の DBMS 上で評価を行った．単純な総和計算の問合せに対しては，NSM で平文を扱う場合よりも MONOMI 方式で DPE の暗号文を扱うほうが高速となり，10 万行のデータに対する実験では CryptDB 方式と比べて 2547 倍の高速化を達成した．一方，MONOMI の結果では p-col(DPE) は row(encrypt) に比べて，TPC-H Q1 が高々 17 倍程度効率的である報告がされている．TPC-H Q1 は 7 つの総和計算を含む一方，本実験における総和計算数は 1 である．この大差が生じた原因の 1 つにバックエンド DBMS が考えられる．MONOMI は PostgreSQL を用いる一方，本研究ではスクラッチから DBMS を構築している．PostgreSQL におけるクエリプロセスの複雑性が性能向上を阻害している可能せがある．我々が得たこの結果は DBMS アーキテクチャが暗号化データ処理の性能に大きく影響を与えることを示唆する．

今後の課題は TPC-H などによるマクロ的評価，暗号文スライス中の桁あふれへの対処，並列処理による高性能化である．

謝辞 本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，JST CREST「EBD : 次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」，JST 育成研究「AS262Z02895H」による．

参考文献

[1] Popa, R. A., Redfield, C. M. S., Zeldovich, N. and Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing., *SOSP*, pp. 85–100 (2011).

[2] Popa, R. A., Redfield, C. M. S., Zeldovich, N. and Balakrishnan, H.: CryptDB: processing queries on an encrypted database, *Commun. ACM*, Vol. 55, No. 9, pp. 103–111 (2012).

[3] Tu, S., Kaashoek, M. F., Madden, S. and Zeldovich, N.: Processing Analytical Queries over Encrypted Data., *PVLDB 6(5)*, pp. 289–300 (2013).

[4] Ferretti, L., Pierazzi, F., Colajanni, M. and Marchetti, M.: Scalable Architecture for Multi-User Encrypted SQL Operations on Cloud Database Services., *IEEE T. Cloud Computing 2(4)*, pp. 448–458 (2014).

[5] Cooney, M.: IBM touts encryption innovation, computerworld (online), available from ⟨http://www.computerworld.com/article/2526031/security0/ibm-touts-encryption-innovation.html⟩ (accessed 2015-04-20).

[6] Ge, T. and Zdonik, S. B.: Answering Aggregation Queries in a Secure System Model., *VLDB*, pp. 519–530 (2007).

[7] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes., *EUROCRYPT*, pp. 223–238 (1999).

[8] Copeland, G. P. and Khoshafian, S.: A Decomposition Storage Model, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985.*, pp. 268–279 (1985).

[9] Schneier, B.: Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)., *FSE*, pp. 191–204 (1993).

[10] Bethencourt, J.: Paillier Library, Advanced Crypto Software Collection (online), available from ⟨http://acsc.cs.utexas.edu/libpaillier/⟩ (accessed 2015-04-20).

[11] GNU project: The GNU MP Bignum Library, GNU-project (online), available from ⟨https://gmplib.org/⟩ (accessed 2015-04-20).

[12] Boneh, D., Crescenzo, G. D., Ostrovsky, R. and Persiano, G.: Public Key Encryption with Keyword Search, *EUROCRYOTO*, pp. 506–511 (2004).

[13] Goldreich, O.: Towards a Theory of Software Protection and Simulation by Oblivious RAMs, *STOC*, pp. 182–194 (1987).

[14] Gentry, C., Goldman, K. A., Halevi, S., Jutla, C. S., Raykova, M. and Wichs:, D.: Optimizing ORAM and Using It Efficiently for Secure Computation., *Privacy Enhancing Technologies*, pp. 1–18 (2013).

[15] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C. W., Ren, L., Yu, X. and Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol., *ACM Conference on Computer and Communications Security*, pp. 299–310 (2013).

[16] Shi, E., Chan, T.-H. H., Stefanov, E. and Li, M.: Oblivious RAM with O((log N)3) Worst-Case Cost., *IACR Cryptology ePrint Archive*, p. 407 (2011).

[17] Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P. and Shi, H.: Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions, *CRYPTO*, pp. 205–222 (2005).

[18] 志村正法，西出隆志，宮崎邦彦，吉浦 裕：秘密分散データベースの構造演算を可能にするマルチパーティプロトコルを用いた関係代数演算，情報処理学会論文誌，pp. 1563–1578 (2010).

[19] 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, *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, ACM, pp. 1395–1406 (2014).