Vol.2017-DBS-164 No.8
2017/1/17

CryptDB の処理時間分析と高速化案

合田真也[†1]　渡辺知恵美[†2]　北川博之[†3]

概要:2011年にRalucaらによって提案された,セキュリティを保護したクラウドデータベースシステムであるCryptDBについて，TPC-C の実行を通して実装していない機能についての調査，各種クエリを処理する際の処理時間の分析を行う．また，インデックスの利用，オニオンの事前復号による処理の高速化を試みる．

1. はじめに

近年，個人情報の漏洩問題やセキュリティ意識の高まりから，機密データの安全性への関心が高まっている．それと同時にデータベースサーバのクラウド化，管理運用のアウトソーシングが普及しつつあるが，データベースサーバの管理運用をアウトソーシングした場合，サーバ管理者が信用できず，機密データの安全性が保障できないという問題がある．この問題を解決するための研究として，暗号化データベースの研究が進められている．2002 年にはHacigums 氏によりサーバのデータを暗号化, 索引を用いてデータ検索を行う手法[1]が提案された．しかしこの手法には索引の内容から暗号化データの内容が推測できてしまう問題があった．暗号化データベースの機密性と効率的な処理の両立についての研究は以降も行われ，2011 年にはRaluca らによって，暗号化したまま効率的な処理が行うことができるデータベースシステムとしてCryptDB[2]が提案された．このシステムはデータを暗号化した状態で効率的に比較や集約，結合を行うことができるシステムである．研究では実装，評価実験まで行われており，実装されたものがオープンソースで公開されているため，誰でも利用可能である．ただし，パフォーマンスの問題や未実装の機能等の問題があり実用化には至っていない．しかし，実際に動作させることのできる暗号化データベースシステムであり，その注目度は高い．2012 年には Stephen らによりCryptDB の改善，高速化を行ったシステムである“MONOMI”[3]が提案されている．

前述の通り実際に利用できる暗号化データベースシステムとしてCryptDB は注目されているが，CryptDB を実際に利用した場合の機能やパフォーマンスについて分析した資料は少ない．本論文ではデータベースのベンチマークツールである TPC-C の実行を通し CryptDB の未実装の機能，パフォーマンスについて調査，分析を行う．全体としてのパフォーマンスのみでなく，CryptDB のシステム内部でかかっている時間についても分析を行う．また，暗号化データベース全体の課題ともなっている処理速度の問題の改善案として，インデックスの追加と CryptDB のオニオン構造の操作による高速化を試みる．

†1 筑波大学情報学群情報科学類
†2 筑波大学システム情報系
†3 筑波大学計算科学研究センター

2. CryptDB

2.1 CryptDB 概要

2011 年に MIT の Raluca らによって提案された, DB サーバの管理者が信用できない場合にも機密データを保護できるデータベースシステムである．データを暗号化した状態で DB サーバに格納し，暗号化した状態でクエリを実行することでデータの機密性を保持する．構成としては図 1 の通り CryptDB プロキシを設置し，利用者から発行されたクエリを CryptDB プロキシが受け取り，暗号化されたデータベースで実行できる形に変換，実行して結果を得る．クエリ実行した結果得られる値は暗号化されているためそれを復号し利用者に送信する．データベースサーバではユーザ定義関数（UDF）を用いて処理を行うため，サーバのシステムに変更を加える必要はない．

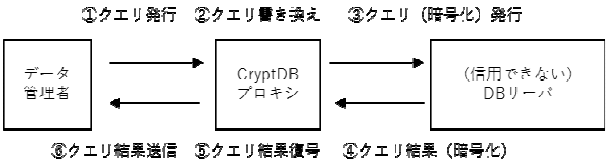

図 1.CryptDB 構成

データを暗号化した状態でクエリを実行するために以下にて紹介する暗号化方式を組み合わせて利用する. なお, 以下には紹介しないが文字列探索を行う SEARCH，大小比較での JOIN を行う OPE-JOIN が提案されている．

2.2 暗号化データへのクエリ実行

暗号化した状態のデータにクエリを実行するために，CryptDB では複数の暗号化方式を組み合わせて利用している．実装されている暗号化方式は以下の 5 つである．

(1) **RND**

ランダムな初期化ベクトルを用いた AES，あるいは Blowfish 暗号による暗号化を行う．一切クエリは実行できないが暗号データから内容を推測することはできない．

(2) **DET**

固定された初期化ベクトルを用いた AES，あるいは Blowfish 暗号による暗号化を行う．暗号化前に同じ値だった値は暗号化を行った後も同じ値となるため，イコール比較を行うクエリが実行できる．

(3) **OPE**

2004 年，Agrawal らにより提案された Order preserving encryption[4]により暗号化を行う. 暗号化前の数値の大小関係が保持されるため, 大小比較を行うクエリが実行できる.

(4) **HOM**

Paillier 暗号[5]による暗号化を行う. Paillier 暗号は準同型暗号であるため，暗号化した状態での加算が可能であり，集約演算（SUM）を行うクエリが実行可能である.

(5) **DET-JOIN**

DET による暗号化を行った値の暗号鍵を変換し，復号することなく二つの値のイコール比較を可能にする．等結合を行うことができる.

2.3 オニオン構造

CryptDB では「2.2 暗号化データへのクエリ実行」で述べたとおり，複数の暗号化方式を利用するが，中には暗号化した状態でもある程度情報を漏洩する暗号化方式がある．例えば，攻撃者は DB サーバのデータを見たとき，データの具体的な内容は暗号化されているため分からないが，DET で暗号化されている場合同じ値が格納されていることやその個数が分かるし，OPE ではデータ同士の大小関係が分かってしまう.

この問題を防ぐために CryptDB ではオニオン構造でのデータの格納を提案している．オニオン構造では，図 2 に示す様に OPE 等の機密性の弱い暗号化方式で暗号化したデータを，RND 等の機密性の強い暗号化方式で暗号化し，DB サーバに格納する．実行時のオニオンの利用方法として, 例えばクエリの実行時に大小比較を行う場合には Order オニオンを利用する．OPE で暗号化した値が必要になるため，RND での暗号化を復号し，OPE で暗号化した値を利用し比較を行う．この構造を用いることで，比較を行わない場合の機密性を保持することができる．CryptDB では値を格納する際, 図 2 の通り 3 種類のオニオンに分け暗号化し，格納する．Eq オニオンではイコール比較，Order オニオンでは大小比較，HOM オニオンでは暗号化したままでの加算を行うクエリをサポートする.

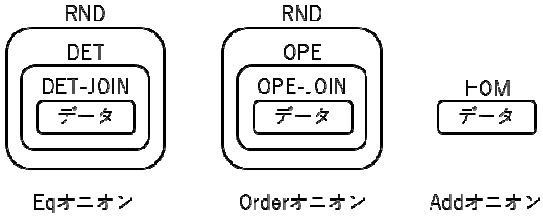

図 2.CryptDB の各種オニオン

下層のオニオンを複合した際に等価関係や大小関係等の情報は漏洩したものとし，再度の暗号化は行われない.

なお，データベース内では図 3 のように 1 つのオニオンに対して 1 つのカラムが作成され, Eq オニオンのカラムを利用することでイコール比較が，Order オニオンのカラムを利用することで大小比較が可能である．なお，図 2 では例としてオニオンが DET,OPE で暗号化された状態まで復号された状態を示す.（テーブル作成後，イコール比較または大小比較が行われるまではそれぞれ RND で暗号化されたデータが格納されている.）

id	income
1	100
2	100
3	200

tableA_Eq	tableA_Order	tableA_ADD	tableB_Eq	tableB_Order	tableB_ADD
3190…4281	1353…3489	(binary)	7215…4457	3884…6120	(binary)
4892…0058	1903…6087	(binary)	7215…4457	3884…6120	(binary)
1482…4602	1992…4880	(binary)	1557…8658	8597…7860	(binary)

図 3.CryptDB 使用時の DB サーバ内データ例

3. 実験概要：TPC-C

今回 CryptDB の性能を評価するためにデータベースのベンチマークツールである TPC-C を利用する. TPC-C では卸売業による処理をシミュレートし，データベースの性能を測定する．処理は以下の注文処理・支払い処理・注文状況確認処理・配送処理・在庫確認処理がそれぞれ 10 : 10 : 1 : 1 : 1 の割合で実行され，1%の割合でロールバックが必要となる．1 分あたりに実行可能な注文処理のトランザクション数が TPC-C のスコアとなる.

(1) **注文処理**

ランダムなアイテムを選択し，値段の確認(SELECT)，新規注文への登録(INSERT)，在庫の減少(UPDATE)，配送処理への登録(INSERT)を行う．TPC-C では一分間に実行できた注文処理の数がスコアとなる.

(2) **支払い処理**

顧客の住所，倉庫の住所，注文内容から注文の料金を計算(SELECT)，支払い(UPDATE)を行う.

(3) **注文状況確認処理**

顧客の注文した商品，配送状況の確認(SELECT)を行う.

(4) **配送処理**

商品の選択(SELECT)，配送の実施(UPDATE)を行う.

(5) **在庫確認処理**

在庫の確認(SELECT)を行う.

デフォルトではデータ規模はテスト時に指定する倉庫数を基準に以下のように設定される.

倉庫数：テスト時に指定する.
配送区域：倉庫数×10
顧客数：配送区域×3,000
在庫：倉庫数×商品数
商品数：100,000

4. CryptDB での TPC-C 実行時の問題点

CryptDB にて TPC-C によるテストを行う際にいくつかの

問題があり，実行することができなかった．ここではその問題点について説明する．また，本来であればCryptDBを改善し問題を解決するべきではあるが，今回の実験は問題点を調査することが目的であり，システムの改修までは範囲としない．そのため，今回はテストプログラムを実行できるよう改造し実験を行った．

4.1 CryptDB上でのTPC-C実行時の問題と解決

(1) カラム型として使用不可能な型が使われている

テストプログラムにて decimal,datetime が利用されているが，表1に示す通り，CryptDBはdecimal,datetimeに対応していない．decimalの代わりにintを利用し，プログラムの処理をintで行うよう合わせて変更した．また，datetimeの代わりにvarcharを利用し，文字列として処理するようプログラムを変更した．

表 1.CryptDBにて利用可能/不可能なカラム型一覧

利用可	TINYINT,SMALLINT,MEDIUMINT,INTEGER,BIGINT, CHAR(),VARCHAR(), TINYBLOB,BLOB,MEDIUMBLOB,LONGBLOB ※数値型はすべてunsignedで作成される
利用不可	FLOAT,DOUBLE,DECIMAL, DATE,DATETIME,TIMESTAMP,TIME,YEAR, ENUM(),SET()

(2) 外部キー制約が利用不可能

CryptDB ではサーバ単体ではデータを他カラムのデータと比較することができないため，外部キー制約を利用することができない．テストプログラムのテーブル作成時に外部キー制約を設定しないよう変更した．

(3) prepared statement が利用不可能

SQL の構文を先にコンパイルしておく高速化手法であるprepared statementはCryptDBで未実装であるため利用できない．テストプログラムではprepared statementを利用せず，クエリを随時発行するよう変更した．

(4) int 型に対してマイナスの値が格納不可能

テストプログラムにてint型のカラムに負の値を格納することがあるが，表1にて示したとおりCryptDBでは数値型はすべてunsignedで作成されるため,実行することができない．正の値を格納，計算の際にプログラムで負の値として計算するよう変更した．

(5) 検索条件内に四則演算を含むクエリが実行不可能

検索条件内に四則演算を含むクエリ（例 WHERE price < max(price) -20）を実行することができない．テストプログラムでは事前に四則演算を行い検索条件に利用するよう変更した．なお，テストプログラムで行われる四則演算は前例のように，値を定数に置き換えることのできるクエリであったため今回の対応が可能であるが，where min_price < price - 30 のような，他カラムを演算した結果を条件とするクエリは実行することができない．

(6) 一部関数（COALESCE()）がクエリ内で利用不可能

CryptDB で未実装の関数があり，実行することができない．テストプログラムでは引数内で NULL でない最初の値を取得する COALESCE()を実行することができなかった．本関数を用いずにクエリを実行，結果が NULL だった場合に規定の値に置き換えるようプログラムを変更した．

(7) サブクエリが実行不可能

サブクエリを実行することができない．テストプログラムではサブクエリで1つの値（商品の値段の最大値等）を取得しているため，サブクエリの値を取得するクエリと本来のクエリの2つに分け実行するようプログラムを変更した．サブクエリで複数値を取得するようなクエリは実行することができない．

5. TPC-C テスト実行

TPC-C によるテストを行うため,本実験では TPC-C の簡易版であるプログラムの tpcc-mysql を利用する．ただし，前述の問題があり CryptDB 上で実行することができないため，テストプログラムの変更を行った．プログラムを変更し CryptDB 環境で実行可能とした tpcc-mysql を以降，tpcc-cryptdb と称する．

5.1 tpcc-cryptdb パラメータ

tpcc-mysql で通常利用するデータ数を CryptDB にて使うと実行速度が非常に遅くなるため，データ数を以下の通り設定している．

倉庫数：5
配送区域：倉庫数×1（デフォルトの 10 分の 1）
顧客数：配送区域×300（デフォルトの 10 分の 1）
在庫：倉庫数×商品数
商品数：10000 （デフォルトの 10 分の 1）
注文：配送区域×300（デフォルトの 10 分の 1）

以降，実験の際には同時コネクト数 1，ramp up 時間（計測を行わない動作時間）60 秒,測定時間 300 秒で実行する．

5.2 実験環境・ハードウェア

実行環境は CryptDB＋MySQL の構成とし，CryptDB のプロキシをクライアントマシンにて動作させる．クライアントとサーバ間は同一 LAN 内に配置し実験を行う．

サーバマシン：

CPU	3.4GHz Intel Core i7-2600
RAM	16GB
MySQL	5.3.34 server

クライアントマシン：

CPU	1.8GHz Intel Core i7-4500U
RAM	4GB

5.3 TPCC-CryptDB 実施

tpcc-cryptdb を，MySQL 環境（暗号化を行わず，CryptDB プロキシも利用しない環境）および CryptDB 環境にて実行し，比較を行う．なお，tpcc-cryptdb では 1 分間に実行した注文処理のトランザクション数がスコアとなる．

表 2.MySQL 環境と CryptDB 環境での TPC-C 実施

tpcc-cryptdb	transaction/m
MySQL 環境	549.600
CryptDB 環境	25.600

MySQL と比較し，CryptDB での実行には約 22 倍の時間がかかった．以降，この実行にかかっている時間について詳細に分析していく．

6. クエリタイプごとの実行速度

tpcc-cryptdb で実行されるクエリが 35 種類あるが，これをクエリタイプごとに分類し，CryptDB 環境での実行時間をそれぞれ計測する．クエリタイプは以下の 8 種類に分類した．

・**SELECT_EQUAL**：イコール比較を含む検索にて選択を行うクエリ，DET を利用する．
・**SELECT_RANGE**：大小比較を含む検索にて選択を行うクエリ，OPE を利用する．
・**SELECT_JOIN**：等結合の後，**SELECT_EQUAL** を行うクエリ，DET-JOIN を利用する．
・**SELECT_SUM**：集約演算(SUM)と選択を行うクエリ，HOM を利用する．
・**INSERT**
・**DELETE**
・**UPDATE_SET**：UPDATE のうち，新たに値を設定するクエリ（例：PRICE = 20）
・**UPDATE_INC**：UPDATE のうち，値に加算を行うクエリ(例：PRICE = PRICE + 20)

tpcc-cryptdb を MySQL 環境，CryptDB 環境にて実施し，以上の 8 タイプについて，それぞれの実行時間を計測した．なお，クエリを実行するためにオニオンの復号が必要になるが，オニオンの復号は全て ramp up 時間（測定を行わない実行時間）に行われるため，実験の結果に影響しない．

表 3.クエリタイプごとの実行速度(ms)

	Mysql	CryptDB	倍率 (Mysql→cryptdb)
SELECT_EQUAL	0.362	15.356	42.362
SELECT_RANGE	0.195	12.587	64.561
SELECT_JOIN	0.544	15.571	28.600
SELECT_SUM	0.494	31.976	64.713
INSERT	0.344	69.628	202.544
DELETE	0.519	11.381	21.940
UPDATE_SET	0.351	32.750	93.220
UPDATE_INC	0.470	54.234	115.432
AVERAGE	0.410	30.435	74.240

表 3 の結果として，DET を利用する SELECT_EQUAL や，OPE を利用する SELECT_RANGE の場合にも他のクエリタイプと同様に実行に時間がかかっていることがわかる．しかし，CryptDB の DET,OPE で暗号化した値は暗号化前の等価関係，大小関係が保存されている状態でサーバに格納されているため，サーバでのクエリ実行時間は変わらないはずである．この実験の結果から，暗号化を行ったこと以外に処理に時間がかかっている原因があることが予想される．この点について次項にて詳細な実験を行う．その他のクエリタイプでは INSERT，UPDATE_INC が CryptDB 環境で特に遅くなっているが，これはデータ挿入・変更時に各オニオンに適合するよう暗号化する必要があることや，処理を行わなければならないカラム数が増えていることが原因である．

7. CryptDB 内での処理時間

7.1 CryptDB 内での処理時間

前項の実験にて，CryptDB 環境での実行の場合，全てのクエリが MySQL で実行する場合と比べて遅くなっていることがわかる．より詳細な実行時間の調査のため，CryptDB での各処理にかかっている時間を計測した．

・**処理準備**：処理前の初期化の処理時間
・**クエリ書き換え**：クエリ内容の暗号化・テーブル名の変更等の処理時間
・**クエリ実行準備**：クエリ実行準備の処理時間
・**クエリ実行**：クエリ実行の処理時間
・**表示準備**：クエリ結果取得から表示までの準備処理時間
・**プロキシ表示**：プロキシ画面にクエリ結果（暗号化）を表示するための処理時間
・**クライアント表示**：クライアント画面にクエリ結果（復号）を表示するための処理時間

表 4.CryptDB 内処理時間(ms)

	SELECT_EQUAL	SELECT_RANGE	SELECT_JOIN	SELECT_SUM
処理準備	0.061	0.111	0.049	0.072
クエリ書き換え	3.301	3.453	5.613	3.692
クエリ実行準備	0.003	0.002	0.003	0.003
クエリ実行	3.764	4.460	3.687	22.978
表示準備	1.315	0.411	2.389	0.794
プロキシ表示	2.155	0.498	0.201	0.324
クライアント表示	4.757	4.066	3.629	4.113
総時間	15.356	12.587	15.571	31.976

表 4 の結果から，SELECT_EQUAL や SELECT_RANGE 等の，暗号化した場合と非暗号化の場合で理論的に実行時間が変わらないはずのクエリ実行にも時間がかかっていることがわかる（なお，MySQL 環境で実行した場合，クライアント表示等も含めたクエリ処理全体で約 0.3ms で実行される）．原因として暗号化ではなく CryptDB プロキシでの処理に時間がかかっていることが予想される．この確認のため，暗号化を行っていないデータベースに対して，暗号化を行わないよう変更した CryptDB プロキシを通してアクセスした場合の処理時間について計測，比較を行う．この実験により CryptDB 内の暗号化部分以外の処理にかかる時間を計測する．

表 5.CryptDB 非暗号化時処理時間(ms)

	SELECT_EQUAL		SELECT_RANGE		SELECT_SUM	
	暗号化	非暗号化	暗号化	非暗号化	暗号化	非暗号化
処理準備	0.061	0.037	0.111	0.191	0.072	0.122
クエリ書き換え	3.301	1.043	3.453	2.250	3.692	1.782
クエリ実行準備	0.003	0.016	0.002	0.064	0.003	0.084
クエリ実行	3.764	3.516	4.460	3.245	22.978	1.034
表示準備	1.315	0.000	0.411	0.000	0.794	0.000
プロキシ表示	2.155	0.000	0.498	0.000	0.324	0.000
クライアント表示	4.757	3.426	4.066	3.857	4.113	3.870
総時間	15.356	8.038	12.587	9.607	31.976	6.892

表 5 より，SELECT_EQUAL,SELECT_RANGE の場合に暗号化を行う場合と行わない場合でクエリ実行時間にほぼ差はないことがわかる．また，暗号化に HOM を使用しておりクエリ実行に処理がかかる SELECT_SUM では暗号化時と非暗号化時で実行時間が大きく異なる．このことから，CryptDB の処理時間の内訳として，暗号化や SQL の内容に関係のない処理に時間がかかっていると思われる．この CryptDB の動作に，SQL の内容に関係なくかかっている時間を計測するべく，次項ではデータ件数を変化させて実験を行う．

7.2 件数を変化させた場合の処理時間変化

データの件数を変化させて同様の実験を行い，クエリ件数に関係なく CryptDB の処理でかかる時間を推定する．データ件数を変化させるため，tpcc-cryptdb のスケールファクターである倉庫数を 2 倍にした場合と 5 倍にした場合を表 5 の結果と比較する．データ数により処理時間が変化するクエリとして SELECT_RANGE を，データ件数により処理時間が変化しないクエリとして INSERT の結果をそれぞれ表に示す．

表 6.データ数を変化させた場合の CryptDB 処理時間

	SELECT_RANGE			INSERT		
	×1	×2	×5	×1	×2	×5
処理準備	0.1106	0.1105	0.0624	0.0618	0.0612	0.0559
クエリ書き換え	3.4532	3.5155	1.9677	53.942	56.711	55.906
クエリ実行準備	0.0016	0.0015	0.0008	0.0008	0.0008	0.0008
クエリ実行	4.4599	6.2661	19.166	4.3582	4.3432	4.7387
表示準備	0.4113	0.406	0.2379	6.1976	6.0668	5.8885
プロキシ表示	0.4976	0.4889	0.2921	6.2095	6.0789	5.8988
クライアント表示	4.0659	4.0526	3.1885	3.663	3.5246	3.4665
総時間	12.587	14.434	24.677	68.234	70.718	70.066

表 6 の通り，データ件数が増加することにより SELECT_RANGE ではクエリ実行時間が増加する．SELECT_RANGE の処理総時間はほぼ線形に増加しており，データ件数が 0 の場合の処理時間，つまりクエリに関係なく CryptDB の処理でかかっている時間は約 10.2ms であると推定される．今回の tpcc-cryptdb の実験結果を見ると，この時間が SELECT クエリの実行時間の大部分を占めており，システムを高速化することは CryptDB の処理時間を高速化する上で重要であることがわかる．

8. 高速化案 1:インデックス追加

CryptDB の処理時間の計測は以上とし，以降は CryptDB の処理を高速化する手法について考える．ここでは，インデックスの利用によるクエリ実行の高速化を試みる．インデックスの状態の調査後，インデックスを作成した状態で tpcc-cryptdb を実施し，効果を検証する．

8.1 インデックス状態の調査

tpcc-cryptdb を実行する際に利用されているインデックスについて調査する．

MySQL 環境にて tpcc-cryptdb でのデータ作成を行った後のインデックスを調べると，主キーのカラムに対するインデックス，およびテーブル作成後に作成するインデックス（CREATE INDEX の実行）が作成されている．CryptDB 環境では主キーのカラムに対しては Order オニオン（大小比較用）のみにインデックスが作成され，CREATE INDEX が実行されたカラムについては Eq オニオン（イコール比較用），Order オニオンの両方にインデックスが作成されている．

8.2 インデックスの追加

一般的に主キーをイコール比較して利用する場合が多いものと思われるが，現状 CryptDB では Eq オニオンにはインデックスが作成されていない．そこで，主キーに対して手動でインデックスを作成し，Eq オニオンにてインデックスを利用できるようにして実行時間を計測する．ただし，データ作成直後にインデックスの作成を行うとオニオンの最上位層のデータ（RND 等）に対してインデックスが作成され，実行時に利用できない．これを防ぐためにオニオンの状態を tpcc-cryptdb 実行後に合わせるよう手動で調整した後，インデックスの作成を行った．

8.3 インデックス追加時の実行速度

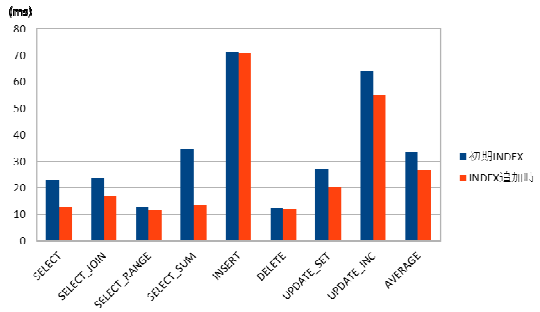

図 4.インデックス追加時の実行速度

図 4 に示すとおり，INSERT，DELETE のクエリではほぼ効果が見られなかったが，他のタイプのクエリでは効果が見られ，全体平均として約 80%の時間で実行することができた．特に主キー3 件を条件とする検索を行う UPDATE クエリでは従来の約 6%の時間で実行が可能になる等，大きな効果が出たクエリも見られる．

9. 高速化案 2：オニオン除去による実行速度

CryptDB での処理時間を増やす要因として，オニオン構造によるものが考えられる．クエリ実行に際して，オニオン構造が上の層（RND 等）まで暗号化されていると，検索条件の定数をオニオンの状態に合わせるための暗号化の回数や，結果の復号に必要な復号の回数が多くなる．オニオン構造を変化させることでこの暗号化，復号の回数を減らし，処理時間の高速化を図る．

9.1 実行後のオニオンの状況

まずはオニオン復号の状況を確かめるべく，テーブル作成時のオニオンと tpcc-cryptdb 実行後のオニオンの状況を調査した．

tpcc-cryptdb のデータ生成を MySQL 環境で実行した場合にはカラムが全部で 92 列作成される．CryptDB で実行した場合には全部で 234 列作成され，tpcc-cryptdb 実行後にはデータ格納後クエリを実行していない場合と比べ，12 件の状態が変化する．これは，tpcc-cryptdb 実行のために 12 列の復号が必要となったことを示す．この内訳としては DET（イコール演算）が必要になったものが 3 件，DETJOIN（等結合）が必要になったものが 2 件，OPE（大小比較演算）が必要になったものが 7 件である．実行後のオニオンの状態の内訳としては RND が 156 件，HOM が 50 件，DET が 20 件，OPE が 7 件，DETJOIN が 2 件である．

なお，CryptDB の実装として，カラム作成時の暗号化状態は RND，または HOM で作成されるが，主キーのカラムに関しては最初から DET で作成されるため，DET を利用するために復号した 3 件に主キーのイコール演算のための復号は含まれない．

9.2 オニオンの事前復号による実行速度

RND や DET 等，複数回暗号化して値を格納しているオニオンについて，暗号化されている最下層の状態（DETJOIN 等）と比較すると，格納する値や演算する定数の暗号化，クエリ結果の復号がより多く必要となり，実行速度は遅くなるはずである．ここではオニオンが複数あることによる実行速度への影響を調査する．

方法として，すべてのオニオンを最下層まで復号した状態でのクエリ実行にかかる時間を計測し，比較を行う．最下層のオニオンについて，Eq オニオンでは DETJOIN，Order オニオンでは OPE，HOM オニオンでは HOM とする．

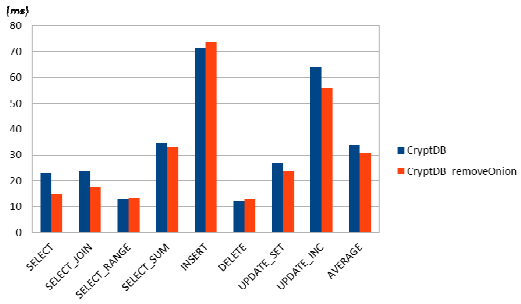

図 5.オニオン復号時の実行時間

INSERT と DELETE を除くクエリタイプで実行時間が減少し，全体平均では約 90%の時間で実行することができた．漏洩しても問題のないデータに関しては，必要な機密性に合わせてオニオンを事前に復号しておくことで処理の高速化が可能である．また，現在の CryptDB では実装されていないが一切暗号化を行わない生データによる格納を行うことで更なる高速化ができるものと思われる．

10. おわりに

TPC-C の実施を通し，CryptDB に未実装である機能の調査を行った．小数のカラム型が利用できないことや負の数の格納ができない等，利用時に気をつけなければいけない点が見つかった．また，tpcc-mysql を改造しての処理時間の分析を行い，CryptDB 内部の処理にかかる時間を測定した．高速化の手法案として，インデックス追加とオニオンの事前復号の 2 点について検討し，効果を測定した．

11. 参考文献

[1]H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra: “Executing SQL over Encrypted Data in the Database-ServiceProvider Model”, Proceeding of the ACM SIGMOD International Conference on Management of Data, pp. 216–227, June 2002.

[2]R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In Proc. of the 23rd SOSP, pages 85–100, Cascais, Portugal, Oct. 2011.

[3]Stephen Tu, M. Frans Kaashoek, Samuel Madden, NickolaiZeldovich Processing analytical queries over encrypted data In Proceedings of the 2013 VLDB, Pages: 289-300

[4]R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data,Paris, France, June 2004

[5]P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of the 18th Annual InternationalConference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Prague, Czech Republic,May 1999.