Vol.2012-DBS-156 No.5
2012/12/12

暗号化ストリームデータ処理における効率化の検討

冨山 克裕[1,a)]　川島 英之[2,b)]　北川 博之[2,c)]

概要：近年，継続的にストリームデータを生成し続けるストリーム情報源の数が増大しつつある．ストリームデータを処理する基盤システムとして，ストリーム処理エンジン（SPE）が開発されている．数千〜数万のストリーム情報源に対して処理を行うためには，SPE には非常に高い演算処理能力が要求される．このような処理の実行にはパブリッククラウドなどの分散並列処理基盤を用いることが有効であると考えられる．しかし，パブリッククラウドは一般に組織のファイヤウォールの外側で第三者により管理されるため，パブリッククラウド上の情報に対して機密性を保持することができない．これに対し我々は，安全性を考慮したストリームデータ処理の実現を目的として暗号化ストリームデータ処理方式の研究を行っている．本稿では，暗号化ストリームデータ処理においてクエリ実行中に効率よく暗号化鍵を更新するための手法を提案する．

1. はじめに

近年のセンシングデバイスの発達に伴い，継続的にデータを生成し続けるストリーム情報源の数が増大しつつある．具体的なストリーム情報源の例として，監視カメラ，ルータを流れるネットワークパケット，株価・為替変動などの金融情報，GPS 端末をはじめとする各種センシングデバイスなどが挙げられる．これらの情報源から得られるデータをストリームデータと呼び，ストリームデータを処理する基盤システムとして，ストリーム処理エンジン（SPE: Stream Processing Engine）がこれまで開発されてきた [1], [4].

ストリーム処理エンジンとは，ストリーム情報源から無限に到着するストリームデータに対して連続的問合せを行うことを可能にするシステムである．連続的問合せとして，選択・射影・結合・集約などの関係演算や複合イベント処理を行うことができる．この問合せは SQL ライクの言語を用いて記述することができ，データが到着する度に評価が行われる．また，メモリ上で即時的に実行されるため，データ永続化を必要とする従来のデータベース管理システム（DBMS）に比べ，処理を高速化できる．従来のデータベース管理システムとストリーム処理との違いを，**図 1** に示す．

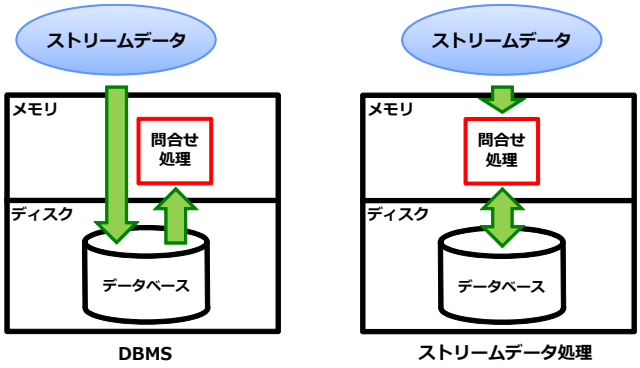

図 1　DBMS とストリームデータ処理の違い
Fig. 1　Difference between DBMS and Stream Data Processing

一方，ストリームデータなどのビックデータに対する処理環境を提供するための技術として，近年パブリッククラウドが注目を浴びつつある．パブリッククラウドとは，インターネットを介して不特定多数の個人または企業に対して計算資源を提供するサービスを指す．ユーザーは，計算資源を自前で持つ場合に比べて運用・維持コストの削減や，資源の柔軟な増減が可能になる．具体的なパブリッククラウドの例として，米 Amazon Web Services による Amazon EC2，米 Microsoft による Windows Azure などが挙げられる．パブリッククラウドはその優れたコストと利便性により，現在急成長を遂げている．しかしその一方，ユーザはパブリッククラウドに対してセキュリティに関する懸念を

[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,
1-1-1, Tennodai, Tsukuba, Ibaraki 305–8577 Japan
[2] 筑波大学 システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba,
1-1-1, Tennodai, Tsukuba, Ibaraki 305–8577 Japan
a) tomi@kde.cs.tsukuba.ac.jp
b) kawasima@cs.tsukuba.ac.jp
c) kitagawa@cs.tsukuba.ac.jp

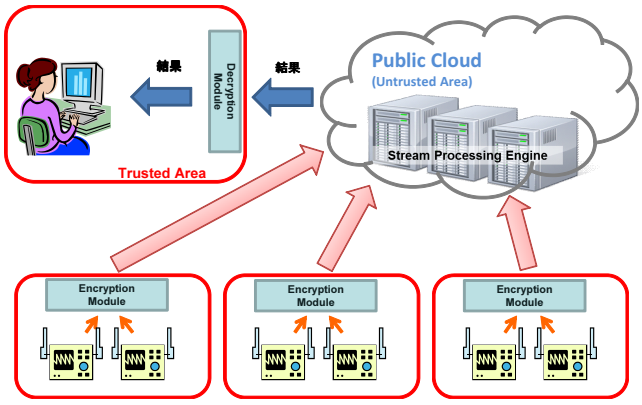

図 2 提案するアーキテクチャの俯瞰図
Fig. 2 The Bird's-eye View of the Architecture

抱いている [3]．パブリッククラウドは一般に，ユーザである企業などのファイヤウォールの外側で第三者により管理される．そのため，パブリッククラウド上に保存された情報を，パブリッククラウド管理者を含む第三者よって覗き見られたり改ざんされる可能性が否定できない．

この問題に対する解決するための手段として，近年，DBMS 上で暗号化されたデータに対して関係演算を実現するための研究が行われている．CryptDB[8] や POP[6] などがその一例である．

我々は安全性を考慮したストリームデータ処理の実現を目的として暗号化ストリームデータ処理方式について研究を行っている．本稿では，これまでの研究で行ってきた，暗号化データストリームを扱う上で課題となるメモリの消費量増大に対処するための 2 つの効率化手法について手短に紹介を行う．その後で，新たに，実行中のクエリを停めることなく暗号化鍵を更新するための手法について提案を行う．我々が研究を行っている暗号化ストリームデータ処理方式のアーキテクチャの俯瞰図を**図 2** に示す．

ストリームデータは Trusted area[*1] で暗号化され，復号されることなく Untrusted area に存在するパブリッククラウドで処理される．パブリッククラウド上で暗号化ストリームデータは一切復号されないため，悪意ある攻撃から守られる．クエリ処理の結果はユーザが存在する Trusted area へ送られ復号される．以下に，本稿における我々の寄与を簡潔に示す．

効率的な暗号化鍵更新手順の提案

暗号化ストリームデータ処理において長期間に渡って同一の暗号化鍵が用いられていると攻撃者に暗号化鍵を破られるリスクが高まるため，一定時間毎に暗号化鍵の更新が行われることが望ましいと考えられる．しかし，クエリ実行中に暗号化鍵を更新すると，処理木の中にそれぞれ異なる 2 つの鍵で暗号化された暗号化タプルが混在することとなり，正しい演算を行うことができなくなる．そこで我々は暗号鍵の移行期間という概念を導入し，クエリの実行を停めることなく段階的に暗号化鍵を更新してゆくための手法について提案を行う．また，暗号化鍵の更新に伴う暗号化モジュール（Encryption module）の計算コストや SPE のメモリ使用量を最小にするような移行期間の長さを見積もるための手法についても提案を行う．

[*1] 本稿では，Trusted area はあらゆる攻撃から保護された安全な領域のことを指し，Untrusted area は攻撃され得る領域のことを指す．

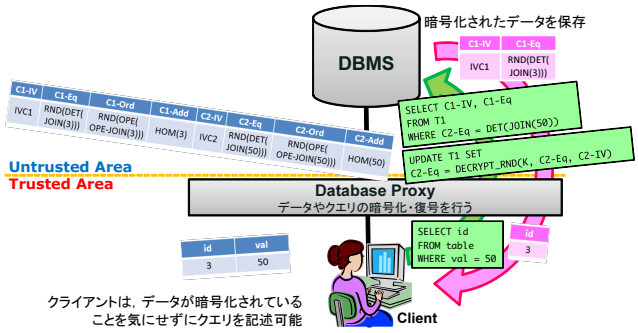

図 3 CryptDB のシステムアーキテクチャ
Fig. 3 System Architecture of CryptDB

本稿の以降の構成は以下の通りである．第 2 章では関連研究として，従来の DBMS 上で暗号化されたデータに対して関係演算を実現する CryptDB の紹介を行う．次に第 3 章で我々が研究を行っている暗号化ストリームデータ処理方式について紹介し，その課題について述べる．第 4 章では，これまでに研究として行ってきた，これらの課題に対する効率化手法について手短に述べる．第 5 章では，新たに，暗号化ストリームデータ処理における効率的な暗号化鍵更新手法について提案を行う．最後にまとめと今後の課題を第 6 章に述べる．

2. 関連研究

2.1 CryptDB

2.1.1 概要

CryptDB は，R. A. Popa ら [8] によって提案された手法である．CryptDB におけるシステムのアーキテクチャを**図 3** に示す．DBMS サーバとクライアントの間に Database Proxy と呼ばれるモジュールを置き，DBMS に保存するデータの暗号化やクエリの書換え，またクエリ結果の復号などを行う．一般に，暗号化された値に対して全ての関係演算を可能とする暗号化アルゴリズムは存在しない．CryptDB では，暗号化された値に対して選択・射影・結合・集約のいずれかの演算を行うことのできる暗号化アルゴリズムを複合的に用い，一つの平文値に対し複数の暗号値を作成する．与えられたクエリに応じてこれらの暗号値を使い分けることで，暗号化された値に対する関係演算を実現する．CryptDB が必要とする暗号化アルゴリズムの特徴を次節に示す．

2.1.2 用いる暗号の特徴

Deterministic（DET）： DET は暗号化された 2 つの値に対し，等価性を確認することができる特徴を持つ．これにより，条件式に等式を持つ選択演算や等結合，GROUP BY や COUNT，DISTINCT などの命令を暗号値に対して実行することができる．DET に用いる具体的な暗号化アルゴリズムとしては AES-CMC[5] が挙げられる．DET が満たす式を以下に示す．

$$x = y \iff DET_K(x) = DET_K(y)$$

Order-Preserving Encryption（OPE）： OPE は暗号化された 2 つの値に対し，不等性を確認することができる特徴を持つ．OPE を用いることにより，レンジクエリや ORDER BY，MIN，MAX，SORT などの命令を暗号値に対して実行することができる．OPE に用いる具体的な暗号化アルゴリズムとしては Boldyreva ら [2] により提案されたアルゴリズムが挙げられる．値が暗号化された状態でも 2 つの値の大小関係を露わにしてしまうため，安全性は DET よりも劣る．OPE が満たす式を以下に示す．

$$x < y \iff OPE_K(x) < OPE_K(y)$$

Homomorphic Encryption（HOM）： HOM は暗号化された 2 つの値の和を求めることができる特徴を持つ．加法準同型暗号である Paillier 暗号 [7] を用い，以下の式を満たす．

$$HOM_K(x) \cdot HOM_K(y) = HOM_K(x + y)$$

3. 暗号化ストリームデータ処理方式

我々は前章で示した CryptDB の考え方を参考にして，暗号化ストリームデータ処理方式について研究を行っている [9]．我々の手法では，Encryption module をストリーム情報源の存在する各 Trusted area に設置し，Decryption module を各ユーザの存在する Trusted area に設置する．暗号化ストリームデータ処理方式のアーキテクチャを**図 4** に示す．

このアーキテクチャの下では暗号化されたデータのみが Untrusted area にあるパブリッククラウドに送られるため，Untrusted area における，のぞき見（Snooping）などの攻撃を防ぐことができる．また，SPE において暗号化されたデータに対して実行することのできない演算は，Decryption module で復号された後に実行される．これを Post-processing と呼ぶ．

Encryption module における暗号化のためのアルゴリズムを Algorithm 1 に，Decryption module における復号のためのアルゴリズムを Algorithm 2 にそれぞれ示す．

3.1 CryptDB との相違点

我々の手法は CryptDB とは決定的に異なる．CryptDB

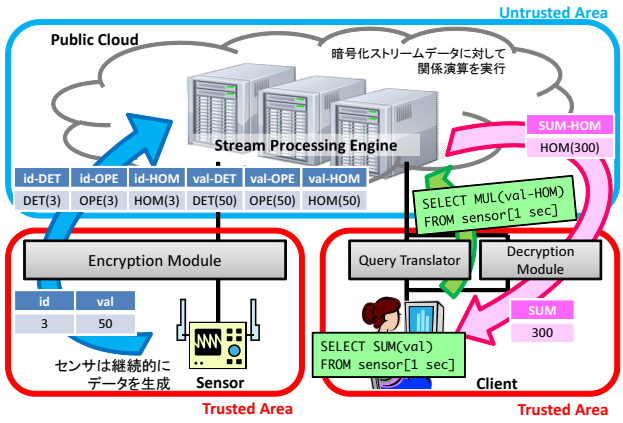

図 4 Encryption & Decryption Modules
Fig. 4 Encryption & Decryption Modules

Algorithm 1 Encryption Procedure

```
Receive the name of stream from a client.
Send ack to the client.
Search schema of the stream from metadata DB. (registered in advance)
for i = 0 to the number of attributes do
    Make DET cipher.
    if the type of attribute_i is integer then
        Make OPE & HOM ciphers.
    end if
    Insert cipher(s) to a buffer.
end for
Send ciphers in the buffer to SPE.
Receive ack from SPE.
```

Algorithm 2 Decryption Procedure

```
Receive the query name from SPE.
Send ack to SPE.
Search schema of the result tuple from metadata DB. (registered in advance)
Receive result tuples from SPE.
for i = 0 to the number of attributes do
    if encryption type of attribute_i is DET. then
        Decrypt a cipher using DET algorithm.
    else if encryption type of attribute_i is OPE. then
        Decrypt a cipher using OPE algorithm.
    else if encryption type of attribute_i is HOM. then
        Decrypt a cipher using HOM algorithm.
    end if
    Insert a plain value to a buffer.
end for
Send data in the buffer to client.
Receive ack from client.
```

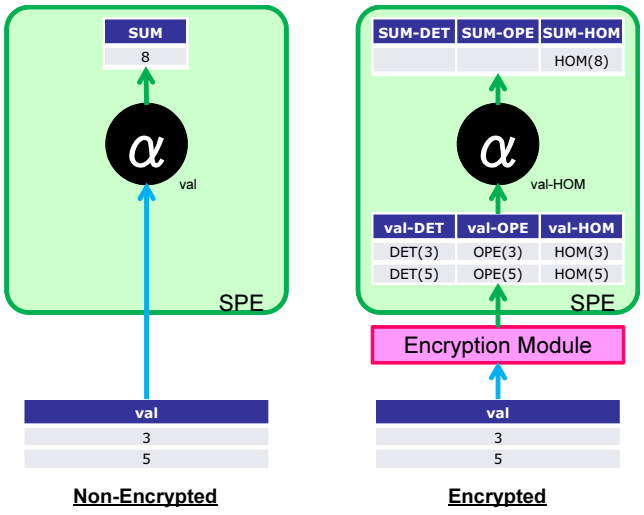

図 5 集約演算子の出力
Fig. 5 Outputs of Aggregation Operator

は Database proxy と呼ばれる単一のモジュールによって暗号化及び復号を実現している．しかし一般的に情報源とユーザとが広範囲に分散しているストリームデータ処理において，このようなスキームが適用できるケースは極めて稀であると言える．そのため，我々の手法では暗号化及び復号を行うためのそれぞれ独立したモジュールを用意する．

3.2 共通鍵の管理

DET として用いる AES-CMC[5]，及び OPE として用いる Boldyreva ら [2] により提案された暗号化アルゴリズムは共通鍵暗号方式である．そのため我々の手法においては，事前に各 Encryption module 及び Decryption module の間で共通鍵を共有しておく必要がある．本稿では，各モジュールは安全な経路を用いて事前に共通鍵を共有しているものとし，鍵を更新するための手順については第 5 章で提案を行う．

3.3 Re-encryption

選択，射影及び結合演算において，各演算子の出力値はそれぞれの入力値から構成される．一方，入力値の合計値などを求める集約演算においては，出力値は入力値と異なる場合がある．**図 5**（左）において，集約演算子が 2 つのタプルの要素 *val* の合計値を計算すると仮定する．この例では，入力タプルの要素 *val* の値は 3 と 5 であり，出力値は 8 である．同様の演算を暗号化ストリームデータ処理で実現する場合を図 5（右）に示す．

集約演算子は要素 *val* の HOM 暗号を用いる．この時，演算子は 8 を表す HOM 暗号を出力する．しかし，SPE は 8 を表す DET 及び OPE 暗号を出力することができない．なぜなら，入力された値に 8 を表す DET 及び OPE 暗号が存在していないためである．そのため本手法において，SPE は**図 6** に示すようなクエリを実行することができない．

```
SELECT a.id
FROM (SELECT id, SUM(val) as s
      FROM sensor[1 min]) as a
WHERE a.s > 500
```

図 6 我々の手法の上で実行できないクエリの例
Fig. 6 An Example of Queries that are not able to Run on the Proposal Technique

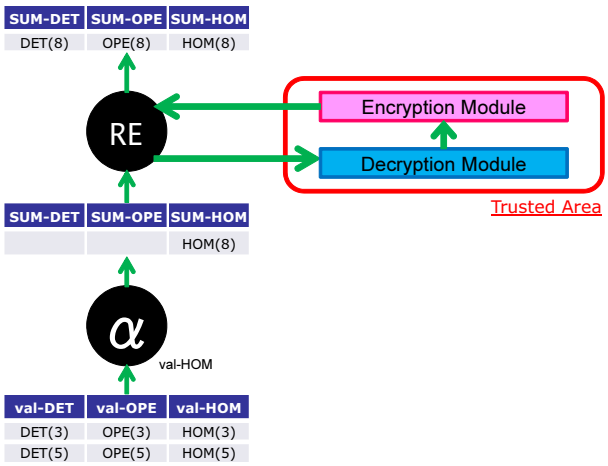

図 7 Re-encryption 演算子
Fig. 7 Re-encryption Operator

図 6 に示すクエリは，集約演算により得られた要素 *val* の合計値に対して選択演算により不等性の評価を行う．よって，集約演算の出力値の OPE 暗号が必要となる．しかし，集約演算は DET 及び OPE 暗号を出力できない（HOM 暗号のみを出力する）ため，SPE は選択演算による不当性の評価を行うことができない．

そこで我々はこれまでの研究の中で Re-encryption 演算子を定義した．Re-encryption 演算子は，HOM 暗号を Trusted area に存在する Decryption module に送信して復号し，平文となった値を Encryption module へ送ることで DET 及び OPE 暗号を生成して出力する．Re-encryption 演算子の概要を**図 7** に示す．

3.4 本方式における課題

前述したとおり，Encryption module は入力されたタプルの各要素に対して 3 種類（DET, OPE, HOM）の暗号値を生成する．そのため，暗号化されたタプルのデータサイズの合計が平文タプルのデータサイズに対して著しく増加することが考えられる．以上より，我々の手法では，以下に示す 2 つの問題点が生じる．

課題 1: タプルサイズの増加: 我々の提案手法では 1 つの平文値に対して 3 つの暗号値が生成される．また，一般に暗号値は平文値よりも大きい．それゆえに，SPE に転送される暗号化タプルサイズは平文タプルに対して 3 倍以上の大きさになると考えられ，通信帯域を圧迫してしまう．

表 1　各演算子が必要とする暗号の種類
Table 1　Required Ciphers for Operators

Operator Type		DET	OPE	HOM
σ	Equality-selection	×		
	θ-selection		×	
π	Projection			
$\bowtie$	Equality-join	×		
	θ-join		×	
α	Summation			×
	Count			×
	Minimum		×	
	Maximum		×	
γ	Group by	×		

課題 2: SPE のメモリ消費量の増加:　課題 1 と同様の理由により，SPE 上の処理木の各シノプシスに暗号化タプルが貯まることで，平文に比べて SPE のメモリ使用量が増加してしまう．

これらの問題に対処するために，我々はこれまでに効率化手法について研究を行ってきた．概要を次章に述べる．

4. 効率化手法

前章で示した課題に対して，我々はこれまでに 2 つの効率化手法を提案してきた [9]．本稿ではページの都合上，これらの手法及び評価実験について手短に紹介する．

4.1 効率化手法 I：データ転送量の削減

ストリームデータ処理ではクエリは事前に SPE に登録されている．よって，クエリを実行前に解析することで，クエリを実行するために必要な暗号の種類を判断することができる．各演算子と，その演算子が必要とする暗号の種類を**表 1** に示す．

効率化手法 I では，クエリを実行前に解析することでクエリ処理に必要な暗号の種類を判断し，それらを Encryption module に告知する．これにより，Encryption module は必要最小限の暗号のみを生成し，暗号化に要するコストの削減と，SPE へ転送されるデータ量の削減を図る．評価実験の結果，本手法の適用により SPE に転送されるデータ量を 90%削減できることを確認できた．

4.2 効率化手法 II：SPE のメモリ使用量の削減

前述した効率化手法 I は，Encryption Module 内部，及び Encryption Module から SPE までの経路において暗号化コストとデータ転送量を削減する手法であった．一方，効率化手法 II では SPE 内部でのメモリ使用量を削減する．SPE に複数のクエリが登録されている場合には，効率化手法 I に加え，処理木中の各演算子に入力される暗号の種類を必要最小限に減らすことで，SPE 全体におけるメモリ使用量の削減が可能となると考えられる．特にウインドウ結合（Window-Join）や集約演算など，シノプシスにタプルを保存する必要のある演算子（ステートフルオペレータ）においては，その後の処理に必要のない暗号を含んだタプルを保持することはメモリ使用量の浪費に繋がる．そこで本手法では，これらの演算子の前に射影演算子を挿入し，その後の処理に必要のない暗号を適宜取り除いてゆくことで SPE のメモリ使用量の削減を図る．評価実験の結果，効率化手法 I ではデータ量を削減できないようなクエリに対して本手法を適用することにより，SPE のメモリ使用量を 11%削減できることを確認できた．

5. 暗号化ストリームデータ処理システムにおける暗号化鍵更新手順の提案

我々がこれまでに研究を行ってきたストリームデータ処理方式は，クエリ実行前に Encryption module，Decryption module 及び Query translator が暗号化鍵を共有することを前提とし，その後の暗号化鍵の更新については論じてこなかった．しなしながら，長期にわたり同一の暗号化鍵を使い続けることは暗号化鍵漏えいのリスクが高まるだけでなく，攻撃者が暗号化鍵を発見しデータの解読を可能にするまでの試行時間を延ばすことにも繋がる [10]．そこで本章では，新たに我々の提案する暗号化ストリームデータ処理システムにおける暗号化鍵更新手順について提案を行う．一般に，ストリームデータ処理は逐次的に発信されるデータに対して継続的にクエリ処理を行うことができる点が利点であり，暗号化鍵の更新を行うためにクエリ実行を停止するのは望ましくない．しかし一方で，クエリ実行中に暗号化鍵を更新し Encryption module が新しい暗号化鍵だけを用いて暗号化タプルを生成すると，SPE 上で実行中の処理木中にそれぞれ異なる暗号化鍵で暗号化されたタプルが混在することとなり，正しいクエリ処理を行うことができなくなる．

そこで我々は，鍵の移行期間という概念を導入することによりクエリ実行を停止せずに暗号化鍵を更新するための手法について提案する．また，暗号化鍵の更新が完了するまでの時間，及び暗号化鍵の更新に要する Encryption module の計算コスト及び SPE のメモリ使用量を抑えるための効率化手法について述べる．

5.1 暗号化鍵の移行期間

暗号化鍵の移行期間とは，実行中の暗号化ストリームデータ処理システムにおいて，クエリ処理に用いる鍵を古い鍵（K_1）から新しい鍵（K_2）に更新するために要する期間を指す．前述したように，クエリ実行中のある時刻に暗号化鍵を更新する（すなわち，移行期間の長さ=0）と，SPE 上で実行中の処理木中にそれぞれ異なる 2 つの鍵で暗号化された暗号化タプルが混在し，正しいクエリ処理を行うことができなくなる．そこで移行期間を設け，移行期

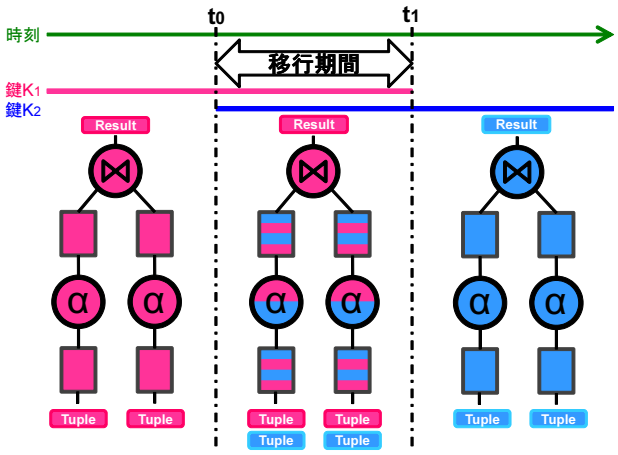

図 8　暗号化鍵更新手順
Fig. 8　Steps of Changing Encryption-keys

間中に Encryption module に入力された平文タプルは古い鍵（K_1）と新しい鍵（K_2）の両方で暗号化を行い，2 つの暗号化タプル（ペアと呼ぶ）を SPE へ送信する．移行期間が一定時間継続することで，実行中の処理木の中に存在する全てのタプルが移行期間中に生成されたペアで満たされる．すなわち，古い鍵（K_1）で作られた暗号化タプルのみを用いてクエリ処理を行っても，新しい鍵（K_2）で作られた暗号化タプルのみを用いてクエリ処理を行っても，復号を行った時に同一の結果を得ることができるようになる．このように処理木中に存在する全てのタプルが移行期間中に生成されたペアで満たされた状態になった後に移行期間を終了し，Encryption module は入力タプルを新しい鍵（K_2）のみで暗号化し SPE へ送信する．同時に，SPE は新しい鍵（K_2）で作られた暗号化タプルのみを用いてクエリ処理を行う．

このように移行期間を設けて一時的に生成する暗号化タプルを 2 重化することにより，クエリ実行を停止せずに暗号化鍵の更新を実現することができる．移行期間を用いて段階的に暗号化鍵を更新してゆく様子を**図 8** に示す．

図 8 において，移行期間の開始時刻を t_0，終了時刻を t_1 と表す．移行期間を設定する際，時刻 t_1 には処理木の中に含まれるすべてのタプルが時刻 t_0 以降に生成された暗号化タプルのペアで満たされていることを保証しなければならない．なぜなら，時刻 t_1 以降，SPE は新しい鍵（K_2）で作られた暗号化タプルのみを用いてクエリ処理を行い，古い鍵（K_1）で作られた暗号化タプルは無視されるため，正しいクエリ結果を得るためには少なくとも新しい鍵（K_2）で作られた暗号化タプルが必要となるからである．次節で，この条件を満たすような最小の移行期間の長さを見積もるための手法について述べる．

5.2　移行期間の長さの見積もり

図 8 において暗号化鍵の移行期間の長さは $t_1 - t_0$ と表すことができる．この間，Encryption module では古い鍵（K_1）と新しい鍵（K_2）の 2 種類の暗号化鍵を用いて暗号化タプルを生成するため，Encryption module が暗号化に要するコストは通常の 2 倍となると考えられる．また，ストリーム情報源からのデータ発信レートが一定であると仮定した時，SPE 上で実行中の処理木中の各シノプシスには最大で通常の 2 倍の暗号化タプルが保管されるため，SPE のメモリ使用量は最大で通常の 2 倍となると考えられる．ゆえに，移行期間の長さ（$t_1 - t_0$）を最小化することができれば，暗号化鍵の更新に要する Encryption module の計算コスト及び SPE のメモリ使用量を抑えることができると考えられる．

移行期間の長さを見積もるためには，対象となる処理木の事前解析が必要となる．処理木中の全てのステートフルオペレータのシノプシスサイズが時間幅によって指定されているとき，移行期間の長さ（$t_1 - t_0$）の最小値を見積もるためのアルゴリズムを Algorithm 3 に示す．

Algorithm 3 Estimating Duration of Renewing Encryption-key

```
1: Receive a continuous query from a client;
2: Translate the query to a plan tree;
3: Merge it to other plan trees when possible;
4: Create all of paths which show routes from root to leaves
   in the plan tree;
5: Duration ← 0;
6: for all path_i ∈ paths do
7:    TotalSize ← 0;
8:    for p ← root node of path_i; p ≠ leaf node; p ← p's child
      node do
9:       if p is a stateful operator then
10:         TotalSize ← TotalSize + Size of synopsis of the op-
            erator;
11:      end if
12:   end for
13:   if TotalSize > Duration then
14:      Duration ← TotalSize;
15:   end if
16: end for
```

Algorithm 3 では，対象となる処理木の全てのパスに対してシノプシスサイズの合計を計算し，その最大値を求めている．このアルゴリズムによって得られる値は，処理木に入力された暗号化タプルが処理木から出るまでに掛かる最大の時間を示したものであり，すなわち前節で述べた条件を満たすことのできる最小の時間となる．

しかしながら，上述した見積もりは以下の 3 点を考慮していない．

- 通信や処理に伴うタプル到着の遅延
- 各 Encryption module 及び SPE の時計の誤差

- シノプシスサイズがタプル数で指定された場合

通信経路の輻輳や各モジュールの持つ時計の誤差により時刻 t_0 以前に生成された暗号化タプルが移行期間後に SPE に到着した場合，SPE はこれらの暗号化タプルを無視するため正しいクエリ結果を得ることができない．また，シノプシスサイズが時間幅でなくタプル数で指定されていた場合は，タプルがシノプシス中に存在する時間を見積もることができないため適切な移行期間の見積もりを行うことができない．これらの点に考慮するためには，Algorithm 3 によって事前に見積もられた移行期間が経過した時に，SPE が実行中の処理木中の各シノプシスに含まれている全てのタプルを確認し，時刻 t_0 以前に生成された暗号化タプルが残っていた場合は移行期間の終了を一定期間延期するような機構が必要となる．

Algorithm 4 De-duplication Operator

```
OID ← φ; {OID denotes a set of TupleID that have been outputted;}
while the query is running do
    Receive an encrypted tuple from previous operator;
    if TupleID ∈ OID then
        OID ← OID − TupleID;
        Discarding the tuple;
    else
        OID ← OID ∩ TupleID;
        Outputting the tuple;
    end if
end while
```

5.3 提案手法の導入

本稿で我々が提案する暗号化鍵更新手順を導入するためには，Encryption module において暗号化タプルを生成する際にいくつかの付加情報を追加する必要がある．また，SPE において一部の既存演算子の挙動を改造する必要がある．

5.3.1 暗号化タプルへの付加情報の追加

Encryption module において暗号化タプルを生成する際，以下に示す 3 つの要素を付加する必要があると考えられる．

EncTime

Encryption module において暗号化タプルが生成された時刻を示す．暗号化タプルが，移行期間中またはその前後のどの期間に作られたかを識別するために必要となる．

TupleID

Encryption module に入力された平文タプル毎にユニークな値を生成する．移行期間中は 1 つの平文タプルから 2 つの暗号化タプル（ペア）が生成されるため，平文タプルが同一であったことを判断するために TupleID が必要となる．

KeyID

タプルの暗号化に用いられた暗号化鍵を識別するための値．SPE でのクエリ処理及び Decryption module で復号を行う際に必要となる．

5.3.2 既存演算子の改造

選択演算子

選択演算子では一般に，条件文の右辺に具体的な値が記述されている．暗号化ストリームデータ処理システムにおいては，右辺に記述されている値は暗号値であるため，鍵の更新に伴ってこの値も動的に変更できるようにする必要がある．

結合演算子

結合演算子は一般に，条件式の左辺にも右辺にも具体的な値を含まないため，改造の必要はないと考えられる．

集約演算子

集約演算子は，シノプシスに 2 種類の鍵で作られた暗号化タプルが混在していた場合に正しい演算を行うことができない．そこで，シノプシスに含まれる暗号化タプルを KeyID の値毎に分けて演算を行えるようにする必要がある．また，演算に用いる暗号化タプルは以下の 3 つの場合によって分ける必要がある．(1) シノプシスに時刻 t_0 以前に生成された暗号化タプルが含まれている場合は，古い暗号化鍵（K_1）を用いて暗号化されたタプルのみに対して演算を行う．(2) シノプシスに含まれる全てのタプルが時刻 t_0 以降に生成されている場合は，古い暗号化鍵（K_1）を用いて暗号化されたタプルと新しい暗号化鍵（K_2）を用いて暗号化されたタプルのそれぞれに対して演算を行う．よって，出力されるタプルも鍵 K_1 と鍵 K_2 のそれぞれで暗号化されたものとなる．(3) 移行期間後（時刻 t_1 以降）は新しい暗号化鍵（K_2）を用いて暗号化されたタプルのみに対して演算を行う．

5.3.3 重複回避演算子の導入

これまで述べたように，移行期間中に Encryption module に入力された平文タプルは古い鍵（K_1）と新しい鍵（K_2）の両方で暗号化を行い，2 つの暗号化タプルのペアが SPE へ送信される．そのため，クエリの処理結果として，ペアになっている 2 つの暗号化タプルが両方とも出力される可能性が考えられる．これら 2 つの暗号化タプルのペアがいずれも Decryption module へ送信されることは通信帯域の浪費に繋がるため，処理木の出力において適切に重複回避を行う必要がある．そこで我々の提案手法では重複回避演算子を定義することにより，処理木の出力で重複タプルの削除を行う．重複回避演算子のアルゴリズムを Algorithm 4 に示す．

表現A

EncTime	TupleID	KeyID	id_DET	val_HOM
1350871980	10	1	$DET_{K1}(1)$	$HOM_{K1}(300)$
1350871980	10	2	$DET_{K2}(1)$	$HOM_{K2}(300)$

表現B

EncTime	TupleID	KeyID1	KeyID2	id_DET1	val_HOM1	id_DET2	val_HOM2
1350871980	10	1	2	$DET_{K1}(1)$	$HOM_{K1}(300)$	$DET_{K2}(1)$	$HOM_{K2}(300)$

図 9 暗号化タプルの表現
Fig. 9 Two Types of Representation of an Encryption Tuple

5.4 移行期間中の暗号化タプル表現方法に関する議論

前節までに示した我々の提案手法では，移行期間中には平文タプル 1 つに対して暗号化タプル 2 つを生成し SPE へ送信した．本節では，これに代わる移行期間中の暗号化タプル表現方法として，古い鍵（K_1）で暗号化された値と新しい鍵（K_2）で暗号化された値を 1 つの暗号化タプルにまとめて表現することを考える．前節までの提案手法で示した表現方法（表現 A と呼ぶ）と本節で考える表現方法（表現 B と呼ぶ）の比較を**図 9** に示す．

表現 B では，Encryption module が SPE に送信するタプルの要素数は常に表現 A の約 2 倍となる．しかし移行期間以外は 1 つの鍵のみを用いて暗号化を行うため，約半分の要素は空の状態で SPE に送信されることとなる．また表現 A においてペアと呼んでいた 2 つのタプルは表現 B では常に 1 つのタプルで表現されるため，TupleID は不要となる他，重複回避演算子は既存の射影演算子で置き換えることができると考えられる．

我々は両表現方法の利点と欠点について，今後の研究の中で比較検討を行っていきたいと考えている．

6. まとめと今後の課題

我々は，安全性を考慮したストリームデータ処理の実現を目的として，暗号化ストリームデータ処理方式について研究を行っている．本稿では新たに，暗号化ストリームデータ処理方式の中でクエリの実行を停めることなく，効率的に暗号化鍵を更新するための手法について提案を行った．また，暗号化タプルの表現方法として表現 A と表現 B の 2 種類を提案し，それぞれの利点と欠点について議論を行った．

今後の課題として，暗号化鍵更新手法についてさらなる検討と評価実験を行いたい．また，GPU や FPGA を用いて暗号化及び復号を高速化する手法について検討を行いたい．

謝辞 本研究成果は，独立行政法人情報通信研究機構（NICT）の委託研究「新世代ネットワークを支えるネットワーク仮想化基盤技術の研究開発」，及び科学研究費基盤研究（C）（#24500106）の助成により得られたものです．

参考文献

[1] Arasu, A., Babu, S. and Widom, J.: The CQL continuous query language: semantic foundations and query execution, *VLDB J.*, Vol. 15, No. 2, pp. 121–142 (2006).
[2] Boldyreva, A., Chenette, N., Lee, Y. and O'Neill, A.: Order-Preserving Symmetric Encryption, *EUROCRYPT*, pp. 224–241 (2009).
[3] Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R. and Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control, *CCSW*, pp. 85–90 (2009).
[4] Gedik, B., Andrade, H., Wu, K.-L., Yu, P. S. and Doo, M.: SPADE: the system s declarative stream processing engine, *SIGMOD Conference*, pp. 1123–1134 (2008).
[5] Halevi, S. and Rogaway, P.: A Tweakable Enciphering Mode, *CRYPTO*, pp. 482–499 (2003).
[6] Hildenbrand, S., Kossmann, D., Sanamrad, T., Binnig, C., Faerber, F. and Woehler, J.: Query Processing on Encrypted Data in the Cloud (2011).
[7] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, *EUROCRYPT*, pp. 223–238 (1999).
[8] Popa, R. A., Redfield, C. M. S., Zeldovich, N. and Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing, *SOSP*, pp. 85–100 (2011).
[9] Tomiyama, K., Kawashima, H. and Kitagawa, H.: A security aware stream data processing scheme on the cloud and its efficient execution methods, *Proceedings of the fourth international workshop on Cloud data management*, CloudDB '12, New York, NY, USA, ACM, pp. 59–66 (online), DOI: 10.1145/2390021.2390033 (2012).
[10] 情報処理推進機構：「安全な暗号鍵のライフサイクルマネージメントに関する調査」調査報告書，情報処理推進機構（オンライン），`http://www.ipa.go.jp/security/fy19/reports/Key_Management/`（参照 2012-11-14）．