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

冨山　克裕†　　川島　英之††　　北川　博之††

† 筑波大学大学院システム情報工学研究科　〒 305–8577 茨城県つくば市天王台 1–1–1
†† 筑波大学システム情報系　〒 305–8577 茨城県つくば市天王台 1–1–1
E-mail: †tomi@kde.cs.tsukuba.ac.jp, ††{kawasima,kitagawa}@cs.tsukuba.ac.jp

あらまし　近年，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ア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を停止することなく効率的に暗号化鍵を更新するための手法を提案する．

キーワード　ストリームデータ処理，リレーショナル演算，暗号化，セキュリティ，パブリッククラウド

Katsuhiro TOMIYAMA†, Hideyuki KAWASHIMA††, and Hiroyuki KITAGAWA††

† Graduate School of Systems and Information Engineering, University of Tsukuba
1–1–1, Tennodai, Tsukuba, Ibaraki, 305–8577 Japan
†† Faculty of Engineering, Information and Systems, University of Tsukuba
1–1–1, Tennodai, Tsukuba, Ibaraki, 305–8577 Japan
E-mail: †tomi@kde.cs.tsukuba.ac.jp, ††{kawasima,kitagawa}@cs.tsukuba.ac.jp

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 [2] [5].

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．従来の DBMS では，まず，到着したストリームを二次記憶に格納してから，必要に応じて主記憶上にロードして問合せ処理を行う．一方ストリームデータ処理では，到着したストリームは主記憶上で即時的に問合せ処理が行われ，必要に応じて二次記憶に格納をする．

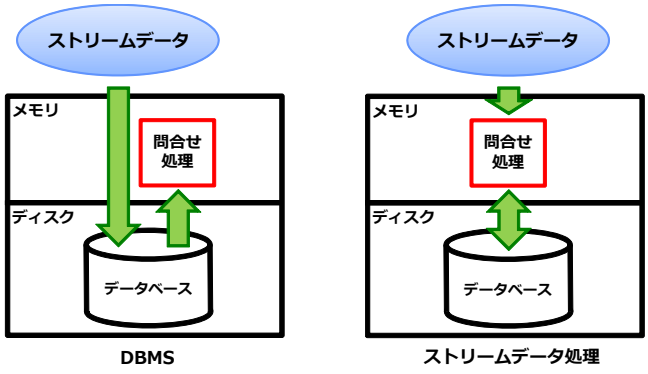

図 1　DBMS とストリームデータ処理の違い

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

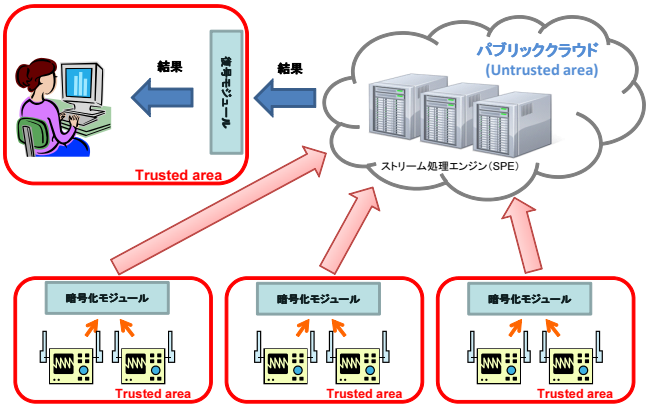

図 2　提案するアーキテクチャの俯瞰図

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，Amazon による Amazon EC2，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抱いている [4]．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ア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 [9] や POP [7] 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データ量増大に対処するための二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(注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データは一切復号されないため，悪意ある攻撃からデータを守ることが可能になると考えられる．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提案を簡潔に示す．

(注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．

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システムにおいて長時間にわたり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 [1]．よって，一定時間ごとに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，クエリ実行中に鍵を更新すると，処理木中にそれぞれ異なる二つの鍵で暗号化された暗号値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．そこで本稿では “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鍵を更新し，その上で，鍵の更新が完了するまでの時間，及び鍵の更新に伴う計算資源のオーバーヘッド（SPE のメモリ使用量，及び各モジュールの計算コストのオーバーヘッド）を準最適化するための手法について提案を行う．

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 [9] 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データベースプロキシ（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すべ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 は暗号化された二つの値（数値や文字列など）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結合演算，グルーピングや COUNT 句，DISTINCT 句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 暗号（CMC モード [6]）が挙げられる．二つの暗号値が等しければ対応する平文値が等しいことがわかるために，安全性は RND よりも若干劣ると言える．DET が満たす式を以下に示す．

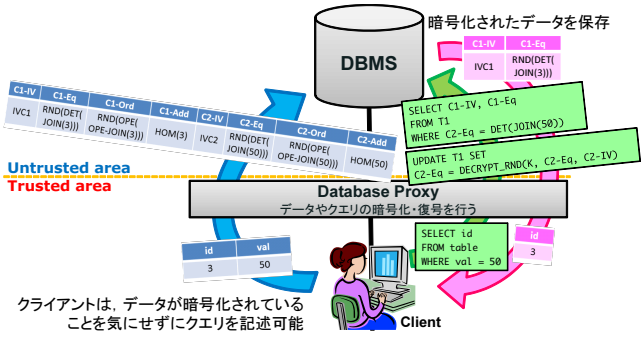

図 3　CryptDB のシステムアーキテクチャ

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

Order-Preserving Encryption（OPE）： OPE は暗号化された 2 つの値（数値）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 句，MIN 句，MAX 句，SORT 句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 [3] 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二つの値の大小関係を露わにしてしまうため，安全性は DET よりも劣ると言える．OPE が満たす式を以下に示す．

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

Homomorphic Encryption（HOM）： HOM は暗号化された二つの値（数値）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 [8] 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 [10]．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．本方式では，ストリーム処理エンジン（SPE）はパブリッククラウドなどの untrusted area に設置することを想定する．また，ストリームデータの暗号化を行うための “暗号化モジュール（encryption module）” はストリーム情報源の存在する各 trusted area に，クエリ結果の復号を行うための “復号モジュール（decryption module）” は各ユーザの存在する trusted area にそれぞれ設置する．

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 SPE へと送られるため，untrusted area におけるのぞき見（snooping）などの攻撃からデータを守ることが可能になる．また，SPE 上でデータが暗号化された状態では実行することのできない処理は，復号モジュールで復号された後に実行する．これを post-processing と呼ぶ．

暗号化モジュールにおける暗号化のためのアルゴリズムを Algorithm 1 に，復号モジュール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．

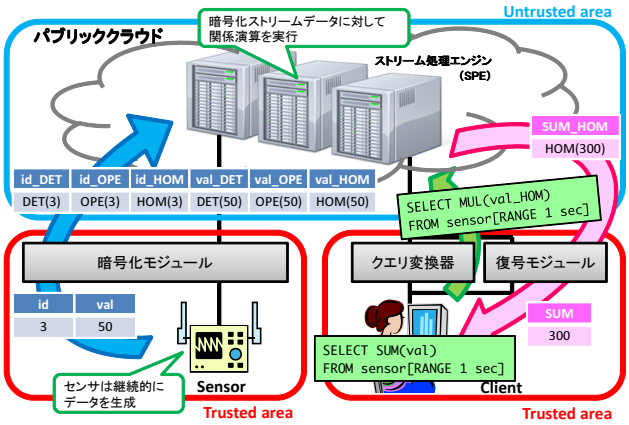

図 4　暗号化ストリームデータ処理方式のアーキテクチャ

Algorithm 1 Encryption Procedure

1: Receive the name of stream from a client.
2: Send ack to the client.
3: Search schema of the stream from metadata DB. (registered in advance)
4: **for** $i = 0$ to the number of attributes **do**
5: Make DET cipher.
6: **if** the type of attribute$_i$ is integer **then**
7: Make OPE & HOM ciphers.
8: **end if**
9: Insert cipher(s) to a buffer.
10: **end for**
11: Send ciphers in the buffer to SPE.
12: Receive ack from SPE.

Algorithm 2 Decryption Procedure

1: Receive the query name from SPE.
2: Send ack to SPE.
3: Search schema of the result tuple from metadata DB. (registered in advance)
4: Receive result tuples from SPE.
5: **for** $i = 0$ to the number of attributes **do**
6: **if** encryption type of attribute$_i$ is DET. **then**
7: Decrypt a cipher using DET algorithm.
8: **else if** encryption type of attribute$_i$ is OPE. **then**
9: Decrypt a cipher using OPE algorithm.
10: **else if** encryption type of attribute$_i$ is HOM. **then**
11: Decrypt a cipher using HOM algorithm.
12: **end if**
13: Insert a plain value to a buffer.
14: **end for**
15: Send data in the buffer to client.
16: Receive ack from client.

3.1 CryptDB との相違点

我々の手法はストリームデータ処理を対象とする点で CryptDB とは決定的に異なる．CryptDB はデータベースプロキシ（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 暗号化鍵の管理

DET として用いる AES 暗号，及び OPE として用いる Boldyreva ら [3] により提案された暗号化アルゴリズムは共通鍵暗号である．また，HOM として用いる Paillier 暗号は公開鍵暗号である．そのため本稿で提案する暗号化ストリームデータ処理方式においては，事前に各暗号化モジュールや復号モジュール，及びクエリ変換器の間で，暗号化及び復号に必要な鍵を共有しておく必要がある．具体的には，DET 及び OPE については各モジュール間で共通鍵を共有し，HOM については各暗号化モジュール間で秘密鍵を，各復号モジュール間で公開鍵をそれぞれ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これらの鍵を共有しているものと仮定し，効率的に鍵を更新するための手順については第 5 章に述べる．

3.3 本方式における課題

前述したとおり，暗号化モジュールは入力されたタプルの各属性に対して 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量の合計は平文タプルのデータ量に対して数倍に増加することが考えられる．以上より，我々の手法では，以下に示す二つの問題点が生じる．

課題 1: タプルサイズの増大: 本方式では，平文タプルの各属性に対してそれぞれ 3 種類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．ゆえに，SPE に転送されるデータ量の合計は平文に対して 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．

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，平文で同様のクエリ処理を行う場合に比べて SPE のメモリ使用量が増加してしまうと考えられる．

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. データ量削減のための効率化手法

前章で示した課題に対して，我々はこれまでに二つの効率化手法を提案してきた [10]．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従来の DBMS と異なり，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，それらを暗号化モジュールに告知する．これにより，暗号化モジュール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暗号化ストリームデータ処理システム全体のスループットを約 5.7〜5.9 倍に改善できることを確認できた(注2)．

表 1 各演算子が必要とする暗号の種類

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	×		

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，暗号化モジュール内部，及び暗号化モジュールから 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 は SPE 内部でのメモリ使用量を削減する．

SPE 上で結合演算や集約演算のようなステートフル演算子は，シノプシスの中に一時的にタプルを保持する．これらのタプルの中には以降の演算子に要求されない暗号値も含まれている可能性があり，このような暗号値をシノプシスに保持しておくことは，明らかに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することにより，その後の処理に必要のない暗号値を適宜取り除いていくことで 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 のメモリ使用量を約 98.5%削減できることを確認できた(注2)．

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

我々がこれまでに提案を行ってきた暗号化ストリームデータ処理方式は，暗号化モジュール，復号モジュール，及びクエリ変換器がクエリ実行前に暗号化及び復号に必要な鍵を共有することを前提とし，その後の鍵の更新については論じてこなかった．しかしながら，広範囲に分散しているストリーム情報源を対象とする我々の想定の下では，クエリ実行中に処理対象のストリーム情報源が追加・削除されることも考えられるため，長期にわたり同一の鍵を使い続けることは鍵の漏えいのリスクに繋がると考えられる．またそれだけでなく，長期にわたり同一の鍵を使い続けることは，攻撃者が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 [1]．以上の理由から，鍵は一定期間ごとに更新し安全性を維持することが必要であると考えられる．

そこで本章では，暗号化ストリームデータ処理方式における

(注2)：これらの結果は，CloudDB'12 において我々が発表した論文 [10] と同様の効率化手法を，実装及び実験環境を改善して再実験した結果であり，論文 [10] に掲載した結果とは異なる点に注意されたい．

効率的な暗号化鍵更新手法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ことが利点であり，鍵の更新を行うためにクエリの実行を停止するのは望ましくない．しかし一方で，クエリ実行中に鍵を更新し，暗号化モジュールが新しい鍵のみを用いて暗号化タプルを生成した場合，SPE 上で実行中の処理木中に新旧それぞれの鍵で暗号化された暗号値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．そこで本章では，“鍵の移行期間” という概念を導入することにより，クエリの実行を停止せずに鍵を更新するための手法を提案する．また，鍵の更新が完了するまでの時間，及び鍵の更新に伴う SPE のメモリ使用量や各モジュールの計算コストのオーバーヘッドを抑えるための効率化手法について述べる．

5.1 評 価 要 素

効率的な鍵の更新を実現するに当たり，本章では以下に示す三つの評価要素を考える．

（1）システム停止時間

SPE に登録されているクエリ中の条件文に含まれる暗号値，処理木中（キュー及びシノプシス）に存在しているすべてのタプル，及び各モジュールの持つ鍵のそれぞれを更新するために，クエリの実行を停止する時間のことを指す．前述したとおり，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ことが利点であるため，システム停止時間は小さい方が望ましいと言える．

（2）鍵の切り替えに要する時間

古い鍵から新しい鍵に切り替えるために要する時間を指す．前述したとおり，古い鍵を使い続けることはいくつかのリスクが存在することから，鍵の切り替えに要する時間は小さい方が望ましいと言える．

（3）計算資源のオーバーヘッド

通常のクエリ処理とは別に発生する，鍵の更新に係る SPE のメモリ使用量のオーバーヘッド，及び各モジュールの計算コストのオーバーヘッドを指し，小さい方が望ましいと言える．

我々はこれら三つの評価要素のうち，システム停止時間を最適化し，その上で鍵の切り替えに要する時間と計算資源のオーバーヘッドを準最適化するような効率化暗号化鍵更新手法を提案する．

5.2 提 案 手 法

5.2.1 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のことを指す．前述したとおり，クエリ実行中のある時刻に一斉に鍵を切り替える（すなわち，移行期間の長さを 0 とする）と，SPE 上で実行中の処理木中に古い鍵（K_1）で暗号化された暗号値のみを持つタプルと，新しい鍵（K_2）で暗号化された暗号値のみを持つタプルが混在し，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，移行期間中に暗号化モジュールに入力された平文タプルは古い鍵（K_1）と新しい鍵（K_2）のそれぞれで暗号化を行い，両方の暗号値を含むタプルを SPE へ送信する．このようなタプルを本稿では “ペアタプル” と呼ぶ．移行期間を用いて段階的に鍵を更新していく様子を図 5 に示す．移行期間が一定時間継続することで，処理木のキュー及びシノプシス中に存在するすべてのタプルがペアタプルで満たされる．すなわち，古い鍵（K_1）で作られた暗号値のみを用いてクエリ処理を行っても，新しい鍵（K_2）で作られた暗号値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．このような状態になった後に移行期間を終了し，暗号化モジュールはペアタプルの送信を終えて，入力された平文タプルを新しい鍵（K_2）のみで暗号化し SPE へ送信する．同時に，SPE 中の各演算子は新しい鍵（K_2）で作られた暗号値のみを用いてクエリ処理を行う．移行期間が終了した直後は処理木中にペアタプルが残存しているが，入力されるタプルは新しい鍵（K_2）で暗号化された暗号値のみを持ち，また各演算子も新しい鍵（K_2）で暗号化された暗号値のみを用いてクエリ処理を行うことから，時間の経過とともにペアタプルは消滅する．

このように移行期間を設け，一時的に生成する暗号値を二重化することにより，クエリの実行を停止せずに鍵の更新を実現することができる．すなわち，前節に述べた評価要素のうち，システム停止時間を最適化することが可能となる．

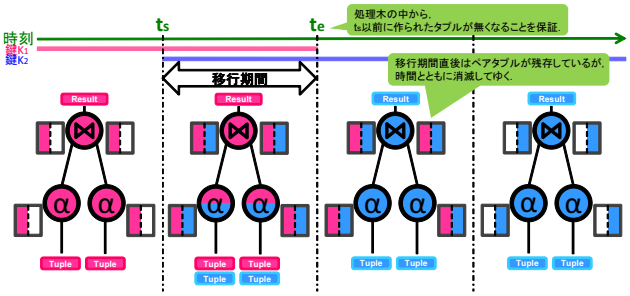

図 5 鍵の移行ステップ

図 5 において，移行期間の開始時刻を時刻 t_s，終了時刻を時刻 t_e と表す．ここで移行期間の終了時刻 t_e は，処理木中に存在しているすべてのタプルが時刻 t_s 以降に生成されたもの，すなわちペアタプルであることを保証できるような時刻にしなければならない．なぜなら，時刻 t_e 以降，SPE は新しい鍵（K_2）で作られた暗号値のみを用いてクエリ処理を行い，古い鍵（K_1）で作られた暗号値は無視されるため，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）で作られた暗号値が必要となるからである．

本章では，処理木中のすべてのステートフル演算子は時間幅で指定されたシノプシスを持つことを前提とする．このとき，処理木を解析することにより移行期間の長さ（$t_e - t_s$）を事前に見積もることが可能となる．しかしながら，各暗号化モジュールで暗号化されたタプルは，SPE に到着するまでの間に通信による遅延が生じる．また，SPE 内部においても処理遅延が発生すると考えられる．ゆえに，実際には通信及び処理遅延を考慮し，移行期間の終了時刻 t_e を動的に変更できるような機構が必要となる．次節で，これらの遅延を考慮した暗号化鍵

更新アルゴリズムについて述べる．

5.2.2 オフセット時間の見積もり

前節に示した移行期間の間，SPE 上で実行中の処理木には葉ノードから順にペアタプル（古い鍵（K_1）と新しい鍵（K_2）による両方の暗号値を含むタプル）が貯まっていく．処理木中の各ステートフル演算子は，自身のシノプシスがペアタプルで満たされるまでは古い鍵（K_1）で作られた暗号値に対して演算を行う．その後，自身のシノプシスがペアタプルで満たされた時刻から，古い鍵（K_1）と新しい鍵（K_2）で作られた暗号値のそれぞれに対して演算を行う．ステートフル演算子 OP_n のシノプシスがペアタプルで満たされる時刻を，時刻 t_{f_n} と表す．また，それぞれのステートフル演算子 OP_n について，移行期間の開始時刻 t_s から時刻 t_{f_n} までの時間を “オフセット時間” と呼び $offset_n$ と表す．

それぞれのステートフル演算子のシノプシスにペアタプルが流入してくるのは，それらの子孫にあたるステートフル演算子が時刻 t_{f_n} 達して，古い鍵（K_1）と新しい鍵（K_2）で作られた暗号値のそれぞれに対して演算を行うようになった後である．よって，処理木の根に近いステートフル演算子の方が，葉に近いステートフル演算子よりもオフセット時間が長くなる．処理木中の各演算子を，根ノードに位置するものから順に OP_0，OP_1，OP_2，$\cdots$ とするとき，ステートフル演算子 OP_n におけるオフセット時間 $Offset_n$ は下式のように表すことができる．ただし，$Range_n$ は演算子 OP_n のシノプシスサイズを意味し，演算子 OP_n がステートフル演算子でないときは $Range_n = 0$ とする．

$$Offset_n = \begin{cases} Range_0, & n = 0 \\ \sum_{i=0}^{n-1} Offset_i + Range_n, & n > 0 \end{cases}$$

上式に基づき，処理木中の各ステートフル演算子のオフセット時間を見積もるためのアルゴリズムを Algorithm 3 に示す．このアルゴリズムでは，はじめに対象となるステートフル演算子を根とする部分木に対して，すべてのパスを抽出している．次に，各パス中のステートフル演算子の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，対象となるステートフル演算子を出るまでに掛かる最大の時間を表したものである．

しかしながら，実際には到着タプルの通信遅延や SPE 内部での処理遅延により，事前に見積もった時刻 t_{f_n} 以降に，古い鍵（K_1）で暗号化された暗号値のみを含むタプルが到着することが考えられる．そこで我々は，遅延時間 d を暗号化ストリームデータ処理システム全体で共有する変数として定義することを考える．ステートフル演算子 OP_n におけるオフセット時間 $Offset_n$，及び遅延時間 d の関係を図 6 に示す．

SPE のインプットアダプタはタプルの到着遅延を逐次検出し，遅延時間 d の値を更新する．また，処理木中の各ステートフル演算子は入力時点でのタプル到着遅延，及び演算子内部での処理遅延を逐次検出し，遅延時間 d の値を更新するとともに，遅延時間 d の値に基づいて時刻 t_{f_n} を動的に変更する．これにより，古い鍵（K_1）で暗号化された暗号値のみを含むタプルが事前の見積もり時刻を過ぎて到着した場合も，これを演算に用いることが可能となる．また，暗号化モジュール及び復号モジュールは d の値を定期的に参照し，移行期間の終了時刻 t_e を動的に延長する．

Algorithm 3 Estimating Offset of Synopsis

```
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: for all node_i ∈ nodes do
5:    Create all of paths which show routes from node_i to leaves
      in the plan tree;
6:    Offset_i ← 0;
7:    for all path_j ∈ paths do
8:       TotalSize ← 0;
9:       for p ← node_i; p ≠ leaf node; p ← p's child node do
10:         if p is a stateful operator then
11:            TotalSize ← TotalSize + Size of synopsis of the op-
               erator;
12:         end if
13:      end for
14:      if TotalSize > Offset_i then
15:         Offset_i ← TotalSize;
16:      end if
17:   end for
18: end for
```

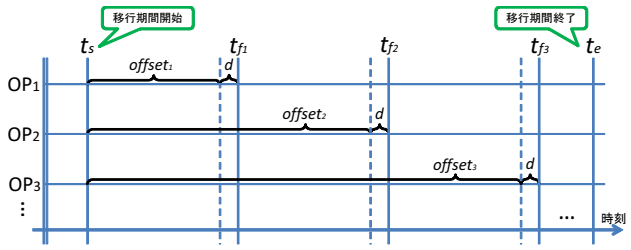

図 6　オフセット時間と遅延時間の関係

SPE のインプットアダプタ，及び各ステートフル演算子がタプル到着遅延を検出するためのアルゴリズムを Algorithm 4 に，各ステートフル演算子が処理遅延を検出するためのアルゴリズムを Algorithm 5 にそれぞれ示す．

Algorithm 4 Detecting Delay for Input Tuple

```
1: for all node_i ∈ nodes do
2:    if node_i is a stateful operator then
3:       dt ← (NowTime − (Offset + d)) − TimeStamp of input
         tuple;
4:       if dt > 0 then
5:          d ← dt;
6:       end if
7:    end if
8: end for
```

これらのアルゴリズムを用いたとき，遅延時間 d の値が頻繁

Algorithm 5 Detecting Delay in Synopsis

```
1: for all node_i ∈ nodes do
2:   if node_i is a stateful operator then
3:     for all tuple_j ∈ synopsis do
4:       if TimeStamp of tuple_j + d < t_s then
5:         d ← t_s − TimeStamp of tuple_j;
6:       end if
7:     end for
8:   end if
9: end for
```

に更新される可能性が考えられる．ゆえに，時刻 t_{f_n} に達して新旧両方の鍵で作られた暗号値のそれぞれに対して演算を行うようになったステートフル演算子 OP_n において，遅延時間 d が更新されることで，時刻 t_{f_n} が現在時刻以降に再設定されてしまう場合が考えられる．このとき，演算子 OP_n に到着するタプルはペアタプルであることが保証されなくなるため，演算子 OP_n は古い鍵（K_1）で作られた暗号値のみを用いて演算を行う状態に戻る．

時刻 t_{f_n} 以前は，到着するタプルに新しい鍵（K_2）で作られた暗号値が含まれていることを保証できない．一方，古い鍵（K_1）で作られた暗号値は，時刻 t_{f_n} によらず，移行期間の終了時刻 t_e 以前に到着するタプルには常に含まれていることが保証される．よって移行期間の終了時刻 t_e に達しない限り，遅延時間 d の更新によって演算子 OP_n における時刻 t_{f_n} が現在時刻以降に再設定されてしまった場合でも，古い鍵（K_1）で作られた暗号値を用いることで正しい演算結果を得られ続けることが保証される．

5.3 提案手法の導入

本章で提案する暗号化鍵更新手法を導入するためには，暗号化モジュール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含める必要がある．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

暗号化モジュールにおいて暗号化タプルを生成する際，以下に示す二つの属性を付加する必要があると考えられる．

EncTime

一般に SPE では，インプットアダプタにおいて，入力される各タプルに時刻情報が付加される．ここで付加される時刻はインプットアダプタを通過する時刻である．しかしながら本章における提案手法では，この時刻情報とは別に，暗号化モジュールにおいて暗号化タプルが生成された時刻（“EncTime” 属性）を付加する必要がある．EncTime 属性は主に，前述した Algorithm 4 や Algorithm 5 において遅延時間を測定するために用いる．

KeyID

タプルの暗号化に用いる暗号化鍵を識別するための値を指す．SPE 上の各演算子において条件評価を行う場合や，復号モジュールにおいて復号を行う際に，評価に用いる暗号値や復号に用いる鍵を選択するために必要となる．

5.3.2 既存演算子の改造

次に，本章で提案する手法を導入するために挙動を改造する必要があると考えられる演算子を以下に示す．

選択（selection）演算子

選択演算子では一般に，登録されている条件式を用いて評価が行われる．我々の暗号化ストリームデータ処理方式において登録されている条件式の右辺は暗号値であるため，鍵の更新に伴って，評価に用いる暗号値も動的に切り替えられるようにする必要がある．

結合（join）演算子

結合演算子は一般に，登録されている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（aggregation）演算子

集約演算子は，シノプシス中に貯まっているすべてのタプルの指定の属性に対して処理を行う．暗号化タプルにおいて，新旧の暗号化鍵を用いて生成された暗号値はそれぞれ異なる属性に格納されているため，時刻に応じて処理に用いる属性を動的に変化させる必要がある．具体的には，時刻 t_{f_n} 以前は古い暗号化鍵（K_1）を用いて暗号化された暗号値を含む属性に対して，時刻 t_{f_n}〜t_e の間は古い暗号化鍵（K_1）及び新しい暗号化鍵（K_2）を用いて暗号化された暗号値を含むそれぞれの属性に対して，時刻 t_e 以降は新しい暗号化鍵（K_2）を用いて暗号化された暗号値を含む属性に対して処理を行う必要がある．

5.3.3 処理木の根ノードにおける重複回避

移行期間中に生成されたタプルペアは，一つの平文値に対して新旧それぞれの鍵を用いて暗号化した暗号値を，両方とも含んでいる．これらは SPE 内部でのクエリ処理のために必要なものであり，復号時にはいずれか一つの鍵で暗号化された暗号値のみあれば良い．そこで，処理木の根ノードに射影演算と同様の機能を導入し，移行期間中に生成されたタプルペアのうち古い鍵（K_1）で暗号化された暗号値を削除する機能を導入することで，SPE から復号モジュールに転送されるデータ量を削減することが可能になる．

5.4 評価実験

本節では，事前に見積もった移行期間の長さに対して実際にどの程度の遅延が生じるのかを確かめるため，暗号化ストリームデータ処理システムにおける SPE 内部での処理遅延時間の測定を行う．なお，通信遅延についてはそれぞれの経路に依存すると考えられるため，本節ではすべてのモジュールを同一マシンで動作させ，処理遅延のみを考えるものとした．実験に用いたマシンの環境を表 2 に示す．

表 2　実験に用いたマシンの環境

CPU	Intel® Xeon® CPU E5640 @ 2.67GHz
Memory	48GB
OS	Ubuntu 10.04.4 LTS (64bits)
Java Runtime	OpenJDK Runtime Environment(IcedTea6 1.11.5)
SPE	SS* (Rev. 134) ※一部機能を追加

また，実験に用いた処理木を図 7 に示す．処理木から，想定される移行期間の長さ（$t_e - t_s$）は 6 秒であることがわかる．

また，処理木に入力するストリーム $s1$ 及び $s2$ に含まれるタプルは，いずれも id 属性の DET 暗号値及び val 属性の OPE 暗号値の 2 つの属性について，それぞれ新旧両方の鍵を用いて暗号化された値を含むペアタプルとし，このうち id 属性の値は常に $DET(1)$ とした．すなわち，処理木中の選択演算と結合演算において選択率（selectivity）は 1 となる．

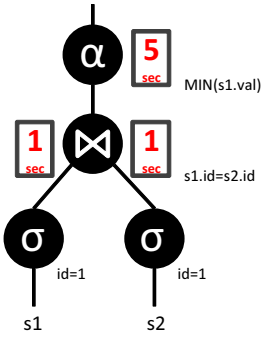

図 7　実験に用いた処理木

処理木から出力されたそれぞれのタプルについて，タプルの暗号化時刻と現在時刻の差を計算することで，処理遅延時間を測定した．この処理木に対して，各ストリーム情報源からの送信レートを変化させながら繰り返し試行を行った．

はじめに，送信レートが 40〜50(tuples/sec) のそれぞれの場合について，処理木から 20 タプルが出力されるまで計測を行い，取った値の幅と平均値を図 8 に示す．このとき，見積もった移行時間（6 秒）からの遅延は最大で約 19 ミリ秒であった．

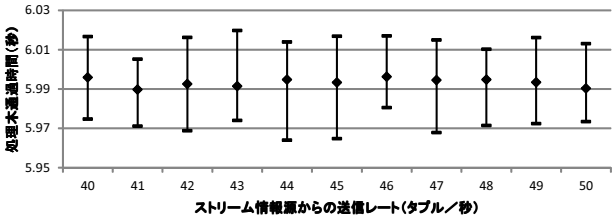

図 8　タプルの処理木通過時間（40〜50(tuples/sec)）

次に，送信レートが 51(tuples/sec) 以上の場合について同様の計測を行った結果を図 9 に示す．送信レートが 51(tuples/sec) を超えると大きな遅延が発生していることがわかる．その原因として，結合演算の選択率（selectivity）が 1 であるために，各ストリーム情報源からの送信レートを r としたときに結合演算子から 1 秒間に出力されるタプル数は $2r^2$(tuples) となり，SPE の性能が限界に達してしまっていることが考えられる．よって，さらに長時間にわたって計測を行ったとき，送信レートが 51(tuples/sec) 以上の場合における遅延時間の最大値はより大きくなっていくと考えられる．

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，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，及びタプルの到着遅延や処理遅延を検出して移行期間を延ばすためのアルゴリズムについて提案した．

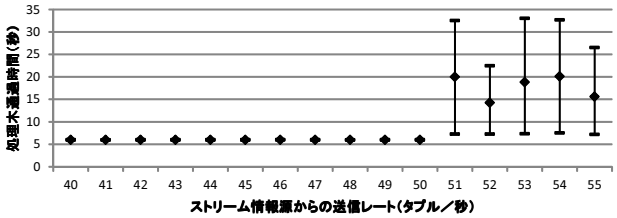

図 9　タプルの処理木通過時間

今後の課題としては，これらのアルゴリズムを我々の研究室で開発中のストリーム処理エンジン及び暗号化・復号モジュールへ実装を行い，さらなる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も検討を行っていきたいと考えている．

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．

文　献

[1] 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」に関する報告書」．http://www.ipa.go.jp/security/fy19/reports/Key_Management/.

[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *VLDB J.*, Vol. 15, No. 2, pp. 121–142, 2006.

[3] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, pp. 224–241, 2009.

[4] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *CCSW*, pp. 85–90, 2009.

[5] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. Spade: the system s declarative stream processing engine. In *SIGMOD Conference*, pp. 1123–1134, 2008.

[6] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *CRYPTO*, pp. 482–499, 2003.

[7] Stefan Hildenbrand, Donald Kossmann, Tahmineh Sanamrad, Carsten Binnig, Franz Faerber, and Johannes Woehler. Query processing on encrypted data in the cloud. 2011.

[8] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pp. 223–238, 1999.

[9] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP*, pp. 85–100, 2011.

[10] Katsuhiro Tomiyama, Hideyuki Kawashima, and Hiroyuki Kitagawa. A security aware stream data processing scheme on the cloud and its efficient execution methods. In *Proceedings of the fourth international workshop on Cloud data management*, CloudDB '12, pp. 59–66, New York, NY, USA, 2012. ACM.