2016 年度 修士論文

完全準同型暗号による

安全頻出パターンマイニングの省メモリ高速化

提出日: 2017 年 1 月 30 日
指導: 山名 早人 教授

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻
学籍番号:5115F016-9

今林 広樹

概　要

近年のクラウド普及及びデータマイニング技術の発展に伴い，データの収集・管理・解析といった一連の処理をクラウド上で行う機会が増加している．その一方で，機密情報の漏洩リスクへの懸念も同時に高まっている．特に医療分野などでは，解析途中・結果の機密性に加え，結果の正確性も求められる．そのため，クラウド上でデータの秘匿性と正確性を保ちながら解析を実行できることは有用である．　完全準同型暗号（FHE: Fully Homomorphic Encryption）は安全かつ正確な処理を実現する方法であり，これまで様々なデータマイニング研究に採用されている．本研究では，FHE を用いてクラウドでデータを秘匿しながら解析することを考え，その解析処理対象として頻出パターンマイニングを取り上げる．　FHE の頻出パターンマイニングへの適用例として，Apriori アルゴリズムを対象にした Liu らのプロトコルがあるが，生成される多量な暗号文，及びその暗号文上での膨大な演算回数により，時間・空間計算量が肥大化することが課題である．また，全データについての処理結果を待機して次の処理に移行するため，データの一斉処理時間や処理遅延が発生する．そこで本研究では，1)暗号文パッキングによる暗号文数の削減，2)暗号文キャッシングによるサポート値計算の高速化，3)暗号文プルーニングによる不要な暗号文の削減，4)ストリーム処理による各データの独立な処理を行うによって，P3CC の課題である時間・空間計算量の削減，及びデータの一斉処理や処理遅延の排除を可能にした．実験評価では，10,000 トランザクションのデータセットにおいて，手法 1)，2)，3)を適用したところ，P3CC の 430 倍の高速化と 94.7%のメモリ使用量削減を達成した．さらに，4)ストリーム処理を適用することにより，　プロトコル実行時間の 23.5%が削減された．

目次

第 1 章　はじめに 1

第 2 章　関連研究 3

2.1　HE による頻出パターンマイニング 3

2.2　FHE による頻出パターンマイニングプロトコル 5

2.3　関連研究のまとめ 6

第 3 章　FHE による頻出パターンマイニングプロトコル 8

3.1　プロトコルの概要 8

3.2　プロトコルに用いる要素技術 9

3.2.1　Apriori アルゴリズム 9

3.2.2　多項式 CRT パッキング 11

3.2.3　CRT 表現暗号文のスロット合計値計算 12

3.3　**P3CC 方式** 13

第 4 章　プロトコルの省メモリ高速化 16

4.1　サポート値計算量の削減 16

4.1.1　暗号文パッキング 16

4.1.2　暗号文キャッシング 18

4.1.3　暗号文プルーニング 20

4.2　処理待機時間・通信時間の削減（隠蔽） 21

第 5 章　実験評価 24

5.1　実験準備 24

5.1.1　データセット 24

5.1.2　実験環境 24

5.1.3　実装 25

5.2　サポート値計算量効率化手法の実験評価 26

5.2.1　暗号文パッキング手法の評価 26

5.2.2　暗号文キャッシング手法の評価 27

5.2.3　暗号文プルーニング手法の評価 27

5.2.4　データサイズに対する計算量評価 28

5.2.5　セキュリティに対する計算量評価 29

5.2.6　公開データセットを用いた計算量評価 30

5.3　ストリーム処理の実験評価 32

第 6 章　おわりに.. 33

第1章　　はじめに[1]

近年のクラウドコンピューティング及びビッグデータ関連技術の発展に伴い，データ収集・管理・解析などの各種処理をクラウド上で行う機会が増えている．組織は顧客・個人情報，特許や企業秘密などの機密情報を蓄積しており，クラウド上でこれらのデータから解析結果を得る場合，解析対象の入力データだけでなく，解析途中のデータや解析結果の出力データもセンシティブな情報となる．特に医薬品や生体情報等のデータは，インサイダーや外部攻撃者の盗聴対象になりやすいため，そのような機密データを扱う医療機関等には，データを秘匿したまま解析を行うクラウド環境が必要となってくる．

本稿では，クラウドサーバへのデータ委託解析を前提とした秘匿技術として，プライバシー保護データマイニング（PPDM: Privacy Preserving Data Mining）について考える[1]．PPDM の研究は，1) 入力プライバシー保護，2) 出力プライバシー保護，3) 暗号化システムの 3 つのアプローチに分類できる．入力プライバシー保護は，クライアントがサーバに解析対象のデータを委託する前に，そのデータに抽象化，ランダム化，ノイズ付加，匿名化等の操作により，センシティブな情報をデータ中から取り除くことができる手法である[2]-[5]．出力プライバシー保護は，サーバが解析結果としてのデータを出力する前に，データへのノイズ付加や摂動化を行うことにより，センシティブな情報の漏洩を防ぐ手法である[6][7]．これらの入力・出力プライバシー保護手法は，1) データの保護範囲が，それぞれ入力データか出力データのどちらか一方である点，2) データの曖昧性を高めることでセンシティブな情報を保護するため，解析結果も曖昧になる点，の 2 点から，機密データの安全性，及び解析結果の正確性が求められる医療分野には適さない．一方で，暗号化システム[8]-[16]は，暗号化されたデータに対して計算を行うことができるため，データの抽象化，匿名化，ノイズ付加などの曖昧性を高める処理を行う必要がない．そのため，データの安全性と解析結果の正確性の双方を同時に満たすことができる．そこで本稿では，PPDM の中でも暗号化システムによるアプローチを選択する．

完全準同型暗号（FHE: Fully Homomorphic Encryption）は，暗号化システムの一つであり，暗号文に対して任意回数の加算・乗算を行える暗号である[17]．この FHE は，統計計算[18]，機械学習アルゴリズム[19][20]，頻出パターンマイニング[15][16]といった様々なデータ解析の研究に応用されている．この FHE を用いた計算の課題点として，膨大な時間・空間計算量を消費することが挙げられる．時間計算量に関しては，暗号文サイズに伴う一つあたりの演算処理の大きさ，及び暗号文数に伴う演算回数の増加により，データ解析に膨大な実行時間を要する．また空間計算量に関しては，大きな暗号文サイズにより，膨大なメモリ領域やストレージ領域を消費する．本稿では，基本的なデータ解析手法として知られる頻出パターンマイニングを取り上げ，FHE の適用により発生する時間・空間計算量を削減することを考える．

---

[1] 情報処理学会論文誌データベース（TOD）にて掲載される論文[32]に基づき記述している．

FHE を頻出パターンマイニングに応用した研究例として，Liu ら[16]の Privacy Preserving Protocol for Counting Candidate（P3CC）がある． P3CC においてサーバは，暗号化されたデータベースに対してサポート値（あるパターンがトランザクション中に出現する頻度）を計算し，その結果をクライアントに送信する．P3CC のサポート値計算には，適用している FHE 方式の性質及びそれに伴うデータベース暗号化方法の性質上，暗号文上での膨大な演算回数を要する．また，全パターンについて処理結果が揃い次第，後続の処理に移行するため，一つ一つの処理に待機時間（処理遅延）や一斉処理時間が発生する．P3CC では特に，1) サーバでのサポート値計算時の膨大な計算時間とメモリ領域（のりきらない場合はストレージ領域），及び 2) 全パターンについてのサポート値計算が終了するまでの待機時間，後続の計算結果（暗号文）の一斉送信や一斉復号にかかる時間が課題となっていた．

そこで本研究では，P3CC のサポート値計算に要する時間とメモリ使用量，サポート値計算の待機時間及び後続の通信時間や復号時間を削減することにより，プロトコルの省メモリ化及び高速化を図る．具体的には，上記の各課題 1), 2)に対して以下の手法を適用することにより実現する．

1) サポート値計算量効率化
   a. 暗号文パッキング（時間・空間計算量削減）
      一つの暗号文で複数の平文を扱うことにより生成される暗号文数削減し，同時にそれに伴う暗号文同士の演算回数も削減した．
   b. 暗号文キャッシング（時間計算量削減）
      サポート値計算の途中結果として生成される暗号文をキャッシング・再利用することにより，類似パターンにおけるサポート値計算を高速化した．
   c. 暗号文プルーニング（空間計算量削減）
      b によりキャッシュされた暗号文のうち，再利用されない暗号文のプルーニングを可能にした．
2) 処理待機時間・通信時間の削減（隠蔽）
   ストリーム処理により，計算処理が終了したサポート値から順に，ファイル書込，通信，ファイル読込，復号の一連の処理を実行する．これにより，あるパターンについて処理時間の長いサポート値計算中に，他のパターンについて各種処理を行うことが可能になり，各種処理のみに要していた時間を削減できる（処理時間の隠蔽）．

本稿の構成は，次に示す通りである．まず第 2 章で関連研究を述べ，第 3 章で FHE による頻出パターンマイニングプロトコルの詳細を説明し，第 4 章でプロトコルの計算量効率化手法を提案する．第 5 章では，提案手法の有用性を示すため，様々なデータセットを用いて実験評価する．第 6 章でまとめと今後の課題を述べる．

第2章　　関連研究

本章では，暗号化システムの中でも完全準同型暗号（FHE: Fully Homomorphic Encryption）を頻出パターンマイニングに適用した関連研究[8]-[16]を紹介する．暗号化システムによるアプローチは，マルチパーティ・コンピューテーション(MPC: Multi-Party Computation)と準同型暗号（HE: Homomorphic Encryption）の 2 つのアプローチに大別できる．各アプローチは，適用するシナリオの前提とデータの秘匿手法の観点から異なる．

MPC は複数パーティ間での通信を前提とし，プログラム回路への入力をパーティ間で秘匿することで秘匿計算を実現する．つまり，各パーティが自身のデータを保持しつつ，また各パーティのデータも受け取りながら計算を実行していくアプローチである．一方で，HE はサーバ・クライアントモデルを前提としており，暗号化されたデータに対し演算を行うことで秘匿計算を実現する．つまり，クライアントがサーバにデータ及び計算を委託することにより，サーバ側で秘匿計算を実行するアプローチである．なお，HE は一定回数までの演算しか行えないが，FHE は任意回数の演算を可能にするため，データマイニングへの適用範囲が広い．表 1 に各アプローチの違いをまとめる．

本研究では，サーバへのデータと計算委託を前提としているため，HE によるアプローチについて関連研究を述べる．2.1 節では，HE による頻出パターンマイニングについて，2.2 節では，本研究に最も関連する FHE による頻出パターンマイニングプロトコルについて，Liu らの P3CC を紹介する．最後に，2.3 節で HE の頻出パターンマイニング適用について関連研究をまとめる．

表 1　各手法適用によるメモリ使用量の変化

	適用シナリオの前提	データの秘匿手法
MPC	複数パーティ間で，各パーティがデータを保持したまま，協力的に各種処理を実行するシナリオ	各パーティがプログラム回路へデータを入力し，回路を通信することで，入力データを秘匿しあう
HE	クライアントがサーバにデータ及び各種処理を委託するシナリオ	データを暗号化することでデータそのものを秘匿する

2.1　HE による頻出パターンマイニング

本節では，暗号化したまま加算もしくは乗算のいずれかを可能とする HE を適用した頻出パターンマイニング研究について述べる．なお，以下の関連研究[8]-[16]では，図 1 のように各トランザクションがパターンを要素としてもつデータベース（図 1(a)）ではなく，トランザクション ID を行に，アイテム ID を列に並べたバイナリ表現のデータベース（図 1(b)）を用いている．これは，暗号化された数値同士の演算を可能にするためである．

Yang ら[8]は，頻出パターンマイニングにおいて，データベース内に出現するパターンの数え上げを秘匿することを目的として，加算に対して準同型性をもつ[2] 楕円 ElGamal[3][21] 暗号を HE として適用した．加法 HE の性質により，サーバ側で秘匿された複数入力の合計値計算を完結できるため，MPC のように複数回の通信を必要とせず，通信回数を一回のみに抑えることができる．1GHz のプロセッサ，512MB のメモリを搭載した PC において，10,000 個の数値を合計するのに 146 ミリ秒の実行時間がかかっている．また，同様の計算を相関ルールマイニングに応用できることを示した．

Goethals ら[9]は，トランザクションを共有するようにアイテム ID で分割された（垂直分割）データベースを前提とし，頻出パターンマイニングのアルゴリズムの一つである Apriori に対して HE を適用することで，サポート値及び各パーティの持つデータを秘匿するプロトコルを提案した．また，Yi ら[10]は，分割しないデータベースを前提とし，同様に HE を Apriori に適用した．これらの HE は加法に対して準同型をもつ Paillier[22]スキームを基にしているため，暗号化データ同士の加算しか行えない．そのため，プロトコル中で必要になる数値比較や乗算は，信頼できるパーティが復号後の平文に対して行っている．なお，Goethals ら[9]及び Yi ら[10]のプロトコルにおいて実装報告はなされていない．

Zhong[11]は，アイテム ID を共有するようにトランザクションで分割された（水平分割）データベースに対して 2-パーティ間の相関ルールマイニングを行うプロトコルを提案した．Goethal ら[9]の頻出パターンマイニングプロトコルで用いられていた Paillier[22]による HE から乗法に対して準同型をもつ ElGamal[21]による HE[4]にスキームを変更し，その上で同様のプロトコルを構築した．この研究においても，実装報告はなされていない．

Zhan ら[12]は，相関ルールマイニングに Paillier[22]暗号による HE を適用し，パターンのサポート値を秘匿計算するプロトコルを提案した．本プロトコルでは，サーバ側で各トランザクション（行）のパターンに対応する要素を暗号化したまま合計し，クライアント側で復号後にパターン長と比較することで，そのパターンが全トランザクションに含まれている頻度を得る．これにより，暗号文上での乗算及び数値比較を行わないプロトコルを実現した．実装報告はなされていない．

Kantarcioglu ら[13]は，相関ルールマイニングに必要なベクトル同士の内積計算について，その計算結果の正確性を下げる代わりに計算時間を短縮可能な手法を提案した．垂直分割

---

[2] ある集合 A の要素同士について演算 B（加算・乗算など）を施したとき，その演算結果も A の要素の一つとなる場合，「B に対して準同型性をもつ」という．

[3] 楕円曲線上の点同士を加算すると，その結果も楕円曲線上の点となる．この性質を利用することで，加法に対して準同型性を持つ楕円 ElGamal 暗号が構築される．

[4] 乗法に対する準同型性をもつ ElGamal 暗号上での計算は，Paillier 暗号上での計算と比較して 8 倍程高速とされる[11].

されたデータベースに対するマイニングを前提としており，Bloom filter[5]とPaillier[22]によるHEを適用することで実現した．本手法の適用により，正確性が約 1%低下する一方で，約 7 倍の高速化を達成した．一方で，3 つ以上のベクトル間での計算はできない．

Kerschbaum[14]は，2 つの集合間での積集合計算（ベクトル要素毎の乗算）について，Boolean filterとHEを適用することで，それぞれ集合サイズと集合要素を秘匿する手法を提案した．このHEは，GoldwasserとMicaliによる暗号[23]であり，2つの暗号化された数値同士の乗算は，平文上でそれらを加算した結果と一致する．平文の法空間は 2 で設定されているため，ベクトルのあるインデックスが同じ値であれば 0，そうでなければ 1 となる．サーバが積集合計算した後，クライアント側でそのベクトルの各要素を復号することで積集合を得る．なお，実装報告はなされていない．

Trans. ID	Item Set
t_1	$\{i_1, i_3, i_4\}$
t_2	$\{i_3, i_5, i_6\}$
t_3	$\{i_2, i_4, i_5, i_6\}$
t_4	$\{i_1, i_2, i_5\}$
t_5	$\{i_3, i_6\}$
t_6	$\{i_1, i_4, i_6\}$

(a)トランザクションデータベース

Items / Trans	i_1	i_2	i_3	i_4	i_5	i_6
t_1	1	0	1	1	0	0
t_2	0	0	1	0	1	1
t_3	0	1	0	1	1	1
t_4	1	1	0	0	1	0
t_5	0	0	1	0	0	1
t_6	1	0	0	1	0	1

(b) アイテム・トランザクションのバイナリ行列

図 1 アイテム･トランザクションデータベース ©IPSJ2017 [32]

2.2 FHE による頻出パターンマイニングプロトコル

Kaosarら[15]は，アイテムIDを共有するようにトランザクションで分割された（水平分割）データベースに対して 2-パーティ間で相関ルールマイニングを行う際の数値比較について，FHEによって暗号化された数値同士での比較を可能にする手法を提案した．また，同様の設定でMPC を用いた場合と比較し，MPC がストレージコスト，通信コスト，計算コストが大きいことから相関ルールマイニングには適さないことを示した．3.40GHz のプロセッサ，16GB メモリを搭載したPCにおいて，FHEによる32bit分の数値比較に7秒程度の実行時間がかかっている．

FHEを頻出パターンマイニングに適用した研究として，Liuら[16] による頻出パターンマイニングプロトコル（P3CC: Privacy Preserving Protocol for Counting Candidates）が挙げられる．P3CC[16]では，暗号文上での数値計算を可能にするため，各トランザクションがアイテム集合をもつ従来のデータベース（図 1(a)）を，行方向に各アイテム ID，列方向に各トランザクション

[5] ある要素が集合に含まれているかどうかを，確率的に判定するためのデータ構造である．含まれる要素を含まれないと判定することはないが，含まれない要素を含まれると判定するため，確率的となる．

ID を並べたバイナリ行列（図 1(b)）に変換している．Liu らが適用した FHE は，整数で暗号文を表現する Dijk ら[24] の方式であり，その性質により，上記バイナリ行列の各要素を暗号化することになる．P3CC では，次の 2 点が問題となる．

1) サポート値計算の時間・空間計算量の問題
   サーバでサポート値計算を行う際に，暗号文上での要素単位の演算回数が膨大となり，それにより計算時間とメモリ領域を要する．
2) 処理待機時間・通信時間の問題
   サポート値計算結果（暗号文）が全パターンについて集まるまでの待機時間，及び暗号化データの一斉送信に伴う通信時間が発生する．また，その後クライアント側での全サポート値の一斉復号時間も発生する．

同論文での実験評価では，ミニマムサポート（サポート値を頻出とするための閾値）を 10%，データセットにはトランザクション数 5,000，アイテム総数 50 を持つ T10I6N50D5kL1k（データ生成器，他パラメータの詳細は 5 章参照），実行環境には Ubuntu12.04 を備えたラップトップ（HP Pavilion dm4）を用いたとき，10,000 秒程度の実行時間がかかっている[16]．

2.3 関連研究のまとめ

表 2 に，2.1 節で紹介した HE による頻出パターンマイニング，及び 2.2 節で紹介した FHE を用いた頻出パターンマイニングプロトコルについてまとめる．

HE を用いた研究[8]-[14]は，加算もしくは乗算のいずれかに対してのみ準同型性をもつ HE を用いているため，いずれかの演算が不足する場合は，クライアントもしくは信頼できるパーティで復号後にその演算を行う必要がある．一方で，FHE は加算と乗算の両方に対して準同型性をもつため，クライアントもしくはパーティにいずれかの演算を依存する必要がなくなる．2.2 節で紹介した P3CC[16]では FHE を用いているため，サーバ側でサポート値まで計算することができる[6]．そのため，頻出パターンマイニングをサーバ側に委託するシナリオでは FHE のほうが適している．

また，FHE で暗号化された数値の比較には，Kaosar ら[15]の研究のように膨大な計算時間がかかる．P3CC[16]では，プロトコルの実行時間を短縮するため，サーバ側で暗号文に対する数値比較を行わず，クライアント側で復号後の平文に対して数値比較を行っている．この点から，高速化のためには暗号文上での数値比較を行わないプロトコルを構築する必要がある．

さらに，P3CC の課題として，2.2 節で述べたように 1) サーバ側のサポート値計算において時間・空間計算量を消費する点，2) サポート値計算終了までの待機時間や暗号化データ一斉送信による通信時間が長い点が挙げられる．プロトコルを省メモリ化・高速化するためには，この 2 点に要する計算量もしくは計算時間を削減する必要がある．

---

[6] 暗号文に対するサポート値の計算には，加法と乗法の双方が必要である．

本研究では，サーバとクライアントの2者間で，クライアントがサーバに秘匿計算を委託することを前提とし，プロトコルの省メモリ化・高速化を目的としているため，次の項目に取り組む．

- HE ではなく FHE を用いることで，サポート値計算の加算と乗算をサーバ側で行う
- 数値比較は暗号文上で行わず，クライアント側で復号後の平文に対して行う
- サポート値計算の時間・空間計算量，及び処理待機時間・通信時間を削減する

表 2　HE／FHE を用いた頻出パターンマイニングプロトコル

	研究目的	HE スキーム	本研究との差異・問題点等
Yang ら[8] 200	複数パーティからの数値（暗号文）を合計するプロトコルの提案.	楕円 ElGamal [21]（加法のみ）	・複数アイテムがトランザクション中に同時出現する頻度のカウントは対象外.
Goethals ら [9]	垂直分割されたデータベースから，サポート値，各パーティの持つデータを秘匿しながら頻出パターンマイニングを行うプロトコルの提案	Paillier [22]（加法のみ）	・分割されたデータベースが前提. ・信頼できるパーティが数値比較や乗算による最終計算を行う.
Yi ら [10]			・クライアント側にて数値比較や乗算による最終計算を行う.
Zhong [11]		ElGamal [21]（乗法のみ）	・クライアント側にて数値比較や加算による最終計算を行う.
Zhan [12]	水平分割されたデータベースから，2-パーティ間でサポート値を計算するプロトコルの提案.	Paillier [22]（加法のみ）	・分割されたデータベースが前提. ・水平分割の性質により，各パターンの存在を確認するための計算方法が異なる.
Kantarcioglu ら [13]	ベクトル間の内積計算を高速化するための手法の提案.	Paillier [22]（加法のみ）	・Bloom filter を用いるため，サポート値の正確性が下がる可能性がある. ・3 つ以上のベクトル間での計算ができない.
Kerschbaum [14]	集合サイズと集合要素を秘匿しながら，2 つの集合間での積集合を計算する手法の提案.	Goldwasser-Micali [23]（加法のみ）	・3 つ以上のベクトル間での計算ができない.
Kaosar ら [15]	FHE によって暗号化された数値同士での比較手法の提案.	SV-FHE [25] [26]（加法・乗法）	・32bit の数値比較で 7 秒程度の実行時間がかかる.
Liu ら [16]	FHE による頻出パターンマイニングプロトコルを構築．サポート値計算における加算・乗算を暗号文上で行う.	DGHV-FHE [24]（加法・乗法）	・サポート値計算に，時間空間計算量を膨大に消費する. ・データの一斉処理により，プロトコル中に各種処理待機時間や処理遅延が生じる.

第3章　　FHE による頻出パターンマイニングプロトコル[7]

本章では，FHE を用いた頻出パターンマイニングプロトコルについて説明する．3.1 節ではプロトコルの概要，3.2 節では，頻出パターンマイニングプロトコルを構築するための要素技術を取り上げる．また 3.3 節では，Liu らの P3CC について具体的に説明する．

3.1 プロトコルの概要

本プロトコルは，信頼できないサーバ（攻撃者モデルとして Semi-honest を仮定）に対しても，頻出パターンマイニングを安全に委託できるプロトコルである． 安全な委託計算を行うためには，データを暗号化してサーバに委託すること，及びサーバは暗号化データに対しマイニングを行えることが必要であり，これを以下で実現する．また，本研究では，頻出パターンマイニング手法として一般的に使われる，Agrawal と Srikant[27]による Apriori アルゴリズムを用いる．この Apriori を暗号文上で実現するためには，加算・乗算の両方が必要となる．これらの要件から，本プロトコルの構築に FHE を用いる．FHE を用いた安全委託型マイニングの大まかな手順は，次の通りである．また，図 2 にその概要図を示す．

① クライアントは，公開鍵と秘密鍵のペアを生成し，公開鍵を用いてデータを暗号化した後，公開鍵と暗号化データをサーバに送る．
② サーバは，クライアントから受け取った要望（クエリ）に対応するデータについて，それを復号することなく，暗号化データのままマイニングを行い，クライアントに結果を返す．
③ クライアントは，マイニング結果として受け取った暗号文を秘密鍵で復号し，要望の結果を得る．

なお，本プロトコルでは，P3CC[16]と同様，暗号文上での数値比較を行わない．これは FHE を用いているため，暗号文上での数値比較ができないことに起因する．なお，近年，有田と中里[28]によって，任意の数値に対する暗号文上での比較手法が提案されたが，条件分岐先の全パターンについて演算を実行する必要があり，計算量が肥大化するため，本研究には適用しない．そのため，P3CC と同様，数値比較が必要な部分に関しては，一度クライアントに暗号文を返却し，クライアント側で復号後に平文上で比較する．上記手順の②で，この手続きが必要となる．具体的には，3.3 節で説明する．

---

[7] 情報処理学会論文誌データベース（TOD）にて掲載される論文[32]に基づき記述している．

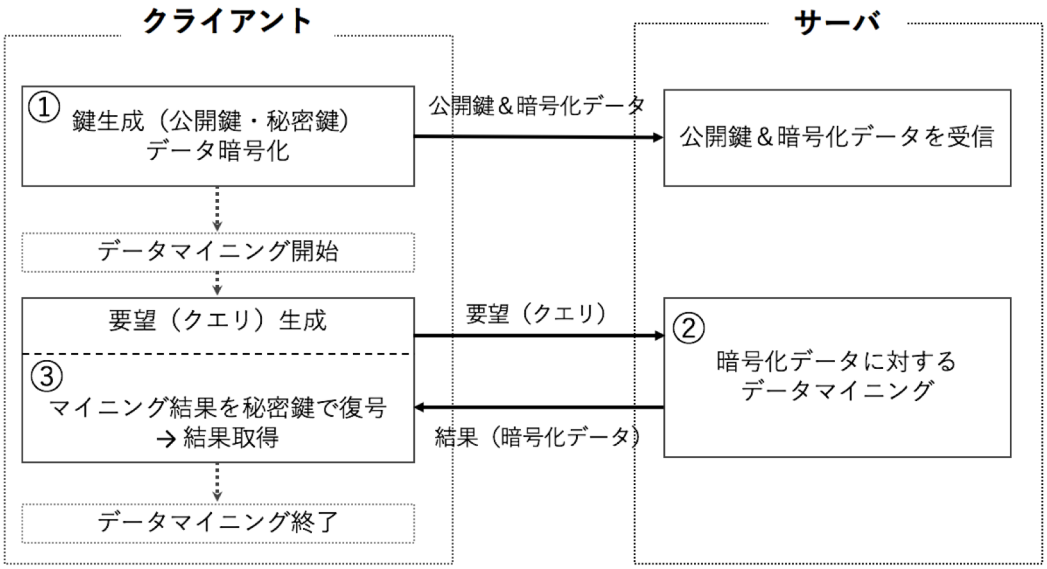


図 2　暗号化データに対するマイニングプロトコルの概要図

3.2 プロトコルに用いる要素技術[8]

本節では，FHE による頻出パターンマイニングプロトコルの構築，及び計算量効率化のために必要な 3 つの要素技術として，1) Apriori アルゴリズム，2) 多項式 CRT パッキング，3) CRT 表現暗号文のスロット合計値計算，を説明する．

3.2.1 Apriori アルゴリズム

Agrawal と Srikant[27]は，頻出パターンを抽出するアルゴリズムとして Apriori を提案した．Apriori で用いるデータベースは，2.2 項で述べたように，各トランザクションがアイテム集合から構成されるものである．本研究では，暗号文上での演算を可能にするため，このデータベースをアイテム･トランザクションのバイナリ行列に変換したものを用いる．以下に，頻出パターンの定義[29]，及び Apriori の手続きを示す[27]．

8 情報処理学会論文誌データベース（TOD）にて掲載される論文[32]に基づき記述している．

定義 1. 頻出パターン[27]

s個の重複しないアイテム集合を$I = \{i_1, i_2, \ldots, i_s\}$, トランザクション集合を$T$とする. 各トランザクション$t \in T$は, Iの中の元からなるアイテム集合をもつ. つまり, tは, i_kをもつなら$t[k] = 1$, そうでなければ$t[k] = 0$を満たすような, Iの部分集合となる. ここで, パターンpをIの部分集合とすると, pのもつ全てのアイテムi_kについて, $t[k] = 1$が成立するときのみ, トランザクションtはpを満足するという. pのサポート値は, pを満足する, Tの中のトランザクション数と等しい. また, そのサポート値が, 与えられたミニマムサポート値以上であるならば, そのパターンは頻出である, という.

Apriori アルゴリズムは, まず, 各アイテムについて, そのアイテムが含まれているトランザクション数をカウントし(サポート値計算), その値がミニマムサポート$minSup$以上のアイテムを頻出アイテム(「パターンの大きさ 1」の頻出パターン)として抽出し, 頻出アイテムの集合 L_1 を得る(アルゴリズム 1 の行 1-4). 続いて, L_1 から「パターンの大きさ 2」の頻出パターン候補の集合 C_2 を生成する(行 7). 例えば, $L_1 = \{\{i_1\},\{i_2\},\{i_3\}\}$としたとき, $C_2 = \{\{i_1, i_2\}, \{i_2, i_3\}, \{i_1, i_3\}\}$が生成される. その後, C_2 の各頻出パターン候補について, その要素となるアイテムが同時に出現するトランザクション数をカウントすることで各頻出パターン候補のサポート値を計算し(行 8-10), サポート値が$minSup$以上となるものを抽出して, 頻出パターン集合 L_2を得る(行 11). 例えば, パターン$\{i_1, i_3\}$のサポート値のみが$minSup$より小さい場合, $L_2=\{\{i_1, i_2\}, \{i_2, i_3\}\}$となる. L_2 は総頻出パターン集合 FPS に保存しておく(行 12). パターンの大きさをインクリメントしながら, 新しい頻出パターン候補の集合が生成されなくなるまで, 上記手続きを繰り返す(行 6-13). 最終的に, パターンの大きさ毎に頻出パターン集合を得る.

アルゴリズム 1 中の countSupport 関数は, 各頻出パターン候補$c \in C_{i+1}$について, バイナリ行列のアイテム列間で要素毎の AND 演算を実行したのち, 全ての要素を足し合わせることにより, cのサポート値を計算する関数である. ここで, 図 1(b) 中のアイテムi_5, i_6に相当する各列を配列として, $\boldsymbol{v}'_{i_5} = (0,1,1,1,0,0), \boldsymbol{v}'_{i_6} = (0,1,1,0,1,1)$と表したとき(3.2.3 項の配列$\boldsymbol{v}$と区別する), 頻出パターン候補 c=$\{i_5, i_6\}$のサポート値を計算することを考える. 具体的には, $\boldsymbol{v}'_{i_5}$と$\boldsymbol{v}'_{i_6}$を要素毎に AND をとり$(0,1,1,0,0,0)$を生成した後, 全要素を足し合わせることで, サポート値 2 を得る(図 3).

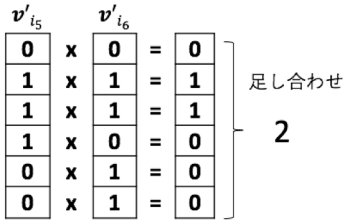


図 3 パターン$\{i_5, i_6\}$のサポート値計算例

アルゴリズム 1. Apriori(I,TDB, minSup) [27] ©IPSJ2017[32]

入力: アイテム集合 I; トランザクションデータベース TDB;
　　ミニマムサポート $minSup$;
出力: 総頻出パターン集合 FPS;

```
1: for each frequent-candidate pattern c ∈ I do
2:    c.support ← countSupport(c, TDB);
3: end for
4: L_1 ← {c ∈ I | c.support ≥ minSup};
5: FPS ← L_1;
6: for (i = 1; |L_i| > 1; i ← i + 1) do
7:    C_{i+1} ← generateFrequentCandidatePatterns(L_i);
8:    for each frequent-candidate pattern c ∈ C_{i+1} do
9:       c.support ← countSupport(c, TDB);
10:   end for
11:   L_{i+1} ← {c ∈ C_{i+1} | c.support ≥ minSup};
12:   FPS ← FPS ∪ L_{i+1};
13: end for
14: return FPS;
```

3.2.2　多項式 CRT パッキング

Smart と Vercauteren[25][26]は，FHE 上に CRT を用いたパッキング手法を提案した．これにより，1 つの平文多項式（係数は整数）で複数の平文（整数）を表現できるため，その平文多項式を暗号化することにより，生成される暗号文の総数を減らすことができる．次の 3 つのステップにより，目的の暗号文を生成する．

① 複数の平文（整数）を要素にもつ配列を生成
② 配列から整数の係数をもつ平文多項式を生成（次数の低い順に係数を並べた配列）
③ 平文多項式の各係数について多倍長整数値に空間を拡張する（暗号化）

ここで生成される平文多項式$f(X)$の次数は FHE パラメータである整数mから生成されるm次円分多項式$\Phi_m(X)$を法としたときの最大次数と一致する．この$\Phi_m(X)$の次数をnとしたとき[9]，$\Phi_m(X)$ はl個のd 次多項式 $F_1(X), \dots F_l(X)$に因数分解できるように設定される$(\Phi_m(X) = F_1(X), \dots F_l(X),$ただし$n = dl)$．ここで CRT により，連立合同方程式$f(X)\ mod\ F_i(X) = \alpha_i\big(1 \le i \le l,\ \alpha_i$は$i$番目につめ込む平文$\big)$を同時に満たす$f(X)$が$F_1(X)F_2(X)\cdots F_l(X)(= \Phi_m(X))$を法として一意に求まる．この$f(X)$は，最大次数を$n'$(ただし$n' < n$)，その係数（整数値）を$a_{n'}$としたとき，$(a_0, a_1, \dots, a_{n'-1}, a_{n'})$（ただし$a_0$は定数項）という配列表現となる．各法空間$F_i(X)$は，平文をつめ込める空間という概念からスロットと呼ばれ，lは$f(X)$がもつスロット

[9] mを正整数とし，オイラー関数$\varphi(m)$を「1 からmまでの整数でmと互いに素となるもの個数」とすると，m次円分多項式$\Phi_m(X)$の次数は$\varphi(m)$（本文中ではn）と一致する．

数に一致する．つまり，1 つの多項式がつめ込める平文数はl個となる．以下では，複数の平文を CRT によって各スロットにもたせた平文多項式を暗号化したものを CRT 表現暗号文といい，多倍長整数型の係数を次数の低い順から並べた配列となる．CRT 表現暗号文同士の演算は，平文を要素にもつ配列同士の要素間の演算として並列化される．

3.2.3 CRT 表現暗号文のスロット合計値計算

Halevi と Shoup[30]は，CRT 表現暗号文が持つ全スロット合計値を算出するためのアルゴリズムとして，TotalSums アルゴリズムを提案した．入力として受け取る CRT 表現された暗号文は，l個のスロットを要素とする配列$\boldsymbol{v} = (v_1, v_2, \ldots, v_l)$（$1 \le i \le l$，スロット$v_i$には$i$番目の平文が含まれる．3.2.2 項参照）とみなすことがでる．TotalSums では，この全スロットに格納されている値の合計値を求め，その合計値を全スロットに格納した配列 $\boldsymbol{u} = (u, u \ \ldots, u)$（ただし，$u = \sum_{i=1}^{l} v_i$）を出力する．手続きをアルゴリズム 2 に示す．ここで， 1)平文からなる配列の各要素をe巡回シフトする操作，及び 2) CRT 表現暗号文$\boldsymbol{u}$の多項式$f_{\boldsymbol{u}}$(X)にX ^{e}を代入後，m次円分多項式$\Phi_m(X)$で法をとる操作（$f_{\boldsymbol{u}}(\mathrm{X}^{e}) \ mod \ \Phi_m(X)$），によって得られる結果は同等となる．つまり，CRT 表現暗号文中の各スロットを平文配列上の各要素とみなした上で，巡回シフト等の操作をすることができる．この操作(2)は，アルゴリズム 2 中で rotate 関数として表す．

アルゴリズム 2. TotalSums(v) [30] **©IPSJ2017[32]**

入力: CRT 表現暗号文（配列）$\boldsymbol{v}$;
出力: CRT 表現暗号文（配列）$\boldsymbol{u}$;

1: $\boldsymbol{u} \leftarrow \boldsymbol{v}$, $e \leftarrow 1$, $n \leftarrow \|\boldsymbol{v}\|$;
2: $k \leftarrow \lceil \log_2 n \rceil$; # n のビット長
3: **for** $(j \leftarrow k-2;\ j \ge 0;\ j \leftarrow j-1)$ **do**
4: $\quad \boldsymbol{u} \leftarrow \boldsymbol{u} + \mathrm{rotate}(\boldsymbol{u}, e)$ *;
5: $\quad e \leftarrow 2e$;
6: $\quad b \leftarrow \mathrm{bit}_j(n)$; # n の j 番目ビット（LSB を 0 番目）
7: $\quad$ **if** $(b = 1)$ **then**
8: $\quad\quad \boldsymbol{u} \leftarrow \boldsymbol{v} + \mathrm{rotate}(\boldsymbol{u}, e)$ *;
9: $\quad\quad e \leftarrow e + 1$;
10: $\quad$ **end if**
11: **end for**
12: **return** $\boldsymbol{u}$

* rotate($\boldsymbol{u}$, e) : $f_{\boldsymbol{u}}(X^e)\ mod\ \Phi_m(X)$ により，$\boldsymbol{u}$ の各要素（スロット）を e 巡回シフト．
（ただし，$f_{\boldsymbol{u}}(X)$ は配列 $\boldsymbol{u}$ の多項式，$\Phi_m(X)$ は m 次円分多項式）

3.3 P3CC 方式

Liu ら[16] は, FHE を用いた安全委託型頻出パターンマイニングプロトコルとして, P3CC を提案した. P3CC が暗号方式として採用した DGHV 方式では, 多倍長整数値qより小さい整数値で暗号文を表現する[24]. このqは FHE のパラメータとして事前に設定される暗号文空間である. また, P3CC は, 図 4 のように行列の要素単位で暗号化を行うため, 各要素の 0 もしくは 1 がq より小さい整数値になる. 暗号化の範囲は行列要素のみであり, トランザクションやアイテムの ID は暗号化されない[16]. Apriori は, 頻出パターン集合を生成する際に, 各頻出パターン候補のサポート値とミニマムサポート値との比較が必要となる. FHE では, その性質上, 暗号化された値同士の比較ができない[10]ため, P3CC[16]では頻出パターンマイニングの途中結果である頻出パターン候補のサポート値をクライアントに返し, クライアントにおいて復号し平文上でミニマムサポートとの比較を行う. 暗号文(多倍長整数で表現)で保持される値は, サーバ側で保持しているアイテム・トランザクションバイナリ行列の各要素, 及び countSupport により得られるサポート値のみである.

: 暗号化範囲

Items / Trans	i_1	i_2	i_3	i_4	...	$i_{N_{item}}$
t_1	1	0	1	1	...	0
t_2	0	0	1	0	...	1
t_3	0	1	0	1	...	1
t_4	1	1	0	0	...	0
:	:	:	:	:	⋱	:
$t_{N_{trans}}$	1	0	0	1	...	1

図 4　バイナリ行列の要素単位暗号化 ©IPSJ2017 [32]

P3CC のプロトコルはアルゴリズム 1 に沿う[16]. 手続きは次の通りである. また, 図 5 に各種処理を対応させて示す.

① クライアントは, 公開鍵と秘密鍵ペアを生成し, 公開鍵でデータベースであるアイテム・トランザクションバイナリ行列の各要素を暗号化する. その後, 公開鍵, 及び各要素が多倍長整数値の暗号化バイナリ行列(図 4)をサーバに送る.

② クライアントは, パターンの大きさが 1 の頻出パターン候補の集合としてアイテム集合 I (平文)をサーバに送る. ただし, I はトランザクションデータベース TDB に出現する全て

[10] 数値をバイナリ化するか, もしくは有田と中里[28]の任意数での比較手法を用いれば, 暗号文上での比較は可能である. しかし, 比較結果も暗号化されているため, 条件分岐先の全パターンについて計算が必要であり, 時間・空間計算量が共に爆発的に増大する.

のアイテムを要素として持つ．各アイテムはアイテム ID によって表現されることにより，アイテム名はサーバ側には秘匿される．

③ サーバは，各パターンc（今，パターンの大きさは 1 であるため，各パターンは 1 つのアイテムから構成され，$c \in I$（平文）となる）に対応するバイナリ行列（図 4）の列（要素は全て暗号文）を読み込み，countSupport（行 2）により暗号文上で各サポート値$c.support$を計算し（行 1-3），クライアントに各サポート値（暗号文，多倍長整数値）を送る．

④ クライアントは，復号された各サポート値$c.support$と$minSup$ とを比較することにより，「パターンの大きさ 1」の頻出パターン集合 L_1 を得る（行 4）．その後，L_1 を総頻出パターン集合 FPS に保存する（行 5）．

⑤ クライアントは，L_1 から generateFrequentCandidatePatterns（行 7）により，「パターンの大きさ 2」の頻出パターン候補の集合 C_2 を生成し，それを平文のままサーバに送る（行 7）．

⑥ サーバは，各頻出パターン候補$c \in C_2$ について対応するバイナリ行列（図 4）の列（要素は全て暗号文）を読み込み， countSupport（行 9）により暗号文上で各サポート値$c.support$を計算し，各サポート値（暗号文，多倍長整数値）をクライアントに返す（行 8-10）．

⑦ クライアントは，復号された各サポート値$c.support$と$minSup$とを比較し，頻出パターン集合 L_2 を生成する（行 11）．その後，L_2 を総頻出パターン集合 FPS に保持する（行 12）．

⑧ パターンの大きさをインクリメントし，⑤，⑥，⑦のプロセスを繰り返す．

このサーバ・クライアント間の複数回の通信中に「サーバがクライアントから得た頻出パターン候補の集合から，頻出パターンを推測できる」というセキュリティ課題が存在する．そのため，P3CC では，ダミーのパターンを頻出パターン候補に加えることで， サーバが頻出パターンを推定する確率を下げることができる α -pattern uncertainty という概念を導入した．これは，真のパターンを推定できる確率を α より小さくすることを保証する．我々の研究においても，P3CC と同様，攻撃者モデルとして Semi-honest 仮定を用い α -pattern uncertainty を採用する．つまり，「サーバはプロトコルに従う中で得られる情報から，ダミーのパターン集合と真のパターン集合とを区別しようと試みる」とする．

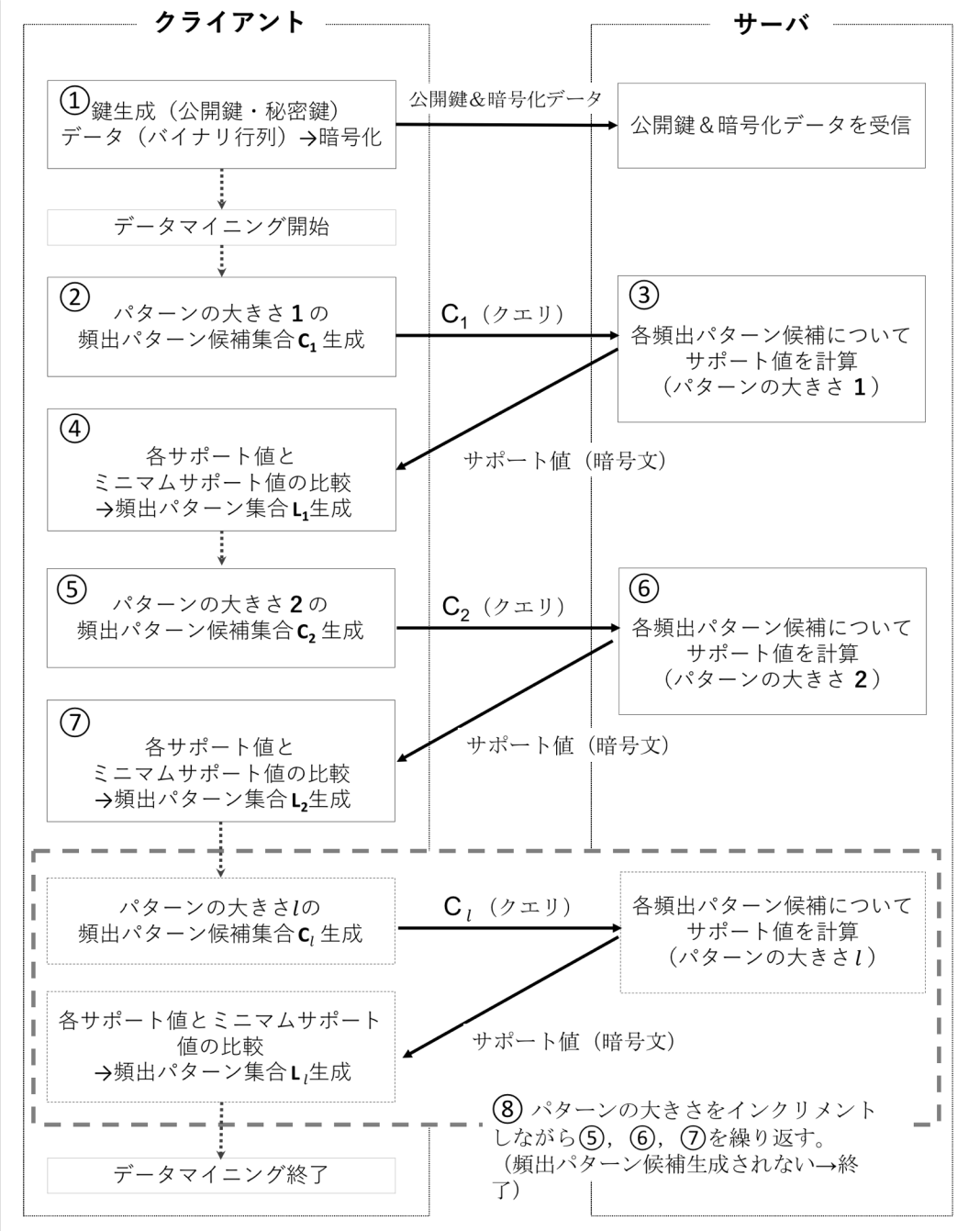


図 5　P3CC による頻出パターンマイニングプロトコルの手続き

第4章　　プロトコルの省メモリ高速化[11]

本章では，3 章で述べた FHE を用いた頻出パターンマイニングプロトコルについて，4.1 節ではサーバでのサポート値計算に要する時間・空間計算量を削減する手法，4.2 節ではサポート値計算の待機時間，後続する通信時間や復号時間を削減する手法，を提案する．

4.1 サポート値計算量の削減

本節では，サーバでのサポート値計算における時間・空間計算量を削減するための手法として，4.1.1 項で時間・空間計算量を削減する暗号文パッキング手法，4.1.2 項で時間計算量を削減する暗号文キャッシング手法，4.1.3 項で空間計算量を削減する暗号文プルーニング手法についてそれぞれ説明する．

はじめに，サーバでのサポート値計算量について具体化する．準備として，3.3 節で述べたように，行と列がそれぞれトランザクション ID とアイテム ID を示すバイナリ行列を用意する．ここで，総トランザクション数（行数）をN_{trans}，総アイテム数をN_{items}（列数 2.1 節定義 1 ではs），CRT 表現暗号文がもつスロット数をlとする．P3CC[16]ではバイナリ行列の各要素について暗号化するため，生成される暗号文数が膨大となり，必要となるメモリ・ストレージ領域の増加，及び暗号文上の演算数の増加をもたらす．具体的には，$N_{trans}\times N_{items}$の数の暗号文が生成され，また，$p_k$を「パターンの大きさ$k$」の頻出パターン候補数，$N_{trans}^{k-1}$を$N_{trans}$の$(k-1)$乗，$t$を頻出パターン候補となるパターンの最大の大きさとしたとき，全パターンのサポート値を計算するために必要な暗号文上での乗算は，$\sum_{k=1}^{t} p_k N_{trans}^{k-1}$回となる．結果として，サポート値に大きな時間・空間計算量がかかる．以下，この乗算回数を暗号文パッキング及び暗号文キャッシングにより削減することを考える．

4.1.1　　暗号文パッキング

FHE 上での Apriori 実行にかかる時間・空間計算量を削減するために，多項式 CRT によるパッキング手法[25][26]を適用する．まず FHE の方式を，整数ベースの DGHV 方式[24]から，多項式ベースの BGV 方式[31]に変換する．これにより，FHE に 3.2.2 項の多項式 CRT パッキングが適用でき，複数の平文を一つの CRT 表現暗号文につめ込むことができるようになる．

これを Apriori に適用する際には，サポート値計算時に必要な乗算回数を減らすため，図 6 のようにバイナリ行列を列単位（アイテム単位）でパッキングする．1つの暗号文につめ込めるのはl個の要素であるため，あるアイテム ID の全トランザクションをつめこむのに必要な暗号文

[11]情報処理学会論文誌データベース（TOD）にて掲載される論文[32]に基づき記述している（ただし，4.1.3 項，4.2 項を除く）．

数は，$\lceil N_{trans}/l \rceil$個となる．ここで，$N_{trans}$が lで割り切れない場合は，余りのr個の要素については，ダミーとして$l-r$個の0と共に 1 つの暗号文につめこむことで実現できる（図 7）．

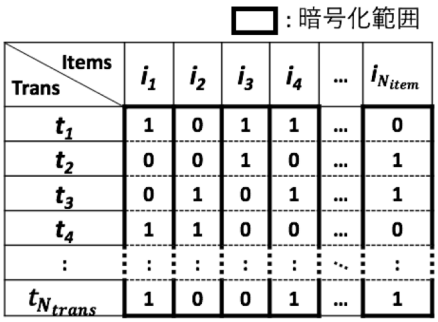


図 6　バイナリ行列の列単位暗号化 ©IPSJ2017 [32]

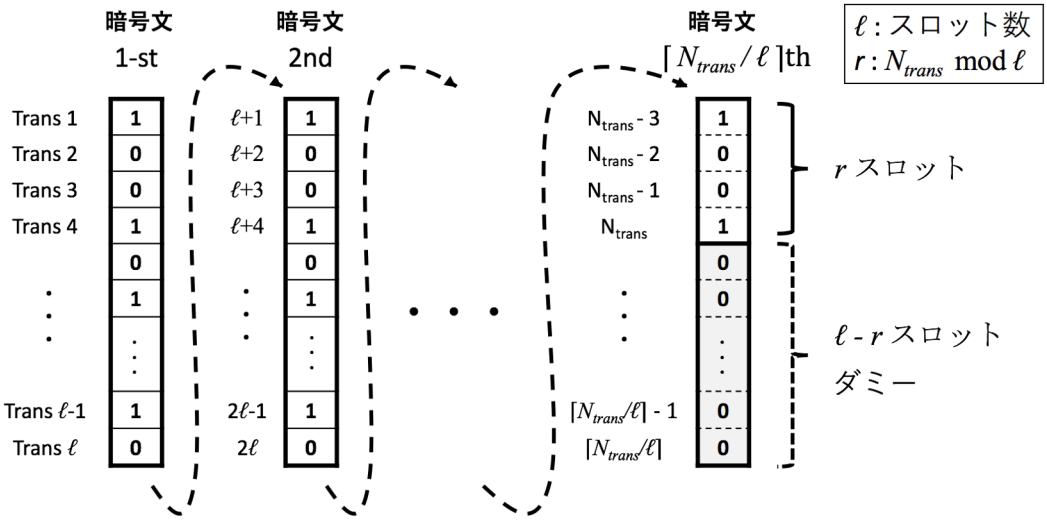


図 7 アイテム列要素の暗号文パッキング　©IPSJ2017 [32]

多項式 CRT パッキング手法を適用することにより，2 つの利点がある．1 点目に，データベースの全要素を暗号化するために必要な暗号文数が，$N_{trans} \times N_{items}$から$\lceil N_{trans}/l \rceil \times N_{items}$に削減され，それに伴い，メモリ使用量も同量削減される．2 点目に，全てのパターンについてサポート値計算に必要な乗算総数が，$\sum_{k=1}^{t} p_k N_{trans}^{k-1}$から$\sum_{k=1}^{t} p_k \lceil N_{trans}/l \rceil^{k-1}$（$p_k$，$N_{trans}^{k-1}$，$t$は上述の通り）に削減されるため，実行時間を短縮できる．図 8 に，パターン例$\{i_1, i_2, i_3\}$のサポート値計算時の乗算について，多項式 CRT パッキング手法の適用前（図 8(a)）と適用後（図 8(b)）の様子を示す．

また, 暗号文 1 つ分の記憶容量について, 1) 単一の平文のみから生成する場合と, 2) l個の平文をつめ込んだ場合は等しくなる. これは, 上記 1), 2)の双方の場合において, 暗号文を生成する途中で同次数の平文多項式表現(係数は整数値, 配列)に一度変換されるためである(3.2.2 項参照). ここで生成される暗号文の記憶容量の大きさは円分多項式の次数(3.2.2 項参照)及び暗号文空間の大きさ(多倍長整数値)に依存し, これらは実験の事前に設定される FHE のパラメータによる. つまり, より多くの平文を多項式につめ込めるパラメータを選定することで, 暗号文に占める単一平文の記憶容量は小さくできるが, 上記 1), 2)の FHE パラメータが同一である限り, 一暗号文に必要な最小単位としての記憶容量は変わらない.

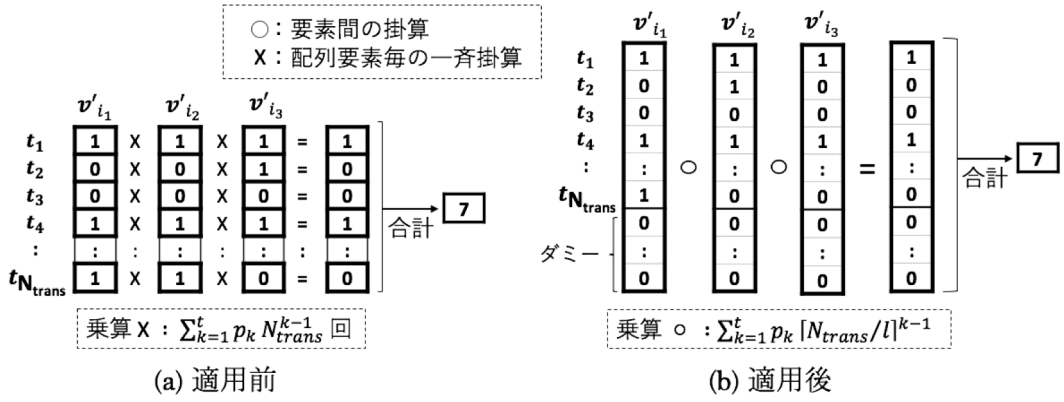


図 8 暗号文パッキング適用による乗算数の比較

4.1.2 暗号文キャッシング

パターンサポート値計算にかかる時間計算量を削減するための手法として, 暗号文キャッシング手法を提案する. このキャッシング手法は,「パターンの大きさk」の各頻出パターン候補について, サポート値計算結果をキャッシュしておき,「パターンの大きさ$k+1$」の頻出パターン候補のサポート値計算時に再利用することで, 冗長な計算を省きサポート値計算を高速化することを目的としている.

具体的に,「パターンの大きさk」の頻出パターン候補$c = \{i_{f(1)}, i_{f(2)}, \dots, i_{f(k)}\}$(ただし, $f(j)\ (1 \le j \le k)$ は $1 \le f(j) \le N_{items}$を満たすアイテム ID)のサポート値を FHE 上で計算する場合を考える. いま, 図 6 の行列においてアイテム$i_{f(j)}$が参照する列(トランザクション集合)を配列として$\boldsymbol{v}'_{i_{f(j)}}$と表し(3.2.3 項の配列$\boldsymbol{v}$と区別する),「$\circ$」を配列要素毎の乗算とすると, $\circ_{j=1}^{k} \boldsymbol{v}'_{i_{f(j)}}$を行うことで, 各要素に乗算された結果を保持した暗号文配列が生成される. c のサポート値は, この配列の全要素を足し合わせることで得られる(3.2.3 項参照). 例えば「パターンの大きさ 4」の頻出パターン候補$\{i_1, i_2, i_3, i_4\}$のサポート値は, $\boldsymbol{X} = \boldsymbol{v}'_{i_1} \circ \boldsymbol{v}'_{i_2} \circ \boldsymbol{v}'_{i_3} \circ \boldsymbol{v}'_{i_4}$を計算し, 配列 $\mathbf{X}$ の全要素を合計して得る. ここで,「パターンの大きさ 4」のサポート値計算の

前に，「パターンの大きさ 3」についても同様の計算を行っているので，$\boldsymbol{v'}_{i_1} \circ \boldsymbol{v'}_{i_2} \circ \boldsymbol{v'}_{i_3}$, $\boldsymbol{v'}_{i_1} \circ \boldsymbol{v'}_{i_2} \circ \boldsymbol{v'}_{i_4}$, $\boldsymbol{v'}_{i_1} \circ \boldsymbol{v'}_{i_3} \circ \boldsymbol{v'}_{i_4}$, $\boldsymbol{v'}_{i_2} \circ \boldsymbol{v'}_{i_3} \circ \boldsymbol{v'}_{i_4}$の 4 つの計算結果を得ていたことになる．

上記のような Apriori のサポート値計算順序を利用し，パターンと計算結果のペアをキャッシングし，再利用することで，冗長な演算を省き，演算回数削減と実行時間の短縮を実現できる．例えば，キャッシュ無しの場合は，$\boldsymbol{X} = \boldsymbol{v'}_{i_1} \circ \boldsymbol{v'}_{i_2} \circ \boldsymbol{v'}_{i_3} \circ \boldsymbol{v'}_{i_4}$と 3 回の演算∘を用いて $\boldsymbol{X}$ を得ていたところを（図 9(a)），$\boldsymbol{Y} = \boldsymbol{v'}_{i_1} \circ \boldsymbol{v'}_{i_2} \circ \boldsymbol{v'}_{i_3}$をキャッシュしておくことで，$\boldsymbol{X} = \boldsymbol{Y} \circ \boldsymbol{v'}_{i_4}$の 1 回の演算∘で $\boldsymbol{X}$ を得ることができるようになる（図 9(b)）．

この暗号文キャッシング手法により，各パターンのサポート値計算にかかる演算∘を 1 回に抑えることができる． Apriori で生成される全頻出パターン候補に対するサポート値計算では，キャッシュ無しでは $\sum_{k=1}^{t} p_k \lceil N_{trans}/l \rceil^{k-1}$回の演算∘が必要であったが，キャッシング手法を用いることで，$\sum_{k=1}^{t} p_k \lceil N_{trans}/l \rceil$回に削減できる．アルゴリズム 3 に，キャッシングを用いた FHE によるサポート値計算のためのアルゴリズムを示す．なお，このアルゴリズム中で，3.2.3 項で説明した TotalSums を用いている．

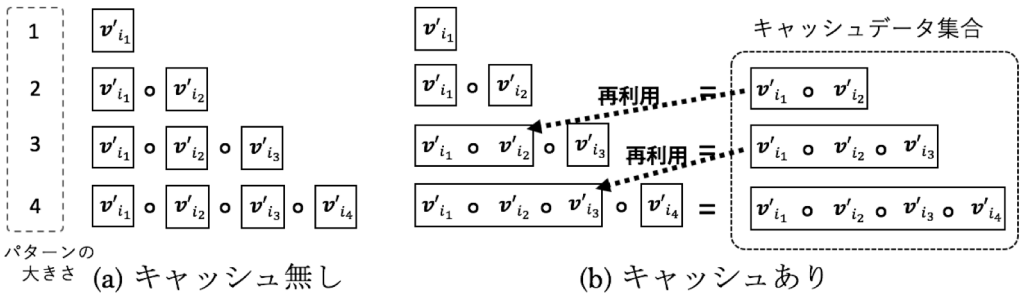


図 9 キャッシングによるサポート値計算 ©IPSJ2017 [32]

アルゴリズム 3. CountSupportWithCache(ET DB, C, CD) ©IPSJ2017[32]

入力: 暗号化トランザクションデータベース $ETDB$;
　　頻出パターン候補の集合 C; キャッシュデータ集合 CD;
出力: サポート値配列 S; CD（更新）;

```
1: S ← ∅;
2: for each frequent-pattern candidate c ∈ C do
3:    k ← |c|;
4:    itemID ← c.pop(k − 1); # k − 1: 最後の要素
5:    c' ← c; # 大きさ k − 1 のパターン {0, 1, ..., k − 2}
6:    hashKey ← setHashKeyfromPattern(c');
7:    cache ← getCachedDatabyKey(CD, hashKey);
8:    col ← getItemColumnfromETDBbyID(itemID);
9:    res ← elementwiseVectorMultiply(cache, col);
          # res[i] ← cache[i] × col[i] (ベクトル要素ごとの乗算)
10:   support ← TotalSums(res); # res の全要素合計
11:   S.append(support);
12:   newHashKey ← makeNewHashKey(c);
13:   CD ← CD ∪ makePair(newHashKey, res); # ペアをキャッシュする
14: end for
15: return S, CD;
```

4.1.3　暗号文プルーニング

暗号文キャッシングにより，全頻出パターン候補についてキャッシュした場合，再利用されない暗号文キャッシュにより余分なメモリを消費することになる．その再利用されない頻出パターン候補を特定し，対応する暗号文のキャッシングを防ぐため，暗号文プルーニング手法を提案する．本手法は，サーバがクライアントから受け取った「パターンの大きさk」の頻出パターン候補集合から「パターンの大きさ$k+1$」の疑似頻出パターン候補集合を生成[12]し，その各疑似頻出パターン候補の部分集合（パターンの大きさk）に含まれない頻出パターン候補をプルーニング対象とする．サーバのメインスレッド内で，ある頻出パターン候補に対応する暗号文がキャッシュされる前に，本手法によりプルーニング対象を選定することで，再利用される可能性のある頻出パターン候補のみをキャッシュする．これにより，暗号文キャッシングの問題であったメモリ使用量増加を抑えることができる．また，これはサポート値計算用のスレッドと別スレッドで実行され，サポート値計算よりも短時間で終了するため，プロトコル実行時間は延長されない．図 10 に「パターンの大きさ 3」の時の具体例とともに，暗号文プルーニング手法を取り込んだプロトコルを示す．

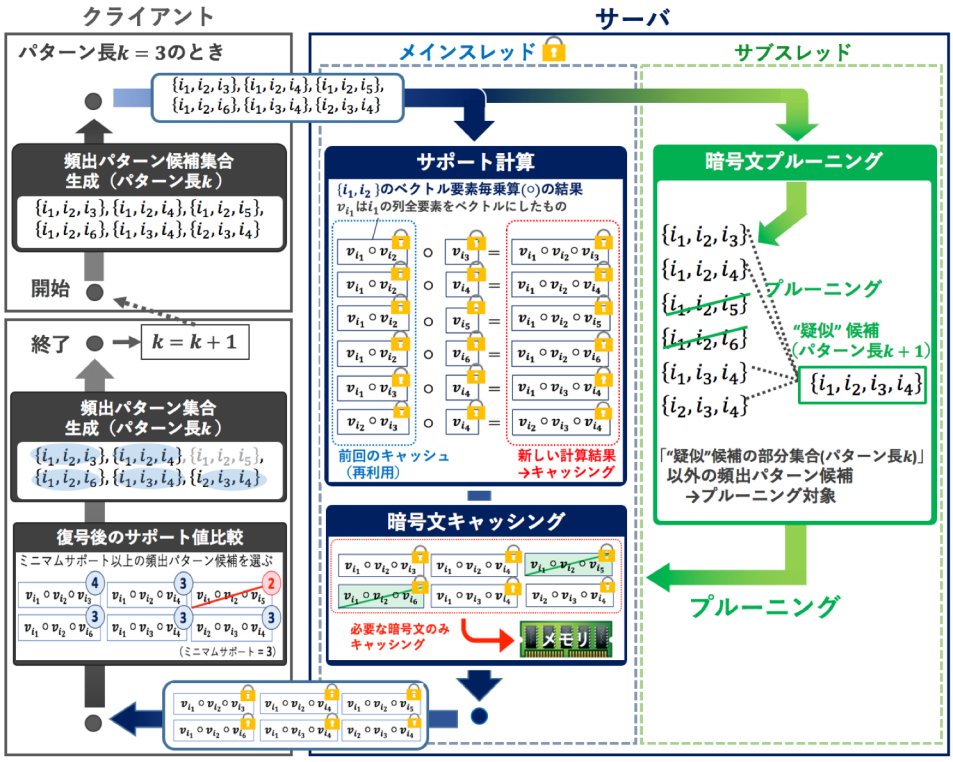


図 10　3 手法適用によるサポート値計算効率化プロトコル

12 疑似頻出パターン候補集合の生成アルゴリズムは，Apriori アルゴリズムで頻出パターン候補集合を生成するアルゴリズムと同じである．

4.2 処理待機時間・通信時間の削減(隠蔽)

本節では, P3CC のもう一つの課題である, サーバ側での全サポート値計算結果の待機時間, 及びそれらの一斉送信による通信時間とクライアント側での全サポート値の復号時間について, これらを削減(隠蔽)する手法とプロトコルを提案する. なお, 3.3 節のプロトコルの手続きに従った上で, 本処理を適用する.

図 11 に, 3.3 節のプロトコル手続きにおいて, サーバが頻出パターン候補集合を受信してから, 全サポート値(暗号文)を送信し, クライアントがそれらを復号するまでのプロセスを具体化する. また, 以下 3 つの課題をどのように解決するかを示す.

- 全サポート値の計算終了までの待機時間
- 全計算結果(暗号文)の一斉送信による大きな通信時間
- クライアント側での全計算結果の一斉復号時間

図 11 に示すように, ①サーバがクライアントから頻出パターン候補集合を受け取った後, ②サポート値計算が完了した頻出パターン候補から順に, ③対応するサポート値のファイル書込, ④通信, ⑤ファイル読込, ⑥復号の一連の処理を行う(ストリーム処理).

これにより, 他の頻出パターン候補について②の処理を行っている間に, ②の処理を既に終えた頻出パターン候補は続きの③～⑥の処理を進めることができるようになる. つまり, ②と③の処理間の待機時間, ④通信時間, ⑥一斉復号時間を, ②サポート値計算を行う処理時間に隠蔽することが可能になり, 結果的にプロトコルの実行時間を削減することができる. 図 12 にストリーム処理適用前と適用後の違いを示す. なお, 図 12 中の各種処理ステップ番号についても, 図 11 中のそれと対応している. また, 以下に疑似コードも示す.

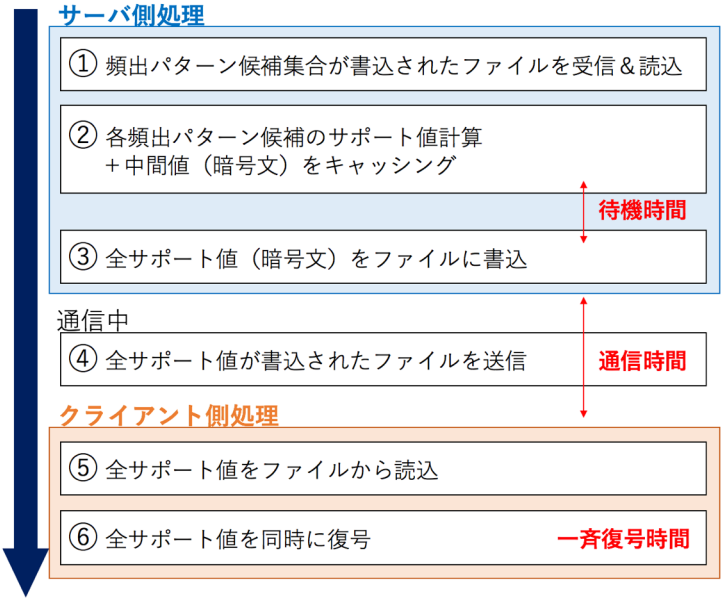
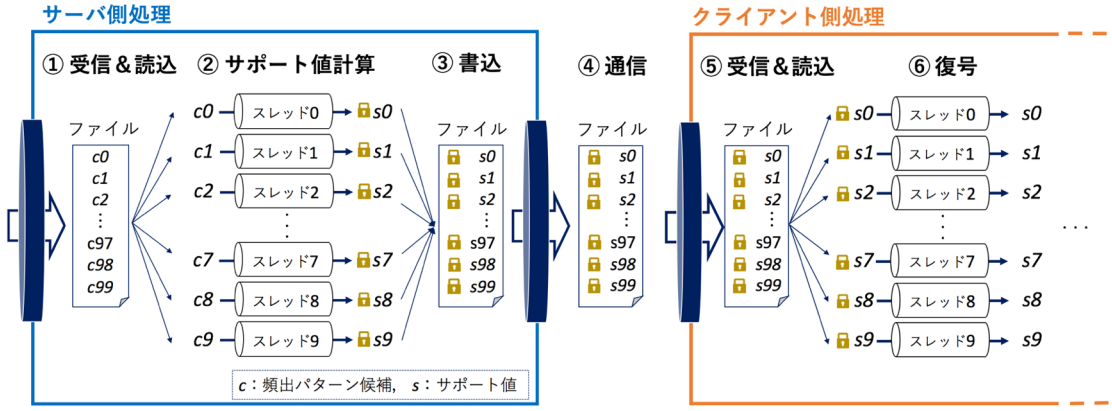
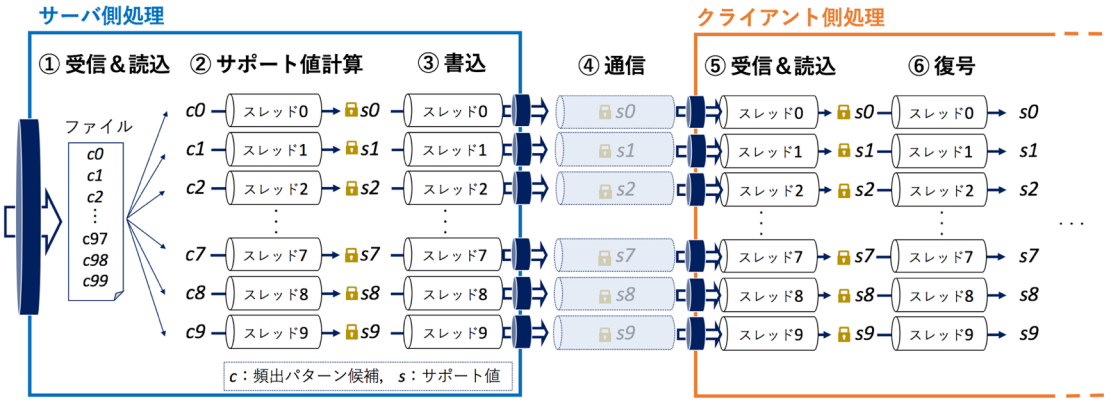


図 11 処理の流れと処理の遅延



(a) ストリーム処理適用前



(b) ストリーム処理適用後

図 12　ストリーム処理適用によるプロトコル比較

疑似コード： ストリーム処理

▷ **サーバ側処理**

◇ 入力：暗号化トランザクションデータベース $ETDB$,
　　頻出パターン候補集合 C;
◇ 出力：各サポート値（暗号文）ファイル $encSupFile$,

```
function main (){
    Foreach (frequent-pattern candidate c ∈ C){
        thread ← findIdleThread(); #待機中のスレッドを探す
        # スレッドに一つずつ頻出パターン候補を流す
        thread.streamSupportCount(c);
    };
};
/* サポート値計算と送信*/
function streamSupportCount(c);
    support ← countSupport(c, ETDB); #サポート値計算
    encSupFile ← writeSupportToFile(support);
    sendSupportFile(encSupFile); #クライアントに送信
};
```

▷ **クライアント側処理**

◇ 入力：各サポート値（暗号文）ファイル $encSupFile$
◇ 出力：サポート値配列 (復号後のサポート値をまとめる) S;

```
function main (){
    /* サブスレッドにて streamFileRead を呼び出す */
    S ← ∅; # サポート値配列を初期化
    encSupFile ← queue.pop(); #キューから取り出す
    thread ← findIdleThread(); #待機中のスレッドを探す
    # スレッドに暗号文を流し，一つずつ復号する
    thread.streamSupportDecrypt(encSupFile);
};
/* サブスレッドで動作する関数 */
function streamFileRead();
    queue ← ∅; # キューを初期化
    count ← 0;
    # 常にファイルを受信し続ける
    While (count < |C|){
        encSupFile ← recvEncSupFileFromServer(); #サーバから受信
        queue.push(encSupFile); #キューに一時的に追加
        count ← count + 1;
    };
};
/* 暗号文の復号 */
function streamSupportDecrypt(encSupFile);
    encSup ← readSupportFromFile(encSupFile);
    support ← decryptEncSup(encSup); #復号
    S.push(support); #サポート値配列に追加
};
```

第5章　　実験評価[13]

本章では，FHE を用いた Apriori 実装に対して，4 章で述べた各種手法をそれぞれ適用し，実験により有用性を評価する．5.1 節では実験準備について説明し，5.2 節では 4.1 節で述べたサポート値計算量効率化手法の適用結果，5.3 節では 4.2 節で述べたストリーム処理の適用結果を述べる．さらに，5.2 節において，1) データセットのトランザクション数を変化させた場合，2)データセットの総アイテム ID 数を変化させた場合，3) α -pattern uncertainty によってダミーセットを追加した場合についても測定し，Apriori の時間・空間計算量評価を行う．なお，本実験では，実行時間を「クライアントが公開鍵と暗号化データをサーバに送った直後から，新たな頻出パターン候補が生成できなくなるまで（3.3 節手続き③-⑧）」とした．

5.1　実験準備

実験準備として，5.1.1 項ではデータセット，5.1.2 項では実験環境，5.1.3 項では実装に用いたライブラリ等について述べる．

5.1.1　　データセット

本実験評価には2種類のデータセットを用いた．1 つ目に，IBM Quest Synthetic Data Generator[14]によって人工的に生成されたデータセットである．パラメータ{T, I, N, D, L}を変化させることで，様々なデータセットを作成できる．ただし，T は 1 つのトランザクションがもつ平均アイテム長，I は最大のパターンの大きさ，N はトランザクション中の異なるアイテム ID 数，D はトランザクション ID 数，L は生成されうる頻出パターン数である．また，2 つ目に，Frequent Itemset Mining Dataset Repository[2] (FIMI) に公開されている chess データセット（3,196 トランザクション，75 アイテム）を用いた． この公開データセットを用いた実験評価は，5.2.6 項及び 5.3 節で行う．

5.1.2　　実験環境

実験環境は，同一ネットワーク内のクライアントとサーバの 2 つのマシンから構成され，10Gb Ethernet で接続されている．クライアントの CPU は Intel Xeon CPU E5-2643 v3（3.4GHz），メモリは 512GB であり，サーバの CPU は Intel Xeon CPU E7-8880 v3（2.3GHz），メモリは 1TB である．OS は，CentOS6.6 である．（図 13）．

---

[13] 情報処理学会論文誌データベース（TOD）にて掲載される論文[32]に基づき記述している（ただし 5.2.3 項，5.3 項を除く）．

[14] http://fimi.ua.ac.be/data/

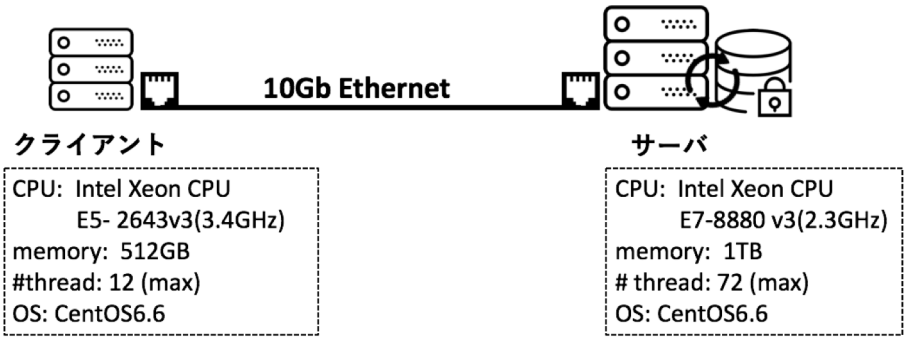


図 13 サーバ・クライアントのマシン環境

5.1.3 実装

本実験は BGV 方式[31]をサポートした公開 FHE ライブラリである HElib[15], 多倍長整数演算を行うためのライブラリである GMP[16], 整数多項式を扱うためのライブラリである NTL[17]を用いて実装した. HElib はパラメータ{*p, r, k, l, c, w*}を設定することで, FHE を構築する. ただし, p^rは平文空間, kはセキュリティパラメータ, lは FHE の回路の深さ(レベル), cはキースイッチ行列の行数, wは秘密鍵に用いるハミング重みである. なお, 本実験では, P3CC [16]と同様に bootstrapping[18]を用いていないため, 暗号文上で適当数の演算を可能とするようにレベル l を設定している. {*p, r, k, l, c, w*} = {2, 14, 80, 10, 3, 64}と設定している(chess データセットの場合は l=20 もしくは l=30). 平文空間2^{14}は D の最大設定値まで扱うための値, レベル 10 は複数回乗算を扱うための値, k, c, wはデフォルト値としている.

本実験では, GMP 多倍長整数演算ライブラリにより, 十分な桁数の整数演算を扱えるようになる. これにより, 暗号化した状態での演算結果の精度を保証する. また, マイニング結果の正確性を確認するため, 1) 行列要素単位暗号化, 2) 列単位暗号化, 3) 平文のまま(暗号化無し)の 3 つの場合において, クライアント側で得られる各頻出パターン候補のサポート値比較を行う.

---

15 http://shaih.github.io/HElib/index.html

16 https://gmplib.org/

17 http://www.shoup.net/ntl/

18 FHE 暗号文には, セキュリティのためにノイズと呼ばれる項が含まれている. このノイズは, 回路レベルlまで演算の度に増加し, ある閾値を超えると復号エラーとなる. bootstrapping は, 暗号文内に lまでの計算結果の正確性を保持したままノイズを小さくできる手法であり, それ故に任意回数の演算が可能となる.

5.2 サポート値計算量効率化手法の実験評価

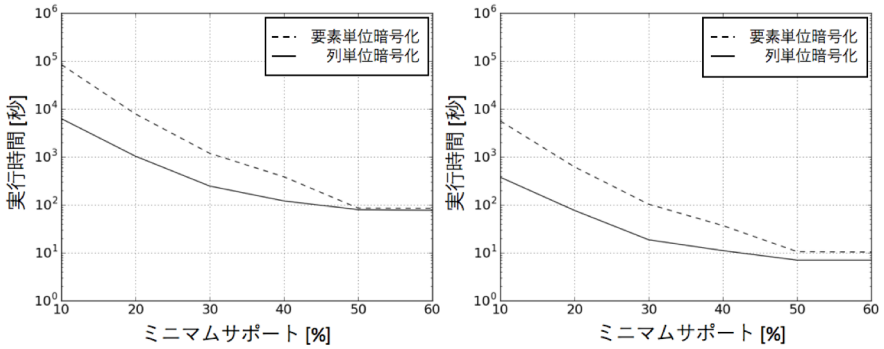
本節では，サポート値計算量を効率化する手法として 4.1 節で述べた，暗号文パッキング，暗号文キャッシング，暗号文プルーニングの 3 手法をそれぞれ適用し，有用性を評価する．

5.2.1 暗号文パッキング手法の評価

暗号文パッキング手法の有用性を評価するため，P3CC の特徴である行列要素単位の暗号化方式，及び提案手法であるパッキング適用による列単位の暗号化方式について，それぞれ Apriori 実行時間及びメモリ使用量を測定し比較する．P3CC では整数ベースの FHE（DGHV 方式）を用いているが，本実験では多項式ベースの FHE（BGV 方式[31]）上で比較する．しかし，暗号文を多項式表現に変更することで， 行列要素単位の暗号化方式実行のための計算時間，及びメモリ使用量に制約が生じる．そのため本実験では，小さいデータセットである T10I6N50D100L1k を用いて比較を行う．

図 14 に，要素単位暗号化方式（パッキング非適用，点線）と列単位暗号化方式（パッキング適用，実線）のそれぞれについて，各ミニマムサポート（10%~60%）による実行時間の推移を示す．図 14 (a)はシングルスレッド実装，図 14 (b)はマルチスレッド実装を示す．マルチスレッドに関して，クライアントではファイル読み書きと暗号化を 12 スレッド，サーバではファイル読み書きとパターンサポート値計算を 24 スレッドで実行している．

要素単位暗号化方式と比較して，列単位暗号化方式は，ミニマムサポート 10%において，シングルスレッド実行で 13.4 倍，マルチスレッド実行で 14.9 倍の高速化を達成した．また，メモリ使用量については，マルチスレッドで 92.3%削減した（445GB から 33.8GB）．なお，シングルスレッドでは，要素単位暗号化方式においてメモリ領域の上限に達し，実測不可能であった．また，両方式から得られる各頻出パターン候補のサポート値，及び平文のままで実行した場合のサポート値が一致したことから，多項式表現による暗号化，及び列単位暗号化方式が正確に機能していることを確認した．



(a) シングルスレッド実行　　(b) マルチスレッド実行（24スレッド）

図 14 暗号文パッキング適用・非適用方式比較 ©IPSJ2017 [32]

5.2.2　暗号文キャッシング手法の評価

暗号文キャッシング手法の有用性を評価するため，列単位暗号化方式，及び列単位暗号化に暗号文キャッシング手法を追加適用した方式について，それぞれ Apriori 実行時間及びメモリ使用量を測定し比較する．なお，同一の基準で比較するため，同じデータセットを用いる．

図 15 に，列単位暗号化方式（キャッシング非適用，点線）と列単位暗号化・キャッシング方式（キャッシング適用，実線）のそれぞれについて，各ミニマムサポート（10%~60%）による実行時間の推移を示す．図 15(a)はシングルスレッド実装，図 15(b)はマルチスレッド実装を示す．なお，4.1 節と同様の適用箇所・スレッド数である．

列単位暗号化方式と比較して，列単位暗号化・キャッシング方式は，ミニマムサポート 10%において，シングルスレッド実行で 1.62 倍，マルチスレッド実行で 1.42 倍の高速化を達成した．また，メモリ使用量については，キャッシングの影響により，シングルスレッドで 12.2%増加（28.9GB から 32.4GB），マルチスレッドで 53.1%増加した（33.8GB から 51.7GB）．なお，キャッシング機能はマイニング結果の精度に影響を与えないため，正確性は保証されたままである．

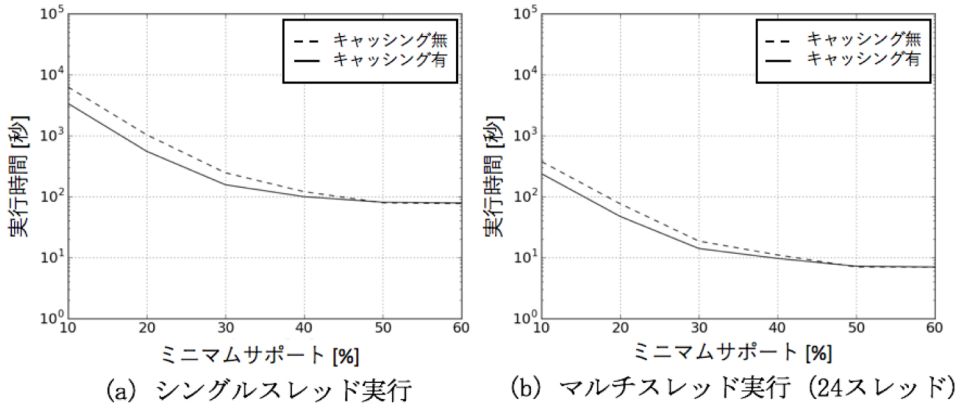


図 15　暗号文キャッシング適用・非適用方式比較 ©IPSJ2017 [32]

5.2.3　暗号文プルーニング手法の評価

暗号文プルーニング手法の有用性を評価するため，様々なデータセットを用いて実行時間，メモリ使用量，キャッシュ数を測定し，評価を行った．表 3 に各データセットにおける各種項目を示す．これらの項目は，「P3CC のみ」，「P3CC に暗号文パッキング・キャッシングを適用」，「P3CC に暗号文パッキング・キャッシング・プルーニングを適用」の順で左から結果を並べている．表 3 の結果より，暗号文プルーニングを適用することでメモリ使用量は削減され，実行時間の増加は見られないことが確認できる．N が 200 のとき最大のメモリ削減（6.09%）を記録した．

表 3　各手法適用によるメモリ使用量の変化

P3CC → P3CC + (a) + (b) →P3CC + (a) + (b) + (c)　(N/A: 1TB以上)

データセット*	N	実行時間 (sec)	メモリ (GB)	キャッシュ数
T6I6N30D10kL1k	30	4,725 → 34.7 → 34.7	428 → 15.9 → 15.7	(none)→ 2,976 → 2,449
T10I6N50D10kL1k	50	26,315 → 79.1 → 79.4	536 → 25.4 → 24.7	(none)→ 8,153 → 7,285
T14I6N70D10kL1k	70	44,986 → 140 → 129	738 → 34.9 → 33.0	(none)→ 15,438 → 13,299
T20I6N100D10kL1k	100	N/A → 265 → 262	N/A → 55.3 → 53.9	(none)→ 30,225 → 27,187
T40I6N200D10kL1k	200	N/A → 1,358 → 1,354	N/A → 230 → 216	(none)→ 164,052 → 151,621
T60I6N300D10kL1k	300	N/A → 2,185 → 2,187	N/A → 373 → 352	(none)→ 266,352 → 246,789

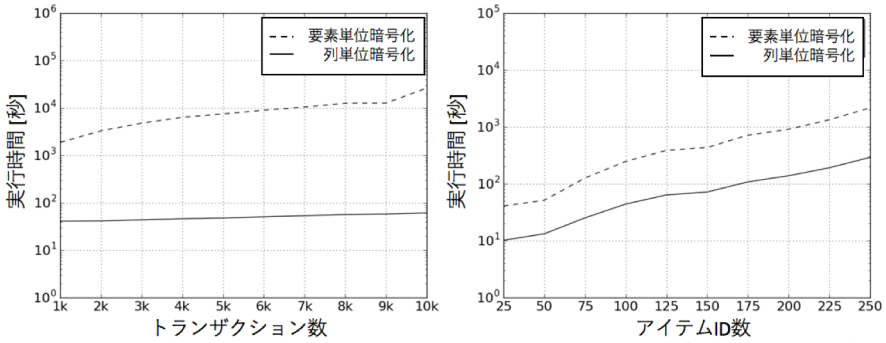
(a): 暗号文パッキング，(b): 暗号文キャッシング，(c): 暗号文プルーニング

5.2.4　データサイズに対する計算量評価

本項では，暗号文パッキングと暗号文キャッシングによる計算量効率化手法が，データセットサイズの変更に依存せず，計算量を削減することを確認する．以下では，トランザクション数とデータセット総アイテム ID 数をそれぞれ変化させる．

まず，データセットのトランザクション数を変化させた場合について，要素単位暗号化方式と列単位暗号化方式の各々にキャッシング手法を適用し，それぞれの実行時間を比較する．まず，データセットは T10I6N50D1kL1k とし，パラメータ D を 1k から 10k まで 1k ずつ増やしながら変化させていく．ミニマムサポートは 20%，サーバ側を 48 スレッドに設定し，各データセットについて実行時間を測定した結果を図 16(a)に示す．D=10k のとき，列単位暗号化方式（実線）は要素単位暗号方式（点線）と比較して 430 倍高速であり，94.7%のメモリが削減された（536GB から 34.5GB）．

続いて，データセットの総アイテム ID 数を変化させた場合について，要素単位暗号化方式と列単位暗号化方式の各々にキャッシング手法を適用し，それぞれの実行時間を比較する．まず，データセットは T5I6N25D100L1k とし，パラメータ T を 5 から 50 まで 5 ずつ，N を 25 から 250 まで 25 ずつ同時に増やしながら変化させていく．ミニマムサポートは 30%，サーバ側を 48 スレッドに設定し，各データセットについて実行時間を測定した結果を，図 16(b)に示す．N=250 のとき，列単位暗号化方式（実線）は要素単位暗号化方式（点線）と比較して，7.42 倍高速で，92.3%のメモリが削減された（473GB から 36.5GB）．



(a) トランザクション数の変化（48スレッド） (b) アイテムID数の変化（48スレッド）

図 16 データサイズについての計算量変化 ©IPSJ2017 [32]

5.2.5 セキュリティに対する計算量評価

本項では，暗号文パッキングと暗号文キャッシングによる計算量効率化手法が， セキュリティ概念（α-pattern uncertainty）の付加に依存せず，計算量を削減することを確認する．そのために，P3CC によるα-pattern uncertainty でダミーセットを追加した場合について，要素単位暗号化方式と列単位暗号化方式の各々にキャッシング手法を適用し，それぞれの実行時間を比較する．パラメータαは「真の頻出パターン候補の集合」を「ダミーを含む全体の頻出パターン候補の集合」の中から推測できる確率である．αが増加するとセキュリティは脆弱になることから，α^{-1}をプライバシーパラメータとして扱うことができる．データセットは T10I6N50D1kL1k，ミニマムサポートは 20%，サーバ側を 48 スレッドに設定する．パラメータα^{-1}を 1（ダミーセット無し）から 6 に 1 ずつ増やしながら， 実行時間を測定した結果を，図 17 に示す．$\alpha^{-1} = 6$のとき，列単位暗号化方式（実線）は要素単位暗号化方式（点線）と比較して，63.3 倍の高速化，80.9%のメモリ削減（177GB から 33.8GB）を記録した．

また，上記 3 つの各ケースについて，要素単位暗号化方式と列単位暗号化方式の両方式から得られる各頻出パターン候補のサポート値，及び平文のままで実行した場合のサポート値が一致したことから，多項式表現による暗号化，及び列単位暗号化方式が正確に機能していることを確認した．

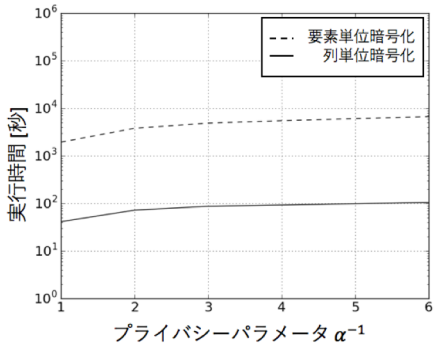


図 17 プライバシーパラメータについての計算量変化 ©IPSJ2017 [32]

5.2.6　公開データセットを用いた計算量評価

本項では，P3CC[16]の評価で用いられている chess データセットを用い，暗号文パッキング手法，暗号文キャッシング手法，暗号文プルーニング手法の有用性を評価する．chess データセットは，3,196トランザクション，75アイテムからなるが，メモリ使用量について測定可能な範囲（1TB 以下に収まる範囲）で実験するため，後半 1,596 トランザクションをデータセットとして用いた[19]．このとき，HElib パラメータは，$\{p, r, k, l, c, w\} = \{2, 11, 80, 20, 3, 64\}$と設定した．

図 18(a)に要素単位暗号化方式（パッキング非適用，点線）と列単位暗号化方式（パッキング適用，実線）のそれぞれについて，各ミニマムサポート（91, 93, 95, 97, 99%）での実行時間を示す．なお，ミニマムサポートの選択は P3CC[16]で用いられている 90～100%の範囲で選択した．並列化は 72 スレッドで行っている．要素単位暗号化方式と比較して，列単位暗号化方式は，ミニマムサポート 91%において，55.3 倍の高速化を達成し，メモリ使用量については，95.8%削減した（687GB から 28.6GB）．また，両方式から得られる各頻出パターン候補のサポート値，及び平文のままで実行した場合のサポート値が一致したことから，多項式表現による暗号化，及び列単位暗号化方式が正確に機能していることを確認した．

次に，図 18(b)に列単位暗号化方式（キャッシング非適用，点線，図 18(a)実線と一致）と列単位暗号化・キャッシング方式（キャッシング適用，実線）のそれぞれについて，各ミニマムサポート（91, 93, 95, 97, 99%）について実行時間の推移を示す．なお，並列化は 72 スレッドで行っている．列単位暗号化方式と比較して，列単位暗号化・キャッシング方式は，ミニマムサポート 91%において，1.56 倍の高速化を達成した．また，メモリ使用量については，キャッシングにより 0.03%増加した（28.64GB から 28.65GB）．なお，キャッシング機能はマイニング結果の精度に影響与えないため，正確性は保証されたままである．ここでさらに暗号文プルーニングを適用することにより，キャッシングにより増加したメモリ使用量が削減（28.65GB→28.64GB）された[20]．また，プルーニング適用によるプロトコル実行時間は 0.01%（96.00 秒→96.01 秒）の増加であり，プロトコルの実行時間を延長することなくメモリ使用量を削減可能なことを確認した．

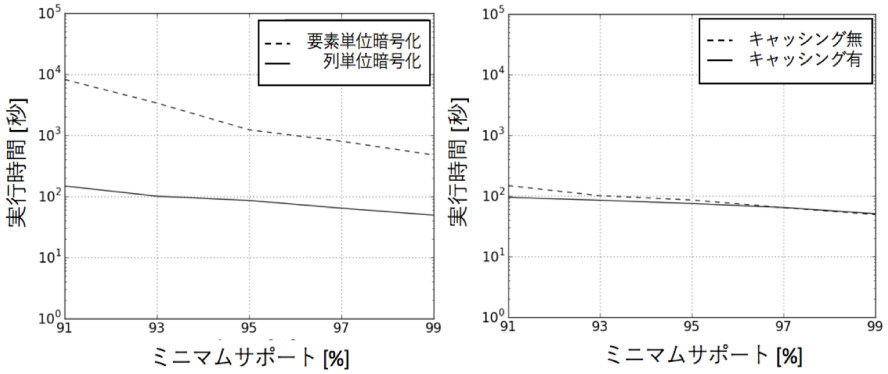
パッキングとキャッシング及びプルーニングを共に適用することで，ミニマムサポート 91%において，全て適用していない要素単位暗号化方式と比較して，86.3 倍の高速化を達成し，メ

---

[19] P3CC [16]で用いられている整数表現による暗号方式と比較し，多項式表現による暗号方式は暗号文のセキュリティが高い一方，一つの暗号文サイズが大きくなる．そのため，chess データセットの全トランザクションを用いた場合，メモリ上限である 1TB を超えて測定不能のため，半量をデータセットとして用いた．なお，P3CC[16]の実験評価では，HP Pavilion dm4 laptop を用いたと表記されているが，CPU・メモリ等の具体的な数値は不明瞭である．

[20] chess データセットの場合，生成される頻出パターン候補数が少ないため暗号文キャッシングによるメモリ増加は微量となり，それ故暗号文プルーニングの効用が見えにくい．本編では総アイテム数を変化させ，様々な人工データセットで評価することにより，有用性を確かめた．

モリ使用量については 95.8%削減した(687GB から 28.4GB)． 表 4 に，各方式を P3CC に同時に適用させた場合について，ミニマムサポートを変化させたときのメモリ使用量を示す．



(a) パッキング手法適用（72スレッド） (b) キャッシング手法適用（72スレッド）

図 18 パッキング・キャッシング手法適用 ©IPSJ2017 [32]

表 4 各手法適用によるメモリ使用量の変化

ミニマムサポート / 適用手法	91%	93%	95%	97%	99%
P3CC	687.1 GB	678.0 GB	678.0 GB	678.0 GB	678.0 GB
P3CC + (a)	28.64 GB	23.73 GB	20.33 GB	20.33 GB	20.33 GB
P3CC + (a) + (b)	28.65 GB	23.74 GB	20.68 GB	20.48 GB	20.40 GB
P3CC + (a) + (b) + (c)	28.64 GB	23.74 GB	20.42 GB	20.48 GB	20. 35 GB

（微差のため，有効数字4桁で記載）

(a): 暗号文パッキング，(b): 暗号文キャッシング，(c): 暗号文プルーニング

5.3 ストリーム処理の実験評価

本実験では，ストリーム処理適用による各実行時間の変化をみる．具体的には，以下の実行時間 A，B の 2 つを測定する． 本実験のデータセットには，Frequent Itemset Mining Dataset Repository[2] (FIMI) に公開されている chess データセット（3,196 トランザクション，75 アイテム）を用い，ミニマムサポートはトランザクション数の 90%と設定した．HElib パラメータは，{*p, r, k, l, c, w*} = {2, 13, 80, 30, 3, 64}と設定した．また，サーバは 24 スレッド，クライアントは 12 スレッドで実行した．

A. 全体の実行時間
   クライアントが暗号化データをサーバに送った直後から，Apriori が終了するまで（3.3 節手続き③から⑧まで）

B. ストリーム処理適用箇所の実行時間
   サーバが頻出パターン候補集合を受信してから，全サポート値（暗号文）を送信し，クライアントがそれらを復号するまで（3.3 節手続き⑥から⑦まで）

表 5 にストリーム処理適用による実行時間 A，B の変化を示す．実行時間 A について，ストリーム処理により，23.5%の時間が削減（1399 秒→1070 秒）された．また，実行時間 B については，26.2%の時間が削減（1252 秒→924 秒）された．また，「ストリーム処理適用箇所の実行時間（B）が全体の実行時間（A）に占める割合」が削減されている（89.5%→86.4%）ことから，従来各種処理にかかっていた時間が，処理時間の長いサポート値計算の時間に隠蔽されていることが確認できる．

表 5 ストリーム処理適用による実行時間の変化

	適用前	適用後	削減時間・割合
実行時間 A	1399 秒	1070 秒	329 秒（23.5%）
実行時間 B	1252 秒	924 秒	328 秒（26.2%）
B/A	89.5%	86.4 %	

第6章　　おわりに

本研究では，FHE による頻出パターンマイニングが 1) 膨大な時間・空間計算量を消費する点，及び 2) 各種処理の待機時間とデータ通信時間に時間を要する点，の 2 点の課題に着目し，それぞれを改善する手法を提案した．具体的には，課題点 1)に関して，サーバでのサポート計算に対して暗号文パッキング手法，暗号文キャッシング手法，暗号文プルーニング手法をそれぞれ適用した．また，課題点 2)に関して，プロトコルにストリーム処理を適用した．実験評価から，これらの手法を適用することで，実行時間とメモリ使用量を大きく削減可能であることを示した．特に，10,000 トランザクションからなる人工データセットにおいて，暗号文パッキング手法，暗号文キャッシング手法，暗号文プルーニング手法の 3 手法を適用したところ，P3CC の 430 倍の高速化と 94.7%のメモリ使用量削減を達成した．さらにストリーム処理を適用することにより，プロトコル実行時間の 23.5%が削減された．

しかし，現在のプロトコルではデータ毎にストリーム処理が適用されているだけであり，複数のデータをまとめた処理や各種処理実行のスケジューリング等が考えられていないこと，ネットワーク帯域や計算資源等の実行環境制限を考慮していないことなど，高速化に向けた改善の余地が残されている．また，現在頻出パターンマイニングの中でも Apriori アルゴリズムを対象にしているが，この問題点として，膨大な数のパターン候補が生成されるため，それに伴うサポート値計算量も膨大となる．一方，ダイナミック・アイテムセット・カウンティング（DIC: Dynamic Itemset Counting）は，サポート値をカウントしながら同時に頻出パターンを決定していくため，パターン候補の生成数を抑えることができる．そこで，次のステップでは，DIC を処理対象とし，課題として想定される通信回数や通信量，サポート値計算の実行スケジューリング等を工夫することを考える．

謝辞

本研究は科学技術振興機構（JST）CREST に支援を頂きました. 本研究を進めるにあたり, 数々のご指導を頂いた山名早人教授に厚く御礼申し上げます. また, 研究や実装への助言, 共著書への協力等を頂いた石巻優さん, 馬屋原昂さん, 佐藤宏樹さん, 安村慶子さん, JungKyu Hun 先輩に深く感謝致します.

参考文献

[1] Gellman, R.: Privacy in the clouds: risks to privacy and confidentiality from cloud computing, *Proc. World Privacy Forum,* 2009.

[2] Evfimievski, A., Gehrke, J. and Srikant, R.: Limiting privacy breaches in privacy preserving data mining, *Proc. ACM SIGMOD-SIGACT- SIGART Symposium on Principles of Database Systems (PODS 2003)*, pp. 211–222, 2003.

[3] Qiu, L., Li, Y. and Wu, X.: Protecting business intelligence and customer privacy while outsourcing data mining tasks, *Knowledge and Information Systems,* vol. 17, no.1, pp. 99–120, 2008.

[4] Tai, C.H., Yu, P.S. and Chen, M.S.: k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining, P*roc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010)*, pp. 473–482, 2010.

[5] Wang, Y. and Wu, X.: Approximate inverse frequent itemset mining: Privacy, complexity, and approximation, *Proc. IEEE International Conference on Data Mining (ICDM 2005)*, pp. 482–489, 2005.

[6] Atzori, M., Bonchi, F., Giannotti, F., et al.: Anonymity preserving pattern discovery, *The International Journal on Very Large Data Baes*, vol. 17, pp. 703–727, 2008.

[7] Bhaskar, R., Laxman, S., Smith, A., et al.: Discovering frequent patterns in sensitive data, *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010)*, pp. 503–512, 2010.

[8] Z. Yang, S. Zhong, and R. N. Wright.: Privacy-preserving classification of customer data without loss of accuracy, *Proc. SIAM International Conference on Data Mining (SDM'05)*, pp. 92-102, 2005.

[9] Goethals, B., Laur, S., Lipmaa, H. and Mielikainen, T.: On private scalar product computation for privacy-preserving data mining, *In International Conference on Information Security and Cryptology,* pp. 104-120, 2004.

[10] Yi, X., and Zhang, Y.: Privacy-preserving distributed association rule mining via semi-trusted mixer, *Data & Knowledge Engineering*, vol. 63, pp. 550-567, 2007.

[11] Zhong, S.: Privacy-preserving algorithms for distributed mining of frequent itemsets. *Information Sciences*, vol. 177, pp. 490-503, 2007.

[12] Zhan, J., Matwin, S. and Chang, L.: Privacy-preserving collaborative association rule mining. In *IFIP Annual Conference on Data and Applications Security and Privacy*, vol. 30, pp. 153-165, 2005.

[13] Kantarcioglu, M., Nix, R. and Vaidya, J.: An efficient approximate protocol for privacy-preserving association rule mining. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, *LNCS*, vol. 5476, pp. 515-524, 2009.

[14] Kerschbaum, F., 2012, May. Outsourced private set intersection using homomorphic encryption. *Proc. ACM Symposium on Information, Computer and Communications Security,* vol. --, pp. 85-86, 2012.

[15] Kaosar, M.G., Paulet, R. and Yi, X.: Fully homomorphic encryption based two-party association rule mining, *Data & Knowledge Engineering,* vol. 76, pp. 1–15, 2012.

[16] Liu, J., Li, J., Xu, S., et al.: Secure outsourced frequent pattern mining by fully homomorphic encryption, *Big Data Analytics and Knowledge Discovery, LNCS,* vol. 9264, pp. 70–81, 2015.

[17] Gentry, C.: Fully homomorphic encryption using ideal lattices, *Proc. Annual ACM Symposium on Theory of Computing (STOC 2009)*, pp. 169–178, 2009.

[18] Naehrig, M., Lauter, K. and Vaikuntanathan, V.: Can homomorphic encryption be practical?, *Proc. 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124, 2011.

[19] Graepel, T., Lauter, K. and Naehrig, M.: Ml confidential: Machine learning on encrypted data, *Information Security and Cryptology–ICISC 2012, LNCS*, vol. 7839, pp. 1–21, 2012.

[20] *Khedr, A., Gulak, G.* and Vaikuntanathan, V.: Shield: Scalable homomorphic implementation of

encrypted data-classifiers, *IEEE Transactions on Computers*, vol. 65, No.9, pp. 2848-2858, 2015.

[21] T. ElGamal.: A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions Information Theory, vol. 31, pp. 469–472, 1985.

[22] P. Paillier.: Public-key cryptosystems based on composite degree residuosity classes, LNCS, vol. 1592, pp. 223–238, 1999.

[23] Goldwasser, S. and Micali, S.: Probabilistic encryption, Journal *of computer and system sciences*, vol. *28*, pp. 270-299, 1984.

[24] Van Dijk, M., Gentry, C., Halevi, S., et al.: Fully homomorphic encryption over the integers, *Advances in cryptology–EUROCRYPT 2010*, LNCS, vol. 6110, pp. 24–43, 2010.

[25] Smart, N.P. and Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes, *Public Key Cryptography–PKC 2010, LNCS*, vol. 6056, pp. 420–443, 2010.

[26] Smart, N.P. and Vercauteren, F.: Fully homomorphic simd operations, *Designs, Codes and Cryptography*. vol. 71, no. 1, pp. 57–81, 2014.

[27] Agrawal, R. and Srikant, R.: Fast algorithms for mining association rules, *Proc. The International Conference on Very Large Data Bases (VLDB 1994)*, pp. 487–499, 1994.

[28] Arita, S. and Nakasato, S.: Fully homomorphic encryption for point numbers, Cryptology ePrint Archive: Report 2016/402, Cryptology ePrint Archive (online), available from <https://eprint.iacr.org/2016/402> (accessed 2017-01-20).

[29] Agrawal, R., Imielin´ski, T. and Swami, A.: Mining association rules between sets of items in large databases, *Proc. the 1993 ACM SIGMOD*, pp. 207–216, 1993.

[30] Halevi, S. and Shoup, V.: Algorithms in helib, *International Cryptology Conference, LNCS*, vol. 8616, pp. 554–571, 2014.

[31] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping, *Proc. The 3rd Innovations in Theoretical Computer Science Conference (ITCS 2012)*, pp. 309–325, 2012.

[32] 今林広樹, 石巻優, 馬屋原昂ほか:　完全準同型暗号による安全頻出パターンマイニング計算量効率化, 情報処理学会論文誌:データベース(TOD), TOD73 号, 2017[21].

[21] 巻数，号数，ページ数は 2017 年 1 月 20 日現在未定.

研究業績

【主著】

1. 国際ワークショップ（フルペーパ）

   Hiroki Imabayashi, Yu Ishimaki, Akira Umayabara, Hiroki Sato and Hayato Yamana: "Secure Frequent Pattern Mining by Fully Homomorphic Encryption with Ciphertext Packing", the 11th DPM International Workshop on Data Privacy Management (DPM), LNCS, vol. 9963, pp. 181-195 (2016.9).

2. 国際会議（ポスター）

   Hiroki Imabayashi, Yu Ishimaki, Akira Umayabara and Hayato Yamana: "Fast and Space-Efficient Secure Frequent Pattern Mining by FHE", Proc. of IEEE International Conference on Big Data 2016 (2016.12).

3. 国内ジャーナル

   今林広樹，石巻優，馬屋原昂，佐藤宏樹，山名早人："完全準同型暗号による安全頻出パターンマイニンク計算量効率化"，情報処理学会論文誌データベース（TOD）（73 号にて採録）．

4. 国内ワークショップ（ペーパ）

   今林広樹，石巻優，馬屋原昂，佐藤宏樹，山名早人："ストリーム処理による安全頻出パターンマイニングの高速化"，データ工学と情報マネジメントに関するフォーラム（DEIM）（2017.3 発表予定）．

【共著】

1. 国内フォーラム（ペーパ）

   安村慶子，石巻優，今林広樹，山名早人:"A Survey on Attribute-based Encryption and its Application in Cloud and Mobile Environment"，第 15 回情報科学フォーラム（FIT），L-009 (2016.9).

2. 国内フォーラム（ペーパ）

   佐藤宏樹，馬屋原昂，石巻優，今林広樹，山名　早人："完全準同型暗号のデータマイニングへの利用に関する研究動向"，第 15 回情報科学フォーラム（FIT），F-002 (2016.9).

3. 国際会議（ポスター）

Yu Ishimaki, Hiroki Imabayashi, Kana Shimizu and Hayato Yamana: "Privacy-Preserving String Search for Genome Sequences with FHE bootstrapping optimization", Proc. of IEEE International Conference on Big Data 2016 (2016.12).

4. 国内ワークショップ（ペーパ）

馬屋原昂，今林広樹，山名早人: "NUMA　マシンにおける完全準同型暗号による安全頻出パターンマイニングの高速化", データ工学と情報マネジメントに関するフォーラム（DEIM）（2017.3 発表予定）.

付録： 本研究で用いたデータセット及び実験結果

本研究で用いたデータセット, 及び実験結果については添付の DVD に収録した. ファイルの詳細は README.txt に記した.