情報処理学会論文誌　データベース　Vol.10　No.1　1–12 (Mar. 2017)

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

今林 広樹[1,a)]　石巻 優[1]　馬屋原 昂[1]　佐藤 宏樹[1]　山名 早人[1]

受付日 2016年9月2日, 採録日 2017年1月10日

概要：医薬品や遺伝子などの機密性の高いデータに対する各種処理をクラウドなどの第三者のサーバ上で行う場合，第三者のサーバからの機密情報漏洩が懸念される．解決策として，機密情報そのものではなく匿名化したデータを第三者のサーバに保存し各種処理を行う方法が考えられるが，医療分野など，処理の正確性が求められる分野では匿名化を採用することが困難である．この問題を解決するため，本稿では，完全準同型暗号（FHE: Fully Homomorphic Encryption）を用いてデータを秘匿した状態で各種処理を行うことを考える．そして，各種処理の対象として頻出パターンマイニングを取り上げる．FHE を用いた各種処理を行ううえでの問題は，膨大な時間・空間計算量を要する点である．FHE の頻出パターンマイニング手法への適用例としては，Apriori アルゴリズムを対象とした Liu らの P3CC があるが，やはり膨大な時間・空間計算量を要する．これに対して本稿では，1) 暗号文パッキングによる暗号文数の削減，および 2) 暗号文キャッシングによるサポート値計算の高速化によって，時間・空間計算量を削減する手法を提案する．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，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．

キーワード：頻出パターンマイニング，秘匿計算，完全準同型暗号，暗号文パッキング

Streamline Computation of Secure Frequent Pattern Mining by Fully Homomorphic Encryption

Hiroki Imabayashi[1,a)]　Yu Ishimaki[1]　Akira Umayabara[1]　Hiroki Sato[1]　Hayato Yamana[1]

Received: September 2, 2016, Accepted: January 10, 2017

Abstract: Private information disclosure from a third-party is concerned as a security risk, while processing highly confidential data such as medicines and genes on the third-party as a cloud server. Although there is a solution that stores and processes the data anonymized from private information on the third-party's server, it is difficult to apply anonymizing techniques to application fields such as a medical field which requires accurate processing. To address this issue, we will perform the processing with concealed data by Fully Homomorphic Encryption (FHE). In addition, we will adopt frequent pattern mining as a processing object. The problem of the processing with FHE is both huge time complexity and space complexity. P3CC by Liu et al. is the first proposed application of FHE for frequent pattern mining, which targets Apriori algorithm and has these particular problems. In this paper, we propose an Apriori based mining method with smaller time and space complexities than P3CC, by adopting two techniques: 1) ciphertext packing that reduces the number of ciphertexts, and 2) ciphertext caching that enables to speed up the support-counting. Our experimental evaluation shows that the proposed scheme runs 430 times faster than P3CC, and uses 94.7% less memory with 10,000 transactions data.

Keywords: frequent pattern mining, secure computing, Fully Homomorphic Encryption, ciphertext packing

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[a)] imabayashi@yama.info.waseda.ac.jp

本稿は，第 11 回 Data Privacy Management（DPM）国際ワークショップ [12] で発表した原稿（著作権保持者は著者）について，実験結果を加え再度評価したものである．

1. はじめに

近年のクラウドの普及にともない，クラウドへのデータ蓄積やクラウド上での各種処理が増加し，機密データ漏洩によるセキュリティリスクへの懸念が高まっている．組織は，顧客情報や行動履歴，医薬品や特許といった機密情報を保持しており，それらデータをクラウド上で解析する際には，被解析データそのものに加え，解析の途中結果や最終結果もまた，センシティブな情報として漏洩リスクを持つ．このような機密情報は，悪意を持った人の盗聴対象となるため，クラウドサーバ上でデータを秘匿したまま解析する仕組みが必要となってきている．

本稿では，委託計算を前提とした，プライバシ保護データマイニング（PPDM: Privacy Preserving Data Mining）を考える [8]．PPDM の先行研究は，1) 入力プライバシ保護，2) 出力プライバシ保護，3) 暗号化システムの 3 つのアプローチに分類することができる．入力プライバシは，サーバ側で解析する前のデータに対して，クライアント側で抽象化，ノイズ付加，ランダム化などの操作を行うことで，入力データ中のセンシティブな情報を保護する [7], [18], [21], [23]．その一方で，出力プライバシは，サーバ側の出力である解析結果に対し，ノイズ付加や摂動化により，センシティブな情報を保護する [4], [5]．3 つ目の暗号化システムによるアプローチでは，暗号化されたデータに対して，マイニングアルゴリズムを適用することにより，プライバシ保護を行う [13], [14], [16]．各アプローチには，利点・欠点が存在する．入力プライバシと出力プライバシのアプローチは，暗号化システムによるアプローチと比較して，短時間で計算可能だが，入力と出力両方のプライバシを同時に満たすことができない．また，これらのアプローチはデータを曖昧にすることで秘匿性を高めるため，マイニング結果も曖昧になりやすい．一方で，暗号化システムを用いた場合は，膨大な計算時間を要するが，サーバ上での途中計算結果まで含めたプライバシ保護に加え，マイニングの数値的正確性を保証する．上記を考慮した結果，本研究では，完全なプライバシ保護を目的とするため，暗号化システムを選択した．

完全準同型暗号（FHE: Fully Homomorphic Encryption）は，そのような暗号システムの 1 つであり，暗号文に対して任意の回数の加算・乗算を行うことができる．Gentry [9] によって FHE が提案され，その後，統計計算 [17]，機械学習アルゴリズム [10], [15]，頻出パターンマイニング [14], [16] といったデータマイニング研究に広く採用されている．

しかし，これらの FHE によるアプリケーションは，時間・空間計算量を大きく消費することが課題となっている．時間計算量に関しては，大きな暗号文サイズと暗号文数にともなう演算数の増加により，データマイニングの実行に膨大な時間を要する．また空間計算量に関しては，暗号文サイズや暗号文数により，メモリやストレージといった計算資源を膨大に消費する．FHE を頻出パターンマイニングに応用した例として，Liu ら [16] の Privacy Preserving Protocol for Counting Candidate（P3CC）がある．P3CC は，Dijk ら [22] による暗号文を整数で表現する FHE（DGHV 方式）上に実装されたものである．P3CC は，アイテム集合を要素として持つトランザクションデータベースを，アイテム・トランザクションのバイナリ行列に変換し，その後，行列の各要素を暗号化している．つまり，暗号文上の計算は要素ごとの加算・乗算として，独立して演算を行うことになるため，生成される暗号文数は行列サイズに線形比例し，膨大なメモリ・ストレージ使用量，通信コスト，暗号文上の計算コストをもたらす．

これらの問題を解決するためには，暗号文数と暗号文上の計算の 2 点を少なく抑えながら，マイニングを実行することが必要である．これを実現するために，本研究では，頻出パターンマイニングへの暗号文パッキングの適用と，暗号文キャッシングアルゴリズムの構築を行った．はじめに暗号文パッキングに関しては，Brakerski ら [6] による暗号文を多項式で表現する FHE（BGV 方式）上に，Smart と Vercauteren [19], [20] による多項式上での暗号文パッキング手法を採用した．パッキングは，複数の多項式に対して中国剰余定理（CRT: Chinese Reminder Theorem）を用い，1 つの多項式として表すことで実現される．この暗号文パッキング手法により，暗号文を複数の平文を格納した暗号文配列として表現（CRT 表現暗号文）でき，その暗号文配列どうしの演算は，配列内要素ごとの演算として並列化される．これにより，P3CC と比較して，より少ない暗号文数と演算回数でマイニングが実行できる．続いて，暗号文キャッシングに関しては，各パターンのサポート値を計算する際に計算途中結果をキャッシュしておくことで，同一パターンのサポート値の再計算を回避するアルゴリズムを構築した．これにより，暗号文上での乗算回数が大きく削減され，サポート値計算の高速化が可能となる．

本研究の貢献は，次の 3 つである．1 つ目に，FHE による頻出パターンマイニングに暗号文パッキングを初めて適用し実験評価した点である．2 つ目に，暗号文パッキング適用により，生成される暗号文数と，それにともなう計算量を削減した点である．具体的には，トランザクション数（バイナリ行列の行数）を N とし，スロット数（暗号文に詰め込める要素数）を l とすると，各列の全要素を詰め込むのに必要な暗号文数，およびそれにともなう演算回数は，ともに N から $\lceil N/l \rceil$ に削減される．3 つ目に，計算途中結果をキャッシング・再利用することで，暗号文上でのパターンサポート値の計算を高速化した点である．

本稿の構成は，次に示すとおりである．まず 2 章で関連研究を述べ，3 章で FHE による頻出パターンマイニングの計算量効率化手法について詳細に説明する．4 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．5 章でまとめと今後の課題を述べる．

2. 関連研究

本章では，暗号化システムを用いたデータマイニングについての関連研究を紹介する．またその中で，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について，Liu らの P3CC を具体的に説明する．

暗号化システムを用いたデータマイニングの研究 [13], [14], [16] 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）の大きく 2 つのアプローチに分類できる．MPC によるアプローチとして，Kantarcioglu ら [13] は，マルチパーティ間でプライバシを保護しながら，分散データベースに対して頻出パターンマイニングを行うためのアルゴリズムを提案した．HE によるアプローチでは，Kaosar ら [14] は，2-パーティでの相関ルールマイニングにおいて，FHE を用いた数値比較手法を提案し，MPC によるアプローチ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適していないことを示した．

本研究に最も関連する研究としては，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）があげられる．Liu らは Dijk ら [22] による，整数で暗号文を表現する FHE を用いており，その性質により，アイテムとトランザクションのバイナリ行列の各要素を暗号化している．それゆえ，P3CC の問題点は，膨大な時間・空間計算量がかかること，また，この計算量がトランザクションサイズに対し線形的に増加していくことである．同論文での実験評価では，ミニマムサポートを 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 4 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．

3. FHE による安全委託型頻出パターンマイニングの計算量効率化

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の課題である膨大な時間・空間計算量について，計算量を削減する手法を提案する．3.1 節でプロトコルの概要を説明する．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．3.3 節では，効率化するための手法として，1) 暗号文パッキングの適用，2) 暗号文キャッシングアルゴリズムの 2 点を説明する．

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，およ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 ら [3] 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，次のとおりである．

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，復号することなく，暗号化データに対しマイニングを行い，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 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，Arita ら [1] 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，3.2.2 項で説明する．

3.2 プロトコルに用いる要素技術

本節では，FHE による頻出パターンマイニングプロトコルを構築し，さらに計算量を効率化するために，必要な 4 つの要素技術として，1) Apriori アルゴリズム，2) P3CC 方式，3) 多項式 CRT パッキング，4) 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 ら [3] 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，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（**図 1** (a)）．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（図 1 (b)）．以下に，頻出パターンの定義 [2]，および Apriori の手続きを示す [3]．

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
Fig. 1　Item-transaction database.

定義 1．頻出パターン [2]

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持つ．つまり，t は，i_k を持つなら $t[k]$=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 の持つすべてのアイテム 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）．たとえば，$L_1 = \{\{i_1\}, \{i_2\}, \{i_3\}\}$ 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）．たとえば，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，パターンの大きさごと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ごとの AND 演算を実行したのち，すべ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（3.2.4 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 $\boldsymbol{v}'_{i_6}$ を要素ごとに 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．

アルゴリズム 1. Apriori(I,TDB, minSup) [3]
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　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 [22]．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 2** (a) 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，暗号化された値どうしの比較ができない[*1]ため，P3CC 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．暗号文で表現されるのは，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（図 2 (a) 参照），および countSupport により得られるサポート値のみである．また，暗号文はすべて多倍長整数で表現される．具体的には次の手続きに従う．ここでは，アルゴリズム 1 の手続きに沿って説明する．

[*1] 数値をバイナリ化するか，もしくは Arita ら [1] 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・空間計算量がともに爆発的に増大する．

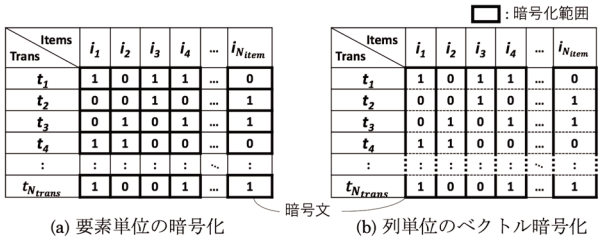


図 2　バイナリ行列の暗号化方法

Fig. 2　Encryption strategy of item-transaction database.

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，および各要素が多倍長整数値の暗号化バイナリ行列（図 2 (a)）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* に出現するすべて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 2 (a)）の列（要素はすべて暗号文）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．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）．

④ クライアントは，L_1 から generateFrequentCandidate-Patterns（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）．

⑤ 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 2 (a)）の列（要素はすべて暗号文）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）．

⑥ 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）．

⑦ パターンの大きさをインクリメントし，④，⑤，⑥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．なお，セキュリティ分析は本稿の目的ではないため，詳細な説明は割愛する．詳細は，Liu らの研究 [16] を参照されたい．

3.2.3　多項式 CRT パッキング

Smart ら [19], [20] 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）を要素に持つ配列を生成．

② 配列から整数の係数を持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[*2]，$\Phi_m(X)$ は l 個の d 次多項式 $F_1(X), \ldots F_l(X)$ に因数分解できるように設定される（$\Phi_m(X) = F_1(X), \ldots F_l(X)$，ただし $n = dl$）．ここで CRT により，連立合同方程式 $f(X)\ mod\ F_i(X) = \alpha_i$（$1 \leq i \leq l$，$\alpha_i$ は i 番目に詰め込む平文）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，$(a_0, a_1, \ldots, a_{n'-1}, a_{n'})$（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，平文を詰め込める空間という概念からスロットと呼ばれ，l は $f(X)$ が持つスロット数に一致する．つまり，1 つの多項式が詰め込める平文数は 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持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

[*2] 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．

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 表現暗号文どうしの演算は，平文を要素に持つ配列どうしの要素間の演算として並列化される．CRT パッキングによる平文多項式の生成に関する数学的な背景は，Smart らの研究 [19], [20] を参照されたい．

3.2.4　CRT 表現暗号文のスロット合計値計算

Halevi ら [11] 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 $\boldsymbol{v} = (v_1, v_2, \ldots, v_l)$（$1 \leq i \leq l$，スロット v_i には i 番目の平文が含まれる．3.2.3 項参照）と見なすことがでる．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，および 2) CRT 表現暗号文 $\boldsymbol{u}$ の多項式 $f_{\boldsymbol{u}}(\mathrm{X})$ 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見なしたうえで，巡回シフトなど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．アルゴリズムと実装の詳細な説明は，Halevi らの研究 [11] を参照されたい．

アルゴリズム 2. TotalSums(v) [11]

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** $(j \leftarrow k-2;\ j \geq 0;\ j \leftarrow j-1)$ **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** $\boldsymbol{u}$

* $\mathrm{rotate}(\boldsymbol{u}, e)$: $f_{\boldsymbol{u}}(X^e)\ mod\ \Phi_m(X)$ 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　プロトコル効率化手法の提案

本節では，3.2 節で説明した 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，時間・空間計算量を削減する手法を提案する．まず準備として，図 2 のような，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，3.2.1 項定義 1 では s），CRT 表現暗号文が持つスロット数を l とする．

P3CC [16] ではバイナリ行列の各要素について，独立に暗号化（図 2 (a) 太線）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，およ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，Apriori 実行に大きな時間・空間計算量が必要になる．

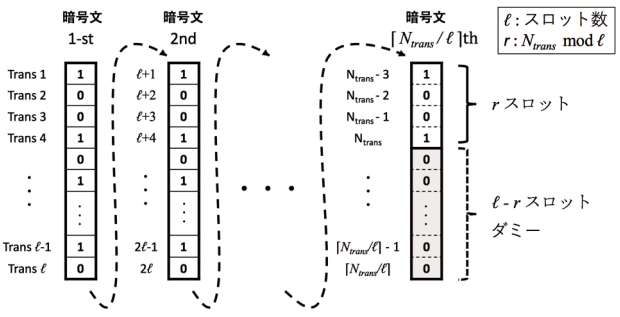


図 3　アイテム列要素の暗号文パッキング
Fig. 3　Ciphertext-packing of item-column-components.

本稿では，FHE を用いた Apriori の実行に必要な暗号文数と演算回数を削減するための手法として，1) 暗号文パッキングと 2) 暗号文キャッシングを説明する．

3.3.1　暗号文パッキングの適用

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 [19], [20] を適用する．まず FHE の方式を，整数ベースの DGHV 方式 [22] から，多項式ベースの BGV 方式 [6] に変換する．これにより，FHE 方式は，複数の平文を多項式 CRT 表現された暗号文に詰め込むことができるようになる（3.2.3 項参照）．

さらに，Apriori の実行に必要な乗算回数を減らすため，バイナリ行列を列単位（アイテム単位）でパッキングする（図 2 (b)）．1 つの暗号文に詰め込めるのは 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詰め込むのに必要な暗号文数は，$\lceil N_{trans}/l \rceil$ 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 とともに 1 つの暗号文に詰め込むことで実現できる（**図 3**）．

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，それにともない，メモリ使用量も同量削減される．2 点目に，すべ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 $\sum_{k=1}^{t} p_k \lceil N_{trans}/l \rceil^{k-1}$（$p_k$，$N_{trans}^{k-1}$，$t$ は上述のとおり）に削減されるため，実行時間を短縮できる．

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 個の平文を詰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 1 度変換されるためである（3.2.3 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（3.2.3 項参照）およ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詰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，1 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．

3.3.2　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 $c = \{i_{f(1)}, i_{f(2)}, \ldots, i_{f(k)}\}$（ただし，$f(j)$（$1 \leq j \leq k$）は $1 \leq f(j) \leq N_{items}$ 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 2 (b) 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（3.2.4 項の配列 $\boldsymbol{v}$ と区別する），「$\circ$」を配列要素ごとの乗算とすると，$\circ_{j=1}^{k} \boldsymbol{v}'_{i_{f(j)}}$ 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（3.2.1 項，3.2.4 項参照）．たとえば「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．たとえば，キャッシュなしの場合は，$\boldsymbol{X} = \boldsymbol{v}'_{i_1} \circ \boldsymbol{v}'_{i_2} \circ \boldsymbol{v}'_{i_3} \circ \boldsymbol{v}'_{i_4}$ と 3 回の演算 $\circ$ を用いて $\boldsymbol{X}$ を得ていたところを（図 **4** (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 $\circ$ で $\boldsymbol{X}$ を得ることができるようになる（図 4 (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 $\circ$ を 1 回に抑えることができる．

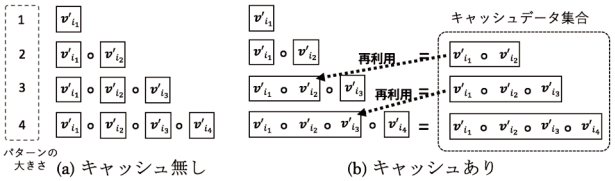


図 4　キャッシングによるサポート値計算
Fig. 4　Support calculation with caching technique.

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，キャッシュなしでは $\sum_{k=1}^{t} p_k \lceil N_{trans}/l \rceil^{k-1}$ 回の演算 $\circ$ 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，3.2.4 項で説明した TotalSums を用いている．

アルゴリズム 3. CountSupportWithCache(ET DB, C, CD)
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.　実験評価

本章では，FHE を用いた Apriori 実装に対し，暗号文パッキング手法と暗号文キャッシング手法をそれぞれ適用し，実験により有用性を評価する．さらに，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（3.2.2 項手続き ②–⑦）」とした．

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[*3] 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 $\{\mathrm{T}, \mathrm{I}, \mathrm{N}, \mathrm{D}, \mathrm{L}\}$ 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持つ

[*3] http://www.philippe-fournier-viger.com/spmf/

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 [*4]（FIMI）に公開されている chess データセットを用いた．この公開データセットを用いた実験評価は，4.4 節で行う．

また，本実験は BGV 方式 [6] をサポートした公開 FHE ライブラリである HElib [*5]，多倍長整数演算を行うためのライブラリである GMP [*6]，整数多項式を扱うためのライブラリである NTL [*7] 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 [*8] 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 $l = 20$）．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．

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

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（暗号化なし）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．

4.1　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，およ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 実行時間およ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）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，およ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．

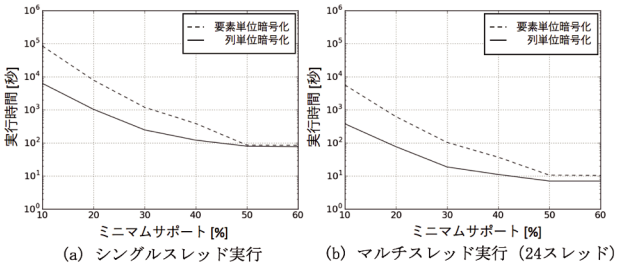


図 5　暗号文パッキング適用・非適用方式比較
Fig. 5　Comparison of packing and non-packing schemes.

図 5 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 5 (a) はシングルスレッド実装，図 5 (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（445 GB から 33.8 GB）．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，およ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，およ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．

4.2　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，およ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 実行時間およ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 6 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 6 (a) はシングルスレッド実装，図 6 (b) はマルチスレッド実装を示す．な

[*4] http://fimi.ua.ac.be/data/
[*5] http://shaih.github.io/HElib/index.html
[*6] https://gmplib.org/
[*7] http://www.shoup.net/ntl/
[*8] 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，それゆえに任意回数の演算が可能となる．

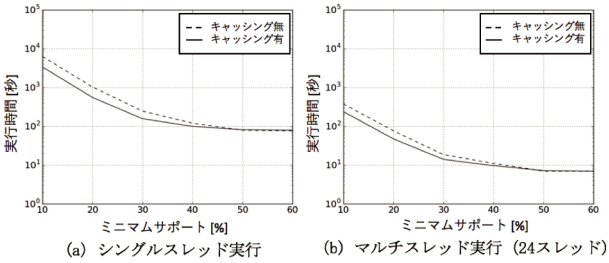


図 6　暗号文キャッシング適用・非適用方式比較
Fig. 6　Comparison of caching and non-caching schemes.

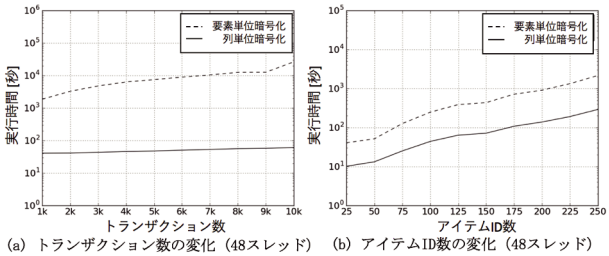


図 7　データサイズについての計算量変化
Fig. 7　Complexities by varing num. of transactions/items.

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（28.9 GB から 32.4 GB），マルチスレッドで 53.1%増加した（33.8 GB から 51.7 GB）．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．

4.3　データサイズとセキュリティに対する計算量評価

本節では，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やセキュリティ概念（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ため，次の 3 つのケースについて，Apriori の計算量評価を行う．

- データセットトランザクション数を変化させた場合
- データセット総アイテム ID 数を変化させた場合
- α-pattern uncertainty でダミーセットを追加した場合

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 1 k から 10 k まで 1 k 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 7** (a) に示す．D = 10 k 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（536 GB から 34.5 GB）．

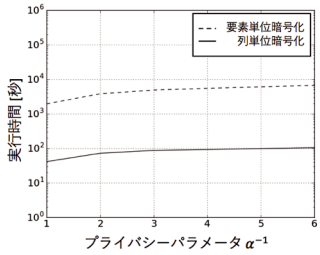


図 8　プライバシパラメータについての計算量変化
Fig. 8　Complexities by varing privacy parameter.

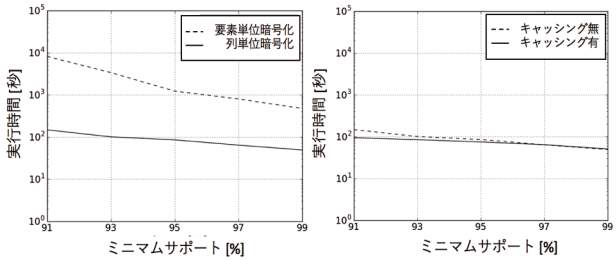
続いて，データセットの総アイテム ID 数を変化させた場合について，要素単位暗号化方式と列単位暗号化方式の各々にキャッシング手法を適用し，それぞれの実行時間を比較する．まず，データセットは T5I6N25D100L1k とし，パラメータ T を 5 から 50 まで 5 ずつ，N を 25 から 250 まで 25 ずつ同時に増やしながら変化させていく．ミニマムサポートは 30%，サーバ側を 48 スレッドに設定し，各データセットについて実行時間を測定した結果を，図 7 (b) に示す．N = 250 のとき，列単位暗号化方式（実線）は要素単位暗号化方式（点線）と比較して，7.42 倍高速で，92.3%のメモリが削減された（473 GB から 36.5 GB）．

最後に，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（ダミーセットなし）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 8** 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（177 GB から 33.8 GB）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，およ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，およ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．

4.4　公開データセットを用いた評価

本節では，P3CC [16] の評価で用いられている chess データセットを用い，4.1 節での暗号文パッキング手法，およ



(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 9　パッキング・キャッシング手法適用

Fig. 9　Adaption of packing and caching techniques.

び 4.2 節での暗号文キャッシ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（1 TB 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[*9]．

図 9 (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，96.0%削減した（687 GB から 27.2 GB）．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，およ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，およ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 9 (b) に列単位暗号化方式（キャッシング非適用，点線，図 9 (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 0.364%増加した（27.2 GB から 27.3 GB）．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．

パッキングとキャッシングをともに適用することで，ミニマムサポート 91%において，どちらも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，メモリ使用量については 96.0%削減した（687 GB から 27.3 GB）．

[*9] 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，1 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 1 TB 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・メモリなどの具体的な数値は不明瞭である．

表 1　各方式のメモリ使用量

Table 1　Memory usage for each scheme.

ミニマムサポート	91%	93%	95%	97%	99%
要素単位暗号化方式	687GB	678GB	678GB	678GB	678GB
列単位暗号化方式	27.2GB	22.3GB	18.9GB	18.9GB	18.9GB
列単位暗号化・キャッシング方式	27.3GB	22.4GB	19.3GB	19.1GB	19.0GB

表 1 に，各方式について，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．

5. おわりに

本稿では，FHE による頻出パターンマイニングの時間・空間計算量を削減することを可能とする，暗号文パッキングの適用手法，および暗号文キャッシング手法を提案した．評価実験から，提案手法は，従来の P3CC による Apriori の実行時間とメモリ使用量を大きく削減することができることを示した．さらにデータセットサイズが大きい場合や，セキュリティを考慮しダミーセットを加えた場合には，より大幅に実行時間とメモリ使用量を削減した．特に，トランザクション数を 10,000 としたとき，従来のプロトコルと比較して，430 倍の高速化と 94.7%のメモリ使用量削減を達成した．

今後の課題として，暗号文キャッシングによるメモリ使用量の増大があげられる．このメモリ領域の圧迫を防ぐためには，不要な暗号文キャッシュを効率的にプルーニングするアルゴリズムの構築が必要である．また，マイニングプロトコルのさらなる高速化のために，データのストリーム処理や，関数実行のスケジューリングを工夫することが必要となる．

謝辞　本研究は，科学技術振興機構（JST）CREST の支援を受けたものである．

参考文献

[1] 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 2016-09-18).

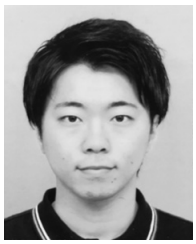
[2] Agrawal, R., Imielin´ski, T. and Swami, A.: Mining association rules between sets of items in large databases, *Proc. 1993 ACM SIGMOD*, pp.207–216 (1993).

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

[4] 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).

[5] 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).
[6] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping, *Proc. 3rd Innovations in Theoretical Computer Science Conference* (*ITCS 2012*), pp.309–325 (2012).
[7] 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).
[8] Gellman, R.: Privacy in the clouds: Risks to privacy and confidentiality from cloud computing, *Proc. World Privacy Forum* (2009).
[9] Gentry, C.: Fully homomorphic encryption using ideal lattices, *Proc. Annual ACM Symposium on Theory of Computing* (*STOC 2009*), pp.169–178 (2009).
[10] 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).
[11] Halevi, S. and Shoup, V.: Algorithms in helib, *International Cryptology Conference*, LNCS, Vol.8616, pp.554–571 (2014).
[12] Imabayashi, H., Ishimaki, Y., Umayabara, A., et al.: Secure frequent pattern mining by fully homomorphic encryption with ciphertext packing, *International workshop on Data Privacy Management* (*DPM 2016*), LNCS, Vol.9963, pp.181–195 (2016).
[13] Kantarcioglu, M. and Chris, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data, *IEEE Trans. Knowledge and Data Engineering* (*TKDE 2004*), Vol.16, pp.1026–1037 (2004).
[14] 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).
[15] Khedr, A., Gulak, G. and Vaikuntanathan, V.: Shield: Scalable homomorphic implementation of encrypted data-classifiers, *IEEE Trans. Comput.*, Vol.65, No.9, pp.2848–2858 (2015).
[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] 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).
[18] 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).
[19] 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).
[20] Smart, N.P. and Vercauteren, F.: Fully homomorphic simd operations, *Designs, Codes and Cryptography*, Vol.71, No.1, pp.57–81 (2014).
[21] Tai, C.H., Yu, P.S. and Chen, M.S.: k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining, *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD 2010*), pp.473–482 (2010).
[22] 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).
[23] 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).



今林 広樹　（学生会員）

2015 年早稲田大学先進理工学部生命医科学科卒業，同大学大学院基幹理工学研究科情報理工・情報通信専攻在学中．暗号技術を用いた秘匿計算の高速化の研究に従事．

石巻 優　（学生会員）

2015 年早稲田大学基幹理工学部情報理工学科卒業，同大学大学院基幹理工学研究科情報理工・情報通信専攻在学中．暗号技術を用いた秘匿計算の高速化の研究に従事．

馬屋原 昂　（学生会員）

2016 年早稲田大学基幹理工学部情報理工学科総代卒業，同大学大学院基幹理工学研究科情報理工・情報通信専攻在学中．暗号技術を用いた秘匿計算の高速化を目的とした HPC の研究に従事．

佐藤 宏樹　（学生会員）

早稲田大学基幹理工学部情報理工学科在学中．暗号技術を用いた秘匿計算の高速化の研究に従事．

山名 早人　（正会員）

1987 年早稲田大学理工学部電子通信学科卒業．1993 年同大学大学院博士後期課程修了．博士（工学）．1993〜2000 年電子技術総合研究所．2000 年早稲田大学助教授．2005 年同教授，現在に至る．2015 年本会理事．大規模データ解析研究に従事．本会シニア会員．

（担当編集委員　荒牧英治）