

完全準同型暗号を用いた 頻出パターンマイニングの実装手法による比較

種村 真由子¹ 小口 正人¹

概要：

ビッグデータなど、大規模なデータの収集・分析による利活用がビジネスなどの分野で進んでいる。大規模なデータを扱う計算には、処理能力の高い計算機システムが必要となるため、外部のサーバに処理を委託することが現実的である。しかし、特に個人情報等の機密データ処理の委託に関しては、情報の漏えい対策が重要となる。プライバシー保護のためには、データの匿名化加工など様々な方法があるが、本研究では、データの機密性保護の方法として、暗号化した状態で計算が可能な完全準同型暗号 (FHE) を使用する。先行研究として、FHE を Apriori アルゴリズムによる頻出パターンマイニングを行うクライアント・サーバ型のシステムについて適用した P3CC がある。このシステムについて、Apriori アルゴリズムを FP-growth アルゴリズムに変更したものを実装し、Apriori アルゴリズムを用いた既存手法との比較を行う。

A Comparison of Implementation Methods of Frequent Pattern Mining Using Fully Homomorphic Encryption

MAYUKO TANEMURA¹ MASATO OGUCHI¹

1. はじめに

ビッグデータ等の大規模なデータを収集し分析することにより、ビジネスなどの分野で利活用が進んでいる。サイズの大きなデータを扱う計算を行うには、処理能力の高い計算機システムが必要となる。システムを用意する事が困難な場合には、クラウド等の外部の計算資源を利用する方法がある。その場合、個人情報や医療データ等、機密性を確保する必要があるデータの委託処理に関しては、送信する情報が漏えいしないよう対策を十分に行う必要がある。機密情報を外部委託して処理する場合においては、完全準同型暗号 (FHE) を用いることで、委託先サーバに平文データを見せずに加算や乗算の計算処理を行うことが可能になる。したがって、信頼できるサーバに対してでなくとも、データの委託処理を行えるという期待がある。FHE の応用例として、Apriori アルゴリズムによるトランザクションデータの頻出パターンマイニングを行った Liu らの P3CC[1] や、その高速化 [2][3] や、分散処理化 [4] につい

ても研究が行われている。P3CC とその高速化についての先行研究は、4 章で説明する。本研究では、Apriori アルゴリズムで処理している部分を FP-growth に変更した頻出パターンマイニングのシステムの実装を行い、先行研究との比較を行う。

2. 完全準同型暗号

2.1 完全準同型暗号の概要

完全準同型暗号 (Fully Homomorphic Encryption, FHE) は、加法準同型性 (式 1) と乗法準同型性 (式 2) の特徴をあわせ持った暗号化手法である。

加法準同型性、乗法準同型性

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m + n) \quad (1)$$

$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) \quad (2)$$

公開鍵暗号方式の機能を持ち、暗号化した状態での暗号文同士の加算、乗算が成立する。すなわち、暗号文同士の

¹ お茶の水女子大学 人間文化創生科学研究科

計算により、暗号化される前の平文を操作することが可能である。したがって、FHE を用いることで、委託先サーバに平文データを見せずに計算処理を委託できるという期待がある。

完全準同型暗号は、2009 年に Gentry が Lattice ベースの実現手法を提案した [5]。各暗号文には、暗号の解読不可能性を高めるため、ランダムなノイズが付加されている。

問題点としては、一般に暗号文と鍵のデータサイズが大きくなることにより処理の計算量が膨大になることと、ノイズの値が暗号文同士の計算を行うたびに増加し、閾値を超えると復号が不可能となることが挙げられる。ノイズの値は、特に乗算を行った際に大きく増加する。bootstrapping という処理を行うことで、暗号文のノイズを refresh することが可能であるが、この処理も計算量が大きいという問題がある。

2.2 Leveled FHE

bootstrapping を使用しない完全準同型暗号の実装として、Leveled FHE がある。Brakerski らによって提唱された [6]。Leveled FHE は、決まった深さ L の論理回路の結果を評価することができる、完全準同型暗号の構造の一つである。事前に与えたレベルに対して、計算の論理回路の深さが小さければ、bootstrapping を行う必要がない。本研究では、bootstrapping を使用しない、こちらの Leveled FHE を使用している。

3. 頻出パターンマイニング

頻出パターンマイニングは、データマイニングの一種であり、大量のデータの中から、相関ルールの抽出を目的とした手法である。例として、トランザクションデータを対象とした。バスケットデータ分析がある。

本研究で扱う頻出パターンマイニングでは、指定したアイテムのうち、各トランザクションがどのアイテムを含んでいるかを、バイナリ行列で表したデータを対象に行う。頻出アイテムセットの判定は、サポート値が指定した値以上であるか否かで行う。サポート値は、全体のトランザクション数に対する、あるアイテムセットが含まれるトランザクション数の割合で表される。代表的なアルゴリズムとして、Apriori と FP-growth がある。各アルゴリズムについての概要を以下に述べる。

3.1 Apriori

Apriori は、アイテム長 1 のアイテムセットから順にサポート値をミニマムサポート値と比較し、頻出アイテムセットを列挙していく、幅優先探索型のアルゴリズムである。4 章で述べる先行研究で使用されている。アイテムの種類が少ない場合でもアイテムの組み合わせの種類は膨大

になり得るため、枝刈りを行うことにより計算量を削減している。あるアイテム長 n のアイテムセットのサポート値が、事前に設定した最小サポート値未満の場合は、そのアイテムセットを含むアイテム長 $n+1$ のパターンも頻出でないと判断し、その後の探索を行わないようにする。実装は比較的容易である。

3.2 FP-growth

FP-growth は、Apriori に対して、アイテムが含まれるものを優先的に探す、深さ優先探索で頻出アイテムセットの抽出を行うアルゴリズムである。

頻出アイテムセットの探索に木構造のデータを用いるという点で、Apriori と大きく異なる。データベースを走査し、トランザクションデータを FP-tree という prefix tree の木構造に格納、それを走査することで頻出パターンを求める。また、Apriori と異なり頻出アイテムセットの候補を列挙しないという特徴がある。データサイズやデータの特性にも依存するが、頻出アイテムの列挙がボトルネックになる Apriori と比較して、探索を効率化できるという期待がある。実装は Apriori よりも複雑である。

FP-tree は、以下のように定義されている [7]。

- (1) 「null」とラベリングされたルートノード、ルートの子として item prefix subtree (アイテムの接頭辞部分木) の組と、frequent-item header table を持つ。
- (2) 各 item prefix subtree は item-name, count, node-link を保持する。count は、そのノードに至る部分パスによって表されるトランザクション数を示す。node-link は FP-tree 内に存在する同名アイテムをもつ次のノードへのリンクであり、存在しない場合は null となる。
- (3) frequent-item header table の各エントリは item-name と head of node-link を保持する。head of node-link は、前項 (2) で述べた node-link の先頭のノードを示すものである。

図 1 に、FP-tree の簡易な例を示す。ただし、図中の長方形は各ノードを示し、実線はノード間の接続、点線矢印は node-link の示す先のノードを指しているものとする。各ノードに書かれている A, B, C は item-name の例、数字は count である。

FP-tree を構築するには、トランザクションデータと、最小サポート値が入力として必要である。

構築手順の概要を以下に示す。

- (1) データベースを 1 回走査し、頻出アイテムのセットと各アイテムのサポート値を計算する。アイテムのサポート値が最小サポート以上の場合に頻出であるとする。

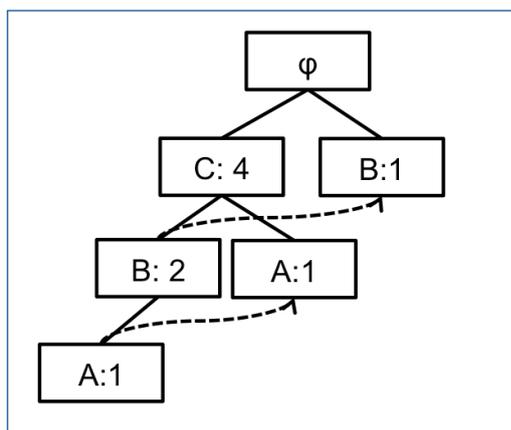


図1 FP-tree の簡易な例

- (2) 頻出アイテムのセットをサポート値の降順でソートする。
- (3) FP-tree のルートに null とする。
- (4) 各トランザクション内から頻出アイテムを取り出し、それらを(2)の順に従ってソートする。
- (5) ソート済みのトランザクションを作成済みの tree に挿入する。トランザクションの最初の要素と、tree のルートの子 N のアイテム名が一致した場合、N の count を 1 増やす。そうでない場合、新しい子ノードを作成し、count を 1 とする。この処理を、トランザクションの要素に繰り返し適用する。

図1に示したFP-treeは、例えば $T_1=\{C\}$, $T_2=\{A, C\}$, $T_3=\{B, C\}$, $T_4=\{A, B, C\}$, $T_5=\{B\}$ のようなトランザクションから構築される。ただし、アイテムA, B, Cは全て頻出アイテムに含まれるとする。FP-treeを一度作成した後は、パターンの探索をFP-treeによって行うため、データベースを走査する必要がない。したがって、アイテム数に依らず、データベース全体を走査する回数は2回となる。

作成したFP-treeを、出現頻度の低いアイテムを含むアイテムの組から順に、木の枝分かれがなくなるまで条件付FP-treeを構築しながら再帰的に走査(FP-growth)を行う。走査手順の概要は以下である。

- (1) 頻出アイテムのリストの中で、一番下(最も非頻出なアイテム)から処理を行う。
- (2) 各頻出アイテムに対するconditional pattern baseを作成。これは、各頻出アイテムに至るまでのノードとその頻度のリストである。
- (3) conditional pattern baseからconditional FP-treeを作成。
- (4) treeに枝分かれが無い場合
 - (a) treeのパスに含まれるノードの各組み合わせに対して、条件付けに使用したアイテムを合わせたパターンを作成する。パターンのサポート値はパスに含まれるノードの最小サポート値とする。

(5) treeに枝分かれがある場合

- (a) 再帰的にアイテムで条件付けしたconditional FP-treeを作成し、走査する。

4. 先行研究

FHEを使用し秘匿データマイニングを行った先行研究について、概要を紹介する。なお、本項で挙げる先行研究で採用されている頻出パターンマイニングのアルゴリズムは、すべてAprioriである。

4.1 P3CC

P3CCは、Liuら(2015)が提案した、完全準同型暗号を用いた安全な頻出パターンマイニング委託システムである[1]。完全準同型暗号の暗号文同士は比較演算が困難であるため、比較演算が必要な部分に関してはクライアントにデータを返し処理を行う。また、暗号文のデータサイズ削減のため、アイテム数、トランザクション数は暗号化されず、各トランザクションに含まれるアイテムを示すバイナリ行列にのみ暗号化を適用している。さらに、クライアント側でダミーデータを加えることで、委託先サーバからの平文データの推測を防いでいる。

4.2 P3CCの暗号文パッキングと暗号文キャッシングによる高速化

高橋ら(2016)は、P3CCをSVパッキングを用いて高速化する手法を提案した。複数の整数をベクトルとして一括に暗号化できるパッキングという方式を用いて、暗号文の個数、暗号文同士の乗算を削減を行った。その結果、パッキングを用いない場合と比較して10倍以上の高速化を実現した。この手法は、Aprioriに限らず、秘匿検索や他のデータマイニングアルゴリズムにも応用することができるとしている[2]。

今林ら(2017)は、完全準同型暗号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、暗号文パッキングの適用手法と暗号文キャッシング手法を提案した。提案手法が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、データセットが大きい場合や、ダミーセットを加えた場合により効果が大きかったとしている。特にトランザクション数10,000のとき、P3CCと比較して、430倍の高速化と94.7%のメモリ使用量削減を実現している[3]。

4.3 P3CCの分散環境への実装とデータベース更新時の処理の高速化

山本ら(2018)は、データベース更新時のAprioriアルゴリズムの高速化を行うFUP(Fast Update)アルゴリズムを用いた秘匿データマイニングシステムの実装を行った。ま

た、マスタ・ワーカ型分散処理を適用し、システムの高速化を行った。分散処理方法には、アイテムセットごとの分割を適用した。その結果、データベース更新時において FUP アルゴリズムを導入した際の再計算の計算時間は、Apriori アルゴリズムによる再計算と比較して約 3~4 倍の短縮が可能になった。また分散処理化によって、マスタ側の計算時間が分散台数に応じて減少している [4]。

5. 手法

本研究では、以下の図 2 のようなシステムを使用する。

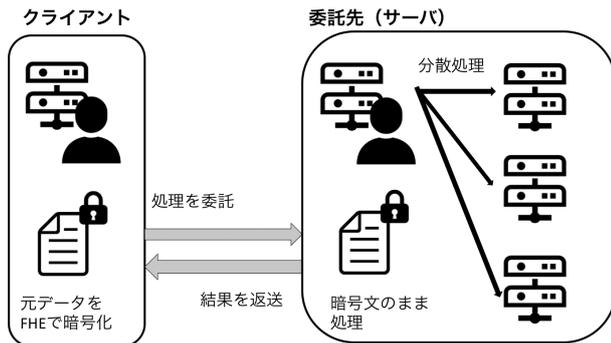


図 2 システム概観

サーバ側の処理にはマスタ・ワーカ型の分散処理を採用している。本研究における実験ではワーカ数を 1 に固定している。本システムで使用する頻出パターンマイニングのアルゴリズムは、Apriori と、FP-growth の 2 つである。双方のプログラムに高橋ら (2016) による先行研究 [2] による暗号文パッキングを使用している。Apriori を使用したプログラムは、山本ら (2018) による先行研究 [4] によるものである。実装は C++ で行い、完全準同型暗号のライブラリは Helib[8]、分散・並列処理のライブラリは MPI[9] を使用している。

以下に、2 種類のアルゴリズムを使用したそれぞれのシステムについての処理を示す。

5.1 Apriori を用いた場合の処理の概要

- (1) クライアントでのデータの準備
 - (a) トランザクションデータを FHE で暗号化する。
 - (b) アイテムの候補を作成し、サーバに送信する。
- (2) サーバでの委託処理
 - (a) クライアントから暗号化されたデータを受信し、復号することなくアイテムのサポート値を計算する。その後クライアントに結果を返送。このときマスタ・ワーカ型の分散処理を行う。
- (3) クライアントでのミニマムサポート値との比較

- (a) サーバからファイルを受信
- (b) サポート値がミニマムサポート値以上のアイテムを取り出す。
- (c) 手順 1b に戻り、アイテムセットのサイズが 1 大きい候補を送信する。候補の数が 0 になるまで繰り返す。

各アイテムセットについて、サポート値の計算をサーバに委託する。サーバ側での比較演算を行えないことから、各アイテムセットとミニマムサポート値との大小比較はクライアント側で行う。

5.2 FP-growth を用いた場合の処理の概要

- (1) クライアントでのデータの準備
 - (a) トランザクションデータを FHE で暗号化する。
 - (b) アイテムの候補を作成し、サーバに送信する。
- (2) サーバでの委託処理
 - (a) クライアントから暗号化されたデータを受信し、復号することなくアイテムのサポート値を計算する。その後クライアントに結果を返送。このときマスタ・ワーカ型の分散処理を行う。
- (3) クライアントでの FP-tree 構築
 - (a) サーバからファイルを受信
 - (b) サポート値がミニマムサポート値以上のアイテムを取り出す。
 - (c) アイテムを頻出順に並び替え、非頻出アイテムを除いたトランザクションを作り直す。
 - (d) FP-tree を構築する。
- (4) クライアント側での FP-tree 走査
 - (a) 構築した FP-tree の走査を行い、結果を出力する。

本プログラムでは、FP-growth の処理のうち、FP-tree を構築する前に行われる、アイテムのサポート値の計算をサーバに委託する。FP-tree の構築と走査はクライアントプログラムにて行っている。FP-tree 構築や走査を行うには多数回の比較演算を行うことが必要であるため、本研究ではサーバに委託せずに処理を行った。クライアントの処理が終了していても、サーバはサポート値の計算結果をクライアントに送信した後はプログラムを終了する。

6. 実験

6.1 実験概要

同一ネットワーク内の 2 台の計算機を用いて、各計算機上でクライアントプログラムとサーバプログラムを動作させた。また、ミニマムサポート値とトランザクション数を

変化させることによる実行時間を確認する。暗号文のレベルは各ミニマムサポート値トランザクション数ごとに、動作する最小値を指定する。

6.2 実験環境

実験で用いた計算機の性能を表 1 に示す。

表 1 実験で用いた計算機の性能

OS	CentOS 6.9
CPU	Intel Xeon プロセッサ E5-2643 v3 3.6GHz 6 コア 12 スレッド
メモリ	512GB

クライアント、サーバとして表 1 に示す同型のマシンを 1 台ずつ使用した。サーバ側では 5 章に示したように、マスタ・ワーカ型の分散処理を行うが、本実験では 1 台に固定している。また、ワーカはマスタと同じマシン上で動作させる。

6.3 実験方法

6.3.1 各実験データに対する実行可能な最低レベルの調査

各トランザクションデータとサポート値の組み合わせに対して、復号不可能とならない最低値を探索した。最低値は各プログラムを実行し、正常終了した際のレベルの最低値とした。

6.3.2 実行時間測定

6.3.1 節に示した方法で求めた、実行可能なレベルの最小値を設定した際の実行時間の測定を行った。

6.4 入力データ

本実験で入力として用いるトランザクションデータは、人工的に作成した。データ生成は、IBM Quest Synthetic Data Generator で行い、平均アイテム長、最大パターンの大きさ、アイテムの種類数、トランザクション数を指定して生成した。各パラメータの値は表 2 のように指定する。

表 2 データ作成時のパラメータ

平均アイテム長	5 (固定)
最大パターンの大きさ	5 (固定)
アイテムの種類数	10 (固定)
トランザクション数	3300, 6600, 9900

6.5 実験結果

6.5.1 各実験データに対する必要最低レベルの調査

パラメータを変化させて作成した各トランザクションデータに対して、Apriori と Fp-growth の計算の過程で復号不可能とならなかった最低のレベルはそれぞれ以下の表 3, 表 4 に示す通りであった。

2 つのアルゴリズムを使用したシステム間での、必要な

表 3 Apriori 使用時の実行可能な最低レベル

		ミニマムサポート値			
		10	20	30	40
トランザクション数	3300	18	18	17	15
	6600	18	18	17	15
	9900	18	18	17	15

表 4 FP-growth を使用時の実行可能な最低レベル

		ミニマムサポート値			
		10	20	30	40
トランザクション数	3300	4	3	3	3
	6600	4	3	3	3
	9900	4	3	3	3

レベルの最低値が大きく異なることがわかる。Apriori を使用した際、トランザクションデータを一旦暗号化した後は、サーバとのデータ送受信のたびに毎回復号し直さない。Apriori の計算のループが多い、すなわち頻出アイテムセット長の最大値が大きくなるほど、暗号文ノイズ蓄積量が増加する。復号不可能にならないようにするためには、あらかじめ設定する暗号文のレベルの値を十分大きくする必要がある。

一方で、FP-growth はサーバとのトランザクションデータのやり取りが 1 回のみの実装であるため、レベルの初期値は 1 回分のサポート値計算を達成できればよいため、必要なレベルの数値が Apriori と比較して小さくなる。

また、サポート値の変化により、候補アイテムセット数が増加することが、トランザクション数よりも必要なレベルに強く影響することがわかる。

6.5.2 実行時間測定

実行時間は、各パラメータ設定に対し 3 回試行した平均を示す。図 3 に Apriori を使用時の実行時間、図 4 に FP-growth 使用時の実行時間を示す。図中の凡例については、クライアントとサーバのどちらで測定した実行時間を示しており、続く数字はトランザクション数である。

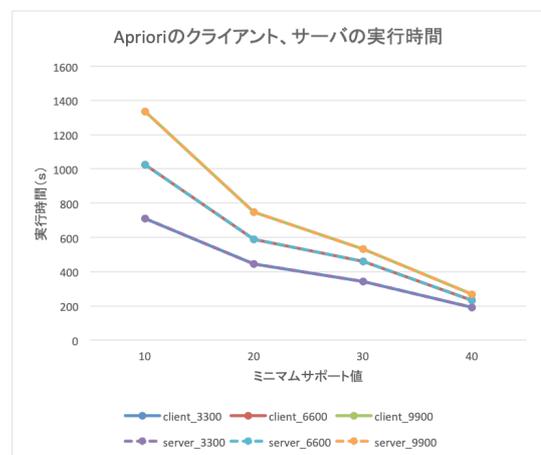


図 3 Apriori 使用時の実行時間

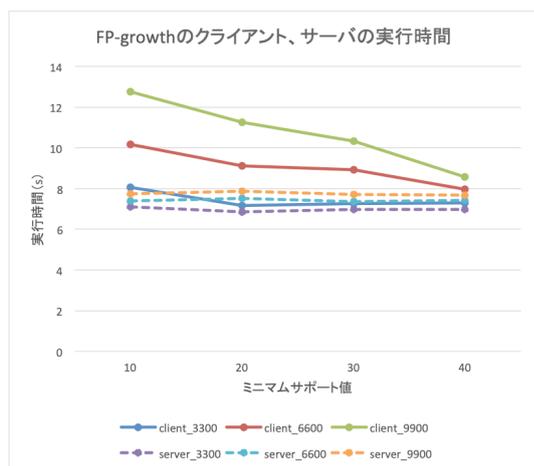


図4 FP-growth 使用時の実行時間

図3のグラフでは線が重なってしまっていることにより、クライアントに関する実行時間が視認できなくなっている。これは、同じトランザクション数での実行時間がクライアントとサーバでほぼ同じであるためである。実行時間に関しては、Aprioriの実行時間に対して、FP-growthの実行時間が全体で見ると約100分の1程度になっていることが確認できる。Aprioriのシステムの方が実行時間が長くなったことについて、レベルの値を大きくするほど暗号文のサイズが大きくなり、処理も計算量が大きくなるためである。FP-growthにおいて、FP-treeの構築と走査も計算量が多い処理であるが、今回の実験で扱ったサイズのデータにおいては、暗号文の処理と比較したとき極めて短いことがわかった。

実験において、2つの異なるシステム間で大きな結果の差が出たことは、アルゴリズム自体の差のみによるものではなく、実装の差が特に大きく影響したと考えられる。現状のFP-growthを使用した実装では、Aprioriを使用したシステムが複数回サーバとやり取りをするのに対して、サーバに委託する処理の割合が小さく、サーバとのデータの送受信回数は1回のみである。処理に時間のかかる、暗号文を操作する処理の回数が少なくなるため、システム全体としても実行時間に大きな差が出たものと思われる。

7. まとめと今後の予定

完全準同型暗号(FHE)を使用したFP-growthアルゴリズムによる頻出パターンマイニングを一部サーバに委託するシステムを実装し、Aprioriアルゴリズムを使用したものとの比較を行った。実行時間の比較においては、FP-growthを使用したシステムが、先行研究のシステムに対して100分の1程度であった。この結果は、アルゴリズム自体の差に加えて、システムの実装による差が顕著にあらわれたものであると考えられる。

今後は、さらに大規模なデータや、性質の異なるデータを入力とした実験や、計算機のリソース使用状況などの測

定を行い、さらに詳しくシステムの特徴を調査していきたい。また、FP-growthのシステムを改善するにあたり、サーバに委託する処理の割合を増加させる際には、クライアントとサーバ間での暗号文データの送受信の回数を抑える工夫が必要になると考えられる。

謝辞

本研究を進めるにあたり、早稲田大学の山名早人先生をはじめ、多くの先生方からのご指導とご協力をいただいております。深く感謝申し上げます。

本研究は一部、JST CREST JPMJCR1503の支援を受けたものです。

参考文献

- [1] J. Liu, J. Li, S. Xu, and B. CM Fung: "Secure outsourced frequent pattern mining by fully homomorphic encryption", In International Conference on Big Data Analytics and Knowledge Discovery, pp. 7081. Springer, 2015
- [2] 高橋卓巳, 石巻優, 山名早人: "SV パッキングによる完全準同型暗号を用いた安全な委託 Apriori 高速化", DEIM Forum 2016 F8-6, 2016
- [3] 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グ計算量効率化", 情報処理学会論文誌データベース (TOD), Vol. 10, No. 1, 2017
- [4] 山本百合, 小口正人: "完全準同型暗号を用いた秘匿データマイニング計算のデータベース更新時の分散処理による高速化", DICOMO2018, 2018
- [5] C. Gentry: "A Fully Homomorphic Encryption Scheme", Doctoral dissertation, 2009
- [6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan: "(Leveled) Fully Homomorphic Encryption without Bootstrapping", ACM Trans. Comput. Theory 6, 3, Article 13 (July 2014), 36 pages, 2014
- [7] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation", 2000
- [8] V. Shoup, S. Halevi: HELib, 入手先 (<<https://shaih.github.com/shaih/HELlib.git>>), (参照 2018-04)
- [9] Open MPI, 入手先 (<<https://www.open-mpi.org/>>), (参照 2019-04)