完全準同型暗号を用いたFP-growthによる頻出パターンマイニングの委託タスク増加による影響の調査

種村 真由子[1] 小口 正人[1]

概要：近年，ビッグデータの利活用が多くの分野で進んでいる．大規模なデータを扱う統計処理を行う際，処理能力の高い計算機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が，外部委託するデータが個人情報等，プライバシに関わる場合は，特に管理に注意する必要がある．本研究では，プライバシ保護のため，外部サーバに送信するデータを完全準同型暗号 (Fully Homomorphic Encryption，FHE) で暗号化する．FHE は，暗号文同士の加算と乗算が成立する公開鍵暗号で，委託先サーバに復号鍵を渡さず，統計処理を行うことが可能となる．これを用いて，頻出パターンマイニングを行うシステムを作成する．本研究では，先行研究において 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っている．しかし，現行のプログラムでは，クライアントがサーバに容易に委託可能なタスクが少ないため，クライアント側の処理の負荷を抑えつつさらにサーバに処理を委託する方法を検討する．また，その処理による実行時間，データサイズなどの増加を調査する．

A study about an effect of increased tasks at a server in secure frequent pattern mining with FHE using FP-growth

MAYUKO TANEMURA[1] MASATO OGUCHI[1]

1. はじめに

近年，ビッグデータの収集・分析などが，ビジネスを中心とする多くの分野で進んでいる．大規模なデータを扱う統計処理を行う際，処理能力の高い計算機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が，外部委託するデータが個人情報等，プライバシに関わる場合は，特に管理に注意する必要がある．多くの場合で行われているプライバシ保護のためのデータ処理の一例としては，個人を特定するような要素をマスキングした上で計算を行うなどの方法がある．

本研究では，プライバシ保護のため，外部サーバに送信するデータを完全準同型暗号 (Fully Homomorphic Encryption，FHE) で暗号化する．FHE は，暗号文同士の加算と乗算が成立する公開鍵暗号で，委託先サーバに復号鍵を渡さず，すなわちデータの内容を見せることなく，統計処理を行うことが可能となる．一方で FHE の性質上, 暗号文同士の比較演算は困難であり，さらに暗号文のデータ量と計算量が多いという特徴もある．FHE については，2 章で説明する．本研究では，これを用いて，頻出パターンマイニングを行うシステムを作成している．頻出パターンマイニングはデータマイニングの一種で，購買データなどの大規模なトランザクションデータを分析するために使われるものであり，どのアイテムの組み合わせが高頻度で出現するか，そこにどのような傾向があるかを調べるための手法である．頻出パターンマイニングと代表的な 2 つのアルゴリズムの Apriori と FP-growth については，3 章で記述する．FHE と頻出パターンマイニングを組み合わせた関連研究として，Apriori を使用した Liu ら (2015) の P3CC(Privacy Preserving Protocol for Counting Candidates) という手法がある．また，P3CC を SV(Smart-Vercauteren) パッキングや暗号文のキャッシュを利用し高速化した例や，サーバ側の分散処理化を行った例もある．先述の通り，本研究では，これらの先行研究において Apriori アルゴリズムで処

[1] お茶の水女子大学大学院
Ochanomizu University

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っている．しかし，以前実装したプログラムでは，クライアントがサーバに容易に委託可能なタスクが少ないことにより，クライアントで多くの処理を行っている．クライアント側の処理の負荷を抑えるためさらにサーバに処理を委託する方法を検討し，また，その処理を行った際の実行時間，使用リソースなどを調査する．

2. 完全準同型暗号

2.1 FHE の概要

完全準同型暗号 (FHE) は，加法準同型性と乗法準同型性の特徴をあわせ持った公開鍵暗号方式である．すなわち，暗号化した状態での暗号文同士の加算，乗算が成立する暗号方式である．すなわち，暗号文同士の計算をしたものを復号すると，暗号化される前の平文同士を直接計算したものと同じ結果が得られる．これにより，FHE を用いて暗号化されたデータは，委託先サーバに平文データを見られることなく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，概念そのものは Rivest ら (1978) によって提案されている [6]．また，2009 年に Gentry が Lattice ベースの実現手法を提案した [1]．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い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，比較演算が困難であ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．ノイズが増加した際は bootstrapping という処理を行うことで，暗号文のノイズを初期値に近い量に減少させ，暗号化した状態での計算が理論上何度でも可能となるが，実際にはこの処理も計算量が非常に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，Brakerski らによって提唱された Leveled FHE がある [7]．本研究で使用するプログラムでは，この Leveled FHE を使用している．Leveled FHE は，事前に設定した深さの論理回路の結果を評価することができる，完全準同型暗号の実装の一つである．暗号文同士の計算によるノイズを減らすために，暗号文の bootstrapping より処理が軽量で，あらかじめ暗号文同士の計算回数が把握できる場合に有効である．

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，相関ルールを抽出することを目的とした手法である．

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，頻出パターンを抽出する処理を行う．頻出であることの判定は，各パターンのサポート値があらかじめ指定したミニマムサポート値以上であるかを比較することにより行う．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，先行研究で使用されている Apriori と，本研究で使用している FP-growth がある．各アルゴリズムについての概要を以下の 3.1 章，3.2 章に示す．

3.1 Apriori

Apriori は，アイテム長 1，すなわちアイテムを 1 つだけ含むパターンから順にサポート値をミニマムサポート値と比較し，頻出と判断されたパターンを列挙していくアルゴリズムである [8]．実装は比較的容易である．アイテムの種類が少ない場合でも，アイテム同士の組み合わせの種類の総数は膨大になり得るという問題がある．したがって，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という枝刈りを行い，計算量を削減している．この手法では，アイテム長が長いパターンが頻出であるとき，アイテム長の分だけ処理が長引くという特徴がある．4 章で述べる先行研究で使用されている．

3.2 FP-growth

FP-growth は，Apriori に対して，FP-growth は，まず頻出パターンを全て含む FP-tree という木構造データを作成し，それを走査することにより，頻出アイテムセットを求めるアルゴリズムである [9]．まず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 の木構造に格納する．その部分木を取り出しながら部分木の枝分かれがなくなるまで再帰的に走査することで，頻出パターンを求める．データベースの走査回数が最低 2 回で済むという特徴がある．章で後述する Apriori と大きく異なる点は，探索に木構造データを用いる，頻出パターンの候補を列挙しないという部分である．また，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．木の規模が大きくなるほど計算量が大きくなり，特にトランザクションデータにおいては，トランザクション数よりもアイテム数の増加が木の規模の拡大に強く影響す

る．実装は Apriori よりも複雑である．

効率化が期待できる点がある一方で，FP-tree の構築と走査には多数かつ頻繁な比較演算が必要であるため，FHE を使用して行うことのできる処理が制限される．

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 [2]．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送し処理を行う．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 [3]．

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 [4]．

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 [5]．

5. システム

5.1 システム概要

本研究で使用するシステムは，すべてクライアント・サーバ型の委託処理を行うものである．クライアント側のマシンには秘密鍵，公開鍵の両方があり，サーバ側には公開鍵のみを暗号化されたデータとともに共有する．サーバ側ではデータを復号する必要のない処理のみを行うこととする．

1 章で述べたように，本研究では以前実装したプログラムと新しく実装したプログラムの比較を行っていく．以下，以前実装したプログラムを旧プログラム，本研究で新たに実装したプログラムを新プログラムと呼ぶ．

5.2 処理の流れ

Leveled FHE はその性質上，可能な計算の種類に制限があることから，頻出パターンマイニングの処理全てをサーバ上で行うことはできない．とくに，比較演算を必要とする処理はクライアントに戻して行う必要がある．したがって，本システムでは頻出パターンマイニング処理のサーバへの委託は，部分的に行っている．具体的には，FP-tree を構築する手前の，データ中の各アイテムの頻出・非頻出を調べる段階の処理までをサーバで行う．これは，FP-tree を構築する際に比較演算を多用するため，サーバ上で FP-tree をそのままの形で構築するのが困難なためである．

旧プログラムの処理を以下に示す．

旧プログラムの手順

(1) サーバの立ち上げと接続準備
サーバ側で通信の受付を行い，クライアントとの接続を確立する．

(2) トランザクションデータ送信前準備
公開鍵と必要なパラメータ等を事前に送信する．

(3) クライアントでのトランザクションデータ送信
クライアント側で保持しているトランザクションデータ（各要素が 0 または 1 のバイナリ行列）を FHE で暗号化し，サーバに送信する．続いてアイテム ID のリスト（暗号化されていないデータ）をサーバに送信する．

旧プログラムの手順

(4) サーバでの委託処理
クライアントから暗号化されたデータを受信する．各アイテムの出現回数を累計する．その後，クライアントに結果を返送する．

(5) クライアントでの FP-tree 構築
サーバから受信したファイルの復号を行う．アイテムごとの出現回数が事前に設定したミニマムサポート値に当たる回数（閾値）以上であるものを頻出と判断する．頻出であったアイテムを取り出し，FP-tree の構築を行う．

(6) クライアント側での FP-tree 走査
構築した FP-tree の走査を行う．

新プログラムでは，旧プログラムの手順 (5) における，頻出であるか否かを調べるため，出現回数と閾値との差を取る部分をサーバに追加で委託する．処理手順は以下の通りである．

新プログラムの手順

(1) サーバの立ち上げと接続準備
サーバ側で通信の受付を行い，クライアントとの接続を確立する．

(2) トランザクションデータ送信前準備
公開鍵と必要なパラメータ等を事前に送信する．

(3) クライアントでのトランザクションデータ送信
クライアント側で保持しているトランザクションデータ（各要素が 0 または 1 のバイナリ行列）と，ミニマムサポート値に対応するアイテムの出現回数の閾値を FHE で暗号化し，サーバに送信する．続いてアイテム ID のリスト（暗号化されていないデータ）をサーバに送信する．

(4) サーバでの委託処理
クライアントから暗号化されたデータを受信する．各アイテムの出現回数を累計する．さらに，別のデータとして各アイテムの出現回数と閾値の差を取る．その後，クライアントに 2 つの処理の結果を返送する．

(5) クライアントでの FP-tree 構築
サーバから受信したファイルの復号を行う．出現回数と閾値の差のデータが 0 以上であるアイテムを頻出と判断する．頻出であったアイテムを取り出し，FP-tree の構築を行う．FP-tree の構築には出現回数のデータを用いる

(6) クライアント側での FP-tree 走査
構築した FP-tree の走査を行う．

旧プログラムと比較して，クライアント，サーバで追加される，または変更される処理には下線を付加して示した．

5.3 実装

プログラム全体は C++により実装されている．FHE を扱うためのライブラリは HElib[10](2016/8 時点の実装)，分散・並列処理のライブラリには Open MPI[11] を使用している．

6. 実験

6.1 実験概要

新プログラムの実装において，旧プログラムと比較した際にクライアントとサーバのプログラム双方に追加の処理が発生した．これによる各プログラムを動作させた際の実行時間，通信量の増加の影響について調査する．

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，サーバとして表に示す同型のマシンを 1 台ずつ使用した．本プログラムはサーバ側がマスタ／ワーカ型の分散処理が可能な実装であるが，本実験では分散処理は行わない．クライアント，サーバとして利用するマシンは同一サブネットに存在するものである．使用する入力データは，IBM Quest Synthetic DataGenerator で生成した人工データである．

6.3 実験方法

同一ネットワーク内の 2 台の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，プログラム中で各処理の所要時間とデータ量を測定した．

データに関するパラメータは表 2 の通りである．

表 2 データに関するパラメータ

アイテム数	30
トランザクション数	9900
ダミーデータ	なし

また，暗号文に関して設定した主なパラメータは表 3 の通りである．

表 3 暗号文のパラメータ

平文空間	2^{14}
セキュリティパラメータ	80
レベル	3
暗号文のパッキング	有り

各測定は 7 回行い，6.4 章に示す結果はそのうち最大値

と最小値を除いた 5 回の平均の数値とする．

6.4 実験結果

6.4.1 実行時間

はじめに，旧プログラムと新プログラムの全体の実行時間を図 1 に示す．左 2 つの項目が旧プログラムのクライアントとサーバ，右 2 つの項目が新プログラムのクライアントとサーバにおける実行時間である．新プログラムではクライアント，サーバともに実行時間が増加している．その増加の割合はクライアントにおいて 27.9%，サーバで 0.236%であった．

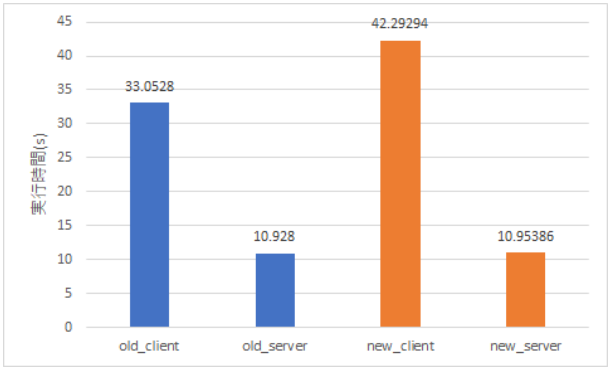

図 1　旧プログラムと新プログラムにおけるクライアント・サーバの全体実行時間

続いて，新プログラムのサーバにおいて増加した処理の内訳の実行時間を図 2 に示す．左から順に，閾値データの受信，差分の計算，差分のファイルへの書き込み，差分のデータの送信にかかる時間を示している．差分の計算にかかる時間が比較的短いことが分かる．サーバにおける計算時間の増加と暗号文同士の減算時間の増加を比較したとき，この規模のトランザクションデータであればファイル入出力にかかる時間の割合の方が大きいことがわかる．

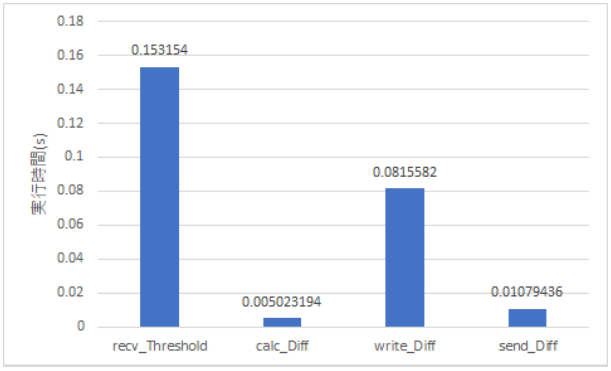

図 2　サーバで増加した処理ごとの実行時間

また，クライアントにおいて増加した処理の実行時間は以下の図 3 の通りである．左から順に，閾値データの暗号化，閾値データの送信，差分データの受信，差分データの復号にかかった時間を示している．最も時間がかかっているのは差分のデータの復号であることが分かる．また，閾値に関連する処理に対して，差分のデータに関連する処理のほうが実行時間が長いことが分かる．

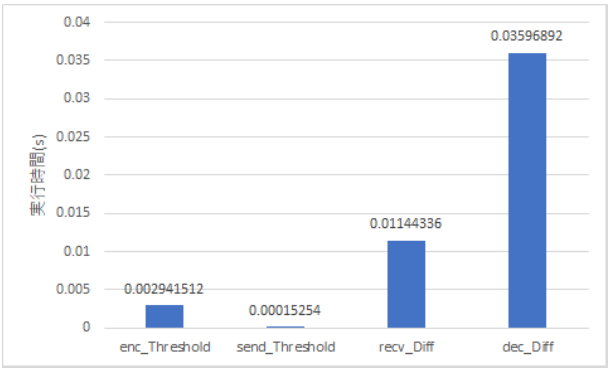

図 3　クライアントで増加した処理ごとの実行時間

6.4.2 データ量

追加の処理において，ファイルの送信や復号などに時間がかかっていることが分かる．そこでクライアントとサーバでやり取りされる追加のデータ量についてまとめたものを以下の図 4 に示す．左の項目から順に，クライアントからサーバに送信した閾値のファイル，クライアントがサーバから受信した差分のファイル，クライアントからサーバに送信したトランザクションデータのファイル，クライアントがサーバから受信したアイテムの出現頻度のファイルの通信量である．右 2 つについては，旧プログラムの実装においても存在するものであり，この図においては参考のために載せている．クライアントにおける通信データ量は，送信の増加は少ないが，受信の増加分が大きい．受信の増加分はアイテムの出現頻度のデータと近い値であることが分かる．また，備考として，閾値のデータサイズは入力データサイズに影響されないため，ほとんど変化がないものである．また，差分のデータは入力データのアイテム数に比例するものである．

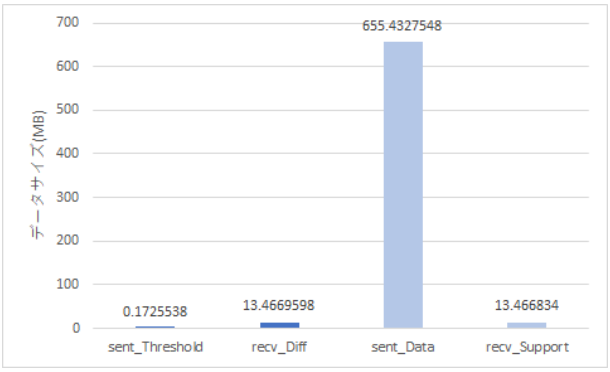

図 4　クライアントで増加した通信データ量

7. 考察

クライアントでの実行時間の増加については，図 2 で示したもののすべての和をとっても図 1 に示した旧プログラムと新プログラムの実行時間の差よりもはるかに小さいため，さらに別の要因で実行時間が伸びている可能性を考慮する必要がある．サーバにおける実行時間の増加は 10^{-3} の範囲であったため，影響は少ないといえる．図 2 で示した，差分を取る計算の計算時間が他の処理と比較して短かった要因としては，FHE の計算においての加算は処理が軽く，結果としてはファイルサイズに関係する処理のほうが長くなったためだと考えられる．図 3 の結果として，閾値に関連する処理よりも差分に関する処理のほうが実行時間が長いことについては，入力データサイズにより増加する差分のファイルサイズが直接的に影響していると思われる．また，差分のファイルサイズは出現頻度のファイルサイズとほぼ同じであるため，新プログラムでの処理の追加により単純にクライアントの受信する結果ファイルのサイズは約 2 倍になると考えられる．

今回は暗号文のレベルを 3 という低い値に設定していたため，これよりも大きなレベルを設定した際の影響はこの限りではない．しかし，サーバにおける加算の追加による影響の少なさが判明したことには，今後さらにサーバ側の処理を追加する際に有効な情報である．

8. まとめと今後の展望

完全準同型暗号を用いた FP-growth において，サーバ側で行う処理を増加させたことにより増えるコストの影響について調査した．実行時間の影響は，その増加の割合はクライアントにおいて 27.9%，サーバで 0.236%であった．クライアントで増加した実行時間は受信ファイルの復号であったが，実行時間の大きな増加原因については判明しなかったため，引き続き調査する必要がある．サーバにおける実行時間の増加は 10^{-3} の範囲であり，影響は比較的小さかった．今後は，今回の実験で得られた処理の増加による実行時間やファイルサイズの増加の影響を考慮しつつ，クライアント上で行われている処理をさらにサーバに移すため，FP-growth のアルゴリズムそのものを編集するなどの方法を検討していく．

9. 謝辞

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] C. Gentry: ”A Fully Homomorphic Encryption Scheme, Doctoral dissertation,” 2009.
[2] 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. 70 - 81. Springer, 2015.
[3] 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.
[4] 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.
[5] 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.
[6] R. L. Rivest et al.: “On data banks and privacy homomorphisms, ”Foundations of secure computation, vol. 4, no. 11, 1978, pp. 169 - 180.
[7] 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
[8] R. Agrawal, T. Imieli nski, and A. Swami: “Mining association rules between sets of items in large databases,” in Acm sigmod record, vol. 22, no. 2. ACM, 1993, pp. 207 - 216.
[9] J. Han, J. Pei, and Y. Yin: “Mining Frequent Patterns Without Candidate Generation,” in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD ’00. New York, NY, USA: ACM, 2000, pp. 1 - 12. [Online]. 入手先 ⟨http://doi.acm.org/10.1145/342009.335372⟩ （参照 2020-05-15）
[10] Shai Halevi and Victor Shoup: HElib. 入手先 ⟨https://github.com/homenc/HElib⟩ （参照 2020-05-15）
[11] Open MPI. 入手先 ⟨https://www.open-mpi.org/⟩ （参照 2020-05-15）