ブロックチェーン上データ解析用オフチェーンDBの改ざん検知と解析結果保証

萱原　正彬†　　Le Hieu Hanh†　　横田　治夫†

† 東京工業大学 情報理工学院　〒 152–8552 東京都目黒区大岡山 2-12-1
E-mail: †{kayahara,hanhlh}@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　暗号資産の基盤技術であるブロックチェーンはデータの耐改ざん性・耐消失性を兼ね備えており，近年様々な分野での応用が期待されている一方，データ解析は不得意と言われている．したがって、データを解析するには別の DBMS に抽出しオフチェーンで解析することが効率的とされるが，抽出後のオフチェーン DB のデータは改ざんされる恐れがある．本研究ではブロックチェーン上とオフチェーン DB 上の両データのハッシュ値の比較によりデータの改ざんを検知し，改ざん検出時は解析に使用するデータを再抽出することで解析における結果保証を目指す．その際，ハッシュ関数をかける対象を変化させ，実験によってそれらのオーバーヘッドや特徴を整理・検証する．

キーワード　ブロックチェーン，オフチェーン DB，Hyperledger，改ざん検知，結果保証

1　はじめに

1.1　研究背景

元は暗号通貨であるビットコインの基盤技術として開発されたブロックチェーンにはデータの耐改ざん性や耐消失性といった強みを備えている．そのため現在，ブロックチェーンを様々な分野に応用する動きが見られている．その分野は金融や保険業界から流通・食品業界まで多岐に渡る．具体的な例としては医療分野に応用した萱原ら [1] の研究がある．これはブロックチェーンベースの電子カルテにプロキシ再暗号化を用いることにより情報の秘匿性に応じて公開範囲を設定し，権限の異なる複数ユーザーへのデータのやり取りや権限失効にも対応したものである．

また，ブロックチェーンに格納されるデータ量は近年急増しており，格納されたデータの解析に対する需要が高まっている．中でも現在も取引が活発に行われるビットコインのブロックチェーンデータ量は 300GB を超え，更に増加し続けている [2]．しかし，現在のブロックチェーンはほとんどがデータの解析を得意とする OLAP システムとしてではなく，データの格納などを得意とする OLTP システムのさらに特殊な種類として設計されている [3]．これは，ブロックチェーンのデータ構造がシンプルなキーバリューストア形式であることや，データ解析においてコンセンサスアルゴリズムのオーバーヘッドが影響してしまうことが原因とされている [4]．したがってブロックチェーンに直接問い合わせを行うデータ分析では小規模なクエリにしか効果を発揮しないと言われている．

実際に予備実験として，1 から 1,000 までの総和計算をブロックチェーン上 (Hyperledger Fabric) と通常のデータベース上 (PostgreSQL) でそれぞれ実行したところ，所要時間が前者は 4,400 秒，後者は 0.6 秒ほどと約 6,800 倍もの差が生じていることが分かった．

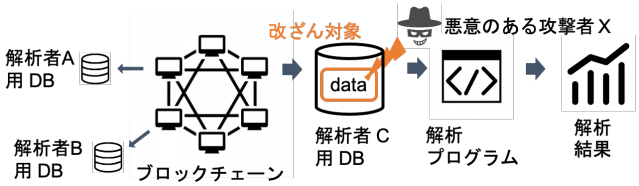

図 1　前 提 条 件

以上を踏まえると，ブロックチェーン上のデータを解析するには別の DBMS を用いたオフチェーンでの解析が効率的とされているが，ブロックチェーンから抽出した後はデータが改ざんされる恐れがあるという課題を抱えており，この課題点についての研究は重要とも言われている [5]．

1.2　前　　提

本研究においては，様々なステークホルダーがブロックチェーンネットワークにアクセスし，その中の解析の役割を担う者が各々のオフチェーンデータベース (オフチェーン DB) に必要なデータを抽出し解析を行うことを前提としている．その際，抽出したオフチェーン DB 内のデータが外部の何者かによる攻撃により改ざんされる場合を想定している．したがって本研究では悪意を持った内部の関係者が存在すること，解析プログラムおよび解析結果の改ざんについては考慮しないものとする．なお，ブロックチェーンの耐改ざん性・耐消失性についても前提とするため，攻撃者のブロックチェーン上のデータに対する改ざんは成功しないものとする．また，本研究において解析プログラムの出力や結果をオフチェーン DB は保持しないものとする．これらの前提の状況を図 1 にまとめる．

1.3　既存研究

Brahma らの研究 [6] では，BAT アルゴリズムを用いたデータベースの侵入検知システムを提案した．近年，侵入検知システムにデータマイニングの技術を適用する研究が活発に行われ

ており，この研究では郡知能アルゴリズムの 1 種である BAT アルゴリズムと距離ベースの外れ値検出アプローチを組み合わせることにより外部の攻撃者によるデータベースへの侵入を検知している．実際にリアルタイムの生体認証データセットを使用して実験を行い，その他のデータマイニング技術を応用した先行研究よりも高い精度で侵入を検知することに成功した．

しかし，この既存研究の課題点として以下の 2 つが考えられる．1 つ目は侵入検知の性能に依然改善の余地があることである．既存研究においては BAT アルゴリズムと呼ばれるデータマイニングの手法を用いているが，実験結果の適合率 (Precision) と精度 (Accuracy) はどちらも 96.5%に留まっている．また，検出の網羅性を示す再現率 (Recall) については言及されていない．2 つ目は仮に改ざんが生じてしまった場合，そのデータを元に復元することができないことである．一般的に改ざん検知システムでは外部からの攻撃により改ざんが生じた場合，その侵入・改ざんの検知は可能であるが基本的に改ざんされたデータの復元までは行われない．

そのため，本研究のようにブロックチェーンのデータ解析に焦点を当てた際は，改ざん検出が全網羅的に行われておらず，改ざんされたデータの復元が困難である点をブロックチェーンの特徴を活かしながら解決していくことが求められる．

1.4 本研究の目的

本研究ではブロックチェーンから抽出したオフチェーン DB のデータ解析の結果が正しいデータによるものであることを保証することを目的とする．この目的を達成するためのアプローチとして，ブロックチェーン上のデータとオフチェーン DB 上のデータにそれぞれハッシュ関数を適用し，両者を比較することで改ざん検知を行う．また，改ざんが検知された場合はブロックチェーンからデータを再抽出した後に解析を行うことで結果が改ざんされていないことを保証する．

1.5 本稿の構成

本稿は以下の通り構成される．2. 節では本研究の前提となる概念を背景知識として説明する．3. 節では本研究の提案手法としてブロックチェーン上のデータ解析に使用するオフチェーン DB の改ざん検知および結果保証を行うシステムの概要を述べる．4. 節では 3. 節の提案手法を実装し，実験によってハッシュ関数をかける対象を変化させた際のオーバーヘッドや特徴の違いを評価する．最後に 5. 節で本研究のまとめと今後の課題を述べる．

2 背 景 知 識

本節では背景知識として，ブロックチェーン，および本研究での提案手法を実装する際に使用したブロックチェーン基盤のフレームワークである Hyperledger Fabric について紹介する．

2.1 ブロックチェーン

ブロックチェーンは 2008 年に Satoshi Nakamoto が発表した論文 [7] の中で仮想通貨であるビットコインの基盤となる技術のことを指す．その後，ブロックチェーンの改善が研究され様々な分野へ応用が行われているため，広義的には「電子署名とハッシュポインタを使用し改竄検出が容易なデータ構造を持ち，且つ，当該データをネットワーク上に分散する多数のノードに保持させることで，高可用性及びデータ同一性等を実現する技術」と定義されている [8]．したがってブロックチェーンは通貨システムに特化した仕組みではなく，分散したネットワーク環境において唯一となるような情報を共有し，その情報に基づいて何らかの処理を行うという極めて汎用的な概念を実用化したものと言える．

表 1　パブリック型とプライベート型の比較

	パブリック型	プライベート型
ノードへの参加	制限なし	制限可能
データ参照・更新	制限なし	制限可能
ビザンチン障害耐性	必須	任意
マイニング報酬	必要	任意

ブロックチェーンには用途や適用するネットワークの種類によっていくつかのパターンが存在し，ブロックチェーン基盤はそれぞれどのパターンを指向しているのかによっておおよそ分類することが可能である．大枠ではパブリック型とプライベート型に分けられ，前者は誰でもネットワークに参加することが可能でありアクセス制御機能が無いためパーミッションレス型とも呼ばれる．後者は信頼する参加者のみにネットワークを限定しておりアクセス制御機能が付随していることが多くパーミッション型とも呼ばれる．それぞれの特徴を表 1 にまとめた．また，プライベート型は企業内で使用する場合と企業連合体を組成して個々の企業間をまたいで使用する場合が存在し，後者をコンソーシアム型と呼ぶ．パブリック型のブロックチェーンの代表例としては Bitcoin Core [9] や Ethereum [10] が，プライベート型およびコンソーシアム型のブロックチェーンの代表例としては後述する Hyperledger が挙げられる．

ブロックチェーンを構成する要素技術として「P2P ネットワーク」「コンセンサスアルゴリズム」「電子署名」「ハッシュ関数」「スマートコントラクト」などが挙げられるが，この内「スマートコントラクト」について詳しく説明する．

スマートコントラクトとはブロックチェーンネットワーク上で動作するプログラムのことである．一般的には，ブロックチェーンに格納されているデータの参照および更新を行う他，管理項目やビジネスロジックなどを独自にカスタマイズすることが可能となっている．このカスタマイズには既存もしくは独自のプログラミング言語を用いており，ブロックチェーンプラットフォームの一種である Ethereum では Solidity と呼ばれる独自のプログラミング言語を使用する．スマートコントラクトのプログラムもブロックチェーンネットワーク参加者間で共有されるため，振る舞いの妥当性が検証可能であり透明性の高いシステムを構築することができる．なお，Ethereum では Contract として，Hyperledger では Chaincode としてスマートコントラクトが導入されており，アプリケーションを通して実行される．

2.2 Hyperledger Fabric

Hyperledger [11] は 2016 年 2 月に Linux Fundation によって開始されたオープンソースのブロックチェーンプラットフォームであり，仮想通貨に限らない広範囲なビジネス用途のブロックチェーン基盤を提供することを目的としたプロジェクトである．2019 年 1 月時点ではテクノロジー・金融・製造・流通・政府機関・教育機関など世界中の様々な業界から 260 以上の企業と組織が参加している．

Hyperledger プロジェクトの 1 つに Hyperledger Fabric [12] と呼ばれるブロックチェーン基盤の実装フレームワークがある．Hyperledger Fabric は当初から企業向けに設計されたものでありモジュール式で汎用性の高いデザインを持つことから，幅広い業界のアプリケーションやソリューションの開発などのユースケースに対応している．

Hyperledger Fabric の主な特徴として「パーミッション型ネットワーク」「迅速なコンセンサスアルゴリズム」「チェーンコード」「多彩な開発環境」の 4 点が挙げられる．

- **パーミッション型ネットワーク**

管理者が不在で誰もがブロックチェーンネットワークに参加可能なパーミッションレス型ではなく，Hyperledger Fabric は選定された複数の信頼性の高い参加者で構成されるパーミッション型ネットワークを前提としている．したがってネットワークに参加するノードやユーザはあらかじめネットワークに登録をしておく必要がある．

Hyperledger Fabric では「チャネル (Channel)」や「組織 (Organization)」といった管理単位が存在する．前者は特定のノード間のみでチェーンコードと台帳を共有し，情報のプライバシーを保護する仕組みであり，後者は Hyperledger Fabric のネットワーク上の参加者を特定のノードや情報に結びつけてグルーピングすることができる論理的な管理単位のことを指す．

- **迅速なコンセンサスアルゴリズム**

Hyperledger Fabric のコンセンサスアルゴリズムは現在 Raft [13] が標準で利用されており，軽度で迅速なコンセンサスが得られる仕組みになっている．そのため，ビットコインのブロックチェーンは毎秒数トランザクション，Ethereum は毎秒数十トランザクション程度のスケーラビリティであるが，Hyperledger Fabric のスケーラビリティは毎秒数百トランザクションに達する．

- **チェーンコード**

Hyperledger Fabric ではスマートコントラクトの機能をチェーンコードと呼ばれるプログラムで実現しており，様々な業務処理を記述，実行することが可能となっている．チェーンコードには以下の 3 つの代表的な機能を有している．

1. チェーンコードを初期化する．
2. クライアントの要求に応じて台帳を更新する．
3. クライアントの要求に応じて台帳の照会結果を返す．

チェーンコードにはチェーンコード ID と呼ばれる識別子が付与されており，1 つの Peer に対して複数の異なるチェーンコードを配置することが可能である．台帳はチェーンコード ID とチャネルの組み合わせ単位で管理されており，これらをまたいだ更新を行うことはできない．

- **多彩な開発環境**

Hyperledger Fabric は Go，Java，Javascript といった多言語での開発に対応しており，Ethereum で使用される EVM(Ethereum Virtual Machine) や Solidity もサポートしている．また，クライアントの機能を独自アプリケーションに組み込むための専用ライブラリである Hyperledger Fabric SDK が存在し，開発中のものを含めると Node.js，Java，Python，Go，REST などに対応している．Hyperledger Fabric SDK で実行できる代表的な機能には以下のものがある．

- チャネルの作成
- Peer ノードのチャネルへの参加
- Peer ノードへチェーンコードをデプロイ
- チャネルのチェーンコードを初期化
- チェーンコードを呼び出してトランザクションを実行
- ワールドステートまたはブロックチェーンに対する照会

3 提案手法

本節ではブロックチェーン上のデータとオフチェーン DB 上のデータにそれぞれハッシュ関数を適用し，両者を比較することで改ざん検知を行うアプローチを用いて，データの改ざん検出を全網羅的に行い，データが改ざんされた場合はデータの復元を正しく行う手法を提案する．始めに提案手法の概要を述べる．その後，解析対象となるデータと 2 種類のメタデータを合わせた LTC-data について，さらに提案手法の手順についてデータの格納時・抽出時・解析時の場面に分けて説明する．なお，本研究ではブロックチェーン上のデータを対象とするため解析を行うデータの更新は行われないものとする．

3.1 提案手法の概要

本研究における提案手法は「格納」「抽出」「解析」の 3 つのフェーズに分かれる．図 2 はメインフェーズであるデータ解析時を示している．なお，解析プログラムを実行する際はプログラムへの入力とハッシュ値の算出を並行して行う On-The-Fly 方式を採用した．

ブロックチェーン上には後述する「**Ledger-data** (L-data)」「**Table-data** (T-data)」「**Category-data** (C-data)」の 3 種類のデータが保持されており，今回解析の対象となるデータは L-data にあたる．この 3 種類のデータについて次小節で詳しく説明する．

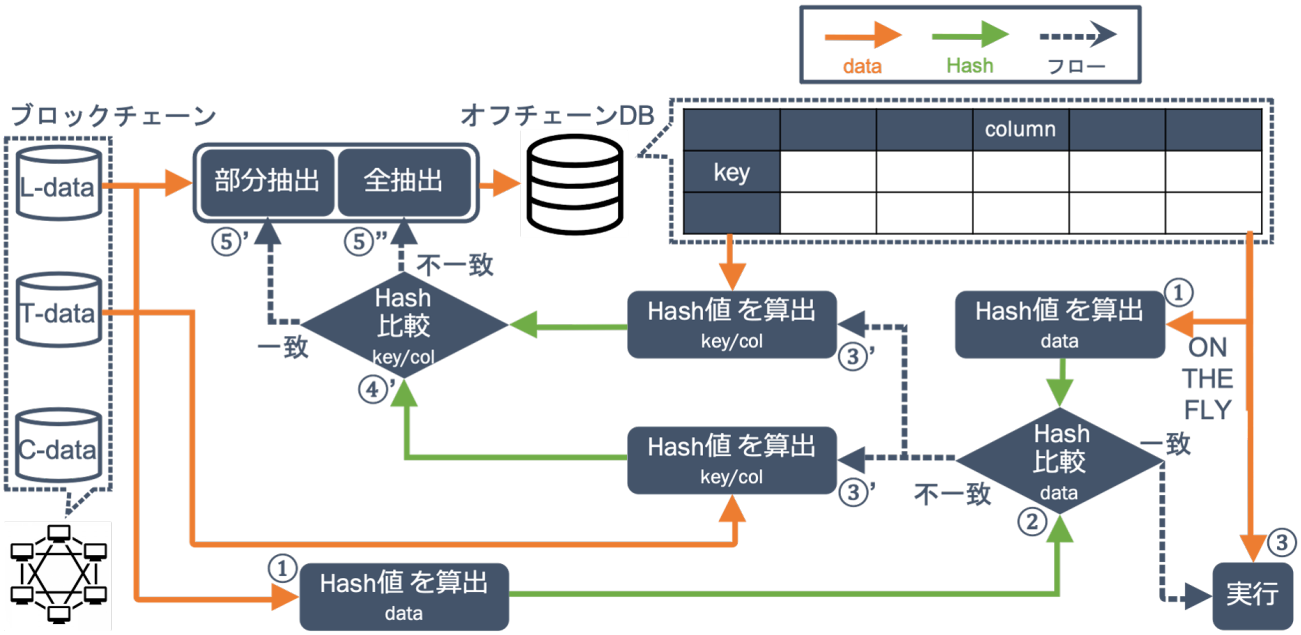

図 2　提 案 手 法

3.2　LTC-data

本提案手法では解析の対象となるデータに加え 2 種類のメタデータを作成する，これらのデータを総称して「**LTC-data**」と呼ぶ．LTC-data はブロックチェーン上に格納されているため改ざんされる恐れが無いことが特徴として挙げられる．また，LTC-data は L-data，T-data，C-data がそれぞれまとまって保存されてはおらず、時系列を含めランダムに保持されている．

L-data，T-data，C-data それぞれの説明については後述するが，以下では説明を行う際の具体例としてブロックチェーンに医療データを格納した「ブロックチェーンベースの電子カルテ」を解析者が解析する場合の使用例を用いて説明する．PID，O-num はそれぞれ患者 ID，医療オーダーナンバーを表している．また，例のおけるデータの記述に関する文法については簡略化しており，Key 情報によってソートして表現している．

以後，必要に応じて L-data，T-data，C-data の Key および Value をそれぞれ，L-key，L-value，T-key，T-value，C-key，C-value と表記する．

3.2.1　L-data

L-data はレジャー情報のことを示しており，格納や抽出，解析の対象となるデータのことである．オフチェーン DB に保管されているデータが改ざんされているかどうかは L-data を基準に判断される．通常のデータベースに格納されるデータは様々なカテゴリごとにテーブルが作成されるが，ブロックチェーンにおける台帳は大半がキーバリューストア形式であるため L-data は様々なカテゴリの情報が入り混じった状態で保持されている．

表 2 は医療データをブロックチェーンに格納した際の L-data の例を示している．L1，L2 は患者情報，L3，L4 は診察情報のデータにあたる．

表 2　L-data の例

Key	Value
L1	{PID:p1, age:24, address:aaa}
L2	{PID:p3, age:8, address:bbb}
L3	{PID:p1, O-num:001, T-type:X-ray}
L4	{PID: p1, O-num:002, T-type:MRI}

表 3　T-data の例

Key	Value
T1	{key:L1,L2, column:PID,age,address, Category:C1}
T2	{key:L3,L4, column:PID,O-num,T-type, Category:C2}

3.2.2　T-data

T-data はテーブル情報のことを示しており，最新のオフチェーン DB の Key 情報とカラム情報を保持している．T-data は L-data の抽出時および解析プログラムを実行する前のハッシュの算出・比較の段階で使用する情報であり，これによりオフチェーン DB に抽出した元データである L-data に正しくアクセスすることが可能となる．

T-data のようなデータを設けない場合，オフチェーン DB のデータの Key 情報に改ざんが生じると正しく元データの L-data を参照することが出来ず，再抽出に失敗してしまう．そのため，T-data は改ざんが発生した際に「どのデータが改ざんされたか」を正しく判断するための重要な情報であり，この情報を改ざんされないようにするためにブロックチェーン上に格納しておくことが必要となる．

表 3 は医療データをブロックチェーンに格納した際の T-data の例を示している．T1 は患者情報をまとめたテーブル情報，T2 は診療情報をまとめたテーブル情報である．

表 4　C-data の例

Key	Value
C1	{key:L1,L2}
C2	{key:L3,L4}

3.2.3　C-data

C-data はカテゴリ情報のことを示しており，ブロックチェーン上の L-data がどのカテゴリに属するかが記載されている．ブロックチェーンの台帳には様々なカテゴリの情報が順不同で保持されている．そのため，データをカテゴリごとに抽出するといった条件付きの抽出時には C-data が必要であり，これらの情報はデータをブロックチェーンに格納する際にそのデータのカテゴリごとに分類し C-data として保持しておく．

表 4 は医療データをブロックチェーンに格納した時の C-data の例を示している．C1 は患者情報を，C2 は診療情報を表している．

3.3　提案手法の手順

本節では提案手法の手順をブロックチェーン上のデータの格納時，抽出時，解析時の 3 つのフェーズに分けて詳しく説明する．

3.3.1　データ格納時

まず，データ格納時の手順を説明する．

①. L-data にデータを格納する．
L-key にはデータの識別子を，L-value にはカテゴリ以外のデータを保持させる

②. C-data に格納したデータのカテゴリ情報を載せる．
C-key にはカテゴリ名を，C-value には C-key のカテゴリに適切なデータの L-Key を保持させる．

C-data は抽出時の条件が L-Key を指定するような基本的なものであった場合は不要であるが，抽出するデータのカテゴリを指定するといった条件の場合はカテゴリの索引機能を持つメタデータの C-data が無いと実行が困難となってしまう．

3.3.2　データ抽出時

次に，データ抽出時の手順を説明する．

①. 抽出する L-data のカテゴリを参照する．
オフチェーン DB に抽出する L-data のカテゴリを指定するような条件の場合は C-data を参照．

②. T-data を作成する．
T-key にはオフチェーン DB におけるテーブル名を，T-value には抽出する L-data の L-key およびカラム情報，そして L-data のデータカテゴリを保持させる．

③. オフチェーン DB にデータを抽出する．

なお，オフチェーン DB に新しいテーブルを作成する場合は T-data も新たに作成する．

データ抽出において始めに T-data を作成することでオフチェーン DB への抽出直後のタイミングにおけるデータの改ざんを防ぐことができる．また，新たな L-data をオフチェーン DB の既存のテーブルへ抽出する際は作成されている T-data に新たに抽出する L-data の L-key を追加する．

3.3.3　データ解析時

最後に，データ解析時の手順を説明する．なお，図 2 の中の番号は以下で説明する手順の番号に対応している．

①. L-data およびオフチェーン DB それぞれのデータにハッシュ関数を適用し，ハッシュ値を算出する．

②. 両者のハッシュ値を比較する．
- ハッシュ値が一致した場合：手順③. へ進む
- ハッシュ値が不一致の場合：手順③'. へ進む

③. ハッシュ値を算出したデータを入力として解析プログラムを実行する．

③'. T-data，オフチェーン DB の Key およびカラム情報にハッシュ関数を適用し，ハッシュ値を算出する．

④'. 両者のハッシュ値を比較する．
- ハッシュ値が一致した場合：⑤'. 改ざんが生じてしまったデータのみを部分抽出する．
- ハッシュ値が不一致の場合：⑤". テーブルの全抽出を行う．

解析プログラムを実行する際はプログラムへの入力とハッシュ値算出を並行して行う On-The-Fly 方式を採用している．また，手順⑤'. における部分抽出は手順①. のハッシュ関数の適用の際に自身が選択した対象の単位で抽出される．例えば，タプル単位でハッシュ関数を適用した場合はテーブル内の 1 つのアイテムのみが改ざんされてしまった際も，改ざんされたアイテムを含むタプルごと再抽出を行う．

オフチェーン DB 上のデータそのものが改ざんされる可能性の他に Key およびカラム情報が攻撃によって改ざんされる場合が考えられる．その際はブロックチェーン上とオフチェーン DB 上とでハッシュ関数を適用するデータの整合性が担保されず，T-data が正常に機能しなくなってしまう．このような状況に陥った場合は改ざんされた部分のみを抽出することができなくなってしまうため，テーブルの全抽出を行う．

3.4　オフチェーン DB について

オフチェーン DB のデータを解析する際にデータが改ざんされている場合は再抽出を行うが，抽出直後に外部からの攻撃が繰り返されることで再抽出のループに入ってしまう可能性がある．こうした状況を避けるためオフチェーン DB は図 2 のよう

表 5　実 行 環 境

CPU	Intel Core M 1.3GHz
Memory	DDR3 8GB 1,600MHz
OS	macOS Catalina 10.15.7
Docker	version 19.03.13, build 4484c46d9d
Hyperledger Fabric	v2.1.1
Node.js	v14.14.0
npm	6.14.10
PostgreSQL	13.0

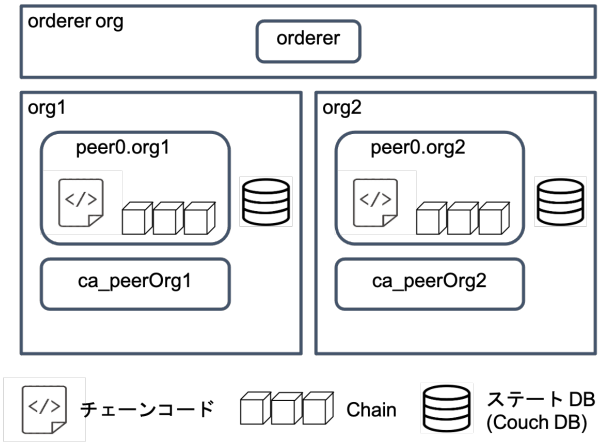

図 3　ブロックチェーンネットワーク構成

な単一の配置の他に，複数のオフチェーン DB を用意することで，Primary/Backup の体制を取ることや，同一のデータか否かを互いのオフチェーン DB 同士がハッシュをかけ合って検証するなどの対策が考えられる．

4 実　　験

本節では提案手法を実装しデータの解析のフェーズにおける改ざん検知およびデータの再抽出に関する評価実験を行う．本実験の実行環境を表 5 に示す．また，ブロックチェーンネットワークは Hyperledger Fabric を用いて構築した．ネットワーク構成は図 3 のようになっている．

4.1 実 験 目 的

本実験ではオフチェーン DB のデータの解析において，解析に使用するデータの改ざんの有無を検証し，データが改ざんされていた場合はブロックチェーン上に保存されている正しいデータを再抽出するという一連の流れについて，各フェーズにおける所要時間を計測することでハッシュ値の算出および比較によるオーバーヘッドを確認する．また，ハッシュ単位の変化によるオーバーヘッドや特徴について整理・検証する．

4.2 実 験 内 容

本実験では後述するデータセットをブロックチェーン上に格納しオフチェーン DB へと抽出を行った後，5 パターンの改ざん操作をオフチェーン DB に対して行う．その後，テーブル，タプル，カラム，アイテムごとにハッシュ関数を適用しハッシュ値を比較することで改ざんを検知し，改ざんが検知された場合はブロックチェーン上に格納されている L-data から再抽出を行う．この操作を 5 回繰り返し後述する計測フェーズごとに平均所要時間を測ると共に 95%信頼区間を記録した．以下ではデータセット，ハッシュ関数のかけ方，計測フェーズ，改ざんパターンについて順に説明する．

4.2.1　データセット

本実験のデータセットとして半角英数字から成る 8 バイトのランダムな文字列をカラム数が 100，タプル数が 10,000 となるように生成しブロックチェーン上に L-data として格納する．なお，本田ら [14] や山田ら [15] が行った電子カルテの解析において，解析に最も重要なテーブルのカラム数がおよそ 100 であったことからこの値を採用した．また，通常の電子カルテにおいて 10,000 タプルは追加されるデータの 2〜3 日分に相当する．

4.2.2　ハッシュ関数のかけ方

本実験ではハッシュ関数として SHA-512 を採用した．解析プログラムは Node.js によって記述されており，オフチェーン DB に SQL 文を実行するとオブジェクトとしてレスポンスが返ってくる．それぞれのハッシュ単位で取得したオブジェクト内のデータを文字列に整形し，その文字列に対し SHA-512 のハッシュ関数を適用している．

4.2.3　計測フェーズについて

本実験では以下の 12 フェーズの所要時間を計測した．

- **HL prep**：解析プログラムがブロックチェーンネットワークに接続するために必要な処理．
- **Extract L**：ブロックチェーン上の L-data の内，解析に必要なデータを読み出す処理．
- **Hash L**：読み出した L-data にハッシュ関数を適用し，ハッシュ値を求める処理．
- **Extract OFF**：オフチェーン DB に保存されている解析に必要なデータを読み出す処理．
- **Hash OFF**：読み出したオフチェーン DB 上のデータにハッシュ関数を適用し，ハッシュ値を求める処理．
- **Comp hash 1**：Hash L と Hash OFF のフェーズで求めたハッシュ値が一致するかを検証する処理．
- **Extract K/C**：オフチェーン DB の Key およびカラム情報をまとめたものを読み出す処理．
- **Extract T**：ブロックチェーン上の T-data の内，現在解析しているデータが含まれるものを読み出す処理．
- **Hash K/C**：読み出した Key およびカラム情報にハッシュ関数を適用し，ハッシュ値を求める処理．
- **Hash T**：読み出した T-data にハッシュ関数を適用し，ハッシュ値を求める処理．
- **Comp hash 2**：Hash K/C と Hash T のフェーズで求めたハッシュ値が一致するかを検証する処理．
- **Update OFF**：改ざんされたデータの再抽出処理

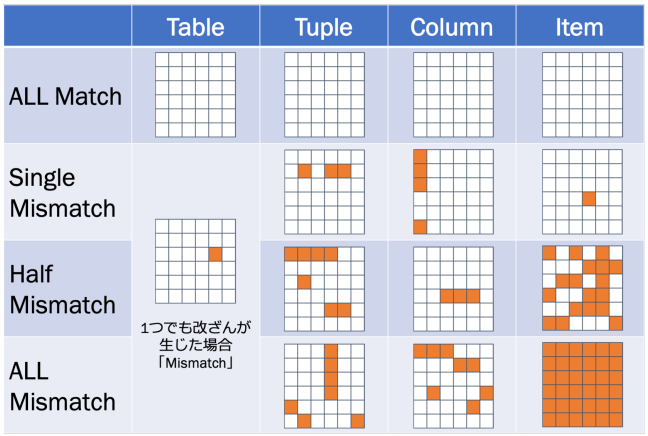

図 4　改ざんパターン

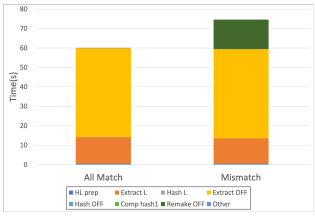

図 5　所要時間 (テーブル)

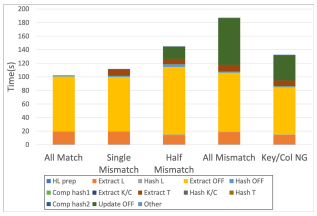

図 6　所要時間 (タプル)

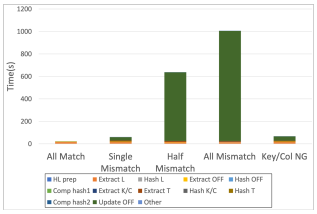

図 7　所要時間 (カラム)

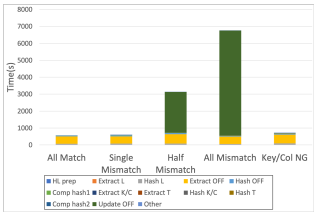

図 8　所要時間 (アイテム)

4.2.4　改ざんパターン

本実験ではデータセットをブロックチェーン上に格納しオフチェーン DB へと抽出を行った後，以下の 5 パターンの改ざん操作をオフチェーン DB に対して行う．

- **All Match**：データの改ざんは検知されない．
- **Single Mismatch**：Comp hash 1 フェーズにおいて 1 回のみ改ざんが検知される．
- **Half Mismatch**：Comp hash 1 フェーズにおいて半数に相当する回数改ざんが検知される．
- **All Mismatch**：全ての Comp hash 1 フェーズにおいて改ざんが検知される．
- **Key/Col NG**：Comp hash 2 フェーズにおいて改ざんが検知される．

なお，All match，Single Mismatch，Half Mismatch，All Mismatch については Comp hash 2 フェーズでの改ざんは検知されない．また，テーブルごとにハッシュ関数を適用した場合は解析を行う任意のデータに対し 1 箇所でも改ざんが生じると全データの再抽出を行うため，まとめて「**Mismatch**」と表現する．なお，**Half Mismatch** のパターンは「**Partial α Mismatch**」の内，$\alpha = 50\%$ とした時の具体例でありパラメータ α は自由に変更ができる．

図 4 はカラム数 6，タプル数 6 のオフチェーン DB に対しテーブル，タプル，カラム，アイテムごとにハッシュ関数を適用した際の All Match，Single Mismatch，Half Mismatch，All Mismatch の改ざん状況を表したものである．オレンジ色に塗られたデータが改ざんされている様子を示している．

4.3　実験結果

All match，Single Mismatch，Half Mismatch，All Mismatch，Key/Col NG の改ざんパターンについて，ハッシュ関数の対象をテーブル，タプル，カラム，アイテムと変化させた際の各フェーズの所要時間およびその合計をそれぞれ図 5，図 6，図 7，図 8 に示す．

4.4　考　　察

ハッシュ関数の対象をテーブル全体とした際は，改ざんが生じた際にどのデータが改ざんされたかを把握する必要がないため，T-data のハッシュ値とオフチェーン DB の Key およびカラム情報のハッシュ値を算出・比較する手順が省かれる．そのため，他のハッシュ単位と比べておよそ 30 秒ほどの処理時間が短縮されることが分かった．10,000 タプル程度のデータ量であれば 1 分程度で改ざん検知のフェーズが終了し，どのような改ざんであっても 15 秒程度でオフチェーン DB の再抽出が完了するため比較的優れたハッシュ単位と言える．

ハッシュ関数の対象をタプル単位とした際は，どのタプルのデータが改ざんされているかを把握することができるため再抽出を効率化することに成功している．実際，Single Mismatch の改ざんでは Update OFF のフェーズにおよそ 10ms ほどしかかかっていない．しかし，Extract L のフェーズにおいては Select の SQL 文を where 句を用いて指定し，タプルの数だけクエリを投げる必要があるため 80 秒程度かかってしまった．タプル数がさらに少ない小規模なオフチェーン DB においては効果を発揮すると考えられる．

ハッシュ関数の対象をカラム単位とした際は，改ざんの検知が完了するまでの時間が他のハッシュ単位と比べて最も早く 20 秒程度であった．これは RDB においてカラム数は頭打ちしてしまうため，大規模データにおいても同様な結果が得られると考えられる．しかし，改ざんが生じた際の再抽出のフェーズでは Single Mismatch の改ざんにおいてもタプル数の数だけ Update の SQL クエリを投げる必要があり，ハッシュ比較において不一致の回数が 1 回増えるごとに Update の必要なクエリの回数がタプル数の分だけ増えてしまう．All Mismatch の改ざんにおいては再抽出のみで約 1,000 秒ほど時間を要するため，外部からの攻撃対象となりにくく改ざんが起こる確率が低いオフチェーン DB においては効果的なハッシュ単位と考えられる．

ハッシュ関数の対象をアイテム単位とした際は，データが改ざんされているかどうかを 1 つずつ検証するため他のハッシュ単位に比べ大幅に所要時間が必要であった．また，改ざんされ

表 6　ハッシュ単位の特徴

改ざん検知数	テーブル単位	タプル単位	カラム単位	アイテム単位
無し	Good	Not Good	**Best**	Worst
少ない	Good	Not Good	**Best**	Worst
中程度	**Best**	Not Good	Worst	Worst
多い	**Best**	Not Good	Worst	Worst

たデータ数が少ない場合は Update OFF のフェーズにおける所要時間はほぼ無視できるが，改ざんされるデータ数が増えると所要時間はカラム単位よりも必要となり，全データが改ざんされている場合は再抽出に 100 分程度かかってしまった．そのためハッシュ関数をかける単位としては適していないと言える．

以上をまとめると，改ざんが生じていなかったり、改ざん検知数が少ない場合はカラム単位が最適なのに対し、改ざん検知数がデータ全体の内の中程度より多くなる場合はテーブル単位がハッシュ対象として最も効果的であることが分かった。また，タプル単位やアイテム単位に関してはハッシュ単位としてはあまり効果的ではないと言える．表 6 はこれらの特徴をまとめたものである．

5 おわりに

5.1 まとめ

本研究ではブロックチェーン上のデータを解析する際に使用するオフチェーン DB のデータの改ざんをブロックチェーン上のデータとオフチェーン DB 上のデータの両者のハッシュ値を算出・比較することで検知する手法を提案し，改ざんが生じていた場合はブロックチェーンからデータを再抽出することで解析における結果保証を実現した．また，Hyperledger Fabric を用いて提案手法を実装し，ハッシュ関数をかける対象をテーブル，タプル，カラム，アイテムと変化させ，5 種類の改ざんパターンについて実験を行いオーバヘッドや特徴を定量的かつ定性的に評価した．そして改ざんされている割合や改ざんのタイプなどによって最も効果的なハッシュ関数のかけ方が変化することを示した．

5.2 今後の課題

今後の課題としてはまず，大規模データでの実験を行うことが挙げられる．本研究で用いたデータセットはデータ量が 8MB と実用的なデータベースにおけるデータセットとは言えない．これは Hyperledger Fabric での実装の際にタイムアウト設定が設けてあり，その時間が 30 秒であった．この制約の元で実験をする必要があったためデータセットが最大でも 8MB 程度となってしまった．タイムアウトの設定はブロックチェーンネットワークへの接続やチェーンコードの呼び出しなど様々な場面で設けてあったため，どの種類でタイムアウトが生じているかを特定し，長時間に渡ってプログラムを実行できる状況を実現することが必要となる．

また，実用的なシステムアーキテクチャでの実験も行うべきと考える．本研究での実験はマルチクライアントサーバーなどの環境では行なっていない．また Hyperledger Fabric におけるネットワーク構成もノード数が 2 と小規模なものとなっている．そのため多くのユーザーがネットワークに接続し，マルチサーバー間でのトランザクションが大量に発生しているような現実的な環境においては，本研究での提案手法における所要時間などのオーバーヘッドにも影響すると考えられ，これらを今後検証していく必要がある．

謝　辞

本研究の一部は，日本学術振興会科学研究費補助金基盤研究 (B)(#20H04192) の助成により行われた．関係者の協力に感謝する．

文　献

[1] 萱原正彬, 本田祐一, 山田達大, Hieu Hanh Le, 串間宗夫, 小川泰右, 松尾亮輔, 山崎友義, 荒木賢二, 横田治夫. ブロックチェーンとプロキシ再暗号化を用いた共有範囲設定可能な医療情報管理. 第 11 回データ工学と情報マネジメントに関するフォーラム, 2019.

[2] Blockchain.com. `https://blockchain.com`.

[3] Kwok-Bun Yue, Karthika Chandrasekar, and Hema Gullapalli. Storing and querying blockchain using sql databases. *Information Systems Education Journal*, Vol. 17, No. 4, p. 24, 2019.

[4] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 30, No. 7, pp. 1366–1385, 2018.

[5] Hoang Tam Vo, Ashish Kundu, and Mukesh K Mohania. Research directions in blockchain data management and analytics. In *EDBT*, pp. 445–448, 2018.

[6] A. Brahma, S. Panigrahi, and J. Mahapatra. Anomaly detection in database using bat algorithm. In *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, pp. 1–5, 2020.

[7] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[8] 日本ブロックチェーン協会. `http://jba-web.jp/`.

[9] Bitcoin core. `https://bitcoincore.org/ja`.

[10] Ethereum. `https://www.ethereum.org`.

[11] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *the proceeding of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, Vol. 310, 2016.

[12] Hyperledger. `https://www.hyperledger.org`.

[13] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 305–319, Philadelphia, PA, June 2014. USENIX Association.

[14] 本田祐一, 山田達大, 萱原正彬, Le Hieu Hanh, 串間宗夫, 小川泰右, 松尾亮輔, 山崎友義, 荒木賢二, 横田治夫. 患者の固有情報及び動的状況を考慮したクリニカルパス分岐要因推定. 第 11 回データ工学と情報マネジメントに関するフォーラム論文集, 2019.

[15] 山田達大, 本田祐一, 萱原正彬, Le Hieu Hanh, 串間宗夫, 小川泰右, 松尾亮輔, 山崎友義, 荒木賢二, 横田治夫. Sid を保持するシーケンシャルパターンマイニングによるクリニカルパスバリアント分析. 第 11 回データ工学と情報マネジメントに関するフォーラム論文集, 2019.