

岐阜大学工学部

電気電子・情報工学科

令和5年度卒業論文

# イーサリアムブロックチェーンを用いた 公文書保管システムに関する研究

三嶋研究室

学籍番号: 1203033012

石上 敬祐

指導教員: 三嶋 美和子 教授

2024年 2月

# 目次

# はじめに

近年, ブロックチェーン技術はその透明性, 改ざん耐性, 分散型の特性を活かし, さまざまな産業や領域での応用が進められている. 中でも, 公文書の保管と管理は, その情報の重要性和永続性から, ブロックチェーン技術の恩恵を受けるべき領域として注目されてきた. 公文書は, 公共の利益を保護するための重要な情報源として機能し, その正確性と信頼性は社会の公正性や透明性を保障する上で不可欠である. しかし, 従来の公文書保管方法は, 物理的なダメージ, 人為的な改ざん, 紛失のリスクなど, 多くの課題を抱えている.

この背景を受け, 本研究では, イーサリアムブロックチェーンを利用した公文書保管システムの実現可能性とその効果を詳細に検討する. また, イーサリアムはスマートコントラクトというプログラマブルなトランザクションの実行機能を持つブロックチェーンプラットフォームであり, この特性を利用して公文書の登録, アクセス, 保護を効率的に行う新しいシステムの `recordex` の提案をする.

提案システムは GCP をフロントエンドの実行環境として, Ethereum Virtual Machine をスマートコントラクトの実行環境として用いる.

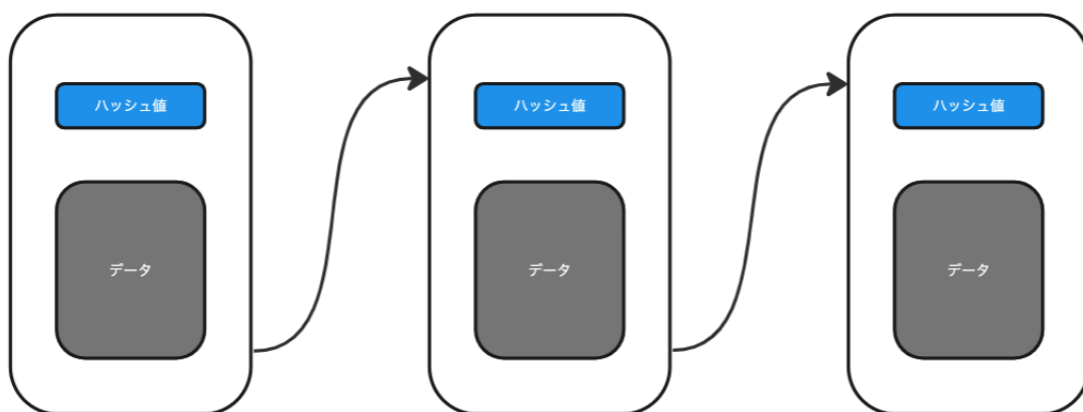
**本論文は n 章で構成される.** 第1章ではブロックチェーン, Ethereum, Ethereum Virtual Machine, スマートコントラクトについて解説する. 第2章では

# 第1章 関連技術

この章では、本研究で提案・実装する公文書保管システムで利用するブロックチェーン、Ethereum, Ethereum Virtual Machine, スマートコントラクトについて解説する。

## 1.1 ブロックチェーン

ブロックチェーンは、データを連続する「ブロック」という単位に分け、それぞれのブロックが前のブロックのハッシュ値（固有の識別値）とナンスと呼ばれる32ビットの値を含む形で連鎖された分散データベースのことを指す。一度書き込まれたデータは、後から変更することが非常に難しくなる特性を持っている。



パブリックブロックチェーンは非中央集権型で誰でも参加可能なネットワークであるため記録されたデータが改ざんされる可能性はゼロではない。しかし、過去に生成されたブロックに記録されたデータを改ざんするためにはそのブロック以降のブロックのハッシュ値も再計算する必要があり、そのような変更は事実上困難と考えられている。

以下ではイーサリアムを例にブロックチェーンの構成要素について、詳細に解説していく。

### 1.1.1 ノード

イーサリアムノードは、イーサリアムネットワークの一部として動作するコンピューターまたはサーバーを指す。各ノードは通常複数のノードと P2P で接続しており、トランザクションの確認やブロックの追加を行う。イーサリアムは結果整合性モデルを採用している分散システムであるため、ある時点では異なるノード間で短期的な非整合が存在する。しかし、全てのノードが最終的には同じデータを保有することとなる。

### 1.1.2 ネットワーク

イーサリアムのパブリックネットワークは、インターネット接続で世界中の誰でもアクセスできる。誰でも公開ブロックチェーン上でトランザクションを読み取りまたは作成し、実行されているトランザクションを検証できる。ピア間のコンセンサスにより、トランザクションとネットワークの状態を追加するかが決まる。

また、イーサリアムにはメインネット、プライベートネット、テストネットの3種類のネットワークが存在する。

- **メインネット**

メインネットは、プライマリ・パブリックのイーサリアム本番環境のブロックチェーンであり、実際の価値を持つトランザクションが分散台帳上で実行されている。

- **プライベートネット**

イーサリアムのメインネットやテストネットに接続されていないネットワークのこと。ここでのプライベートとは保護されていて安全という意味ではなく、先に述べたようにメインネットやテストネットから分離されていることを意味している。

- **テストネット**

テストネットは、プロトコルやスマートコントラクトの開発者が、メインネットへデプロイする前に、実際の運用環境でプロトコルの更新や将来的なスマートコントラクトの双方をテストするためのネットワークである。これは一般のウェブ開発における、本番とステージングサーバと同じようなものと考えられる。

メインネットにデプロイする前に、テストネットで作成したコントラクトコードをテストする必要があり、既存のスマートコントラクトと統合する分散型アプリ(Dapp)では、ほとんどのプロジェクトはコピーがテストネットにデプロイされている。[2]

### 1.1.3 トランザクション

イーサリアムトランザクションとは、コントラクトではなく人間によって管理されたアカウントである外部所有アカウント(EOA)によって開始されたアクションを指す。

EVM の状態を変更するトランザクションは、ネットワーク全体にブロードキャストされる必要があります。すべてのノードは、EVM で実行されるトランザクションのリクエストをブロードキャストできる。その後、バリデータがトランザクションを実行し、結果の状態の変更をネットワークに伝播する。(結果整合性モデル) [3]

イーサリアムネットワーク上でトランザクションが伝播される際、全てのノードが同時にそのトランザクションを認識するわけではない。ネットワークの遅延やノードの接続状態などの要因で、トランザクションの伝播には時間がかかる場合がある。しかし、時間が経てば結局すべてのノードがそのトランザクション情報を持つことになる。また、2つ以上のノードが同時に新しいブロックを発見した場合、短期的にネットワーク上で異なるブロックチェーンのバージョンが存在することがある。しかし、時間が経つと、一つのチェーンが他よりも長くなることで、ネットワーク上のノードはそのチェーンを採用するようになり、結果的に一貫性が達成される。

### 1.1.4 マークルツリー

イーサリアムのブロックには多数のトランザクションが含まれており、これらのトランザクションを効率的に検証するためにマークルツリーが使用される。具体的には、ブロック内の全トランザクションのハッシュ値を元に、マークルルートという一つのハッシュ値を生成する。

このマークルルートはブロックヘッダに格納され、ブロックのデータ整合性を確認する際に使用される。特定のトランザクションがブロックに含まれているかどうかを検証するためには、そのトランザクションのハッシュと、それに関連する一部のハッシュ値（マークルパス）のみで検証することが可能である。これにより、ブロック全体をスキャンすることなく、効率的にデータの確認や検証が行える。

### 1.1.5 コンセンサスアルゴリズム

イーサリアムは2022年9月にコンセンサスメカニズムをプルーフ・オブ・ステーク (PoS)に移行した。

(2022年9月以前に使用されていたコンセンサスメカニズムであるPoSのアーキテクチャの一部であるマイニングの解説はここでは省くことにする。)[6]

PoS における主な参加者は以下の4つである。

- バリデータ
- ステーカー
- ステッキングプール
- デリゲーター

#### バリデータ

新しいブロックの生成とトランザクションの検証を担当する。その際、バリデータは、自分の保有するコインをステークとしてネットワークに預け、その信頼性を保証する。

活動: バリデータは、トランザクションを収集し、ブロックを形成し、それを他のノードに提案、提案されたブロックが正しいと認識されれば、新たなブロックがチェーンに追加される。バリデータはブロックを正確に生成することで報酬を受け取る[7] が、不正行為を行うとステークの一部を失うリスクがある (スラッシング)。[8]

#### ステーカー

ステーカーは、自分のコインをステークとしてバリデータに提供する一般ユーザーである。彼らは直接バリデータとして活動することではなく、ステッキングプールを通じて間接的にネットワークに貢献する。ステーカーは自分のコインをステッキングプールに預け、そのプールが生成するブロックに基づいて報酬を受け取る。

#### ステッキングプール

ステッキングプールは、多数の小規模ステーカーからコインを集め、一つの大きなステークとして機能する。これにより、小規模ステーカーもバリデータのような役割を間接的に果たすことができる。[9] プールのオペレーターは集めたステークを使ってバリデータとして機能し、ブロックを提案する。プールは生成したブロックに対する報酬をプール参加者に分配。ただし、プールが不正行為を行った場合、そのペナルティも分配されることになる。

## 1.2 イーサリアム

2009年にサトシ・ナカモトが開発したビットコインの「分散型コンセンサスのツールとしての基盤となるブロックチェーン技術」に注目したヴィタリック・ブテリンが開発を始めたブロックチェーンがイーサリアムである。ビットコインとイーサリアムとの違いは**スマートコントラクトというデジタル資産を任意のルールを実装したコードで直接制御するより優れた基盤がある、スマートコントラクトという一種のアプリケーションを実行するための非中央集権化された分散型のプラットフォームがある**という2点である。[10]

ブロックチェーンとしての基本的な概念は1.1、コンセンサスアルゴリズムは1.1.5の解説と同じであるため、本説ではスマートコントラクトとそのスマートコントラクトの実行基盤である分散型アプリケーションプラットフォームについて解説する。

### 1.2.1 Ethereum Virtual Machine (EVM)

Ethereum Virtual Machine (EVM) とはイーサリアムクライアントを実行する数千の接続されたコンピュータによって維持される 1 つのエンティティとして存在する、分散型の状態マシンである。

イーサリアムではスマートコントラクトという機能のために、全アカウントとその残高を保持するだけでなく、予め定義されたルールに従ってブロックごとに変化し、任意のマシンコードを実行できるマシンの状態を保持する必要がある。このマシンの状態を保持しているイーサリアムネットワークに参加している、イーサリアムクライアントを実行するためのコンピュータの集まりを Ethereum Virtual Machine (EVM) と呼ぶ。[11]

### 1.2.2 ガス

ガスとは、イーサリアムネットワーク上で特定の操作を実行するために必要な計算労力を測定する単位のことである。イーサリアムでは、トランザクションを実行するには計算リソースが必要であり、そのリソースの対価として料金を支払う必要がある。これにより、イーサリアムはスパム攻撃に対する脆弱性を解消し、無限の計算ループに陥ることを防ぐ。計算料金は、ガス代として支払われる。

ガス代は、操作のガス使用量に、ガス単位当たりのコストを乗じた額となっている。この料金は、トランザクションが成功したか失敗したかに関係なく支払われる。

支払うガス代の合計は下の 2 つのコンポーネントに分けられる。

- **base fee**

トランザクションをブロックに含めるのに必要な最低料金。ブロックごとに料金が料金が変わる。ベースフィーは現在のブロックとは別個に計算され、前のブロックによって決定されます。これにより、おおよそのトランザクションフィーを予測できる。ブロックが作成されるとこのベースフィーは「バーン」され、流通から削除される。

- **priority fee (チップ)**

バリデータへ支払われるチップ。競合するトランザクションよりも高いチップを支払うことで、同じブロック内の他のトランザクションよりも優先的にトランザクションが実行されるようになる。

[12]

### 1.2.3 トランザクション

イーサリアムトランザクションとは、コントラクトではなく人間によって管理されたアカウントである外部所有アカウント(EOA)によって開始されたアクションを指す。特に、EVM の状態を変更するトランザクションはネットワーク全体にブロードキャストされる必要がある。すべてのノードは、EVM で実行されるトランザクションのリクエストをブロードキャストできる。この後、バリデータがトランザクションを実行し、結果の状態の変更をネットワークに伝播する。

トランザクションにはフィー(手数料)が必要であり、これが 1.2.2 で解説したガスである。

送信されたトランザクションには次の情報が含まれる。

- **from**  
送信者のアドレス。なお、コントラクトアカウントはトランザクションを送信できない。
- **recipient**  
受信アドレス (外部所有のアカウントの場合、トランザクションは値を転送する。コントラクトアカウントの場合、トランザクションはコントラクトコードを実行する。)
- **signature**  
送信者の識別子。送信者の秘密鍵がトランザクションに署名し、送信者がこのトランザクションを承認したときに生成される。
- **nonce**  
連続的に増加するカウンターで、アカウントから送信されるトランザクション番号を示す。
- **value**  
送信者から受信者に送金する ETH の量 (WEI 単位: 1ETH は 1e+18wei と等価)。
- **input data**  
任意のデータを含むオプションのフィールド。
- **gasLimit**  
トランザクションで消費できるガスユニットの最大量。EVMが各計算ステップで必要なガス単位を指定する。
- **maxPriorityFeePerGas**  
バリデータへのチップとして含める際に消費されるガス代の上限。
- **maxFeePerGas**  
トランザクションに対して支払う用意があるガス単位あたりの最大手数料 (baseFeePerGasとmaxPriorityFeePerGasを含む)

[13]



### 1.2.4 スマートコントラクト

スマートコントラクトは、契約条件が満たされたときに自動的に実行するコンピュータコードに契約をデジタル化する。

従来の契約における最大の問題の1つは、信頼できる個人が契約事項に遂行する必要があることである。たとえば、競輪の賭けに勝ち、契約の条件が満たされたとしても、他者が契約を履行すると（つまり、掛け金を支払う）信頼しなければならない。しかし、スマートコントラクトでは誰かを信頼する必要はなく、契約条件が満たされたときに結果が自動的に実行される。

[14]

スマートコントラクトは1度デプロイすると、デプロイ時に作成されたコントラクトアドレスに紐づけられた処理は修正できないため、注意が必要である。（現代のアプリ開発でよく採用される、アジャイル開発[15]やトランクベース開発[16]などの頻繁にリリースが行われる開発手法と比較した場合。）

本研究では、公文書保管システムのためのスマートコントラクトを構築する。

### 1.2.5 イーサリアムの課題




現在のイーサリアムには主に次の3つの課題がある。

- スケーラビリティ
- 手数料（ガス代の高騰）
- 新たなブロックが生成されるまでの時間

イーサリアムのユーザー規模が拡大するにつれ、イーサリアムブロックチェーンの処理能力は限界に達しつつある。これによりイーサリアムネットワークの使用コストが増加しているため、シャーディング[17]やレイヤー2[18]などの「スケーリングソリューション」の必要性が高まってきている。

[19]

また、イーサリアムは Solana[20] や TON Coin[21] のようなブロックチェーンと比較すると、新たなブロックが生成に時間がかかることも課題の1つである。

	 TON	 Ethereum	 Solana
Block time ①	5 sec.	12 sec.	1 sec.
Time-to-finality ①	Under 6 sec.	10–15 min. ②	6.4 sec.
Simple transaction performance ①	High	Potentially high	High
Complex transaction performance ①	High	Low	Very low ②
Sharding support ①	Max. 2 <sup>60</sup> shards per workchain	Max. to 2 <sup>6</sup> shards	None
Cross-shard communication ①	Near-instant	Slow time-to-finality	None

Comparing TON, Ethereum, and Solana [22]

## 第2章 recordex のプロトコル

本研究では、公文書がいつ、誰によって変更されたかをブロックチェーン上に記録するプロトコルの提案を行う。このプロトコルを通じて、公文書がいつ、誰によって、どこを変更したかを正確に記録できるように、そして、その変更履歴を誰もが追うことができるようになることを目的とする。

提案システムにおけるエンティティは「文書の記録者」、「記録コントラクト」、「文書のアップロード者」の3者に分類され、それぞれがイーサリアムアカウントを使ってシステムを利用する。これら3種類の利用者はブラウザを通じて各コントラクトへの情報の記載や取得を行う。文書の記録者はブラウザを通じてブロックチェーンへのファイルハッシュ値の書き込み、ドキュメントファイルをオブジェクトストレージに保存する。利用者の関係を図2.1に、各利用者の役割を表2.1に、記録コントラクトの概要を表2.2に示す。

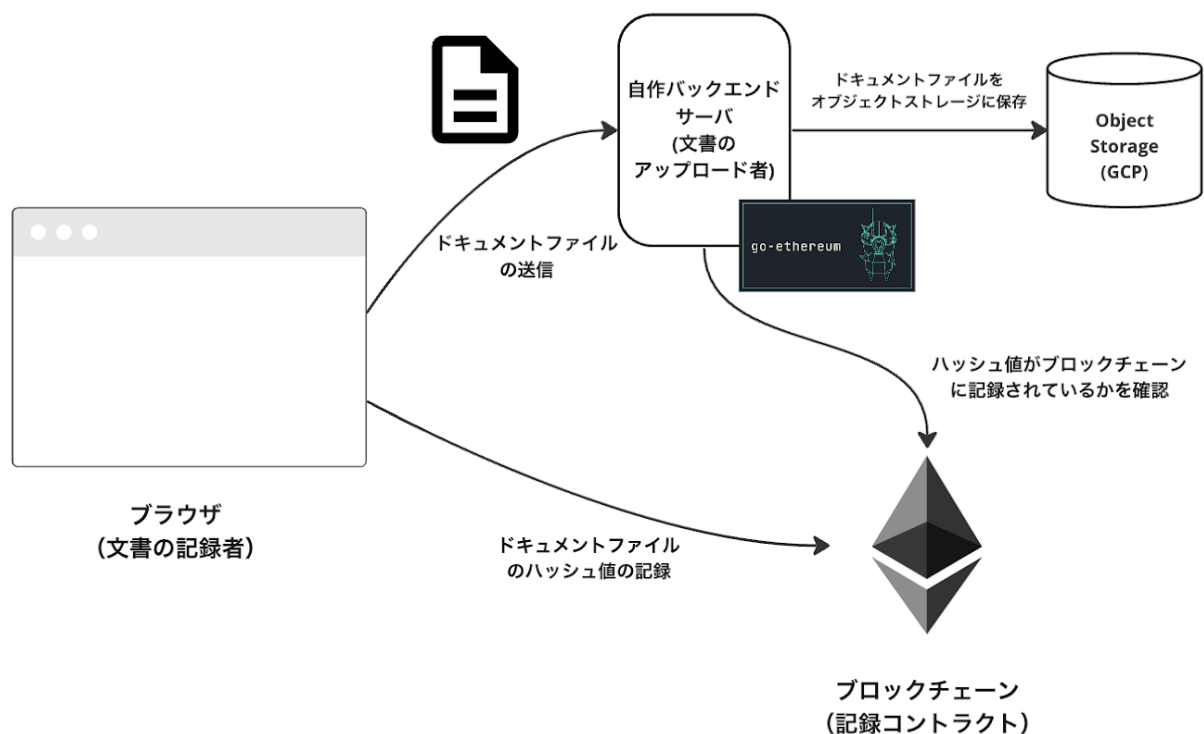


図2.1: 利用者の関係

表2.1: 各利用者の役割

利用者	役割
文書の記録者	文書ファイル本体をバックエンドに、文書のハッシュ値を記録するトランザクションを記録コントラクトに送る。
記録コントラクト	文書の記録者が作成したトランザクションを検証し、ファイルのハッシュ値をブロックチェーンに記録する。
文書のアップロード者	文書の記録者から送られてきたファイルのハッシュ値がブロックチェーンに記録されていることを確認し、ファイルをオブジェクトストレージに保存する。

表2.2: 記録コントラクト概要

利用者	役割
権限管理コントラクト	記録コントラクトの各処理を実行できるアカウントを管理する.
ハッシュ値記録コントラクト	文書のハッシュ値をブロックチェーンに記録する.

## 2.1 文書の記録者サイドの処理

文書の記録者サイドで行う処理は以下の3つの処理からなる (図2.2 参照).

1. 文書ファイルのハッシュ値を求める
2. 求めた文書ファイルのハッシュ値を記録コントラクトに送る
3. 文書ファイル本体を自作バックエンドサーバに送る

### 1. 文書ファイルのハッシュ値を求める

ブラウザで SHA-256 ハッシュ関数を用いて文書ファイルのハッシュ値を求める.

### 2. 求めた文書ファイルのハッシュ値を記録コントラクトに送る

求めたハッシュ値を記録コントラクトに送る.

### 3. 文書ファイル本体を自作バックエンドサーバに送る

文書ファイル本体をバックエンドサーバに送り, オブジェクトストレージに文書ファイルを保存する.

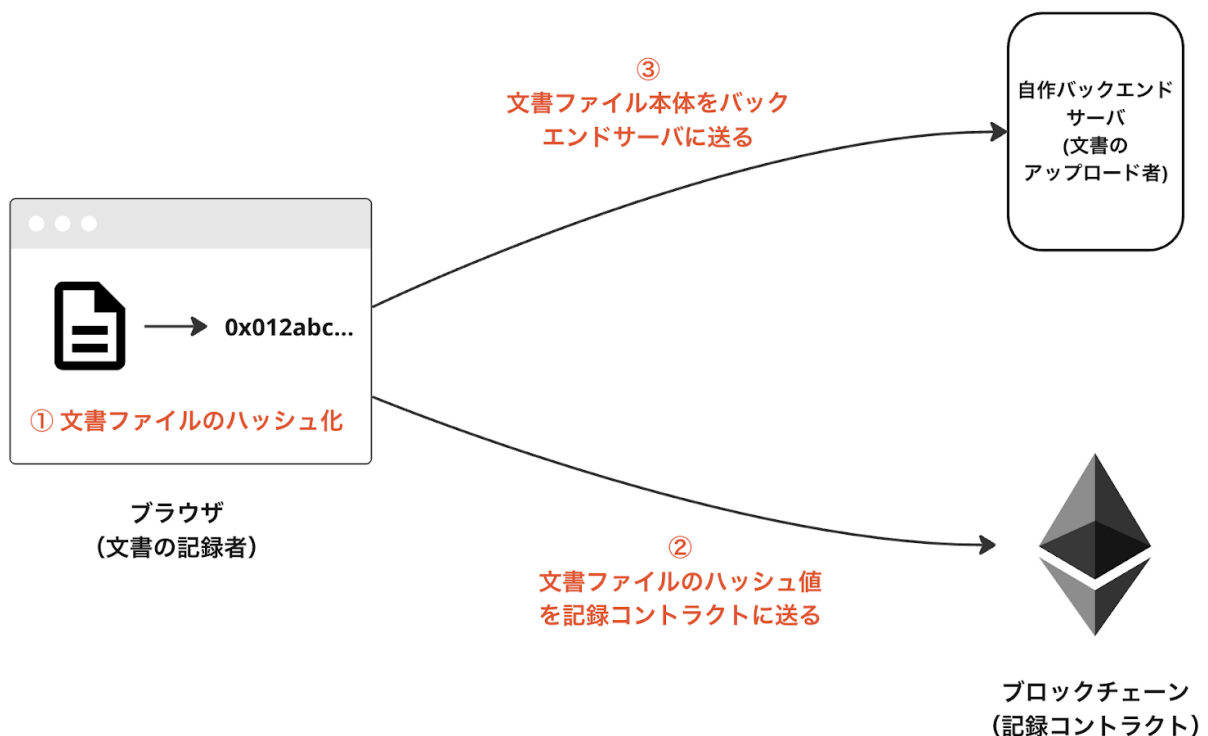


図2.2: 文書記録者サイドでの処理

## 2.2 記録コントラクトの処理

記録コントラクトで行う処理は以下の3つの処理からなる (図2.3 参照).

1. ブロックチェーンにハッシュ値を記録する権限があるかを確認
2. ファイルハッシュ値の履歴にコンフリクトが起きていないかを確認
3. ファイルのハッシュ値をブロックチェーンに書き込む

### 1. ブロックチェーンにハッシュ値を記録する権限があるかを確認

トランザクションを送信したイーサリアムアドレスが、ブロックチェーンにファイルハッシュ値を書き込む権限があるかを確認. もし権限を持っていない場合、トランザクションの revert を行う.

### 2. ファイルハッシュ値の履歴にコンフリクトが起きていないかを確認

1つ前のファイルバージョンのハッシュ値が正しく指定されているかを確認. もしファイルハッシュ値が正しく指定されていない場合、トランザクションの revert を行う.

### 3. ファイルのハッシュ値をブロックチェーンに書き込む

ファイルのハッシュ値をブロックチェーンに書き込み、ファイルの最新バージョンを更新する.

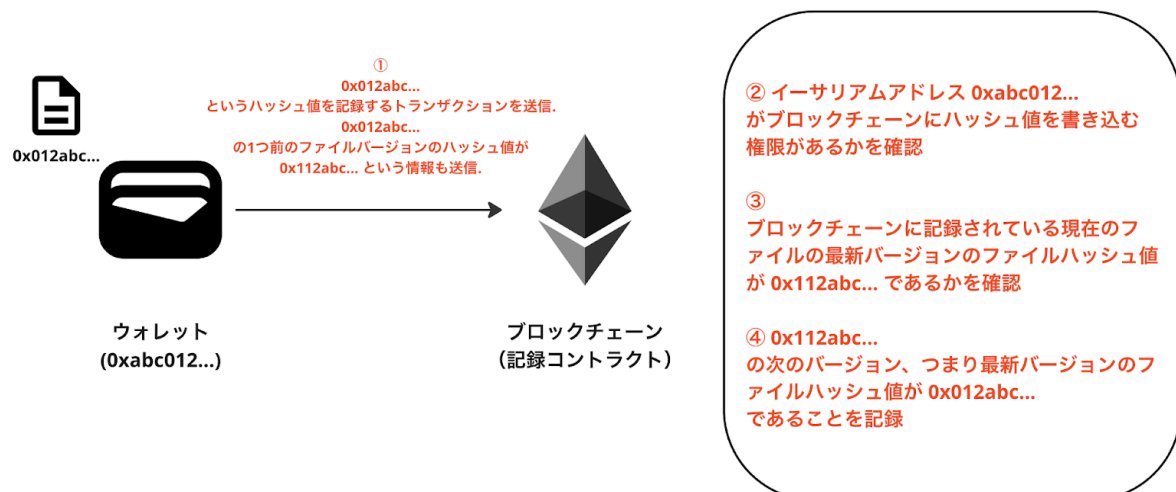


図2.3: 記録コントラクトでの処理

### 2.2.1 ファイルハッシュ値の履歴にコンフリクトが起きていないかを確認する理由

ファイルのハッシュ値の履歴を正確に追うには下の2つの制約を満たす必要があり、その中でファイルのハッシュ値にコンフリクトが起きていないかを確認する必要がある。

1. ファイルハッシュ値の履歴を記録する際に、バージョン1のファイルを変更したものがバージョン2のファイルを変更したものと扱われてはいけない (図2.4 参照)。つまり、ファイルバージョン1のファイルを変更したファイルはバージョン2として扱われるべきで、バージョン3として扱われてはいけないのである。
2. バージョン1を変更した結果として、バージョン2のハッシュ値をブロックチェーンに記録しようとしたとき、すでにバージョン2がブロックチェーンに存在しているならば、バージョン2のハッシュ値が2つ存在してしまうことを防ぐ (ハッシュ値のコンフリクトを防ぐ) べきである。つまり、新たなバージョン2のハッシュ値を記録するトランザクションは revert されるべきである。

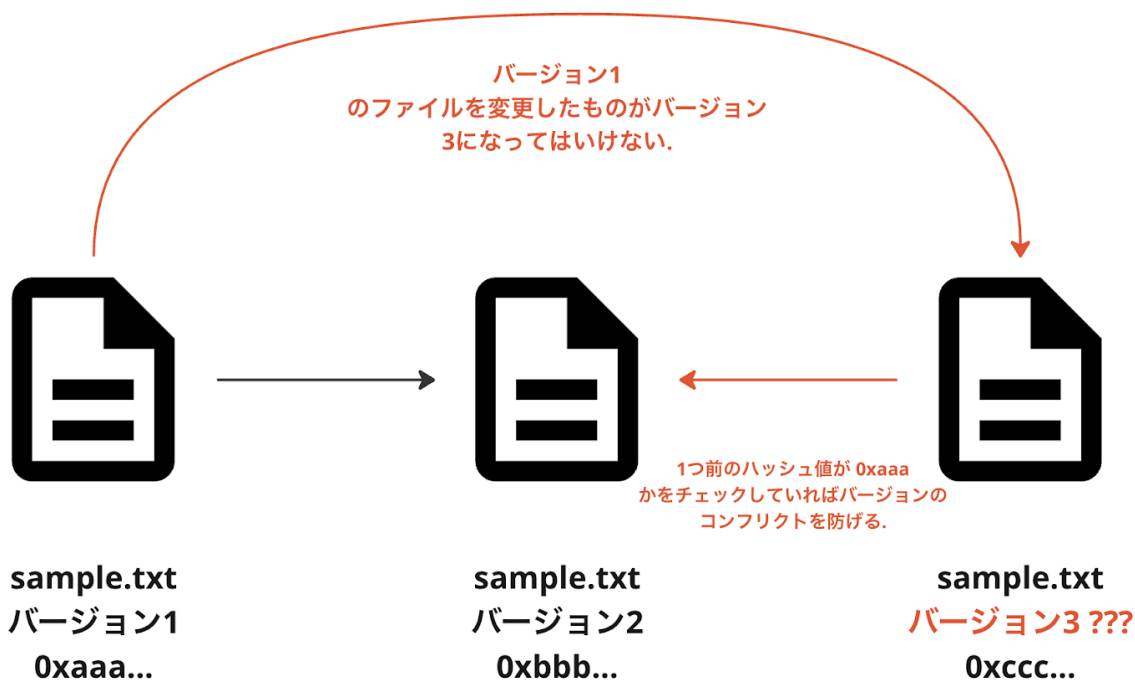


図2.4: バージョンのコンフリクト例

## 2.3 文書のアップロード者の処理

文書のアップロード者サイドで行う処理は以下の2つの処理からなる (図2.5 参照)。

1. ブラウザから送られた文書ファイルのハッシュ値がブロックチェーンに記録されているかの確認
2. ドキュメントファイル本体をオブジェクトストレージに保存

### 1. ブラウザから送られた文書ファイルのハッシュ値がブロックチェーンに記録されているかの確認

ブラウザから「文書ファイル」と「文書ファイルのハッシュ値をブロックチェーンに記録するトランザクションの ID」を送ってもらう。文書ファイルのハッシュ値を記録するトランザクションがイーサリアムブロックチェーンのブロックに取り込まれたことを確認した後、

そのトランザクションに記録されている文書ファイルのハッシュ値とブラウザから送られた文書ファイルのハッシュ値が同じかどうかを確認する。

## 2. ドキュメントファイル本体をオブジェクトストレージに保存

ドキュメントファイル本体をオブジェクトストレージに保存する。

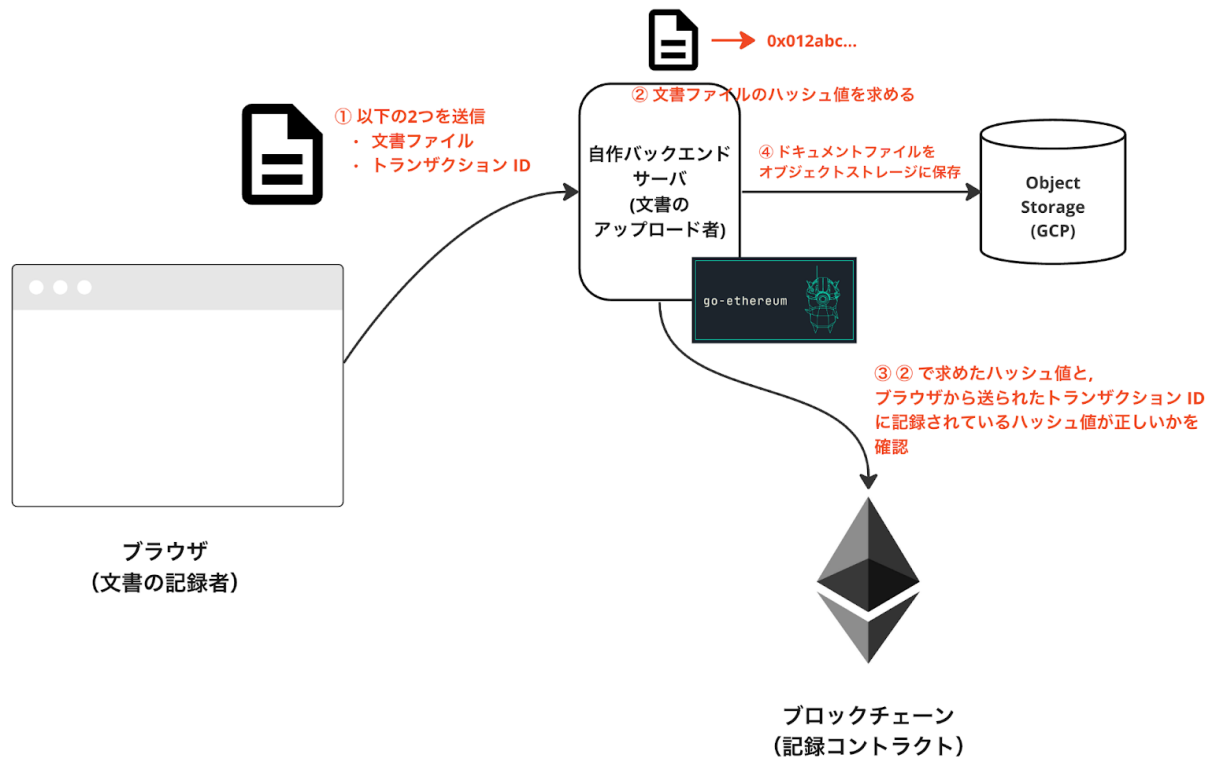


図2.5: 文書のアップロード者の処理

## 第3章 recordex の実装

aaabbb

# 参考文献

[1] マスタリングイーサリアム

<https://www.oreilly.co.jp/books/9784873118963/>

[2] ネットワーク

<https://ethereum.org/ja/developers/docs/networks/>

[3] トランザクション

<https://ethereum.org/ja/developers/docs/transactions/>

[4] プルーフ・オブ・ステーク

<https://ethereum.org/ja/developers/docs/consensus-mechanisms/pos/>

[5] マイニング

<https://ethereum.org/ja/developers/docs/consensus-mechanisms/pow/mining/>

[6] マージ

<https://ethereum.org/ja/roadmap/merge/>

[7] Proof of Stake における報酬

<https://ethereum.org/ja/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/#rewards>

[8] スラッシング

<https://ethereum.org/ja/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/#slashing>

[9] ステーキングプール

<https://ethereum.org/ja/staking/>

[10] Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.  
By Vitalik Buterin (2014).

[https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum\\_Whitepaper\\_-\\_Buterin\\_2014.pdf](https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf)

[11] Ethereum Virtual Machine (EVM)

<https://ethereum.org/ja/developers/docs/evm/>

[12] ガスとフィー(手数料)

<https://ethereum.org/ja/developers/docs/gas>

[13] トランザクション

<https://ethereum.org/ja/developers/docs/transactions/>

[14] スマートコントラクト入門

<https://ethereum.org/ja/smart-contracts>

[15] アジャイル

<https://www.nri.com/jp/knowledge/glossary/lst/aa/agile>

[16] トランクベース開発

<https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development?hl=ja>

[17] シャーディング

<https://ethereum.org/ja/developers/docs/scaling#sharding>

[18] レイヤー2

<https://ethereum.org/ja/developers/docs/scaling#layer-2-scaling>

[19] スケーリング

<https://ethereum.org/ja/developers/docs/scaling>

[20] Solana

<https://solana.com/ja>

[21] TON Coin

<https://ton.org/>

[22] Blockchain analysis

<https://ton.org/analysis>



