Vol.2021-DPS-186 No.38
Vol.2021-CSEC-92 No.38
2021/3/15

分散台帳に従うデータ流通制御に向けたオフチェーンデータの紐づけ方式

大橋 盛徳[1,a)]　藤村 滋[1]　中平 篤[1]

概要：特定のファイルではなく、ある関係者間で生成されるデータファイル全体を管理するために、分散台帳とオフチェーンデータとの新しい紐づけ方式を考案した。本方式は、カメレオンハッシュの同一ハッシュを生成する特徴を利用した ID とその ID 検証方法および、分散台帳上に記録した制御情報へのオフチェーンデータを起点としたアクセス方法を含む。本方式を使うことで、オフチェーンデータの更新に際し、オフチェーンのみの処理で制御情報との紐づけを更新することができる。Ethereum を使い、本方式の実現性を検証し、データファイルの生成や更新のプロセスにおいて、紐づけ更新およびその検証が可能であることを確認した。

キーワード：分散台帳、ブロックチェーン、データ流通制御、ファイル流通制御、カメレオンハッシュ、Ethereum

1. Introduction

ブロックチェーン技術 [1][2] は仮想通貨の基盤として登場して以来様々な分野での利用が検討され、将来有望な技術と見られてきた [3][4][5][6]。ブロックチェーン技術の活用分野は様々であるが、その多くのユースケースに共通する構造がある。ブロックチェーンの台帳上で、メタデータを扱い、ブロックチェーン外でそのメタデータに対応するデータファイルを取り扱うという構造である。この構造をとる初期のユースケースに Proof of Existence(PoE)[7] がある。PoE はデータファイルのハッシュ値をとり、そのハッシュ値をデータファイルの ID としてブロックチェーンに記録することで、存在の証明を行うものである。ハッシュ値やそれに類する値（フィンガープリント）をデータファイルの ID としてブロックチェーンのメタデータに記録する紐づけ方式は、ブロックチェーンを使ったデータ管理の基本形となっている。

ブロックチェーン上に記録したメタデータとデータファイルとのハッシュ値やそれに類する値を ID とした紐づけは、複数の類似データファイルの中から正規とするデータファイルを指定し、確定する効果がある。そのため、PoE のようなデータファイルの存在証明のほかにも、ゲームアイテム [8] や美術品 [9] など唯一性を持つデジタルコンテンツをデジタルアセットとして流通させる場合や、貿易取引 [10] などのように複数プレイヤーの間で、正規とするデータファイルを指定しながらプロセスを進める場合で有益である。

一方で、ハッシュ値を ID とした紐づけの方法は、正規とするデータファイルを一部のある特定ファイルだけに固定しにくい場合や固定する意義が薄い場合には有効とは言えない。例えば、動画コンテンツの制作過程のように、データファイルに編集や加工を加えながら、複数の関係者間でデータファイルを流通させていく場合は、ある特定のファイルではなく、ある関係者間で生成されるデータファイル全体をまとめて管理することが求められる。

生成されるデータファイル全体を管理するためには、データファイルが更新されるたびにメタデータとの紐づく ID を修正する必要がある。しかしながら、ブロックチェーンは台帳更新において合意形成が必要であり、その性能は Public 型ブロックチェーンであるほど一般的に遅くなる [11]。さらに登録にコストがかかることや、登録したログにより活動実態が他者に明らかになることへの懸念から、すべてのデータファイルを登録するインセンティブが働きにくく、登録されないデータファイルが生み出されてしまう問題が予想される。

我々は、生成されるデータ全体を管理するために、データをアセットとして管理する従来の仕組みの考え方を変えて、台帳に記載した制御ポリシーであるメタデータを参照

[1] 日本電信電話株式会社　NTT サービスエボリューション研究所
239-0847 神奈川県横須賀市光の丘１－１
[a)] shigenori.ohashi.ur@hco.ntt.co.jp

して、データを管理する仕組みを検討する。この仕組みは、ブロックチェーンの台帳でアセットととして一部のデータを管理する従来の仕組みを機能的に補うものである。

この仕組みの実現のためには、前述した制御ポリシーであるメタデータとデータファイルとの紐づけの課題のほかに、データのオーナー毎に制御ポリシーを設定する場合を想定した制御ポリシーの発見の仕組みも必要である。

本論文は、前述した背景を踏まえ考案した

(1) ブロックチェーン上のメタデータとブロックチェーン外（オフチェーン）のデータファイルとのカメレオンハッシュを活用した ID による紐づけ方式
(2) 管理用メタデータへのアクセス情報とデータファイルのパッケージ方式

を紹介する。

考案した紐づけ方式は、カメレオンハッシュ [12] の同一ハッシュを生成する機能に着目した。本方式によりオフチェーンでメタデータとデータファイルの紐づけ更新が可能になる。またデータのパッケージ方式は、Ohashi[13] の提案した方式を拡張し、様々なファイル形式に対応した。Ohashi[13] では、メタデータを各主体の管理のもとブロックチェーンに置き、InterPlanetary File System(IPFS)[14] にデータファイルとともにメタデータへのアクセス情報を紐づけて格納することで、データファイルを起点にしたメタデータの発見を可能にしている。

本論文の構成は、次の通りである。2 章で関連研究についてまとめ、3 章で我々が目指す制御ポリシー型のデータ管理の特徴を説明する。4 章で考案方式について解説し、5 章で考案方式に基づく実装システムについて説明する。6 章で考案方式の評価と考察を行い、7 章でまとめを述べる。

2. Related Works

2.1 Blockchain

ビットコイン [1] から始まるブロックチェーン技術は、複数の管理主体による相互監視により不正や改ざんを検出しながら台帳を更新・維持することを可能にした仕組みである。トランザクションや台帳の構造には様々なものが提案されているが、最初のブロックチェーンのユースケースであるビットコインは、ブロックが鎖状につながった台帳構造をしている。ブロック間はハッシュ値により結び付けられ、過去のブロックやそこに収められたトランザクションの書き換えや削除は実質的に難しい。トランザクションの実行権限は、電子署名により検証がされ、対応する秘密鍵を持つユーザ以外は実行させることができない。トランザクションの検証や実行は、ブロックの作成と合意のプロセスで管理主体により行われる。ブロック生成合意プロセスの速度がブロックチェーンネットワークで処理できるトランザクション量に大きく影響する。ビットコインのブロック生成合意プロセスは Proof of Work と呼ばれ、約 10 分に 1 回ブロックが生成される。ブロックにはサイズの上限があり、含められるトランザクションには限りがある。そのため、ビットコインの処理速度は約 5TPS と言われている。分散型アプリケーションの登録ができる Ethereum[2] では、少し性能が改善され 10〜15TPS とされている。ブロックチェーンプラットフォームのスケーラビリティの問題は、広く認知され、スケーラビリティ向上の技術開発も進められている [15]。近い将来解決することも期待されるが、通常のサーバクライアントシステムに比べてブロック生成合意プロセスがあるブロックチェーンや分散台帳技術は、処理性能の面では基本的に劣る。

トランザクションで実行できる処理は、プラットフォームにより異なる。ビットコインでは、基本的に送金処理ではあるが、OP_RETURN というスクリプトコードを使うことで送金以外の情報を書き込むことができる。初期のブロックチェーンの応用研究ではこの OP_RETURN を使いメタデータを埋め込んでいた。Proof of Existence も OP_RETURN を活用した例である。しかし、格納できるデータ量は少なく、送金以外の用途への活用は Ethereum の登場後に大きく広がった。Ethereum では、コントラクトと呼ばれる分散型アプリケーションをユーザが作成し追加することができる。コントラクト毎にアドレスが付与され、コントラクトは互いに区別される。コントラクトは送金と同じくトランザクションにより実行させることができ、情報を埋め込むだけでなく、トランザクションに載せたデータに対する演算も可能になったため、様々な用途への応用が広がった。Ethereum の登場によりブロックチェーンの汎用化が進んだが、データ管理という用途では、データ自体を格納することにはならなかった。その理由は三つ考えられる。

一つ目は、合意プロセスへの高負荷である。台帳の更新には、台帳の更新分をブロックとして生成し合意するプロセスが必要である。この合意プロセスにおいてトランザクションを含むブロックを共有する必要がある。そのため、データファイル本体をトランザクションに格納することは、ブロックの伝送遅延を引き起こすとともに、コントラクト実行の際の装置負荷を大きくするためスケーラビリティを低下させる要因となる。

二つ目は、台帳のサイズの肥大化である。ブロックチェーン技術は基本的に参加者がそれぞれ独立してブロックやトランザクションの検証ができるように台帳データをそれぞれの参加者にレプリケーションしている。例えば、100 ノードからなるブロックチェーンシステムにトランザクションを使用して 1MB のデータを保存した場合、100MB のデータ領域が 1 トランザクションで消費されてしまう。そのため、データファイル本体をブロックチェーンに格納することは非合理的と言える。Public 型のブロックチェーンネットワークでは、格納するデータ量に従い手数料を多

く取る設計となっている場合もある。

三つ目は、データセキュリティである。ブロックチェーンに登録した過去のデータは基本的に書き換えができない。そのため、暗号化したデータファイルの暗号鍵が危殆化したとしても、削除や再暗号化をすることができない。Public 型のブロックチェーンネットワークではなく、限られた参加者で限られた用途で運用されるコンソーシアム型のブロックチェーンネットワークの場合には、用途に合わせネットワークをカスタマイズすることで、これらデータファイル本体をブロックチェーンに格納する場合の三つの影響を低減できるが、なくすことは難しい。

今日では、オフチェーンでの処理を組み合わせる事はブロックチェーンの活用を目指す上で重要な観点となっている [16]。

データファイル本体を格納することには非合理的であるブロックチェーンであるが、データファイルを流通管理するには有効な手段でもある。それはトークンによるデータファイルのアセット化である。

2.2 Token

トークンとは、あるモノが持つ価値をパッケージして流通可能に表現したものであり、アセット化の方法である。トークンはその中にパッケージするものの性質に従い、ファンジブルトークンとノンファンジブルトークンの二つに区別される [17]。ファンジブルトークンは、同種のトークンと相互交換が可能なものを指す。例えば、仮想通貨である。ノンファンジブルトークンは同種のトークンと相互交換が基本的にできないものである。例えば、ゲームのアイテムや車、美術品などの唯一性をもつものが該当する。

ブロックチェーンを使ってデータファイルをアセット化するには、ノンファンジブルトークンが使われる。ノンファンジブルトークンの仕様で有名なものには ERC721[18] があり、トークンの送付やトークンの数の確認、所有者の確認などの基本関数が定義されており、仮想通貨のようにトークンを送受することができるようになっている。

トークンは先に述べたようにある特定データを代替し、流通させることを目的とした概念である。そのため、加工や編集で生成されるデータファイル全体の管理や流通制御を目的とすると、アセット化とは異なるトークンの考え方が必要となる。つまり、ある特定データファイルだけを対象とするアセット型のトークンの使い方だけではなく、データファイル全体を管理制御する制御ポリシー型のトークンの使い方が必要である。我々はトークンをデータの制御を司るポリシーの格納物として使うことを考えている。

2.3 台帳上情報の発見方法

ブロックチェーン上の情報を取得するためには、あらかじめアクセスする対象情報の台帳上の場所やインターフェース情報を知る必要がある。例えば、Ethereum の場合は、コントラクトにアクセスするには、台帳上のアドレスと ABI と呼ばれるインターフェース情報を知ることが必要である [19]。多くのブロックチェーンを使ったユースケースでは、参照先を固定することで、台帳上の情報発見の問題を回避している。しかしながら、データ主体が意識的にデータファイルの管理やその活用に参加する場合、データ主体毎に制御ポリシーを自ら管理したいというニーズが生まれると考えられる。そのため、データ主体毎に制御ポリシーやその格納先が異なる場合にも対応したデータ管理方式が求められる。

データファイル毎に紐づく制御ポリシーが異なることを想定するとデータ利用者がデータファイルを起点に制御ポリシーを発見できる仕組みが求められる。この仕組みの実現方法として、大きく二つ考えられる。一つ目は、データファイルの識別子と制御ポリシーへのアクセス情報との変換システムを介し制御ポリシーを発見する方法である。二つ目は、データファイルに制御ポリシーへのアクセス情報を保持し、そのアクセス情報を使い制御ポリシーを発見する方法である。

一つ目の方法は、DNS（Domain Name Service）に類するシステムとして構築することが考えられる。分散型の DNS には、例えば、Namecoin[20] や Ethereum の ENS（Ethereum Name Service）[21] という実装がある。これらの実装を使うことで、データファイルの識別子から制御ポリシーへのアクセス情報に変換するシステムの構築が期待できる。制御ポリシーがどのブロックチェーンネットワークに保持されていようと、このシステムでデータファイルの識別子から制御ポリシーへのアクセス情報に変換する規則とすることでデータファイル側への情報埋め込みをする必要なく、データファイルから制御ポリシーへのアクセスが実現される。しかしながら、データ主体には DNS に類するシステムへの登録情報と制御ポリシーという複数の情報を管理する手間が増えてしまう。

二つ目がデータファイル側に制御ポリシーへのアクセス情報を保持する方法である。これは Ohashi[13] で採用されている方法である。Ohashi では、IPFS 側にデータファイルとアクセス情報を格納し、それらを繋ぐオブジェクトを追加することで両者を結び付けている。Ohashi の方式はデータファイル側にアクセス情報を保持する仕組みが必要であるが、制御ポリシーの管理だけで済むメリットがある。我々は、この方式を拡張し、IPFS のネットワーク外での流通も想定した方式を考案した。

2.4 カメレオンハッシュ

ハッシュ関数とは、任意の長さのデータから固定長の数値（ハッシュ値）を得るための関数のことを指す。ハッシュ関数、特に暗号学的ハッシュ関数は、その出力の一様

性が高く、別々の入力から同じ値が出力される衝突の発生確率は最小限に抑えられている。カメレオンハッシュとは、トラップドアを持つハッシュ関数またはその出力値のことである。トラップドアを使うことで、容易に同じハッシュ値を出力する入力（衝突条件）を見つけることができる。カメレオンハッシュの実装は Krawczyk and Rabin[12] によって提案されて以降様々な実装が提案されている。実装毎に入力は少し異なるが、カメレオンハッシュのアルゴリズムは、鍵の生成（GK）、ハッシュ値の計算（CH）、衝突条件の発見（CF）の 3 つのフェイズからなる。

GK　ハッシュ鍵 (hk) とトラップドア (td) 対を出力する
CH　ハッシュ鍵 (hk)、メッセージ (m)、乱数 (r) を入力にハッシュ値 (v) を出力する
CF　トラップドア (td)、メッセージ (m')、メッセージ (m) と乱数 (r) のペアを入力に、乱数 (r') を出力する

トラップドアは秘密鍵の役割を持ち、任意のメッセージに対して同じ出力値となる正しい乱数を生成できることが、トラップドアの持ち主の証明となる。

3. 制御ポリシー型データ管理

我々が目指す制御ポリシー型のデータ管理についてアセット型のデータ管理と比較し、特徴を整理する。我々が、アセット型と呼称するのは分散台帳上にトークンを生成し、そのトークンを一つの資産として流通させる方式である。この方式は、資産化したトークンが価値を持ち、お金のように売買できることが特徴である。トークンに価値が見いだされるため、分散台帳上の情報を主、オフチェーンのデータファイルを従とする関係となる。一方で我々が目指す制御ポリシー型のデータ管理は、制御用のポリシー情報を分散台帳上に格納するため、分散台帳上に資産化されるわけではない。あくまでデータ側に価値が見いだされるため、分散台帳上の情報が従、オフチェーンのデータファイルが主となる関係となる。整理すると**表 1** の通りになる。

制御ポリシー型のデータ管理の狙いは、アセット型で管理されるべき最終のデータファイルに至るまでに生成される途中経過のデータファイルも含めて、複数の関係者間で生成されるデータファイル全体を管理することである。各データファイルは新規作成や加工編集、紐づく制御ポリシーの新規作成や変更など絶えず変化し続け、時間を経るほどにデータファイルの総量や、データファイルと制御ポリシーの関係も増え続けることが考えられる。このような場合に対応できる制御ポリシーとデータファイルの紐づけが本論文のテーマである。次章で紹介する考案方式は、本テーマに対応する方式の一つであり、制御ポリシーとデータファイルとの紐づけにおいて、紐づけを維持するためのブロックチェーンへの再登録コストやそのユーザビリティ、活動が他者に明らかになる心理的障壁を問題とし、その軽減を目指したものである。

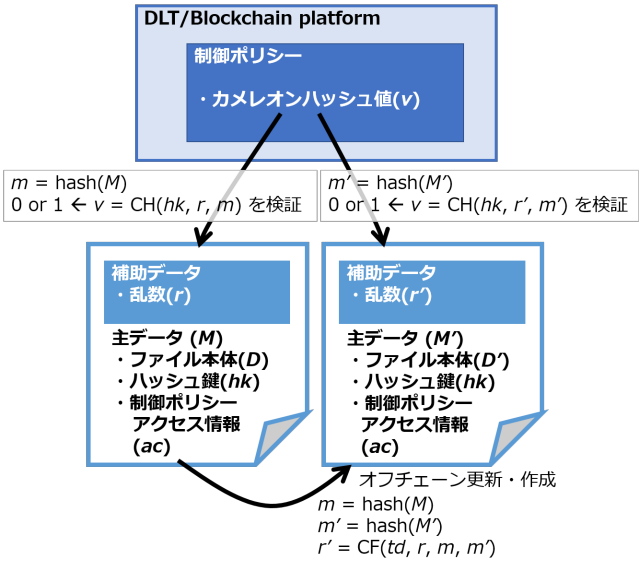

図 1　カメレオンハッシュを使った紐づけ方式概略

4. 考案方式

我々の考案方式は以下 2 つの部分からなる。

(1)　ブロックチェーン上の制御ポリシーとオフチェーンのデータファイルとのカメレオンハッシュを使った ID による紐づけ方式
(2)　制御ポリシーへのアクセス情報とデータファイルのパッケージ方式

これら 2 つの方式について順に説明する。

4.1 ブロックチェーン上の制御ポリシーとデータファイルとのカメレオンハッシュを使った ID による紐づけ方式

本方式は、通常はハッシュ値が ID として使われる制御ポリシーとオフチェーンのデータファイルとの紐づけにカメレオンハッシュを ID として使う方式である。ハッシュ値を ID とするメリットは、暗号学的にデータファイルを起点に算出できることである。カメレオンハッシュも同様のメリットを享受できる。カメレオンハッシュを ID とする場合も、データファイルを入力に暗号学的に ID となるカメレオンハッシュを算出する。

我々の考案方式の概略は**図 1** のようになる。カメレオンハッシュを ID として制御ポリシー側に記録し、ハッシュ鍵や乱数をオフチェーンのデータファイル側に記録する。ここで、主データ (M) の示す範囲には、ファイル本体のほかにハッシュ鍵も含む。ファイル本体 (D) を更新した際には、オフチェーンで乱数の再計算を行い、カメレオンハッシュによる紐づけを確保する仕組みである。

通常のハッシュ値は、同一内容以外はファイルごとに

表 1　アセット型と制御ポリシー型のデータ管理方式の違い

	アセット型	制御ポリシー型
分散台帳上の情報	資産の本体（権利書等）	制御情報（制御ポリシー）
台帳上情報とオフチェーンデータとの関係	主 - 従	従 - 主
方式思想の違い	データの資産化	データの制御

ハッシュ値が変わるため、ファイルの加工や編集によりファイルが変化したり、新たに生み出される度に制御ポリシー側を更新する必要がある。カメレオンハッシュの場合は、トラップドアを持つユーザであれば、ファイル側の更新や編集に対して常に対応する乱数を計算でき、制御ポリシー側の更新をせずに制御ポリシーとの紐づけを確保することができる。分散台帳上の情報をアセットとしてとらえるとオフチェーンのみの操作で紐づけをアップデートする本方式は、改ざん操作になるため、有益ではないが、分散台帳上の情報を制御ポリシーととらえると低コストで紐づけを確保できるメリットを見出せる。次にカメレオンハッシュによる紐づけの生成と更新と検証の流れを述べる。

まず生成の手順を**図 2** に沿って述べる。ユーザは、カメレオンハッシュのハッシュ鍵とトラップドアの対を生成するとともに、制御ポリシーを分散台帳に記録する。ユーザは、制御ポリシーで制御する対象のファイル本体 (D) を生成し、そのファイル本体に加えてハッシュ鍵 (hk) を記録する。主データ (M) のハッシュ値をメッセージ (m) として、乱数 (r) とハッシュ鍵 (hk) からカメレオンハッシュ (v) を生成する。ユーザは、生成したカメレオンハッシュ (v) をIDとして制御ポリシーに記録し、補助データに乱数 (r) を格納する。

次に図 1 を使い更新の手順について述べる。トラップドアを持つユーザは、ファイルの本体を更新する $(D \to D')$。すると主データも $M \to M'$ と更新される。ユーザは、トラップドア (td) と、更新前の主データのハッシュ値 (m) と乱数 (r)、更新後の主データのハッシュ値 (m') を入力として、新たな乱数 (r') を計算する。そして、新しい乱数 (r') を更新されたファイルの補助データに記録する。

最後に検証の手順について述べる。ユーザは主データ (M) からハッシュ値 (m) を生成し、それをメッセージとして乱数 (r) と主データに記録されたハッシュ鍵 (hk) からカメレオンハッシュ (v) を生成する。カメレオンハッシュの値を制御ポリシーのカメレオンハッシュと比較し、一致することを確認する。トラップドアを持たないユーザがファイルの編集した場合、正しい乱数を生成できないため、紐づけの検証は通過しない。正規のユーザがファイルを更新する場合は、乱数を生成し直すだけなので、ユーザ側の負荷も低く抑えることができる。

ここまでの議論は、カメレオンハッシュの紐づけに注目してその動作を説明した。次章では、データファイルを起点に制御ポリシーを発見する仕組みについて述べる。

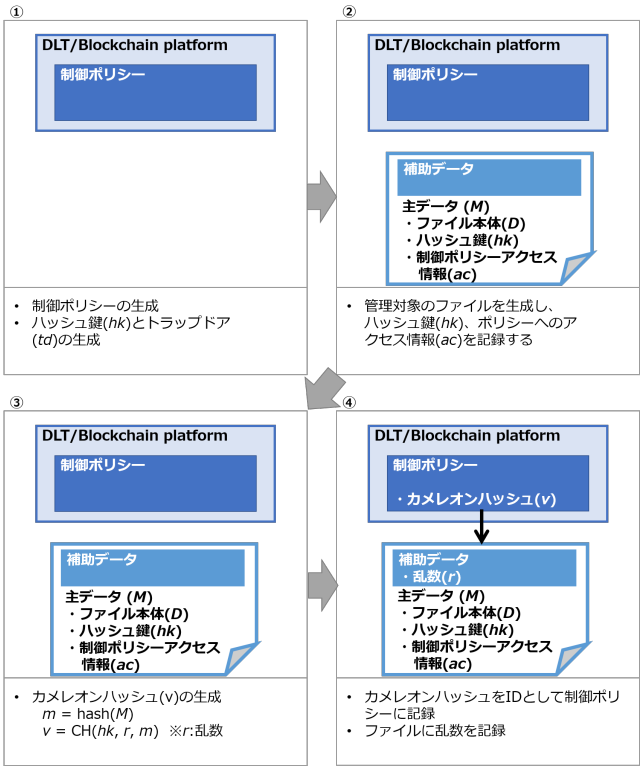

図 2　カメレオンハッシュを使った紐づけ生成

4.2　データのパッケージ方式

制御ポリシーをデータファイルを起点に発見する方法には、DNS などの方法があるが、我々は、図 1 図 2 にあるように制御ポリシーへのアクセス情報をファイル側に持つ方式を考案した。ファイル側にアクセス情報を保持する方法として、データファイルのヘッダ等への埋め込みと、アクセス情報のファイルとデータファイルとをアーカイブ等によって一つのファイルとしてまとめる方法が考えられる。ファイルのヘッダ等への埋め込みは、既存のファイル仕様を拡張する必要があり、影響を受けるシステムが広範囲にわたる。そこで、アクセス情報を記録したアクセス情報ファイルをデータファイルとアーカイブで一つにまとめる方式を採用した。ファイル仕様を拡張する必要がないので、既存システムへの影響を最小限に抑えつつ、様々な仕様のファイルを制御ポリシーと紐づけできる。

ハッシュ鍵と乱数、制御ポリシーへのアクセス情報を明らかにしてデータパッケージの概要を図にすると**図 3** になる。アーカイブでまとめる方法を用いると、図で示すファイル本体は、複数のファイルから構成されていても構わないため、データファイルの管理単位を柔軟に変えられるメリットがある。

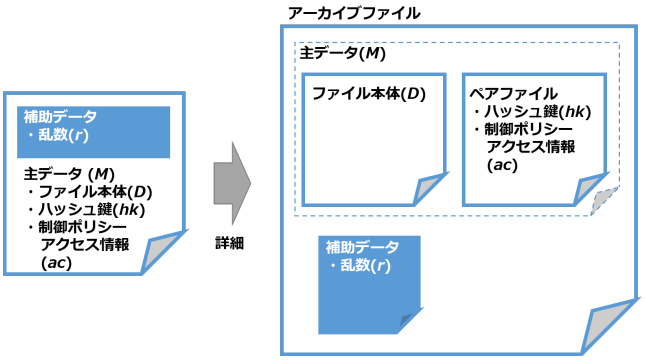

図 3 ファイルのパッケージ方式概略

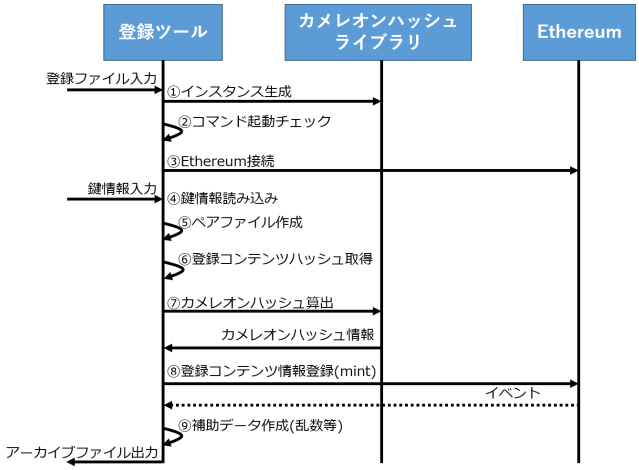

図 4 データファイルの登録シーケンス

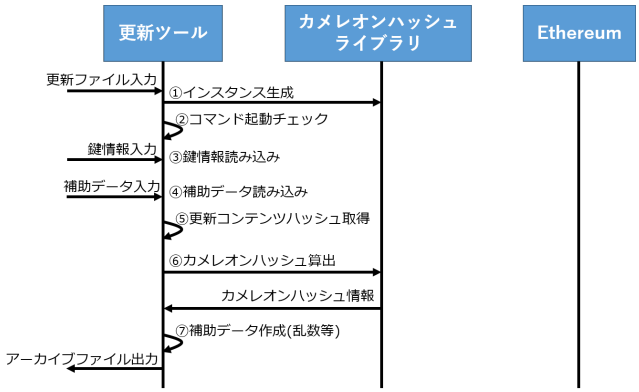

図 5 データファイルの更新シーケンス

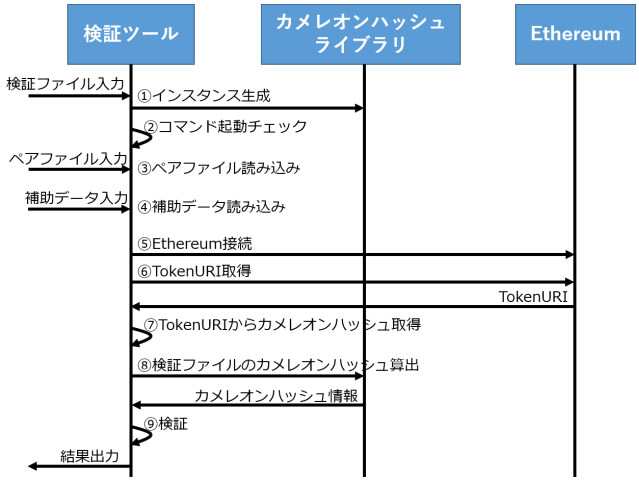

図 6 カメレオンハッシュの検証シーケンス

5. 評価用システム構築

考案方式の実現性を検証するために構築した評価用システムについて説明する。評価用システムは、鍵生成、データファイルの登録、データファイルの更新、カメレオンハッシュの検証の 4 つの評価ツールからなり、各ツールは Nodejs（Javascript）で作成した。

分散台帳には Ethereum を使い、データファイルの紐づけ先として openzeppelin/contracts(v.3.1.0)[22] の ERC721 をベースとしたスマートコントラクトを作成した。このスマートコントラクトには、ERC721 の仕様に対して、**表 2** の関数を追加で定義している。TokenURI にはカメレオンハッシュを登録する仕組みとなっている。

カメレオンハッシュのアルゴリズムには、Ateneise and de Medeiros の離散対数問題をベースとした方式 [23] を利用した。カメレオンハッシュアルゴリズムは Go 実装のものを利用し「GopherJS」、「node-go-reguire」を使用して nodejs モジュール化した。

データファイルの登録、更新、カメレオンハッシュの検証ツールのシーケンスを**図 4**、**図 5**、**図 6** に示す。シーケンスは、更新において、分散台帳を参照していない点が考案方式の特徴である。

その他のソフトウェア条件は**表 3** にまとめる。

6. 考案方式の評価

考案方式と従来のハッシュ値による紐づけ方式でデータファイルの登録と更新、検証の過程を比較し、処理に要する時間から評価用システムを利用した場合の操作性を評価する。

従来方式は分散台帳を用いた一般的な情報登録、更新、検証の方法とした。従来方式の登録処理は、図 4 から④⑤⑦⑨の処理を除いたものである。従来方式の更新処理は、図 5 から③④⑥⑦の処理を除き、図 4 の⑧に相当する処理を加えたものである。従来方式の検証処理は、図 6 から③④⑧の処理を除き、ハッシュ値を計算する処理を加えたものである。

6.1 評価方法

表 4 に示した種類のファイルについて、登録と更新と検証に要する時間を 100 回測定し、その平均値を考案方式と従来方式で比較する。

6.2 評価環境

評価用ノートパソコンに Hyper-V で VM を立て、Ethereum と検証用ツールのプロセスを起動させた状態

表 2 ERC721 仕様への追加関数一覧

コントラクト名	関数名	説明
Document	mint	ERC721 トークンの払い出しと TokenURI 情報の登録を行う
	burn	払い出し済みの ERC721 トークンと TokenURI の削除を行う
	setTokenURI	指定したトークン ID と一致する ERC721 トークンに TokenURI 情報を設定する

表 3 評価用システムソフトウェア条件

対象	ソフトウェア名	ヴァージョン
評価ツール	Nodejs	v12.16.1 以上
	npm	6.13.4 以上
カメレオンハッシュライブラリ	Go 言語	1.12.16
	GopherJS	1.12-3
コントラクト	Solidity	0.6.x

表 4 評価用ファイル

ファイルの種類	登録時サイズ	更新時サイズ
TEX	43KB	53KB
PDF	180KB	561KB
EPS	1.8MB	10.3MB

表 5 評価用環境

ホスト	OS	Windows 10 Enterprise バージョン 1909
	CPU	Core i5-6300U
	Memory	8GB
ゲスト	OS	Ubuntu バージョン 20.04
	CPU	2 コア
	Memory	4GB

で測定する (**表 5**)。Ethereum のマイニング方式は Proof of Authority(PoA) とし、マイニング間隔は Ethereum のメインネットワークの平均ブロック生成間隔を参考に 13 秒とした。ハッシュアルゴリズムは SHA-256、カメレオンハッシュの鍵長は 2048bit とした。

6.3 結果

登録、更新、検証処理のファイル種類毎の比較結果をそれぞれ**図 7**、**図 8**、**図 9** に示す。また、全ファイルの処理時間を平均し、各処理における従来方式と考案方式を比較したものを**図 10** に示す。

登録、更新、検証の処理において、ファイルの種類（サイズ）による差は、ハッシュ値を生成する処理時間に表れるが、非常に差は小さく体感的な違いは感じられない。

登録処理では、考案方式の処理時間は、従来に比べて 2 秒程度増加する結果となった。カメレオンハッシの生成や補助データ、ペアファイルの生成に時間を要した結果である。

更新処理では、考案方式の処理時間は、従来に比べて 7〜8 秒程度減少する結果となった。考案方式では、カメレオンハッシュの衝突条件の計算が追加されているが、従来方式では、分散台帳に登録する時間が追加されるため、トータルでは、考案方式の方が処理時間が少なくなった。

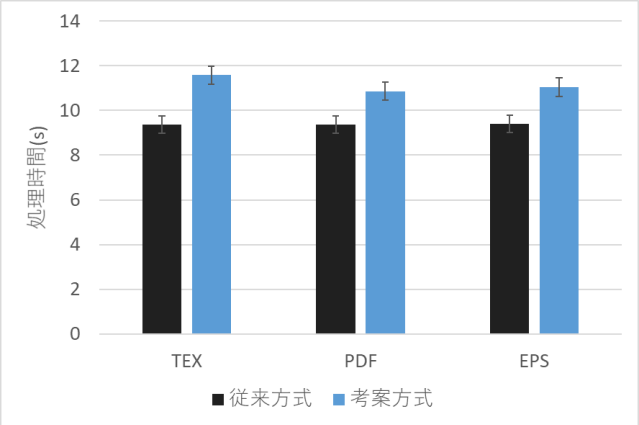

図 7 登録処理時間に対するファイルの違いによる影響

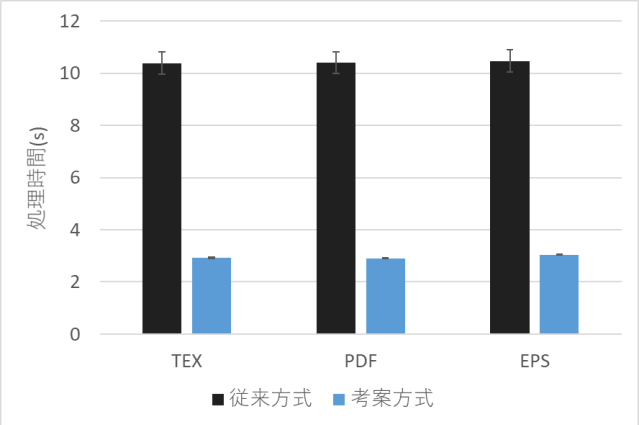

図 8 更新処理時間に対するファイルの違いによる影響

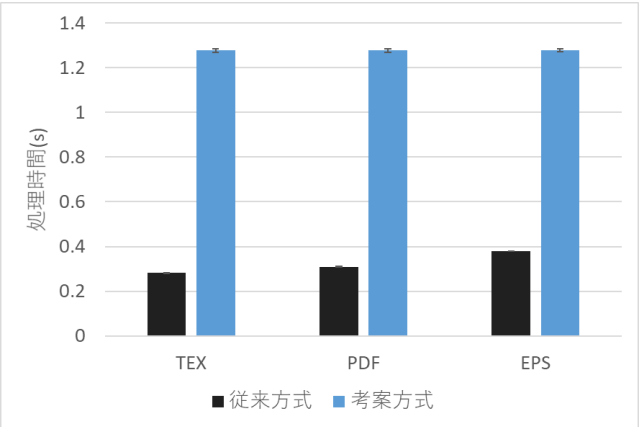

図 9 検証処理時間に対するファイルの違いによる影響

検証処理では、考案方式の処理時間は、従来に比べて 1 秒程度増加する結果となった。考案方式では、カメレオンハッシュの計算が追加されているためである。

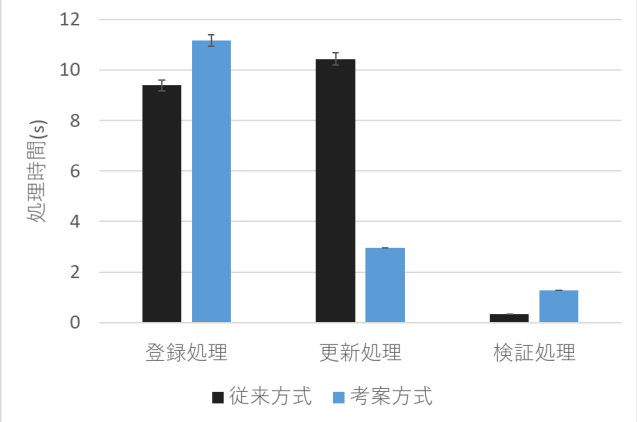

図 10 考案方式と従来方式の各処理における処理時間の比較

6.4 考察

考案方式は、登録時には従来に比べて処理時間がかかるものの、更新時には従来に比べて処理時間が短く済むことを確認できた。今回の評価実験では、分散台帳は 13 秒間隔でブロックが生成される理想に近い条件での実験であるため、実際のシチュエーションでの利用では、トランザクションの伝搬遅延やブロック生成間隔のばらつきの影響が加わる。更新処理においては、考案方式では分散台帳の更新を行わないため、実際のシチュエーションで加算される遅延時間は、更新処理において従来方式と考案方式の差を拡大させると考えられる。

考案方式は、紐づけを維持するためのブロックチェーンへの再登録コストやその低いユーザビリティ、活動が他者に明らかになる心理的障壁を問題とし、その軽減を狙ったものである。その観点においては、本結果から考案方式は想定する効果を持つと考えられるが、カメレオンハッシュの再計算に時間を要しており、オフチェーンでの紐づけ更新において、より低負荷な方式の検討は引き続き必要だと考えられる。

7. まとめ

本研究は、特定のファイルではなく、その特定ファイルに至るまでにある関係者間で生成されるデータファイル全体を管理するために、分散台帳とオフチェーンデータとの新しい紐づけ方式を考案した。本方式は、紐づけを維持するためのブロックチェーンへの再登録コストやその低いユーザビリティ、活動が他者に明らかになる心理的障壁を問題とし、その軽減を狙ったものである。本方式は、カメレオンハッシュの同一ハッシュを生成する特徴に着目した ID と、データファイルと関連情報とのパッケージ方法からなり、オフチェーンデータの更新に際し、オフチェーンのみの処理で制御ポリシーとの紐づけを更新できる。Ethereum を使った評価用システムによる検証によって、考案方式の実現性と狙い通りの効果が期待できることを確認した。

参考文献

[1] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008).
[2] Wood, G. et al.: Ethereum: A secure decentralised generalised transaction ledger.
[3] Swan, M.: *Blockchain: Blueprint for a new economy*, O'Reilly Media, Inc. (2015).
[4] Andoni, M., Robu, V., Flynn, D., Abram, S., Geach, D., Jenkins, D., McCallum, P. and Peacock, A.: Blockchain technology in the energy sector: A systematic review of challenges and opportunities, *Renewable and Sustainable Energy Reviews*, Vol. 100, pp. 143–174 (2019).
[5] Chen, G., Xu, B., Lu, M. and Chen, N.-S.: Exploring blockchain technology and its potential applications for education, *Smart Learning Environments*, Vol. 5, No. 1, p. 1 (2018).
[6] Reyna, A., Martín, C., Chen, J., Soler, E. and Díaz, M.: On blockchain and its integration with IoT. Challenges and opportunities, *Future generation computer systems*, Vol. 88, pp. 173–190 (2018).
[7] PoEx: Proof of Existence.
[8] Dapper Labs, Inc: CryptoKitties — Collect and breed digital cats!
[9] Startbahn, Inc: Startrail Whitepaper.
[10] NTT DATA Corporation: INTERNATIONAL TRADE DATA SHARING PLATFORM USING BLOCKCHAIN TECHNOLOGY.
[11] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C. and Rimba, P.: A taxonomy of blockchain-based systems for architecture design, *2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE, pp. 243–252 (2017).
[12] Krawczyk, H. and Rabin, T.: Chameleon Hashing and Signatures, *IACR Cryptol. ePrint Arch.*, Vol. 1998, p. 10 (1998).
[13] Ohashi, S., Watanabe, H., Ishida, T., Fujimura, S., Nakadaira, A. and Kishigami, J.: Token-Based Sharing Control for IPFS, *2019 IEEE International Conference on Blockchain (Blockchain)*, IEEE, pp. 361–367 (2019).
[14] Protocol Labs: IPFS powers the Distributed Web.
[15] ethereum: Ethereum 2.0 Specifications.
[16] Eberhardt, J. and Tai, S.: On or Off the Blockchain? Insights on Off-Chaining Computation and Data, pp. 3–15 (online), DOI: 10.1007/978-3-319-67262-5_1 (2017).
[17] Pillai, B., Biswas, K. and Muthukkumarasamy, V.: *Blockchain Interoperable Digital Objects*, pp. 80–94 (online), DOI: 10.1007/978-3-030-23404-1_6 (2019).
[18] Entriken, W., Shirley, D., Evans, J. and Sachs, N.: ERC-721 Non-Fungible Token Standard.
[19] Solidity: Contract ABI Specification.
[20] Namecoin: Namecoin.
[21] True Names LTD: Ethereum Name Service.
[22] OpenZeppelin: Contracts.
[23] Ateniese, G. and de Medeiros, B.: On the key exposure problem in chameleon hashes, *International Conference on Security in Communication Networks*, Springer, pp. 165–179 (2004).