Computer Security Symposium 2018
22 - 25 October 2018

ブロックチェーンを用いた認証・認可システムの設計と実装

江澤 友基[†1]　瀧田 愼[†2]　白石 善明[†2]　髙野 泰洋[†2]　毛利 公美[†3]　森井 昌克[†2]

概要： インターネット上には多数のアプリケーションサービスが存在し，その多くはサービス毎に独立してユーザのデータを蓄積している．認証・認可システムがサービスに依存せずクライアント情報や権限情報をサービス間で共有し，データをより広く利用できるようになれば，データはより大きな価値を生み出すと期待される．このとき，データにアクセスするクライアントの確認や，当該クライアントの利用権限の確認が円滑に行えることが望まれる．本稿では，ブロックチェーンを認証情報と認可情報の保管場所として利用しサービスサーバから独立した認証・認可システムの設計と，Ethereum と Apache Web サーバを用いた実装について述べている．

キーワード：分散台帳，スマートコントラクト，認証，認可，アクセス制御，Ethereum

Design and Implementation of Authentication and Authorization System with Blockchain

Yuki Ezawa[†1]　Makoto Takita[†2]　Yoshiaki Shiraishi[†2]
Yasuhiro Takano[†2]　Masami Mohri[†3]　Masakatu Morii[†2]

Abstract: There are a lot of application services on the Internet, and most of them store user's data independently. If an authentication and authorization system does not depend on services and shares client information and authority one among services, more data can be used and these data are expected to create greater value. Towards IoT / Cyber Physical Systems society, it is desirable that authentication and authorization of clients accessing data to create new value can be performed smoothly across data store services. This paper gives a design of the authentication and authorization system independently from service providers by using blockchain as a storage of information on authentication and the authorization, and shows the implementation using Ethereum and Aapche's Web server.

Keywords: Distributed Ledger, Smart Contracts, Authentication, Authorization, Access Control, Ethereum

1. はじめに

現代においてデータは経済の発展に必要不可欠な資源である．企業間ではその独占のために日々熾烈な争いを繰り広げ，自社の管理サーバにデータを蓄積している．経済のさらなる発展を見込むために，この大量のデータを共有し利用することで，互いにメリットのある知見を交換することが期待される．ここで，この大量のデータが生まれる源の一つである IoT デバイスに注目する．企業は IoT デバイスを現場に投入することによって，大量のデータを取得しようとしている．このデータ資源をスマートコントラクトによって管理することを考える．デバイスから得られるデータの取引手順を定めた契約をスマートコントラクトによって記述し，その契約に参加する主体間で自動的に取引できるようにすることで，より効率的なデータの活用が期待される．このような取引を実現させるためには，参加主体が何者であるのかを証明するための認証システムと，それに基づいて取引への参加の可否などを決定するための認可システムの整備が必要不可欠である．この認証・認可システムが特定のサービスや組織に依存せずにクライアント情報や権限情報をサービス間で共有できれば，クライアントはそれらのデータを権限に応じて利用でき，データはより大きな価値を生み出す方向に活用できると期待される．

ブロックチェーン技術は，各国の銀行がその技術のフィンテックへの適用可能性について研究を始めて以降，急速にその他の様々な分野への応用について議論がなされるようになった[1][2]．ブロックチェーンとは，不特定多数のノードに分散された台帳上にデータを記録し，その台帳を各ノードが監視することでデータの改ざんを防ぐ技術である．ブロックチェーンが注目されている理由として，改ざんが困難な形でデータを台帳上に記録できること，さらに台帳が公開されているため情報の透明性が保たれることという特徴が挙げられる．データベースに記録された情報が中央集権的に管理されていると履歴を残さずに書き換えることが容易である．ブロックチェーンネットワークでは，不特定多数のノードに分散された公開台帳を互いに監視することで情報の不正な書き換えが困難であり，情報の真正性・透明性が保証される．

この特徴は認証基盤の構築に応用できる．IoT デバイスの認証情報をブロックチェーン上に記録することで，そのデバイスを管理する単一のサービスに限らず様々なサービスでの取引を期待できる．本稿では，ブロックチェーンを認証データベースおよび認可データベースとして利用することで認証基盤を検証サーバから分離した認証・認可システムを設計する．さらに，Apache Web サーバとブロックチェーンプラットフォームの Ethereum による実装を示す．

2. 関連事例

ブロックチェーンを認証データベースとして利用する考え方に基づいた認証サービスとして Onename[3]が挙げら

†1　神戸大学工学部電気電子工学科
Faculty of Engineering, Kobe University
†2　神戸大学大学院工学研究科電気電子工学専攻
Graduate School of Engineering, Kobe University
†3　岐阜大学工学部電気電子・情報工学科
Faculty of Engineering, Gifu University

れる．Onename は Blockstack というブロックチェーンネットワークをデータベースとして用いる．ユーザが作成した ID 情報をそこへ登録し，ユーザ間で信頼度のフィードバックを行うことにより認証基盤としての活用を目指している．他に生体認証とブロックチェーンを連携させたシステム[4]や，ユーザの個人情報を第三者が確認し暗号化してブロックチェーン上に登録することで他のサービスがレベルの高い認証を行える ShoCard[5]というサービスが存在する．これらはいずれも人を対象として設計された認証システムである．

Ethereum ブロックチェーンを用いてユーザの認証・認可を試みた先行研究[6]が存在する．この研究では，ユーザおよびプロバイダがともに Ethereum アドレスを所有することでプロバイダは公開鍵によるユーザの検証ができる．さらに，プロバイダの構築したコントラクト上へのユーザのアドレスの登録を条件とした認可システムの設計をしている．この手法は IoT デバイスの認証・認可に適用可能な意味で注目できるが，Ethereum について検討しているのみであり，認証・認可のシステムの一般的な構成方法を与えているとは言い難い．本研究ではブロックチェーンを用いた認証基盤をより一般的に構成できる方法を与えることを目標としている．

3. ブロックチェーン

ブロックチェーン技術において，ブロックを生成するためにはネットワークに参加するノードが台帳に改ざんや不正な記録がないという合意を形成する．合意形成には，Proof of Work (PoW)[7] などが用いられる．PoW では，ネットワーク参加者が計算資源を用いて特定の条件を満たす値を先に探し出す競争を行い，最初にその値を見つけたノードが新しいブロックを追加する．そのブロックを他のノードが検証し，内容に不正がないことが確認されれば合意の形成がなされる．PoW はパブリックチェーンの合意形成に利用される．一方，コンソーシアムチェーンにおいて利用されるプロトコルに Practical Byzantine Fault Tolerance (PBFT)[8]などがある．PBFT ではコアノードと呼ばれる信頼できる一部のノードに取引承認の権限を与え，合議によって取引の承認を行う．これにより，ファイナリティ（決済完了性）と高いスループットが得られる．

スマートコントラクトを搭載するブロックチェーンの概要は図 1 のようになる．ブロックチェーンの各ブロックには，ネットワーク上のストレージに格納された値の変更命令を記述したデータであるトランザクションに加え，トランザクションを通して変化する値を表すデータであるステートに関するデータ States Root が格納されている．States Root は Storage Root を持つノード情報のハッシュからなるハッシュツリーのルートハッシュを持つノード情報のハッシュ値を示す．Storage Root はネットワーク上の各ストレージのステートからなるハッシュツリーのルートハッシュを持つノード情報のハッシュ値を示す．トランザクションは，この各ストレージのステートに変化を加えるために発行される．

スマートコントラクトを利用するにはまず，クライアントノードはアプリケーションを通して，関数を実行してトランザクションを発行するようスマートコントラクトに命令する．命令を受けたスマートコントラクトはブロックチェーンネットワーク上の計算リソースを使って関数を実行し，計算結果を記述したトランザクションを発行する．ネットワーク上のノードはこのトランザクションを読み取ってストレージのステートを変化させる．このようにして発行されたトランザクション群についてコンセンサスが形成されると，このトランザクション群を含んだブロックはブロックチェーンに追加され，新たなブロックの生成が始まる．

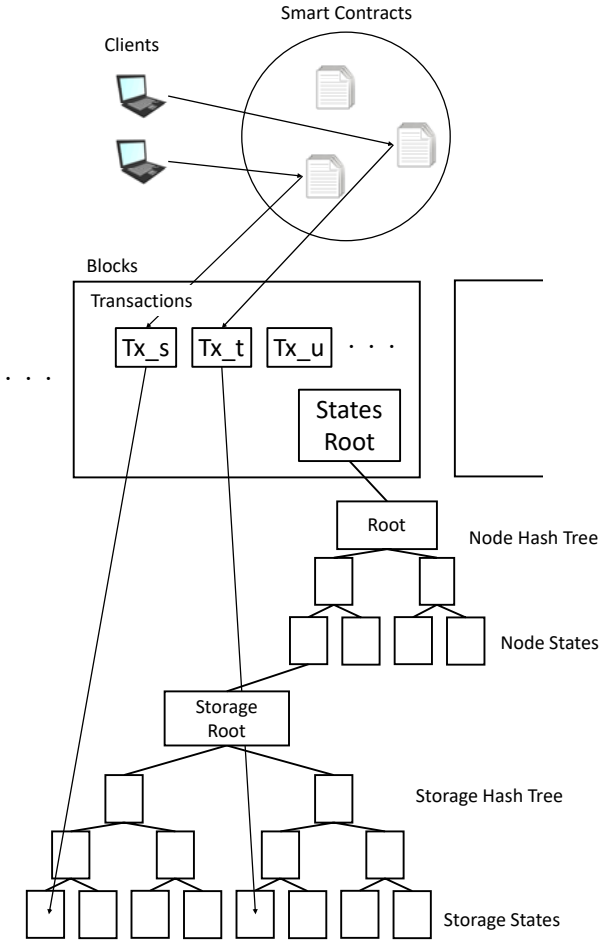

図 1　ブロックチェーンの概要

4. ブロックチェーンを用いた認証システム

本研究では，認証データベースが検証サーバから独立し，証明クライアントを含めた三者が互いに独立した関係を維持できるような認証基盤の構築を目指している．これを実現するためには，次の要件を満たさなければならない．

要件 1 : 証明クライアントが認証基盤へ ID 情報を登録・失効できる．

要件 2 : 検証サーバが認証基盤に登録された ID 情報を参照できる．

要件 3 : 検証サーバは認証基盤を利用して証明クライアントを認証できる．

本章では不特定多数のノードによる参照が可能な分散台帳上に認証データベースを構築することで，以上の要件を満たす認証基盤を構築する．

4.1 システム構成

証明クライアントと検証サーバの二者に認証コントラクトを加えた三者を参加主体とする．各々は以下のモジュールから構成される．システムの流れは図 2 のようになる．

証明クライアント

A) ID 情報登録部
B) 認証要求部
C) ID 情報失効部

認証コントラクト

A) ID 情報登録処理部
B) 検証処理部
C) ID 情報失効処理部

検証サーバ

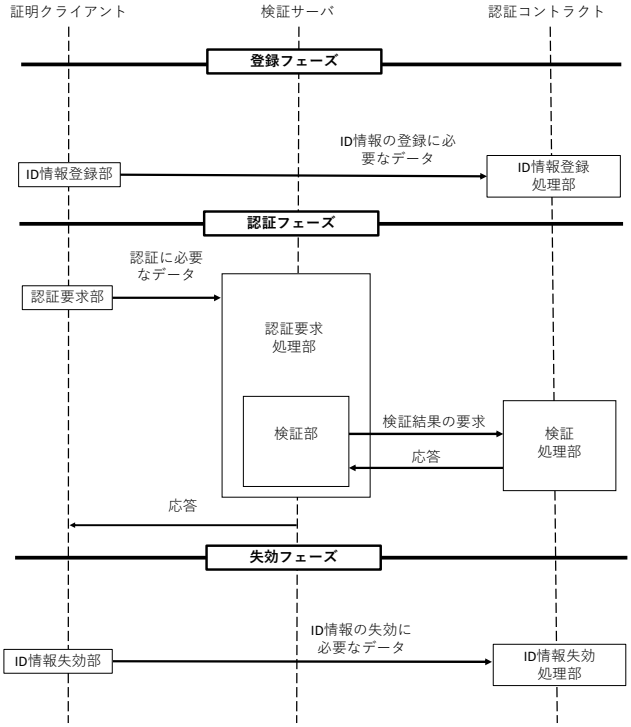

図 2　認証システムの動作

表 1　変数 – 認証コントラクト

変数名	機能
id	クライアントを識別するための ID を格納する.
spec	クライアントを認証するための値を格納する．この変数は一つ以上なければならない.

表 2　関数 – 認証コントラクト

関数名	機能
regClient (id, spec)	コントラクト内に同じ ID がなければ新たに ID 情報を格納する.
unregClient (id, spec)	ID に紐づけられるコントラクト内の変数をすべて初期化する.
checkClient (id, spec)	ID と固有情報がコントラクトに登録されているか確認する

A)　認証要求処理部
B)　検証部

これら八つのモジュールから，三つのフェーズ – ID 情報登録フェーズ・認証フェーズ・ID 情報失効フェーズが構成される．各フェーズについて説明する.

4.1.1 ID 情報登録フェーズ

Step-Rg1 証明クライアントは ID 情報登録部で認証コントラクトへ ID 情報の登録に必要なデータを送信する.

Step-Rg2 認証コントラクトは ID 情報登録処理部で ID 情報を登録する.

4.1.2 認証フェーズ

Step-An1 証明クライアントは認証要求部で認証に必要なデータを検証サーバへ送信する.

Step-An2 検証サーバは認証要求処理部で証明クライアントの認証に必要なデータを受信する.

Step-An3 検証サーバは検証部で認証コントラクトに対し ID 情報が正しいか確認を要求する.

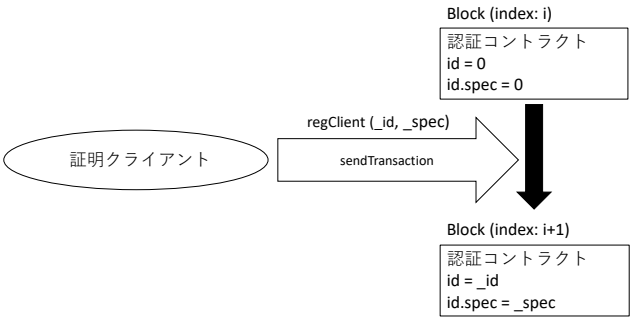

図 3　ID 情報登録部

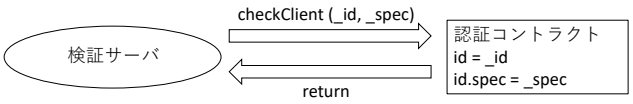

図 4　検証部

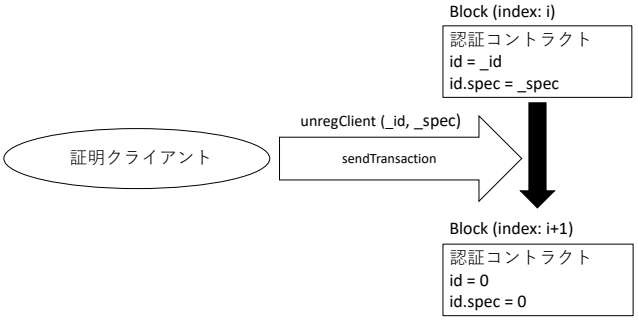

図 5　ID 情報失効部

Step-An4 認証コントラクトは検証処理部で受信したデータに応じて検証サーバに対し応答を送信する.

Step-An5 検証サーバは認証要求処理部で，認証クライアントからの応答を受けて証明クライアントに OK/NG の結果を返す.

4.1.3 ID 情報失効フェーズ

Step-Rv1 証明クライアントは ID 情報失効部で認証コントラクトへ ID 情報の失効に必要なデータを送信する.

Step-Rv2 認証コントラクトは ID 情報失効処理部で ID 情報を失効する.

4.2 設計

証明クライアントの情報を保持するため，認証コントラクト内に表 1 のような変数を定義する．さらに，ID と固有情報を紐づけるためのマッピングを定義する. これにより，変数 spec に格納されるデータは変数 id の値によって固有の値となる．これらの変数を扱うための関数を，表 2 のように定義する．認証コントラクトの内容に変更を加える関数を用いる際にはアカウントからトランザクションを発行する必要があり，regClient と unregClient がこれに当てはまる．ソフトウェアが行うシステムの内部処理について，三つのフェーズに分けて説明する．参加主体を証明クライアント，認証コントラクト，検証サーバの三者とする.

4.2.1 ID 情報登録フェーズ

証明クライアントは図 3 のように関数 regClient を用いて ID と固有情報を認証コントラクトに登録するトランザクションを発行する．登録を要求した ID にまだ固有情報が登録されていなければ ID と固有情報の登録は成功する．すでに ID に固有情報が登録されていれば登録は失敗する.

4.2.2 認証フェーズ

証明クライアントは ID と固有情報を検証サーバに送信する. 検証サーバは証明クライアントから受け取った ID と固有情報を引数として，図 4 のように関数 checkClient によ

って認証コントラクトに ID 情報が登録されているか検証する．

4.2.3 ID 情報失効フェーズ

証明クライアントは図 5 のように関数 unregClient を用いて認証コントラクトに登録されている ID 情報を失効するトランザクションを発行する．ID 情報が認証コントラクトに登録された内容と一致する場合には失効に成功し，そうでない場合には失敗する．

5. ブロックチェーンを用いた認可システム

本章では，アプリケーションサーバへのアクセス認可を要求する認可クライアントに対し，認可サーバ及びアプリケーションサーバはトークンとして Cookie を提供し，Cookie によって認可クライアントのセッション管理を行う認可システムを構築する．ここでの認可クライアント及び認可サーバはそれぞれ 4 章の証明クライアント及び検証サーバと同じ主体を表すが，役割を明確にするために表記を変えて説明する．

Cookie (HTTP Cookie) とはサーバの指示に従ってクライアントに保存された情報のことを指す．クライアントに Cookie が保存されているとき，クライアントはサーバに Cookie の所有を伝える仕組みになっている．Cookie の仕様は 2018 年 8 月現在 RFC6265[9]に則っており，HTTP レスポンスでは表 3 のような属性が規定されている．

アプリケーションサーバが認可クライアントのアクセス制御をする際，認証コントラクトに登録された認可クライアントの ID 情報だけではきめ細かいアクセス制御を実現できない．そこで，本稿ではこれを満足する機能を有するコントラクトを構築することを考える．このコントラクトを認可コントラクトと呼ぶことにする．

認可コントラクトは認証コントラクトに登録されている ID 情報を引き継ぎ，アプリケーションサーバは指定した ID をある追加属性 - サービス ID をもつ ID として認可コントラクトに登録できる．この際，アプリケーションサーバはこのサービス ID のオーナーでなければならない．認可コントラクト内のサービス ID に認証コントラクト内の ID を紐づけることで，疑似的に ID にサービス ID を持たせることができる．サービス ID のオーナーアプリケーションサーバ 以外のアプリケーションサーバもこのサービス ID を参照できるようにシステムを構築することで，多様なサービス ID によるアクセス制御が実現できる．

5.1 システム構成

システムの流れは図 6 のようになる．参加主体は認可クライアント，アプリケーションサーバ，認可サーバ及び認可コントラクトの四者とする．各々は以下のモジュールから構成される．

認可クライアント

A) アクセス要求部

アプリケーションサーバ

A) 認可登録部
B) アクセス要求処理部
C) 認可失効部

認可サーバ

A) リダイレクト処理部

表 3 Cookie の属性

属性	説明
Name	Cookie の名前と値
Expires	Cookie の日時有効期限
Max-Age	Cookie の秒数有効期限
Domain	Cookie を送信するドメイン
Path	Cookie を送信するディレクトリ
Secure	SSL/TLS 通信時のみサーバへ送信
HttpOnly	HTTP ヘッダ以外から取得不可

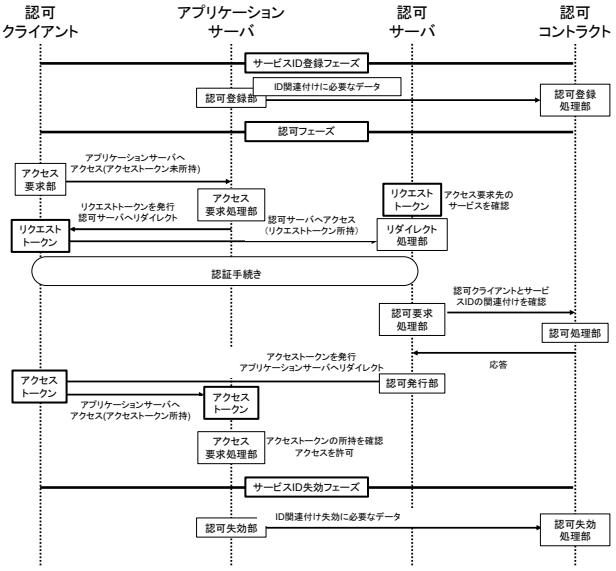

図 6 認可システムの動作

B) 認可要求処理部

認可クライアント

A) 認可登録処理部
B) 認可処理部
C) 認可失効処理部

これら 11 のモジュールから三つのフェーズ – サービス ID 登録フェーズ・認可フェーズ・サービス ID 失効フェーズが構成される．各フェーズについて説明する．

5.1.1 サービス ID 登録フェーズ

アプリケーションサーバが認可クライアントの ID をサービス ID をもつ ID としてアプリケーションサーバの ID に関連付ける．

Step-SRg1 アプリケーションサーバは認可登録部で認可コントラクトへ ID にサービス ID を発行するために必要なデータを送信する．

Step-SRg2 認可コントラクトは認可登録処理部で ID にサービス ID を発行する．

5.1.2 認可フェーズ

認可クライアントがアプリケーションサーバへアクセスを要求した際のアクセス制御に関するフェーズである．アプリケーションサーバは認可クライアントがアクセス要求してきた際，指定したトークンの所有をアクセス認可の条件とすることができる．このトークンをアクセストークンと呼ぶこととする．一方，認可クライアントがアクセストークンを所有していなかった場合，アプリケーションサーバは認可クライアントのアクセス認可取得を促すためのトークンを認可クライアントへ与える．このトークンをリクエストトークンと呼ぶこととする．リクエストトークンは認可サーバへ認可要求を行う役割と，リダイレクトする際のアクセス要求先サービス及びアクセスに必要なサービス ID を認可サーバに伝える役割をもつ． 本フェーズのフローは以下のようになる．

Step-Az1 認可クライアントはアクセス要求部で，アプリケーションサーバにアクセスする．

Step-Az2 アプリケーションサーバはアクセス要求処理部で認可クライアントがアクセストークンを所持していないことを受けて，認可クライアントにリクエストトークンを発行し，認可サーバへリダイレクトする．認可クライアントがアクセストークンを所持している場合は Step-Az9 と同様となる．

Step-Az3 認可クライアントはリクエストトークンを所持した状態で認可サーバへリダイレクトされる．

Step-Az4 認可サーバはリダイレクト処理部で認可クライアントがリクエストトークンを所持していることを受けて，認証手続きを行う．認可クライアントがリクエストトークンを所持していない場合，認可サーバは認可クライアントのアクセスを拒否する．

Step-Az5 認証手続きに成功すると，続いて認可サーバは認可要求処理部で，認可クライアントのアクセス要求先のサービス ID が認可クライアントの ID がもつサービス ID に含まれているか，認可コントラクトに確認を要求する．

Step-Az6 認可コントラクトは要求を受けて，認可処理部で認可クライアントのサービス ID の確認を行う．その後，認可サーバへ結果を返す．

Step-Az7 認可サーバは検証結果を受けて，認可発行部でアクセストークンを発行し，認可クライアントをアプリケーションサーバへリダイレクトする．

Step-Az8 認可クライアントはアクセストークンを所持した状態で，アプリケーションサーバへリダイレクトされる．

Step-Az9 アプリケーションサーバは，アクセス要求処理部で認可クライアントが正しいアクセストークンを所持していることを確認し，アクセスを許可する．

5.1.3 サービス失効フェーズ

アプリケーションサーバのサービス ID に対する認可クライアントの ID の関連付けを失効する．

Step-SRv1 アプリケーションサーバは認可失効部で認可コントラクトへ指定 ID のサービス ID の失効に必要なデータを送信する．

Step-SRv2 認可コントラクトは認可失効処理部で ID に発行されたサービス ID を失効する．

5.2 設計

Web サーバの動作を実現するために，認可コントラクト，アプリケーションサーバ及び認可サーバで表 4〜6 のような関数を定義する．ソフトウェアが行うシステムの内部処理について，三つのフェーズに分けて説明を行う．参加主体は，認可クライアント，アプリケーションサーバ，認可サーバ，認可コントラクトの四者とする．

5.2.1 サービス ID 登録フェーズ

アプリケーションサーバは regSid を用いて認可クライアントの ID をサービス ID に関連付けるトランザクションを発行する．関連付け登録を要求する ID がまだ関連付けされていなければ登録は成功する．

5.2.2 認可フェーズ

まず認可クライアントはアクセス要求部でアプリケーションサーバへアクセスする．続いてアプリケーションサーバはアクセス要求処理部で認可クライアントのアクセスを受けて，verify_access_token がリクエストオブジェクトからアクセストークン cookie を読み取って検証する．この関数は検証結果と読み取ったサービス ID を返す．検証結果が正しく，読み取ったサービス ID が空でなければ，check_sid がこのサービス ID とアプリケーションサーバの指定するサービス ID が一致するか確認し，異なればアクセス拒否コードを返す．一致すればアクセスを許可する．アクセストークン cookie がないなどの理由で検証結果が正しくない場合，make_req_token がクエストオブジェクトからリクエストトークン cookie を生成する．リクエストトークン cookie には，アクセス要求先 URI と，アクセス要求先のサービス ID 及びそのハッシュが記述される．このハッシュは，サービス ID と認可サーバの秘密鍵から生成する．その後，生成したリクエストトークンを認可クライアントに発行し，認可サーバへリダイレクトする．

認可サーバは認可クライアントのリダイレクトを受けて，まずリクエストトークン cookie の所有を確認する．認可クライアントがリクエストトークン cookie を持っていなければ，no_cookie_error がアクセスを拒否する旨を伝える．所有が確認できたら，make_login_scree が認可クライアントの ID と固有情報を受け取るモジュールを生成する．受け取った後，authenticate が認可クライアントの認証手続きを行う（この手続きについては 4 章を参照）．さらに verify_sid がサービス ID が偽造されていないか検証した後．認可コントラクト内で定義される関数 checkSid を呼び出して，認可クライアントの ID がコントラクト内のサービス ID に関連付けられているか確認する．これらの手続きに問題が生じた場合，再度認証手続きからやり直す．問題がなかった場合，make_access_token がリクエストオブジェクト及び ID，サービス ID からアクセストークン cookie を生成する．アクセストークン cookie には，アクセス要求元の IP アドレス，トークン生成時刻，ID，サービス ID，有効期限，及びそれらの値と認可サーバの秘密鍵から成るハッシュが記述される．go_to_uri が生成したアクセストークンを認可クライアントに発行し，アクセス要求先のアプリケーションサーバへリダイレクトする．アプリケーションサーバは再度，認可クライアントがアクセストークン cookie を所有してい

表 4　関数 – 認可コントラクト

関数名	説明
regSid (sid, id)	サービス ID に ID が関連付けられていなければ関連付ける
unregSid(sid, id)	サービス ID に ID が関連付けられていれば関連付けを解除する
checkSid(sid, id)	サービス ID に ID が関連付けられているか確認する

表 5　関数 – アプリケーションサーバ

関数名	説明
verify_access_token (r)	アクセストークン cookie を検証し，検証結果とトークンに記述されたサービス ID を返す
check_sid (sid)	トークンに記述されたサービス ID を引数とし，アプリケーションサーバの指定するサービス ID と異なればアクセス拒否コードを返す
make_req_token (r)	リクエストトークン cookie を生成して返す

表 6　関数 – 認可サーバ

関数名	説明
no_cookie_error ()	認可クライアントがリクエストトークン cookie を所有していなければアクセス拒否画面を生成する
authenticate (id, spec)	認可クライアントの ID，固有情報を引数として認可クライアントの認証情報を確認し，結果を返す
verify_sid (sid, hash)	トークンに記述されたサービス ID とハッシュ値を引数としてサービス ID を検証し，結果を返す
make_access_token (r, id, sid)	アクセストークン cookie を生成して返す
go_to_uri (r, req_uri, access_token)	アクセストークン cookie を発行し，認可クライアントをアクセス要求先にリダイレクト
make_login_screen ()	認可クライアントから ID と固有情報を受け取るモジュールを生成する

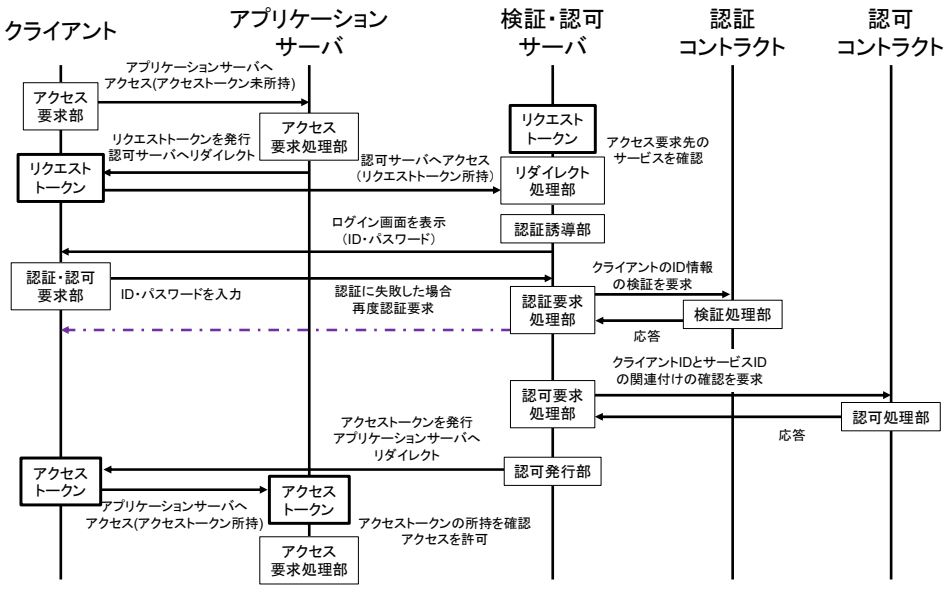

図 7　実装した認証・認可システムの流れ

るか確認し，所有が確認できたらアプリケーションサーバは認可クライアントのアクセスを許可する．

5.2.3 サービス ID 失効フェーズ

アプリケーションサーバは unregSid を用いて認可クライアントの ID とサービス ID との関連付けを解除するトランザクションを発行する．関連付け解除を要求する ID が関連付けされていれば登録は成功する．

6. プロトタイプ

6.1 開発環境

OS については，Ethereum による認証基盤，および Apache による認可システムの動作確認を CentOS7 上で，認証コントラクトのデプロイを Windows10 上の Mist Wallet で行い，両 OS 間で台帳の共有を行った．その他の環境は表 7 のようになっている．

6.2 実装

6.2.1 Ethereum を用いた認証システムの実装例

まず Ethereum によって認証コントラクトを実装する．ここでは説明を簡単にするためにパスワードのハッシュを固有情報として登録した例をあげるが，公開鍵証明書を登録して認証するなどの秘密の情報を登録しないのが望ましいことに注意されたい．

Solidity によって認証コントラクトのプロトタイプを記述し，Mist Wallet を使用してコントラクトのデプロイを行う．コントラクト内に構造体 ClientInfo を定義し，その中に二種類の変数 – owner と passwd_hash を定義する．構造体 ClientInfo 内の変数 owner には ID 登録を行った Ethereum アドレスが，passwd_hash には登録パスワードの keccak256 ハッシュ値を uint 型に変換した値がそれぞれ格納される．ClinetInfo は ID から呼び出せるようにマッピングする．このマッピングにより，ID と固有情報の関連付けが実現する．さらに，表のような機能を有する関数を定義する．checkClient はコール関数なのでブロックチェーンを読み込むだけでよく，トランザクションを発行する必要はない．一方，その他の関数はトランザクション関数であるためブロックチェーンに変更を加えるためのトランザクションを発行する必要がある．

続いて Ethereum を用いた認証システムの実装について説明する．ID による認証を行う前に登録フェーズで証明クライアントは Geth を用いて ID 情報の登録を行う．証明クライアントは Geth 上で関数 regClient を用いて ID・パスワードを登録する．パスワードはコントラクト内で keccak256 ハッシュ値に変換された状態で記録される．ID 情報を失効する際には，失効フェーズで証明クライアントは Geth 上で関数 unregClient を用いて ID・パスワードを失効する． ID 情報の登録後，認証フェーズで証明クライアントは ID による認証要求ができる．関数 checkPassword を ID・パスワードを用いて呼び出すと，入力が正しければ true を，正しくなければ false を返す．これによって検証サーバは ID 情報が登録されていることを確認できる．

表 7　開発環境

OS	Windows10 / CentOS7
Web サーバ	Apache-2.2.24
Apache モジュール	mod_perl-0.10.0
コントラクト開発言語	Solidity-0.4.11
Ethereum クライアント	Geth-1.8.1
Ethereum ウォレット	Mist Wallet-0.8.10

6.2.2 Ethereum と Apache を用いた認証・認可システムの実装例とその実行

アプリケーションサーバおよび検証・認可サーバには Apache httpd[10] を用いる． また， Apache の拡張には mod_perl と呼ばれるモジュールを使用した．mod_perl とは，Perl 言語で記述されたプログラムを高速で実行できる機能を持った，Apache のモジュールである．このモジュールを使用することで，Perl での Apache API の利用が可能になる．

Apache httpd によってアプリケーションサーバ及び検証・認可サーバを構築し，ここに Etheruem による認可コントラクトを組み込むことで，認可システムを実装する．さらに，この認可システムに Ethereum による認証システムを組み込むことで，Ethereum と Apache による認証・認可システムを実装する．このシステムの流れは図 7 のようになる．

まず Ethereum によって認証コントラクトと同様にして認可コントラクトを実装する．コントラクト内にサービス ID とクライアントの ID のマッピングを定義する．これにより，コントラクト内でクライアントの ID を格納する配列をサービス ID に関連付けることができる．

続いて認証・認可システムの実装及び実行例について説明する．ここで，クライアント及びアプリケーションサーバはあらかじめ認証コントラクトに ID 情報を登録してい

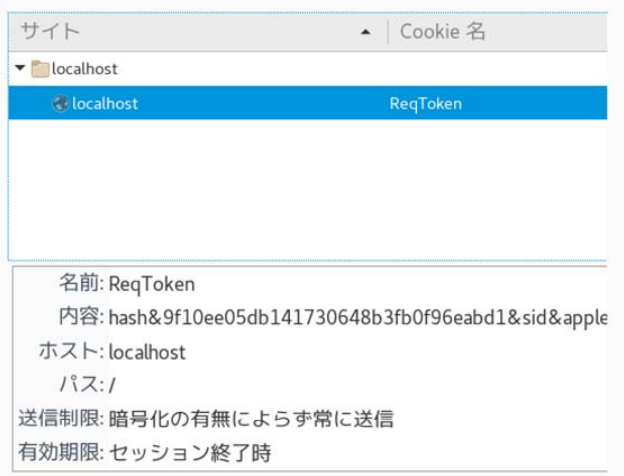

図 8　リクエストトークン cookie

図 9　認証画面

るものとする．アプリケーションサーバはサービス ID 登録フェーズで，Geth 上で関数 regSid を用いてサービス ID にクライアントの ID の関連付け登録を行う．関連付けを解除する際には，アプリケーションサーバは関数 unregSid を用いてサービス ID 失効フェーズでクライアントの ID の関連付けを解除する．

続いて，認証・認可フェーズについて，各モジュール毎に順を追って説明する．ここで，検証・認可サーバの URI を http://localhost/authLogin/ ，アプリケーションサーバの URI を http://localhost/application/ とし，アプリケーションサーバは Apache の config ファイルで URI に割り当てるサービス ID を指定しておく．以下に本システムの基本的な流れを記す．CentOS 上で Apache を起動し，アプリケーションサーバ及び検証・認可サーバを立ち上げておく．

【アクセス要求部】クライアントは Web ブラウザを起動し，アプリケーションサーバに http://localhost/application/index.html へのアクセスを要求する．

【アクセス要求処理部】アプリケーションサーバはクライアントがアクセストークン cookie を所有しているか確認する．クライアントがアクセストークン cookie を所有していればアクセスを許可する．クライアントがブラウザからの初回アクセスなどでアクセストークン cookie を所有していなければ，アプリケーションサーバはクライアントにリクエストトークン cookie を発行し，検証・認可サーバ http://localhost/authLogin/へリダイレクトする．

【リダイレクト処理部】認可サーバはクライアントがリクエストトークン cookie を所有しているかどうか確認する．クライアントがリクエストトークン cookie（図 8）を所有していなければ検証・認可サーバはアクセスを拒否する．

【認証誘導部】クライアントがリクエストトークン cookie を所有していればクライアントのブラウザに認証画面（図 9）を表示する．

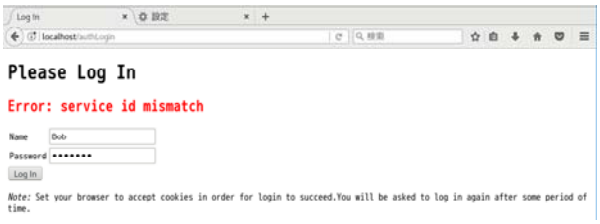

図 10　サービス ID が正しくない

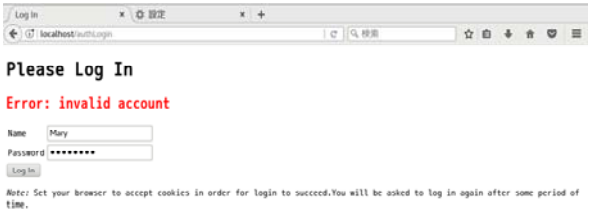

図 11　ID 情報が正しくない

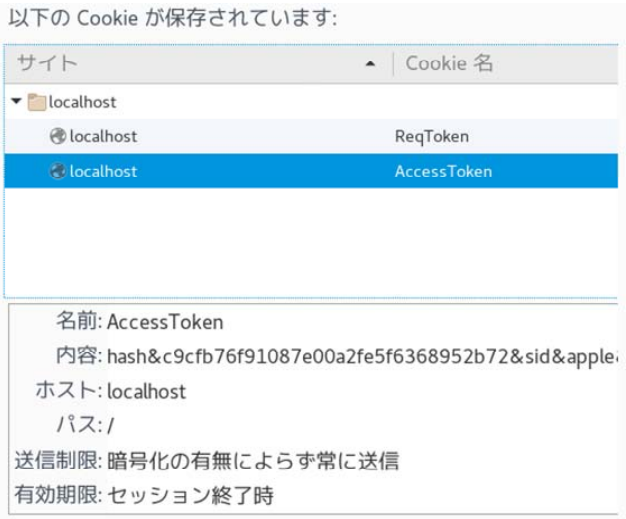

図 12　アクセストークン cookie

【認証・認可要求部】クライアントは認証コントラクトに登録済みの ID 情報をブラウザに入力し，検証・認可サーバに送信する．

【認証要求処理部】検証・認可サーバは認証コントラクトの関数 checkClient によってクライアントの ID 情報を検証する．検証結果が正しくなければ再度認証画面（図 10）を表示して認証情報の入力を要求する．正しければ認可要求部に進む．

【検証処理部】検証・認可サーバから受け取った ID 情報が認証コントラクト内に登録されているか確認し，登録されていれば true を，されていなければ false を返す．

【認可要求処理部】検証・認可サーバは認可コントラクトの関数 checkSid によってクライアントの ID がサービス ID に関連付けられているか確認する．関連付けられていなければ再度認証画面（図 11）を表示する．関連付けられていれば認可発行部に進む．

【認可処理部】検証・認可サーバから受け取ったサービス ID とクライアント ID との間の関連付けが認可コントラクト内に登録されているか確認し，登録されていれば true を，されていなければ false を返す．

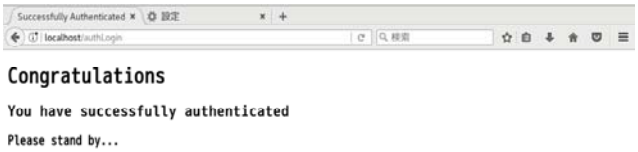

図 13　リダイレクト待機画面

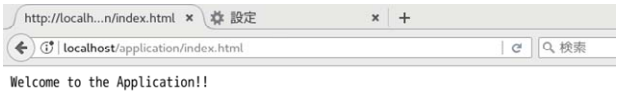

図 14　アプリケーション画面

【認可発行部】アクセストークン cookie（図 12）を発行し，クライアントをアプリケーションサーバにリダイレクト（図 13）する．
【アクセス要求部】アクセストークン cookie を取得したクライアントはリダイレクトされ，再度アプリケーションサーバの URI である http://localhost/application/index.html へのアクセスを要求する．
【アクセス要求処理部】アプリケーションサーバはクライアントがアクセストークン cookie を所有しているか確認する．所有していればアクセスを許可する（図 14）．

以上が本稿における，Ethereum と Apache を用いた認証・認可システムの実装とその実行例である．

7. おわりに

本研究ではサービスのクライアント情報や権限情報をサービス間で共有して利用できることを目的として，ブロックチェーンを用いた認証・認可システムの設計とその実装例について述べた．一般的なブロックチェーン技術で利用できることを意図した認証・認可システムという点で関連する事例とは異なるものである．本提案内容をより一般的なフレームワークとして提供することが今後の課題である．

謝辞　本研究の一部は JSPS 科研費 16K00184，18K04133 の助成を受けたものである．

参考文献

[1] 山藤敦史，箕輪郁雄，保坂豪，早川聡，近藤真史，一木信吾，金子裕紀: “金融市場インフラに対する分散型台帳技術の適用可能性について,” August 2016, JPX ワーキング・ペーパー. https://www.jpx.co.jp/corporate/research-study/workingpaper/tvdivq0000008q5y-att/JPX_working_paper_No15.pdf
[2] 経済産業省商務情報政策局: “ブロックチェーン [分散型台帳] ，シェアリングエコノミーを活用した新たな産業社会に向けて,” 産業構造審議会情報経済小委員会分散戦略 WG (第 4 回) 事務局資料. http://www.meti.go.jp/committee/sankoushin/shojo/johokeizai/bunsan_senryaku_wg/pdf/005_s01_00.pdf
[3] “Onename,” https://onename.com/, visited on 09-08-2018
[4] HITACHI:“PBI を使った安全なブロックチェーンシステム,” 13 January 2018, http://www.hitachi.co.jp/rd/portal/report/2018/socialsec/index.html visited on 04-06-2018
[5] “ShoCard,” https://shocard.com, visited on 04-06-2018.
[6] Mukesh Thakur. “Authentication, Authorization and Accounting with Ethereum Blockchain,” https://helda.helsinki.fi/bitstream/handle/10138/228842/aaa-ethereum-blockchain.pdf, visited on 09-08-2018
[7] Satoshi, N.. “Bitcoin: A Peer-to-Peer Electronic Cash System,“ https://bitcoin.org/bitcoin.pdf, p.3, visited on 19-08-2018
[8] “Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric),” https://www.researchgate.net/profile/Harish_Sukhwani/publication/320649195_Performance_Modeling_of_PBFT_Consensus_Process_for_Permissioned_Blockchain_Network_Hyperledger_Fabric/links/5abbc97aaca27222c75367cc/Performance-Modeling-of-PBFT-Consensus-Process-for-Permissioned-Blockchain-Network-Hyperledger-Fabric.pdf, visited on 19-08-2018
[9] A.Barth, U.C.Berkeley: “HTTP State Management Mechanism,” https://www.ietf.org/rfc/rfc6265.txt, visited on 09-08-2018
[10] “Apache HTTP Server Project,” https://httpd.apache.org/, visited on 19-08-2018