情報カプセルとブロックチェーンを用いたデジタルコンテンツの流通

木下研究室　　201503841　山下恵生

研究背景

- 近年、ネットワークの発達によりデジタルコンテンツの流通が盛んになっている。
- しかし、デジタルコンテンツの不正利用が多発している
- コンテンツ購入後のユーザ間での自由なやり取りができない

著作者の権利を保護してユーザ間での自由なやり取りができるような技術が必要

問題点

- デジタルコンテンツの不正利用を防止する、つまり著作者の権利の保護が必要
- デジタルコンテンツをユーザ間でやり取りをするためには利用権の受け渡しが必要
  - 従来のDRM（デジタル著作権管理）の枠組みではこれらを確実に実行させるのが難しい

情報カプセルとブロックチェーンを利用したコントラクトコードで確実に実行できる

先行研究

- スマートプロパティと既存のDRM技術を組み合わせることにより不正コピーの防止を実現するシステムが提案されているが、コンテンツを転々流通させるシステムのセキュリティモデルが示されてない。

Take-Grantモデルを採用

DRM（デジタル著作権管理）

- Digital Rights Management　=　DRM
  デジタルデータに対する著作権管理技術の総称
  (例:電子透かし、アクセス制御、暗号化、情報カプセル)

コンテンツの流通・再生に制限をかける

Ethereum(イーサリアム)

●スマートコントラクト・分散型アプリケーションの構築プラットフォーム

スマートコントラクトとは・・・契約行動をプログラム化し、自動的に実行しようとするもの

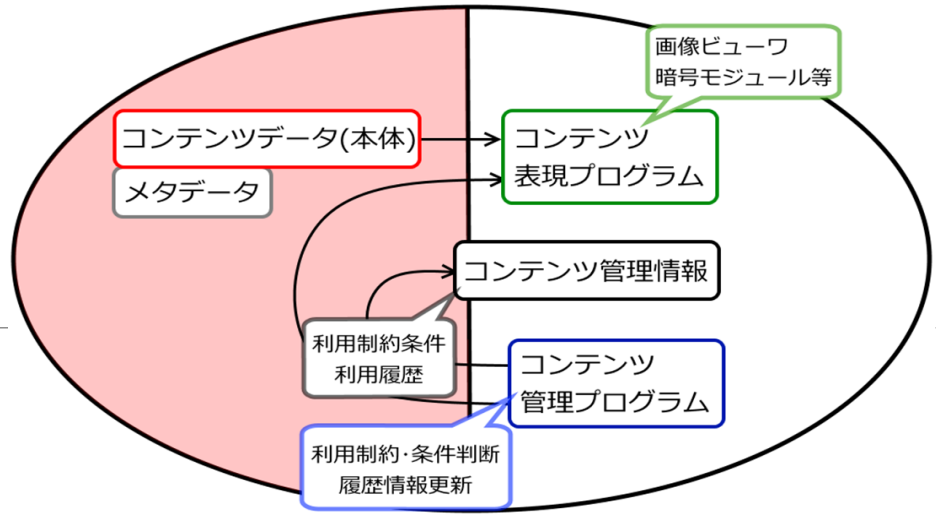
●ビットコインの次に時価総額が大きい仮想通貨

特徴

➢ユーザが誰でも自由にスマートコントラクトのプログラミング及びコード実行を行うことが可能

➢独自のブロックチェーン（P2P）ネットワークをもつ

情報カプセル



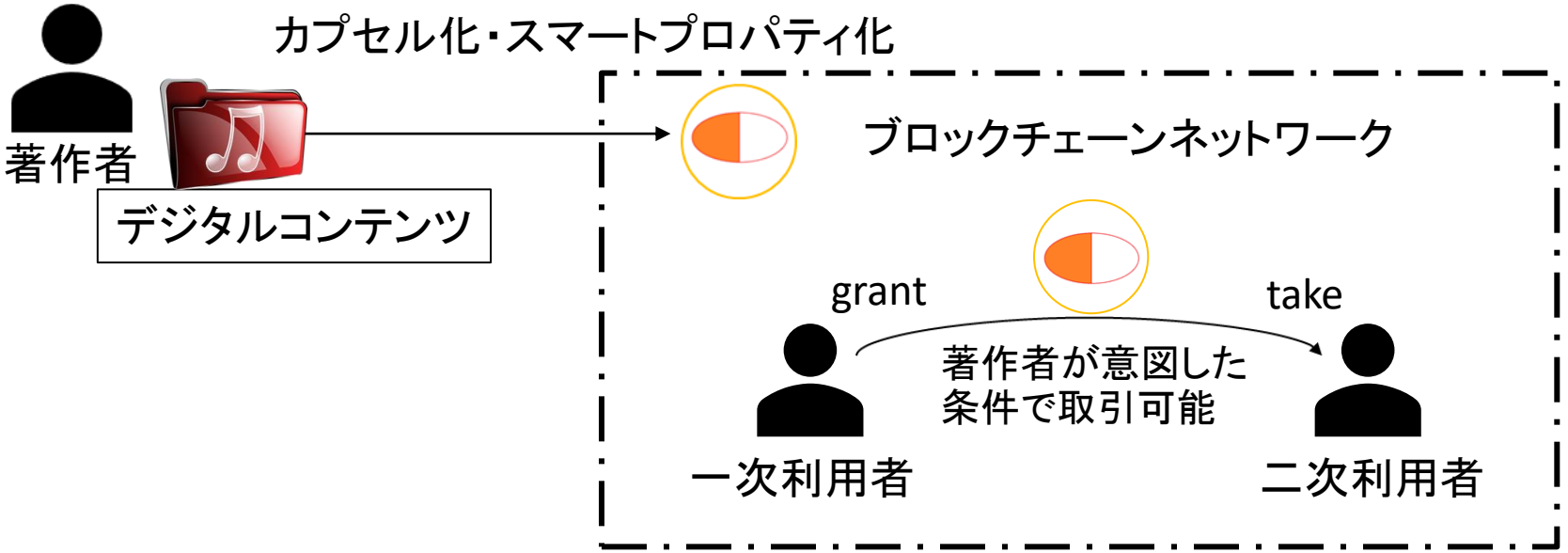
- 情報カプセルは、権利者に関する情報、アクセス権や利用条件などのメタデータをエージェント同士で行う必要がある。
- エージェントの現在の所有者が誰であるかを記録し常に更新しておく必要がある。
- そこで、メタデータをブロックチェー ンに記録することでエージェントのアクセスが容易になる。
- さらに、ブロックチェーンを応用したスマートプロパティおよびスマートコントラクトと連携させることでコンテンツに関する権利や利用条件などを信頼できる第三者なしで保証することが可能となる。

提案

・デジタルコンテンツをカプセル化する（情報カプセル）

・Take-Grantモデルを情報カプセルのメタデータであるアクセス権の委譲に利用

・情報カプセルをブロックチェーンネットワークで流通させる



コントラクトコード

・プログラム埋め込み型ブロックチェーン(例:Ethereum)に書くプログラムコードのこと

・EVMという仮想マシンによって実行される

➢コントラクトコードにTake-Grantモデルを記述

スマートプロパティ

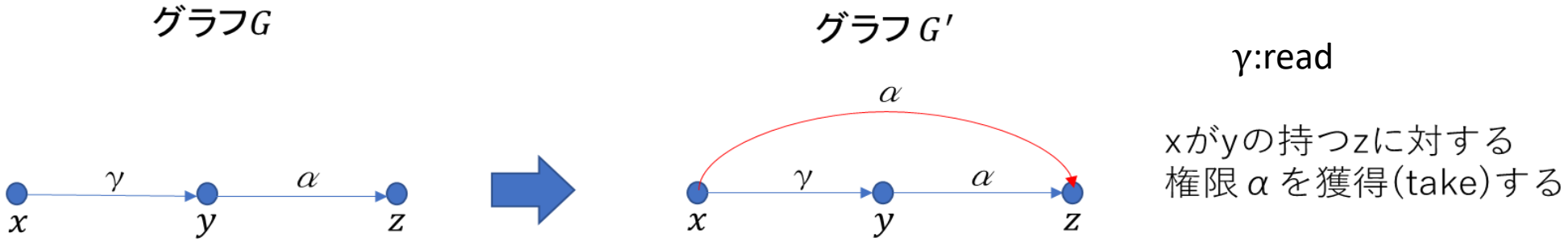
コントラクトコードに記述することで実現可能

従来のスマートプロパティは、情報カプセルの権利者の移転を記述することは可能であったが、コンテンツに対するアクセス権は従来のDRMの枠組みに依存していた。コントラクトコードと従来のDRMの枠組みを連携させることでスマートプロパティによる権利管理を確実なものとすることができる。

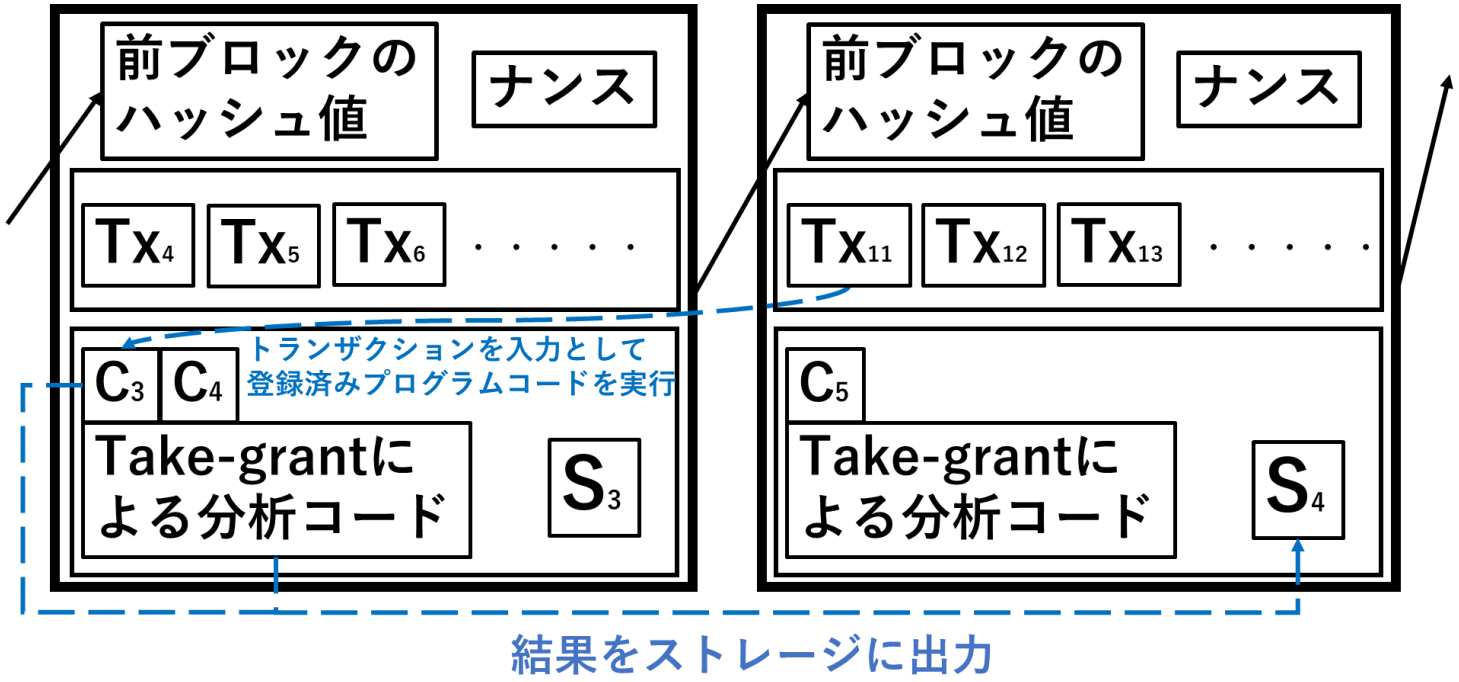
→アクセス権の移転にTake-Grantを使う

Take-Grantモデル

- 権限の委譲に関わる操作 take と grant を含む基本操作を設定し、その基本操作を持つシステムのセキュリティをグラフ理論に基いて分析する。
- システムの保護状態は、主体（および対象）を頂点、許可をラベル付きの辺とする有限の有向グラフによって表される。
- Take,Grant,Create,Call,Removeの五個の基本操作からなる。



ブロックチェーンに埋め込む Take-Grantのコントラクトコード



情報カプセルとみなす！

実装

- Gethによるプライベートネットでブロックチェーンネットワークの構築

Gethでできること

- アカウントの作成
- マイニング
- コントラクトの作成・実行　etc…

コマンド プロンプト - geth --networkid "1367204618" --maxpeers 0 --nodiscover --datadir "C:¥tools¥ethereum¥Geth1.6.5¥home¥eth_private_net" --rpc --...

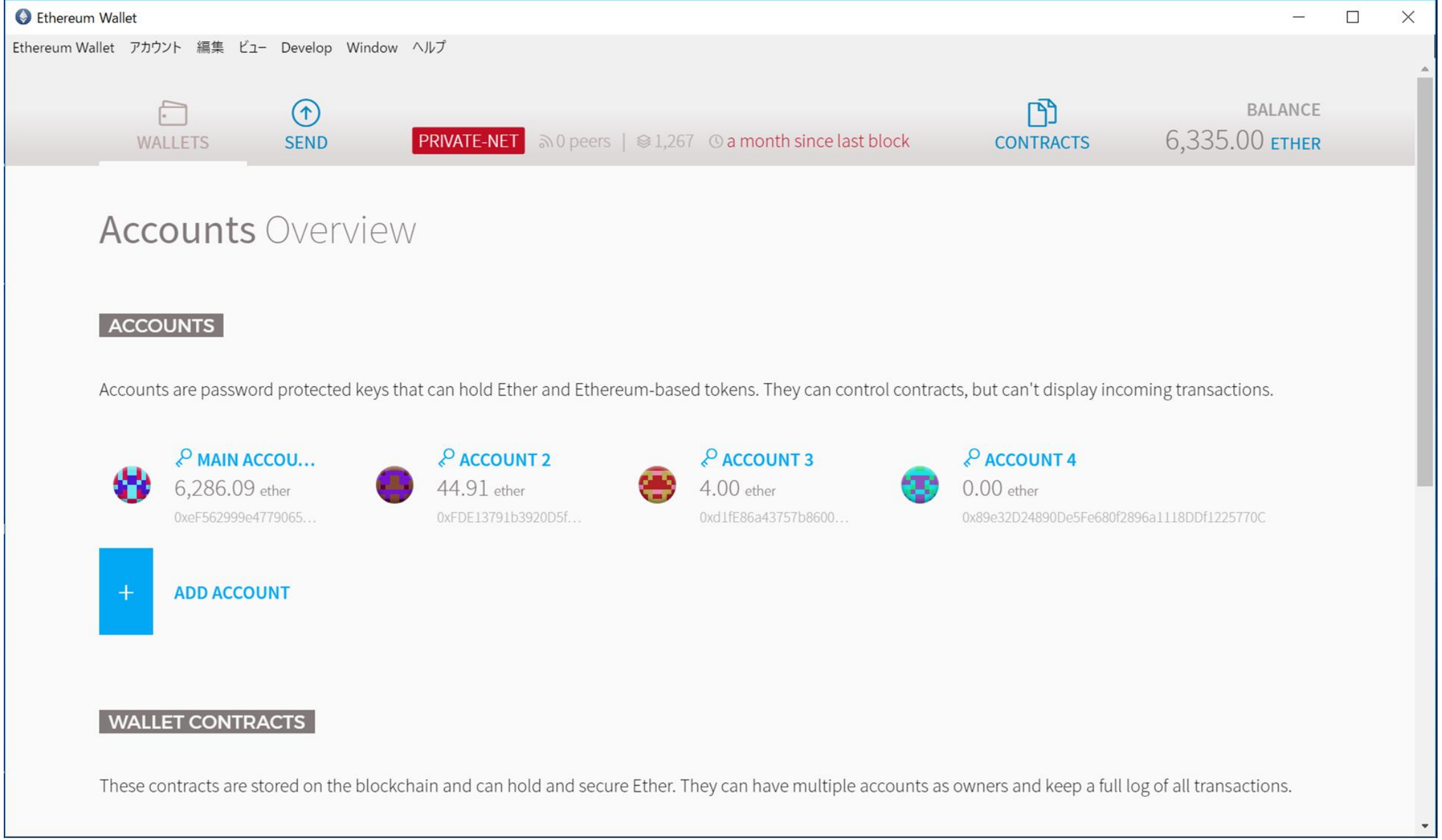
```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:¥Users¥megumu>cd ./

C:¥Users¥megumu>cd ..//

C:¥Users>cd ..//

C:¥>tools
'tools' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。

C:¥>cd tools

C:¥tools>cd ethereum

C:¥tools¥ethereum>geth --networkid "1367204618" --maxpeers 0 --nodiscover --datadir "C:¥tools¥ethereum¥Geth1.6.5¥home¥et
h_private_net" --rpc --rpcaddr "localhost" --rpcport "8545" --rpccorsdomain "*" --rpcapi "eth,net,web3,personal" --targe
tgaslimit "20000000" console 2>> C:¥tools¥ethereum¥Geth1.6.5¥home¥eth_private_net¥geth_err.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.6.5-stable-cf87713d/windows-amd64/go1.8.3
coinbase: 0xef562999e4779065bfe62f1aa43bec57d3951f4f
at block: 1267 (Thu, 20 Dec 2018 12:23:23 JST)
 datadir: C:¥tools¥ethereum¥Geth1.6.5¥home¥eth_private_net
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> _
```

実装

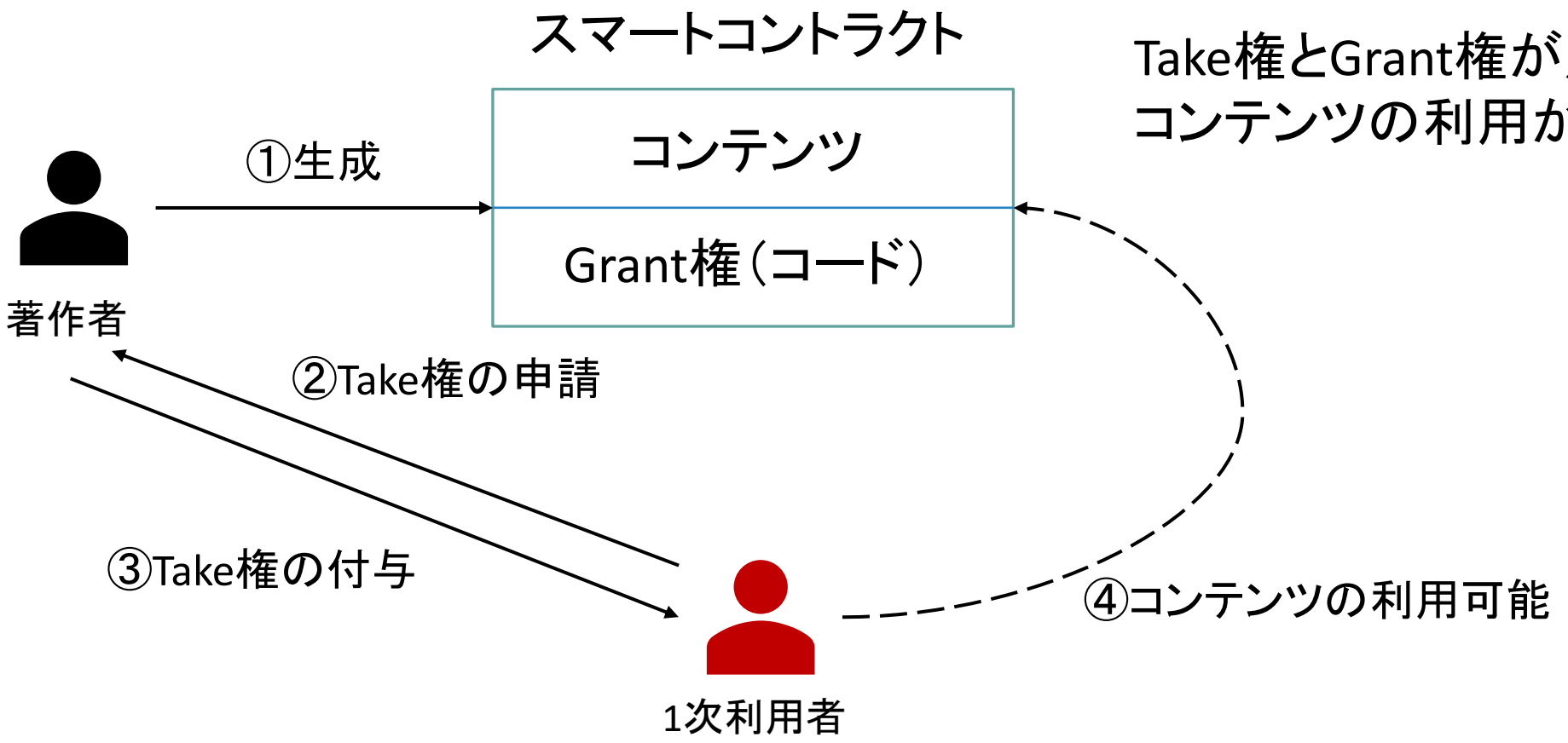
- Mist Walletによるスマートコントラクトの開発

Mist Walletでできること

- アカウント情報の確認が容易になる
- スマートコントラクトの開発が容易になる
- 繋がっているネットワークの確認　etc…

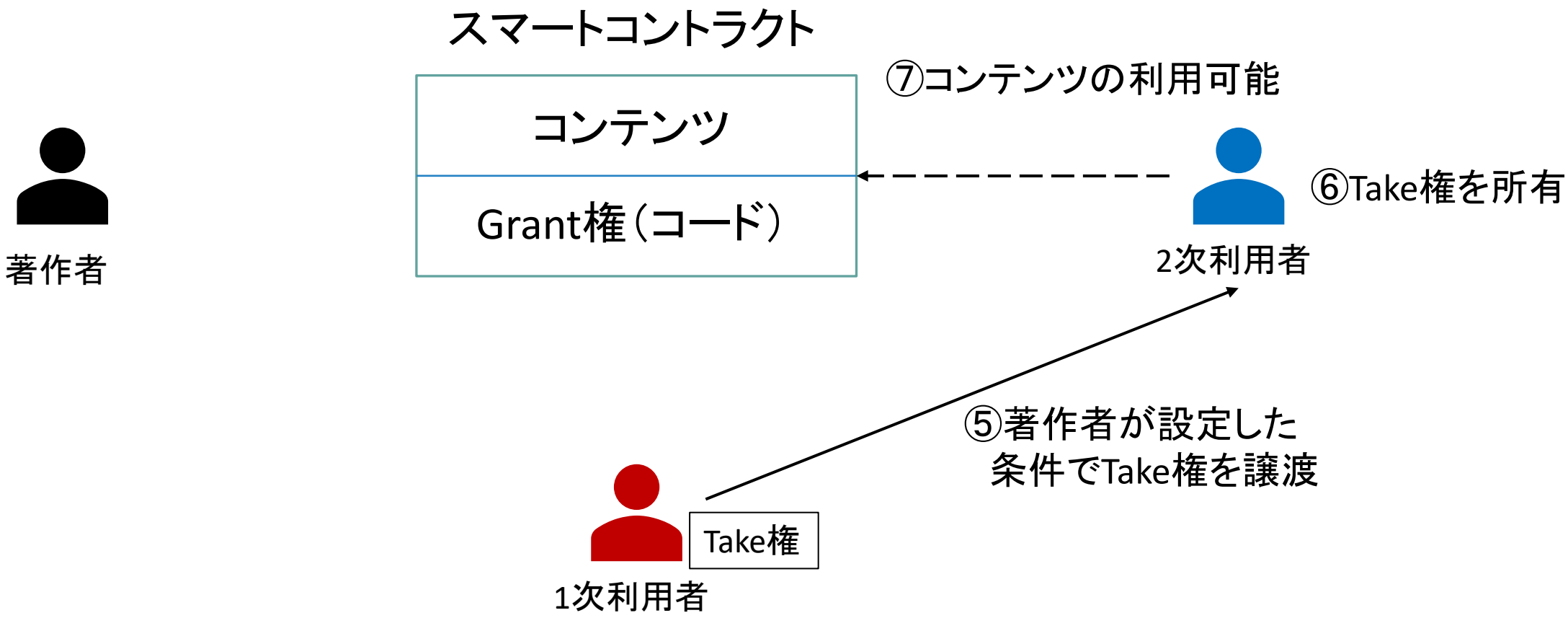

Ethereum Wallet
Ethereum Wallet アカウント 編集 ビュー Develop Window ヘルプ
WALLETS
SEND
PRIVATE-NET
0 peers | 1,267 a month since last block
CONTRACTS
BALANCE
6,335.00 ETHER
Accounts Overview
ACCOUNTS
Accounts are password protected keys that can hold Ether and Ethereum-based tokens. They can control contracts, but can't display incoming transactions.
MAIN ACCOU...
6,286.09 ether
0xeF562999e4779065...
ACCOUNT 2
44.91 ether
0xFDE13791b3920D5f...
ACCOUNT 3
4.00 ether
0xd1fE86a43757b8600...
ACCOUNT 4
0.00 ether
0x89e32D24890De5Fe680f2896a1118DDf1225770C
+
ADD ACCOUNT
WALLET CONTRACTS
These contracts are stored on the blockchain and can hold and secure Ether. They can have multiple accounts as owners and keep a full log of all transactions.

実装　コントラクトコードの設計



Take権とGrant権が照会できれば
コンテンツの利用が可能となる

実装　コントラクトコードの設計



マイドキュメント¥研究室¥マイフォルダ¥卒論（提出）¥TakeGrant.sol - sakura 2.2.0.1

ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)

```
pragma solidity ^0.4.11;
contract TakeGrant{
    // 利用者の構造体
    struct User {
        string name;    //利用者名
        address addr;   //利用者のアドレス
        uint num;       //利用者番号
        bool take;      //Take権をtrueなら持っている、falseなら持っていない
    }

    // Grant権の構造体
    struct Grant {
        string author; //著作者名
        address cont;   // コンテンツのアドレス
        bool status;    // trueの場合はコンテンツ利用可能
    }

    string public conState;     // 「OK」ならコンテンツの利用が可能
    address public author;      // 著作者のアドレス
    address public content;     // コンテンツのアドレス
    address public checkAdd;    // アドレスチェック用
    uint public UserNum;        // 利用者番号
    bool public checkTake;      // Take権チェック用

    mapping (address => User) public users;// Userを格納するマップ
    mapping (uint => Grant) public grants; // Grantを格納するマップ

    /// 著作者の権限チェック
    modifier onlyAuthor() {
        require(msg.sender == author);
        _;
    }
    /// コンテンツの権限チェック
    modifier onlyContent() {
        require(msg.sender == content);
        _;
    }
    /// Takeの権限チェック
    modifier onlyTake() {
        checkTake = false;
        checkTake = users[msg.sender].take;
        require(checkTake == true);
        _;
    }

    /// コンストラクタ
```

```
    /// コンストラクタ
    /// コンテンツのアドレスを引数に取る
    function TakeGrant(address _content) {
        author = msg.sender;
        content = _content;
        UserNum = 0;

        // Grant権の設定をする
        Grant g = grants[0];
        g.author = "Yamashita Megumu";
        g.cont = content;
        g.status = true;
    }

    /// 利用者がTake権を申請
    function ApplyForTake(string _myName) public {
        //既に同じアドレスが登録されていたら処理を終了する
        checkAdd = users[msg.sender].addr;
        require(checkAdd == 0);

        // 利用者の設定をする
        User u = users[msg.sender];
        u.name = _myName;
        u.addr = msg.sender;
        u.take = false;
    }

    /// 著作者がコンテンツ利用可能な利用者登録
    function SetUser(address _user) public onlyAuthor{
        //申請されてなければ処理を終了する
        require(users[_user].addr != 0);

        UserNum++;

        // 利用者の設定をする
        User u = users[_user];
        u.num = UserNum;
        u.take = true;
    }

    ///Take権を持つ利用者がコンテンツ可能な利用者登録
    function PassTake(address _newUser) public onlyTake{
        //申請されてなければ処理を終了する
        require(users[_newUser].addr != 0);

        UserNum++;
```

```
        UserNum++;
        // 利用者の設定をする
        User u = users[_newUser];
        u.num = UserNum;
        u.take = true;
    }
    /// コンテンツの利用時に呼ばれる関数
    function UseOfContent() public onlyTake{
        // コンテンツの利用が一時停止されていたら処理を終了する
        require(grants[0].status == true);
        // 利用に成功したら表示
        conState = "OK";
    }
    /// コンテンツ利用を一時停止する
    function contentStop1(string _msg) public onlyAuthor {
        Grant g = grants[0];
        g.status = false;
        conState = _msg;
    }
    function contentStop2(string _msg) public onlyContent {
        Grant g = grants[0];
        g.status = false;
        conState = _msg;
    }
    /// コンテンツ利用を可能にする
    function contentStart1() public onlyAuthor {
        Grant g = grants[0];
        g.status = true;
        conState = "OK";
    }
    function contentStart2() public onlyContent {
        Grant g = grants[0];
        g.status = true;
        conState = "OK";
    }
    /// 著作者が特定の利用者に対してコンテンツ利用を不可にする
    function ban(address _banUser) public onlyAuthor {
        User u = users[_banUser];
        u.take = false;
    }
    /// コントラクトを破棄するための関数
    function kill() public onlyAuthor {
        selfdestruct(author);
    }
}
[EOF]
```

●結論

・スマートコントラクトにTake-Grantモデルを記述することによってシステムのセキュリティを分析し、デジタルコンテンツの利用者同士でのやり取りが実質的に可能となった

●今後の課題

・コントラクト実行時のコスト(Gas)を抑えたコードの設計

・実際のデジタルコンテンツとどのようにして連携するか