2024-2025

Gestion d'Accès et Authentification via SmartContracts Ethereum et MetaMask

Alexandros DRIMIS
Friedrich Nihat RÖBEN
Omar ABDELKEBIR

TECHNOLOGIES BLOCKCHAIN
PROFESSEUR : OLIVIER PONCHAUT

Table des matières

1. Introduction

Ce document décrit étape par étape le processus de réalisation d'un système de gestion d'accès et d'authentification basé sur les smart contracts Ethereum et l'utilisation de MetaMask. Ce projet combine une logique blockchain (via Solidity) avec une interface utilisateur web pour offrir une solution d'accès protégée.

2. Conception et écriture du Smart Contract

A. Ecriture du code Solidity

- Pour le code, nous avons utilisé l'IDE en ligne, Remix (https://remix.ethereum.org)
- On créé un nouveau workspace, danc celui-ci on ajoute un dossier 'contracts' et enfin à l'intérieur de ce dernier nous avons créé notre fichier 'auth-contract.sol'. Celui-ci est donc un fichier 'Solidity' avec une extension '.sol'.

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleAccess {
    // Mapping pour stocker les adresses ayant accès
    mapping(address => bool) public hasAccess;

    // Prix d'accès en ETH
    uint256 public constant ACCESS_PRICE = 0; // 0 ETH pour la démo

    // Event émis quand quelqu'un obtient l'accès
    event AccessGranted(address indexed user);

    // Fonction pour obtenir l'accès
    function getAccess() public payable {    27942 gas
        require(msg.value >= ACCESS_PRICE, "Wrong amount sent");
        require(!hasAccess[msg.sender], "You already have access");

        hasAccess[msg.sender] = true;
        emit AccessGranted(msg.sender);
    }

    // Fonction pour vérifier l'accès
    function checkAccess(address user) public view returns (bool) {
        return hasAccess[user];
    }
}
```

Figure 1 : Code Solidity du Smart Contract

B. Objectif du Smart Contract

- Permettre aux utilisateurs d'obtenir un accès en échange d'un paiement (dans le cas présent 0 ETH, pour la démonstration).
- Fournir une méthode de vérification dans Remix pour confirmer si un utilisateur possède les droits d'accès. Pour cela il suffit d'entrer l'adresse de l'utilisateur dans la fonction 'checkAccess' après le déploiement. S'il a accès, celle-ci retournera 'true', sinon 'false'.

C. Compilation, Déploiement et Obtention d'ETH Sepolia

Une fois que le contrat a été écrit, il est possible de le compiler et de le déployer. Si votre seul but est d'écrire un Smart Contract, vous pouvez le déployer dans l'environnement par défaut 'Remix VM (Cancun)' qui vous permettra de le tester via des transactions simulées et non connectées à un réseau réel.

Dans notre cas, puisque nous souhaitons également implémenter une connexion MetaMask avant que l'utilisateur puisse payer un accès, nous avons choisi l'environnement 'Injected Provider – MetaMask'. Vous pouvez ensuite vous connecter au réseau Sepolia, qui est un réseau de test public pour l'Ethereum et ensuite y déployer le Smart Contract.

Attention : Pour pouvoir déployer le contrat, des frais de gas sont nécessaires, ceux-ci peuvent être payé avec des testnet ETH et non avec des ETH réels. Pour en obtenir, il vous suffit d'ajouter du Sepolia sur votre Wallet MetaMask via un faucet.

Dans notre cas, nous avons utilisé celui de Google Cloud :

https://cloud.google.com/application/web3/faucet/ethereum/sepolia

Pour rappel, les frais de transaction ("gas fees") servent à exécuter les fonctions du contrat. Ces frais sont nécessaires pour rémunérer les mineurs ou validateurs qui traitent les transactions sur la blockchain. Sur le testnet Sepolia, ces frais sont très faibles et peuvent être couverts avec les ETH de test obtenu via un faucet.

D. Récupérer l'adresse et l'ABI

- Vous pouvez copier l'adresse du contrat dans la section 'Deployed Contracts' de l'onglet de déploiement.

Figure 2 : Emplacement de copie de l'adresse du contrat

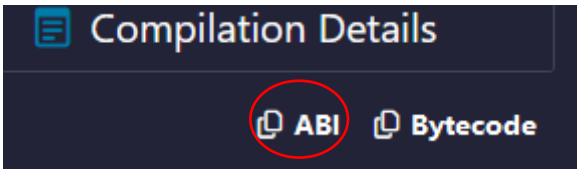

Figure 3 : Emplacement de copie de l'ABI du contrat

- L'ABI (Application Binary Interface) est générée automatiquement par Remix après la compilation du contrat. Elle décrit toutes les fonctions, événements, et structures du contrat. Pour le récupérer, rendez-vous dans l'onglet 'Solidity Compiler' après la compilation et appuyez sur le bouton de copie au bas de la section.

3. Conception de l'interface Utilisateur Web

A. Code HTML et JS

Le code est assez simple et l'interface n'est pas poussée car ce n'est pas le but. Toutefois il est intéressant de mentionner que Javascript utilise une librairie Web3 dans le code, afin de pouvoir communiquer avec le smart contract. Lors de votre lecture du code vous remarquerez que l'adresse du contrat y a été insérée mais également l'ABI.

B. Objectifs du site web

- Permettre aux utilisateurs de se connecter à MetaMask (Attention, si vous lancez le site en local, il faut utiliser un serveur pour pouvoir utiliser MetaMask : python -m http.server, puis rendez-vous sur localhost :8000 par défaut).
- Interagir avec le smart contract déployé
- Afficher un contenu protégé une fois que l'accès est obtenu.

C. Explication du flux

1. Connexion avec MetaMask
   - L'utilisateur clique sur le bouton **"Connecter avec MetaMask"**
   - Une fenêtre MetaMask s'ouvre pour demander la connexion
   - Une fois connecté, l'adresse de l'utilisateur est affichée
2. Achat d'Accès
   - L'utilisateur clique sur le bouton **"Obtenir l'accès"**
   - Une transaction est envoyée à la blockchain pour appeler la fonction getAccess() du smart contract
   - Bien que le prix soit de 0 ETH, des frais de gas doivent être payés pour exécuter la transaction
   - Si la transaction réussit, un message de succès s'affiche. (Attention, la transaction ne se fait pas immédiatement il faut attendre +-10 secondes. Cela est dû au fait que les mineurs privilégient les transactions avec les plus hauts frais afin de gagner plus.)
3. Vérification de l'accès
   - En interne, le contrat stock l'adresse de l'utilisateur comme ayant accès

4. Visualisation des transactions

Puisque nos transactions sont effectuées sur la blockchain, il est possible de les observer publiquement, pour cela il suffit de se rendre sur un site recensant toutes les transactions Ethereum, et dans notre cas plus précisément Sepolia. Le site que nous avons utilisé est le suivant : https://sepolia.etherscan.io/

L'adresse du contrat est la suivante :
« 0xf8e81D47203A594245E36C48e151709F0C19fBe8 »

Et l'adresse du Wallet que nous avons utilisé principalement pour les tests est la suivante : « 0x565191938F9Ca9F51173EB54F98a357a0Cc665DE »

Une fois sur le site, il vous suffit de rechercher une adresse pour visualiser toutes ses transactions, toutefois lorsque vous cherchez l'adresse du contrat, il y aura beaucoup plus d'adresses que prévues car puisqu'elle est publique et accessible, tout le monde peut interagir avec. En l'occurrence, pour ce contrat, il s'agit probablement de bots automatiques explorant en permanence la blockchain. Le hash et la méthode de

transaction ne sont pas repris dans les captures ci-dessous pour avoir une meilleure visibilité.

Block	Age	From		To	Amount	Txn Fee
7384658	2 hrs ago	0x56519193...a0Cc665DE	IN	0xf8e81D47...F0C19fBe8	0 ETH	0.00052338
7384643	3 hrs ago	0x56519193...a0Cc665DE	IN	0xf8e81D47...F0C19fBe8	0 ETH	0.00067344
7384643	3 hrs ago	0x56519193...a0Cc665DE	IN	0xf8e81D47...F0C19fBe8	0 ETH	0.00067344
7372306	46 hrs ago	0x159cA92b...92618A555	IN	0xf8e81D47...F0C19fBe8	0.05 ETH	0.00011133
7365293	2 days ago	0xA321fFb5...8C087D86E	IN	0xf8e81D47...F0C19fBe8	0 ETH	0.00040419
7364067	3 days ago	0xdc4A11eE...6b667a9cd	IN	0xf8e81D47...F0C19fBe8	1 wei	0.00056761

Figure 4 : Dernières transactions avec le contrat

Dans le cas du Wallet, il y a uniquement les transactions liées au compte, vous verrez que les 'OUT' sont des transactions pour interagir avec le SmartContract, et les transactions 'IN' sont des ETH Sepolia récupérées via un faucet.

Block	Age	From		To	Amount	Txn Fee
7384671	2 hrs ago	0x9a0C272b...EcB561d55	IN	0x56519193...a0Cc665DE	0.05 ETH	0.00054662
7384658	2 hrs ago	0x56519193...a0Cc665DE	OUT	0xf8e81D47...F0C19fBe8	0 ETH	0.00052338
7384643	2 hrs ago	0x56519193...a0Cc665DE	OUT	0xf8e81D47...F0C19fBe8	0 ETH	0.00067344
7384643	2 hrs ago	0x56519193...a0Cc665DE	OUT	0xf8e81D47...F0C19fBe8	0 ETH	0.00067344
6984146	59 days ago	0x56519193...a0Cc665DE	OUT	0xf8e81D47...F0C19fBe8	0 ETH	0.00014423
6984113	59 days ago	0x56519193...a0Cc665DE	OUT	0xf8e81D47...F0C19fBe8	0 ETH	0.0001335
6984099	59 days ago	0x97149BC6...90D1c1661	IN	0x56519193...a0Cc665DE	0.04072627 ETH	0.00009743

Figure 5 : Dernières transactions du Wallet

5. Conclusion

Pour conclure, vous l'avez remarqué mais l'accès du projet actuel ne mène pas à une ressource exploitable. Il s'agissait de comprendre et démontrer le fonctionnement d'une connexion sécurisée via MetaMask et ensuite de pouvoir effectuer une transaction sur la blockchain grâce au SmartContract. Toutefois, on pourrait se projeter dans le futur et prendre pour exemple une boutique d'habits en ligne sur le Web3. Pour se connecter à son compte, on utiliserait MetaMask, qui ne récolte pas nos données privées comme le font la plupart des sites actuellement, et au lieu de payer avec notre carte bancaire ou autre, il serait possible de payer en Ethereum, évitant aux boutiques de devoir passer par un tiers.