

COMPTE RENDU DE TP :

Développer, déployer et interagir avec un contrat intelligent sur Ethereum

# PIPERAUD Alexandre

## 07/09/2019

Ce rapport comporte l’intégralité des recherches, manipulations et résultats obtenus à la cour du TP du 07/09/2019. Il aborde les différents points clef approchés au cours de ce dernier, étant :

* La prise en main des outils « Remix » et « Metamask ».
* Le développement d’un « Smart Contract ».
* Le déploiement d’un « Smart Contract ».
* Les interactions avec un « Smart Contract ».

Cours : Blockchain

Enseignant : OZCAN Christophe

|  |
| --- |
| **Prise en main de l’outil Metamask** Dans un premier temps, nous devons générer un « Wallet » dans le but d’obtenir une paire de clef permettant de communiquer avec un réseau « Blockchain » publique basé sur « Ethereum » (Ethereum Sopsten). Pour ce faire, nous utiliserons l’outil « Metamask » dans un navigateur Firefox afin de pouvoir utiliser et traiter gratuitement des transactions.  Nous générons ainsi la clef publique ci-contre, que nous utiliserons pour l’ensemble des manipulations et transactions du TP :  *0xc447d09EE09ee8973d2e3D393495D42Fc771765d*  Afin de pouvoir effectuer des transferts et payer le coût des transactions que nous génèrerons, nous allons simuler l’envoi de tokens Ethereum sur notre clef privé par le biais de l’outil.  Nous pouvons ainsi observer, grâce à l’outil « Etherscan » que nous utiliserons pour l’analyse des transactions, l’envoi de la ressource vers notre wallet :    Figure 1 : Résumé de la transaction pour la génération de tokens    Figure 2 : Etat des crédits après transaction  Nous pouvons aussi observer que nous avons bien la génération d’un bloc chainé, contenant l’intégralité des données attendues (Id du block, has du block et hash du block précédent, le « Nonce » …) :    Figure 3 : Détail du block généré pour la transaction  Une fois le compte crédité, nous allons pouvoir effectuer des transactions à travers la blockchain, vers d’autres entités. Afin de vérifier si nous arrivons bien à générer des transactions, nous allons effectuer un transfert d’un montant de 1 ETH vers la clef publique suivante :  *0xc25a95A1D4a59A0E56f188f9C966A3Dad518100F*  Pour cela, nous utilisons Metamask et renseignons l’ensemble des informations demandées pour l’envois des tokens :    Figure 4 : Interface d’envois d’ETH de l’outil Metamask  Nous pouvons en relever la transaction suivante, confirmant bien la présence dans la blockchain de l’envoi du montant préalablement indiqué :    Figure 5 : Détails de la transaction sur l’envoie des token ETH Prise en main de l’outil Remix et déploiement d’un Smart Contract Afin de générer un nouveau Smart Contract dans le réseau, nous nous baserons sur un déjà développé et disponible sur la platforme Github par le biais de l’adresse suivante : « *https://github.com/cozcan/TP\_Election* ». Ce dernier permet la déclaration de candidats ainsi qu’un système de vote pour ces derniers.  Afin de pouvoir modifier, compiler et soumettre les contrats au réseau, nous utiliserons l’outil Remix.  Une fois les fichiers développé sous l’IDE Solidity importés dans l’environnement, nous demandons à l’outil de compiler notre contrat afin de récupérer un fichier « ABI » (standard permettant l’interaction avec le contrat) et un fichier « Byte Code » (constructeur du contrat). Ces derniers sont disponibles avec ce document et seront utilisés pour la création du smart contract dans la blockchain.  Nous pouvons désormais demander le déploiement du contrat par le biais de l’outil. Ce dernier vas donc envoyer, faire valider et appliquer les fichiers générés afin de créer un nouveau smart contract aillant une adresse ip public propre. Nous pouvons relever la transaction de ce déploiement ci-dessous :    Figure 6 : Détails de la transaction pour le déploiement d’un smart contract  Nous pouvons constater que dans la transaction ci-dessus, nous avons :   * La présence de la clef publique du contrat :   0xcaca723851462e2ba0469745e30c0fa492cbf5da   * Un cout de transaction plus élevé que le coup de transaction fournit dans l’exemple du polycopié. En effet, les coûts de transactions sont bien plus élevés, puisque l’état de la blockchain et différente (les conditions différents). Lors de notre transaction le nombre de transaction est plus important et donc le coût de traitement de ces dernières augmente (plus de nœuds présents donc plus de transferts nécessaires).  Interactions avec un Smart Contract Une fois le contrat déployé, nous devrions pouvoir interagir avec en luis envoyant des transactions passant en paramètre des valeurs traitées par ses fonctions. Pour plus de simplicités, nous utiliserons l’interface de l’outil.  Nous allons donc générer des utilisateurs au sein de ce dernier :   * Génération d’un candidat « PIPERAUD » :   Nous pouvons relever la transaction suivante émise :    Figure 7 : Détail de la transaction permettant l’appel à une fonction du Smart Contract  Suite à cette transaction, nous pouvons observer que le réseau à bien accepté et diffusé l’information. Nous avons désormais un nouvel utilisateur à l’id n°1 :    Figure 8 : Visualisation des données de l’ID 1 du Smart Contract sur l’outil Remix   * Génération d’un candidat « Other » :   Nous pouvons relever la transaction suivante émise :    Figure 9 : Détails de la seconde transaction permettant l’appel à une fonction du Smart Contract  Nous pouvons de nouveau constater la génération d’un nouvel utilisateur au sein de la bloackchain pour notre Smart Contract :    Figure 10 : Visualisation des données de l’ID 2 du Smart Contract sur l’outil Remix  Nous pouvons observer ici que nous avons bien une incrémentation du nombre de candidats enregistrés auprès de notre contrat sur le réseau.  Suite à la génération de ces deux candidats, nous pouvons bien observer que le contrat suite ce que nous avons généré. Ce dernier génère bien la création de candidats lors de l’appel à cette fonctionnalité avec l’intégralité des valeurs nécessaire (Id, Nom, Nombre de votes) et retourne bien le nombre total de vote. Nous pouvons aussi observer la présence de l’adresse du détenteur du contrat (aillant les privilèges sur ce dernier) :  *0xc447d09EE09ee8973d2e3D393495D42Fc771765d*  Etant le propriétaire du contrat, il est donc normal que nous y retrouvions notre adresse publique.  Nous allons désormais pouvoir tester la fonctionnalité de vote afin d’observer les interactions lorsque nous ou un utilisateur externe (clef publique différente) tente d’interagir avec notre contrat :   * Emission d’un vote pour le premier candidat de la liste (ID 1) :   Nous pouvons donc relever la transaction suivante pour l’émission d’un vote :    Figure 11 : Détails de la transaction permettant l’appel à une fonction de vote du Smart Contract    Figure 12 : Traces générées par le contrat lors de l’exécution de la fonction  Nous pouvons observer ci-dessus que nous avons bien eu une réaction au sein du code du contrat avec la modification d’une valeur. Lorsque que nous vérifions l’état de notre candidat par le biais de l’interface graphique, nous pouvons aussi observer ce changement :    Figure 13 : Visualisation des données de l’ID 1 du Smart Contract sur l’outil Remix  De plus, nous pouvons aussi observer que le nombre total de votes a été automatique incrémenté.   * Emission d’un vote pour le premier candidat de la liste (ID1), en provenance d’une autre clef public :   Nous pouvons donc relever la transaction suivante émise par le détenteur de la clef public :  *0xc8134fa8c874359e9aa8ecd005af6a409446a59a*    Figure 14 : Détails de la transaction permettant l’appel à une fonction de vote du Smart Contract depuis une autre clef public  Nous pouvons donc observer que nous avons bien eu un vote émis par le propriétaire d’une autre clef sur notre contrat. Gestion et administration d’un Smart Contract Nous pouvons désormais administrer le Smart Contract utilisé jusqu’à afin de céder les droit à un autre utilisateur de la blockchain. Pour ce faire, nous utiliserons de nouveau l’interface Remix e récupérant l’adresse public d’un autre utilisateur et en la renseignant dans le champ approprié. Nous utiliserons l’adresse publique ci-contre :  [*0xc8134fa8c874359e9aa8ecd005af6a409446a59a*](https://ropsten.etherscan.io/address/0xc8134fa8c874359e9aa8ecd005af6a409446a59a)  Nous pouvons observer les détails de la transaction émis sur le réseau :    Figure 15 : Détails de la transaction permettant le transfert de propriété de ce dernier    Figure 16 : Traces générées par le contrat lors de l’appel à la fonction de transfert de propriété  Nous pouvons observer que nous avons bien eu l’appel à une méthode de notre contrat et pouvons relever la présence des clefs publiques respectives de l’émetteur et du récepteur des privilèges.  De plus, nous pouvons observer par le biais de l’interface graphique un changement dans la section propriétaire :    Figure 17 : Visualisation du propriétaire du Smart Contract sur l’outil Remix  Nous pouvons donc conclure que nous avons bien effectué et diffusé le transfert des propriétés de notre contrat à un autre tiers.  Aillant vue la présence de propriété sur les contrats et la possibilité de leurs transfert, nous pouvons désormais manipuler les fonctions afin de leurs ajouter des sécurités et restrictions. En effet, nous pouvons au sein du code, lors de la déclaration des méthodes du contrat, mettre des restrictions sur l’utilisation de ces dernières. Comme le montre l’exemple ci-dessous, nous pouvons déclarer la stricte utilisation de notre méthode au propriétaire du contrat :    Figure 18 : Extrait de code d’une méthode sans restrictions d’utilisations    Figure 19 : Extrait de code d’une méthode avec restriction d’utilisation à « onlyOwner »  Nous pouvons donc rajouter une sorte d’héritage à notre fonction, qu’y sera appelée lorsque la méthode « addCandidate » sera utilisée.  Nous déclarons donc une méthode « modifier » qu’y sera appelée avant son exécution :    Figure 20 : Extrait de la méthode de control d’utilisation des méthodes du contrat  La déclaration ci-dessus va donc vérifier au sein de la requête émise que la clef public correspond à celle enregistrée dans les données du contrat (celle de la clef propriétaire), et rejettera avec un message d’erreur toute transaction utilisant une autre clef.  Afin de pouvoir tester ce fonction, nous aurions pu de nouveau émettre une demande d’ajout de candidat auprès du contrat avec la clef public propriétaire et une autre clef d’un autre tiers. |
|  |
|  |