Développer un contrat/programme sur Ethereum

Stéphane Bortzmeyer
stephane+pses@bortzmeyer.org

Pas Sage en Seine / Hacker Space Festival, 2 juillet 2016

Qu'est-ce qu'Ethereum

- Une **chaîne de blocs** (oui, comme Bitcoin)
- Haut niveau : une structure de données publique et sécurisée. Tout le monde peut vérifier. Pas d'organisme de confiance.
- Bas niveau : les transactions sont signées par l'envoyeur, regroupées en **blocs**, et les blocs sont chaînés. Insérer un bloc nécessite une **preuve de travail**.
- Une transaction exécute du code (un petit programme)

Les ethers

- La monnaie d'Ethereum
- Générée par le **minage** (le même processus qui insère les blocs)

Mon fric

```
> cab()
  eth.accounts[0]: 0xaf8e19438e05c68cbdafe33ff15a439ce6742972
      balance: 121.560302052611260406 ether
  eth.accounts[1]: 0x2dda57ee99c806477ba05114801543f48ab3d338
      balance: 0 ether

> eth.sendTransaction({from: eth.accounts[0],
        value: web3.toWei(25, 'ether'),  to: eth.accounts[1]})
"0xa8845821d8454637c7882fac583390d46850a03302f031a5cbf039c6987930d7"

> cab()
   eth.accounts[0]: 0xaf8e19438e05c68cbdafe33ff15a439ce6742972
       balance: 96.559882052611260406 ether
   eth.accounts[1]: 0x2dda57ee99c806477ba05114801543f48ab3d338
       balance: 25 ether
```

Programmable

- Jusque là, aucune différence avec le Bitcoin
- L'importante nouveauté d'Ethereum : c'est une machine de Turing. Le code exécuté est écrit dans un langage de Turing.
- On peut donc tout faire (tout ce qu'on sait programmer)
- Ethereum n'est pas que pour la monnaie

À quoi ça sert

- Tout ce qui doit être vérifiable par tous et contrôlé par personne
- Monnaie, cadastre, registre de noms (cf. Namecoin), « uberiser Uber (disruption !) »
- En revanche, inadapté aux gros calculs : la machine virtuelle Ethereum est lente et chère
- Ces programmes sont nommés des **contrats**
- Ils peuvent s'exécuter sans intervention humaine (conséquences philosophiques, légales et politiques importantes)

Le langage machine

- Niveau assembleur (une machine à pile)
- `PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x90 DUP1 PUSH1 0x11 PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN STOP PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x0 CALLDATALOAD`
- On ne va pas l'utiliser pour programmer mais il faut se rappeler que c'est ça qui est stocké et exécuté dans la chaîne

Solidity

- Le principal langage de programmation utilisé actuellement
- Langage impératif
- Un contrat est composé de variables globales, rémanentes, et de fonctions
- Un contrat peut appeler d'autres contrats

Un environnement très spécial

- Attention, la chaîne de blocs peut être déroutante
- On modifie une variable, on ne verra le résultat que lorsque le bloc aura été miné
- Interactions avec l'extérieur très limitées (tous les mineurs doivent avoir le même résultat : pas question de requêtes HTTP externes)
- Déboguage très difficile (pas d'interface utilisateur)
- Ordre des transactions pas contrôlable

Un contrat : Soleau

- But : permettre de prouver l'antériorité d'un fichier (roman, chanson, programme, données) pas encore publié, en stockant son condensat (*hash*) dans la chaîne (donc, sans révéler le contenu). *Trusted Timestamping*
- La chaîne est ordonnée : chaque bloc a un numéro, croissant de manière monotone (et le bloc a une date)
- On enregistre le condensat, la date et l'adresse du déposant
- Plus tard, il suffira de produire le fichier pour prouver à tous son antériorité

soleau.sol 1/3

```
contract Soleau {

  uint price = 1 ether;
  struct Record {
    address holder;
    bool exists;
    uint created;
  }
  mapping (string => Record) _records;
```


soleau.sol 2/3

```
  function record(string hash) returns (bool success,
                                        bool already, uint block) {
    if (msg.value < price) {
      success = false;
      msg.sender.send(msg.value); /* We're nice, we refund */
      return;
    } /* Else we keep the money */
    if (_records[hash].exists) {
      success = true;
      already = true;
      block = _records[hash].created;
    } else {
      _records[hash].exists = true;
      _records[hash].holder = msg.sender;
      _records[hash].created = now;
      success = true;
      already = false;
      block = _records[hash].created;
    }
  }
```

soleau.sol 3/3

```
  function get(string hash) returns (bool success,
                                     uint block, address holder) {
    if (_records[hash].exists) {
      success = true;
      block = _records[hash].created;
      holder = _records[hash].holder;
    } else {
      success = false;
    }
  }
}
```


Un contrat : FindByHash

- But : accès par le contenu (ICN, *Information-Centric Networking*, cf. RFC 7476 et RFC 6920)
- Stocker le·s URL permettant d'accéder à un contenu donné
- Contenu identifié par son condensat

findhash.sol 1/4

```
contract FindHash {

  uint price = 0.01 ether;

  address _manager;

  struct Bucket {
    bool exists;
    uint num; // Never decreases: we can only add records, not remove them.
    mapping(uint => Record) records; // Multi-valued mappings not
    // possible, we need two levels of mapping
  }
  struct Record {
    bool exists;
    string uri;
  }
  mapping(string => Bucket) data;
```


findhash.sol 2/4

```
  function FindHash() {
    _manager = msg.sender;
  }

  function set(string key, string value) returns (bool) {
    bool result;
    // We do not check that key is a hash: it allows the caller to
    // choose the algorithm s·he likes
    if (msg.value < price) {
      msg.sender.send(msg.value); /* We're nice, we refund */
      return false;
    } else { // Gimme money
      result = _manager.send(msg.value);
      if (!result) {
          throw;
      }
    }
```

findhash.ol 3/4

```
    if (data[key].exists) {
      data[key].records[data[key].num] = Record(true, value);
    }
    else {
      data[key].exists = true;
      data[key].records[0] = Record(true, value); // We do not check the synt
  // URI
    }
    data[key].num++;
    return true;
  }
```


findhash.sol 4/4

```
  // Retrieve the values

  function num_of(string key) constant returns (uint) {
    return data[key].num;
  }

  function get(string key, uint index) constant returns (bool, string) {
    if (!data[key].exists || !data[key].records[index].exists) {
      return (false, "");
    }
    return (true, data[key].records[index].uri);
  }
}
```

Récupérer les valeurs depuis la console JavaScript

```
/* Get all the URIs for a given hash */

findhashC = eth.contract(abi);
findhash = findhashC.at("0x78db9a1dfe1b46c4361ac91cda111f5366ddc0e5");

function getAlls(hash) {
    for (var i = 0; i < findhash.num_of(hash); i++) {
        result = findhash.get(hash, i);
        console.log("#" + i + " " + result);
    }
}
```

```
% geth --preload get-urls.js --exec 'getAlls("cf4163b8f4c13b915e246ea7d2792
#0 true,https://localhost/
#1 true,https://www.bortzmeyer.org/toto
```


Piège : la sécurité

- Langage de Turing $\longrightarrow$ tout est possible. Y compris du code malveillant (ce n'est pas par hasard que le langage de Bitcoin est si limité)
- Environnement nouveau et surprenant pour les programmeurs
- De l'argent en jeu (et l'argent rend fou)
- Les contrats sont censés se débrouiller seuls. Que faire en cas de bogue ?

La crise de The DAO

- Un fonds d'investissement en ethers. Des gens mettent des ethers, d'autres proposent des projets à financer, les investisseurs votent. Pas d'entreprise ou d'association.
- Dans les 100 Meuros récoltés
- Un voleur a trouvé une bogue et a volé le tiers des fonds
- Une intervention manuelle a permis de fruster le voleur (alors que le contrat était censé être la loi, et être autonome)

Exemples de pièges pour le programmeur

- Un contrat peut en appeler en autre mais cet autre ne fait pas partie de la transaction (exceptions non propagées $\rightarrow$ testez le code de retour !)
- Pas de distinction entre compte et contrat : vous croyez envoyer de l'argent à une adresse mais vous exécutez du code (*fallback function*)
- Si une fonction n'est pas réentrante, un appel par un contrat que vous appelez peut changer l'état

Leçons à en tirer

Pour tous :

- Les contrats ont des bogues
- Réfléchissez avant d'investir ! Pour adultes seulement !
- Imaginez ce que serait devenu le Web s'il y avait des bogues dans les codes PHP et JavaScript

Pour les programmeurs :

- Solidity ne vous empêche pas de vous tirer une balle dans le pied
- Keep calm et relisez et faites relire vos programmes (attention à l'hubris)
- Passer à des langages fonctionnels ?
- Validation formelle du code ?

Piège : la confiance et la validation

- Pourquoi les gens enverraient-ils du fric à votre contrat ?
- Une bonne idée ne suffit pas, il faut aussi avoir confiance dans sa réalisation
- D'où vient la confiance ?
  - Code source disponible et lisible (c'est le minimum)
  - Évaluation par plusieurs experts
  - Mais la chaîne exécute du code machine : il faut aussi vérifier que ce code correspond au source.

Piège : la mise à jour

- Tous les programmes ont des bogues
- Les contrats aussi
- Comment mettre à jour ? Le code à une adresse donnée de la chaîne est immuable.
  - Prévoir la possibilité de migrer les fonds vers le nouveau contrat (cela ouvre une nouvelle voie d'attaque)
  - Avoir une indirection entre le nom et l'adresse du contrat (cela ouvre une nouvelle voie d'attaque)

Conclusion

- Une technologie géniale
- Plein de distractions pour les informaticiens
- C'est nouveau et disruptif. Donc :
  1. Attention avant d'y confier trop d'argent
  2. Difficile de prévoir le futur
- Pour les pionniers et les courageux explorateurs