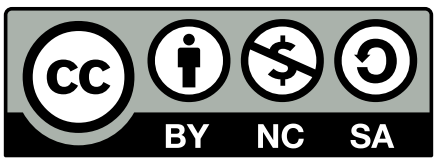
université de BORDEAUX

Étude et implémentation de Smart-Contracts pour Blockchain

Paul Hermouet
<paul.hermouet@etu.u-bordeaux.fr>

Master CSI, Université de Bordeaux, France

05 septembre 2018



Introduction

- Outil d'exécution symbolique pour Smart Contrats
- Protocole d'enchères en tant que preuve de concept
- État de l'art blockchain et IoT

Plan

Plan

Un peu d'histoire...

université de BORDEAUX

1992 Cynthia Dwork / Moni Naor inventent le concept de preuve de travail (*Pricing via processing or combatting junk mail*).

1998 Nick Szabo : *Bit Gold* : système monétaire numérique basé sur la preuve de travail.

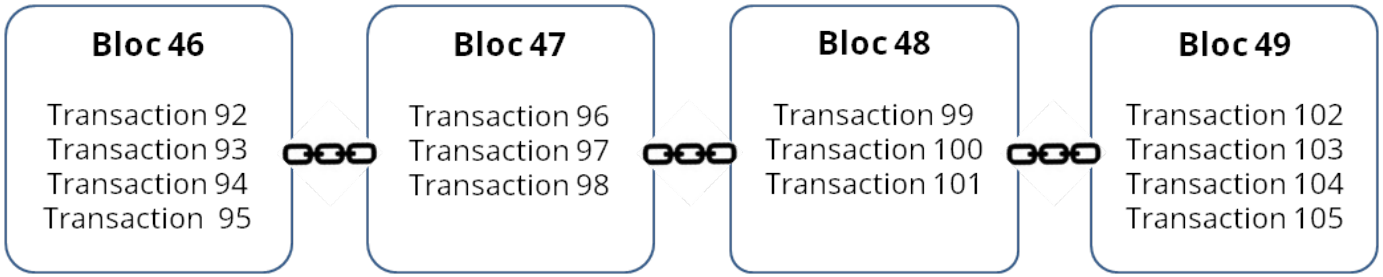
2008 Satoshi Nakamoto : *Bitcoin : a peer to peer electronic cash system*

2013 Vitalik Buterin crée Ethereum : *Ethereum White Paper* (formalisation).

2015 Lancement du premier réseau Ethereum.

Blockchain

- Un réseau basé sur une blockchain :
  - est distribué.
  - possède une une chaîne de blocs = un historique de transactions.
  - chaîne de blocs => solde de chaque utilisateur.
- $tx = (envoyeur, destinataire, montant)$



Diffusion

- Tous les utilisateurs possèdent une copie de l'historique.
- Tous les utilisateurs ont des voisins dans le réseau.
- Transmission des transactions aux voisins.
- Délai $\longmapsto$ registre différents
- Preuve de travail $\longmapsto$ Validation d'un bloc $\longmapsto$ Consensus

Smart Contracts

université de BORDEAUX

- Smart Contract : un compte avec en plus :
  - un bytecode
  - un stockage
- Ethereum Virtual Machine (EVM)
  - pile d'exécution
  - accès au stockage + état du réseau
- $tx = (envoyeur, destinataire, montant, data)$

Plan

Failles

université de BORDEAUX

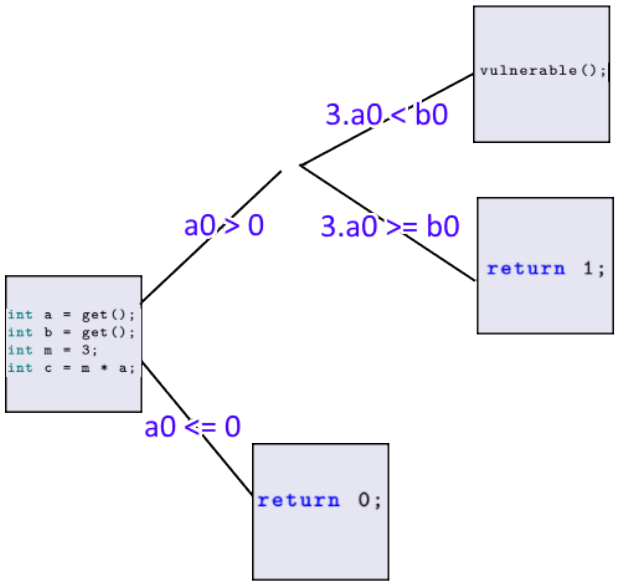
```
1     function batchTransfer(
2         address[] _recipients,
3         uint _amount) public {
4
5         uint totalAmount = _recipients.length * _amount;
6         require(totalAmount <= balances[msg.sender]);
7         balances[msg.sender] -= totalAmount;
8         for (uint i = 0; i < _recipients.length; i++) {
9             balances[_recipients[i]] += _amount;
10        }
11    }
```

Exécution Symbolique

université de BORDEAUX

- Arbre d'exécution
  - Nœuds : état d'exécution
  - Branches : contraintes

```
int a = get();
int b = get();
int m = 3;
int c = m * a;
if (a > 0)
    if (c < b)
        vulnerable();
    return 1;
return 0;
```



EVM Bytecode

...60035460001460105 7...

1 - PUSH 0x03
3 - SLOAD
4 - PUSH 0X00
6 - EQ
7 - PUSH 0X10
9 - JUMPI

STACK
0x03

EVM Bytecode

...60035460001460105 7...

1 - PUSH 0x03
3 - SLOAD
4 - PUSH 0X00
6 - EQ
7 - PUSH 0X10
9 - JUMPI

STACK
STORAGE(3)

EVM Bytecode

...60035460001460105 7...

1 - PUSH 0x03
3 - SLOAD
4 - PUSH 0X00
6 - EQ
7 - PUSH 0X10
9 - JUMPI

STACK
0x00
STORAGE(3)

EVM Bytecode

...60035460001460105 7...

1 - PUSH 0x03
3 - SLOAD
4 - PUSH 0X00
6 - EQ
7 - PUSH 0X10
9 - JUMPI

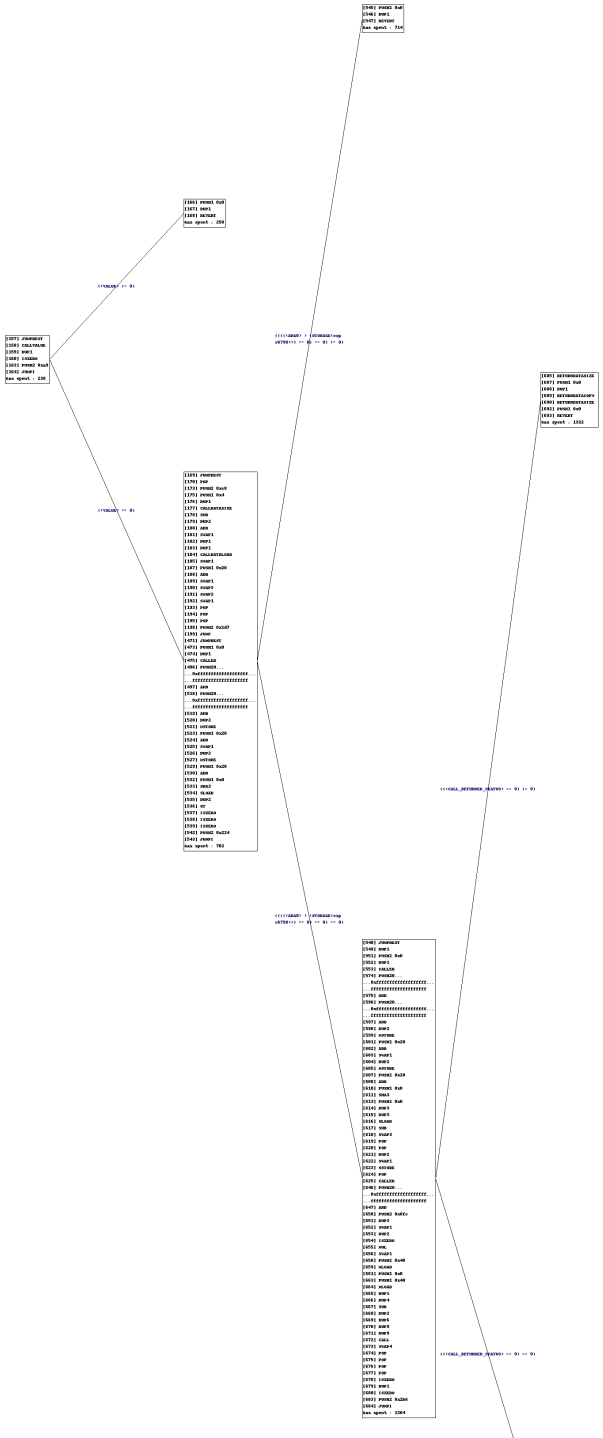
STACK
(STORAGE(3) == 0)

Sur les Smart Contracts

université de BORDEAUX

- **input** : bytecode
- émulation de l'EVM
- construction de l'arbre d'exécution
- pattern-matching
- **output** : rapport sur les failles (+ view)

```
1      function withdraw (uint _amount) public {
2          require(_amount <= balances[msg.sender]);
3          balances[msg.sender] -= _amount;
4          msg.sender.transfer(_amount);
5      }
6  }
```



Plan

- **E. Blass et F. Kerschbaum - Strain : A secure auction for blockchains**
- Enchères sous pli cachetées
- Décentralisé $\longmapsto$ blockchain
- Comparer deux nombres en n'en connaissant seulement un
- Chiffrement de Goldwasser-Micali (problème de résiduosité quadratique)
- Entiers de Blum : $n = p \cdot q$ (p et q premiers, $p \equiv q \equiv 3 \ (mod\ 4)$)
- $(\frac{-1}{n}) = 1$ et -1 n'est pas un résidu quadratique modulo n
- $\forall a \in \mathbb{Z}/n\mathbb{Z}, a^{\frac{(p-1)(q-1)}{4}} = 1$ si a est un résidu quadratique modulo n,
  $= -1$ sinon.

Chiffrement

université de BORDEAUX

Algorithm 1: GenKeys

$p, q \xleftarrow{\$} \{x \in \mathbb{N}, x \text{ prime}, x \equiv 3 \pmod 4\}$;
$pk \leftarrow p \times q$;
$sk \leftarrow \frac{(p-1)(q-1)}{4}$;

Algorithm 2: Encrypt(pk, b)

$r \xleftarrow{\$} (\mathbb{Z}/n\mathbb{Z})^*$;
$c \leftarrow r^2 \times (-1)^b \pmod{pk}$;

Algorithm 3: Decrypt(sk, pk, c)

if $(c^{sk} = 1 \pmod{pk})$
    $b \leftarrow 0$;
else
    $b \leftarrow 1$;

Variante AND

Algorithm 4: $\text{Encrypt}^{AND}(pk, b)$

if $(b = 0)$

$(a_1, a_2, ..., a_\lambda) \xleftarrow{\$} \{0,1\}^\lambda$;

return $(Encrypt(a_1), ..., Encrypt(a_\lambda))$

else

return $\overbrace{(Encrypt(0), ..., Encrypt(0))}^{\lambda \ times}$

Algorithm 5: $\text{Decrypt}^{AND}(sk, pk, c)$

for (i from 1 to λ)

if $(Decrypt(sk, pk, c[i]) = 1)$

return 0

return 1

Homomorphismes

université de BORDEAUX

- Homomorphismes
  - **XOR** : $DecryptOneBit(EncryptOneBit(b_1) \times EncryptOneBit(b_2)) = b_1 \oplus b_2$
  - **NOT** : $DecryptOneBit(-EncryptOneBit(b)) = NOT(b)$
  - **AND** : $EncryptOneBit^{AND}(b_1) = (c_{1,1}, ..., c_{1,\lambda})$
    $EncryptOneBit^{AND}(b_2) = (c_{2,1}, ..., c_{2,\lambda})$
    $DecryptOneBit^{AND}((c_{1,1} \times c_{2,1}, ..., c_{1,\lambda} \times c_{2,\lambda})) = b_1 \ AND \ b_2$
- Porte logique pour comparaison
  - $F = \bigvee_{i=0}^{n} F_i$
    $F_i = x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} (x_j = y_j)$
  - F est vraie si et seulement si x $>$ y
- Protocole
  - A envoie le chiffré de son montant $c_A = Encrypt(pk_A, m_A)$ aux autres participants.
  - B calcule $c_{A,B} = Encrypt(pk_A, m_B)$.
  - B envoie $cmp_{A,B} = F(c_A, c_{A,B}$ à A.
  - A déchiffre $cmp_{A,B}$

Questions ?