

Introduction to blockchain technology and smart-contract programming

Sylvain Conchon

University Paris-Saclay

- ▶ Introduction
- ▶ Couche P2P : sécurité et cohérence des données
- ▶ Bitcoin
 - ▶ Modèle UTXO
 - ▶ Langage Script
 - ▶ Clients légers, Arbres de Merkle, Lightning Network

INTRODUCTION

Qu'est-ce qu'une Blockchain ?

Une blockchain est comme un grand **livre de comptes**

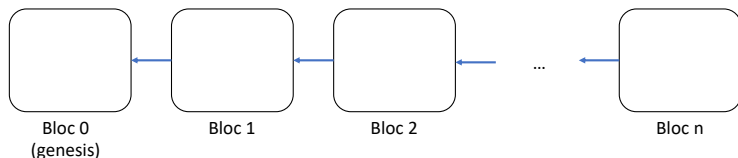


Chaque **page** de ce livre correspond à un **bloc** où sont enregistrées des **transactions** (d'argent ou autre)

Dans le monde des blockchains, ce livre de comptes est appelé un **registre** (*ledger* en anglais).

Chaînes de blocs

La structure de données qui contient ces blocs est similaire à une **liste chaînée**. Chaque bloc i contient un pointeur vers le bloc $i - 1$.



Le premier bloc (celui qui commence la blockchain) est traditionnellement nommé **genesis**.

Transactions

Les transactions enregistrées dans les blocs impliquent **deux** “clients” :

- ▶ celui qui est à l'**origine** de la transaction
- ▶ celui qui est le **bénéficiaire** de la transaction

Les transactions contiennent d'autres informations, comme une date et un montant

date	de	à	montant
03/01/2021 21:56	Marc	Alain	0.01 B
03/01/2021 21:58	Marc	Julie	0.02 B
03/01/2021 21:59	Julie	Bob	0.005 B
⋮	⋮		

À partir de ces transactions contenues dans tous les blocs, on tient à jour les comptes...

Comptes bancaires

Client	Montant
#1	100 ₮
#2	20 ₮
#3	50 ₮
⋮	⋮

Registre distribué

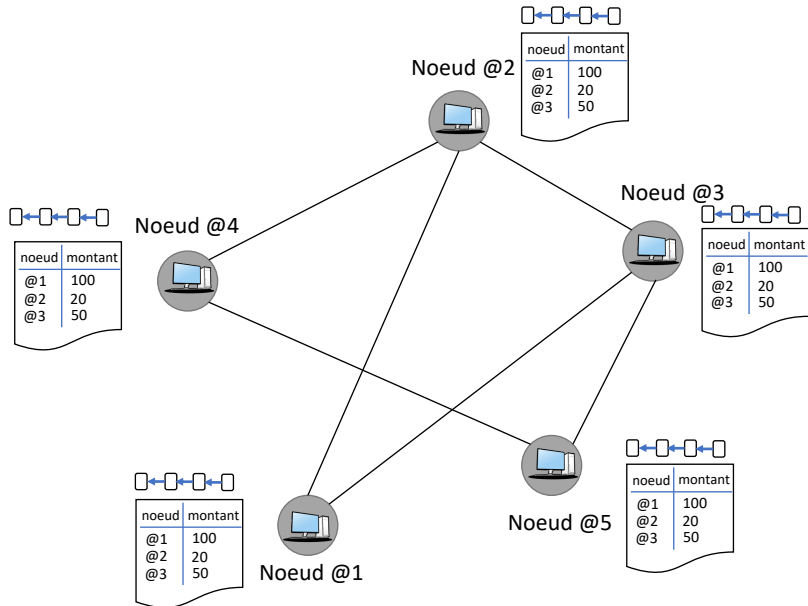
La particularité de la blockchain est que ce registre n'est pas stocké sur un serveur centralisé.

Il est **partagé** (ou recopié) sur un grand nombre d'ordinateurs à travers le monde entier qui sont tous **interconnectés**.

On parle de **registre distribué**, sans aucun organe central de contrôle.

L'intérêt d'une structure distribuée est une très grande résistance **aux pannes** (des serveurs) et aux **attaques de sécurité**.

Architecture décentralisée



Quelques problèmes sous-jacents à une blockchain

Comment s'assurer qu'un bloc **ne va pas être supprimé** de la chaîne ? (c'est-à-dire qu'une page du registre ne soit supprimée)

Comment vérifier l'**intégrité** d'un bloc ? C'est-à-dire que personne ne peut modifier le contenu des transactions ou en ajouter, en supprimer ?

Comment faire pour que toutes les machines aient **la même version** du registre ?

Comment faire pour être sûr qu'une personne **ne dépense pas plus** qu'elle n'a d'argent ?

Comment s'assurer qu'une personne **ne dépense pas l'argent d'une autre** ?

Etc.

Pour résoudre ces problèmes, l'implémentation d'une blockchain repose (entre autre) sur :

- ▶ une architecture de **réseau pair-à-pair** (P2P)
- ▶ un protocole de **consensus distribué**
- ▶ des primitives **cryptographiques**

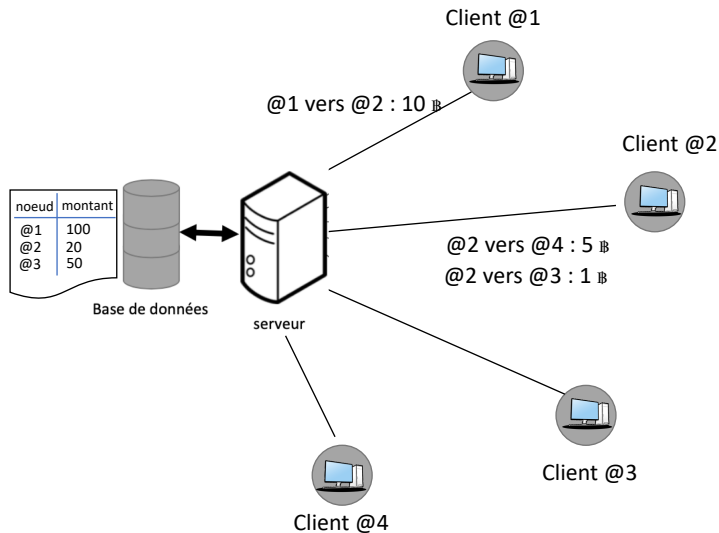
RÉSEAUX P2P

Un réseau Pair-à-Pair (Peer-to-Peer en anglais) est constitué d'un ensemble d'ordinateurs qui s'**organisent entre eux** pour s'échanger de l'information (fichiers, etc.)

Contrairement à une architecture centralisée du type **client/serveur**, avec un serveur et des clients qui s'y connectent, les ordinateurs d'un réseau P2P jouent **à la fois** les rôles de client et de serveur.

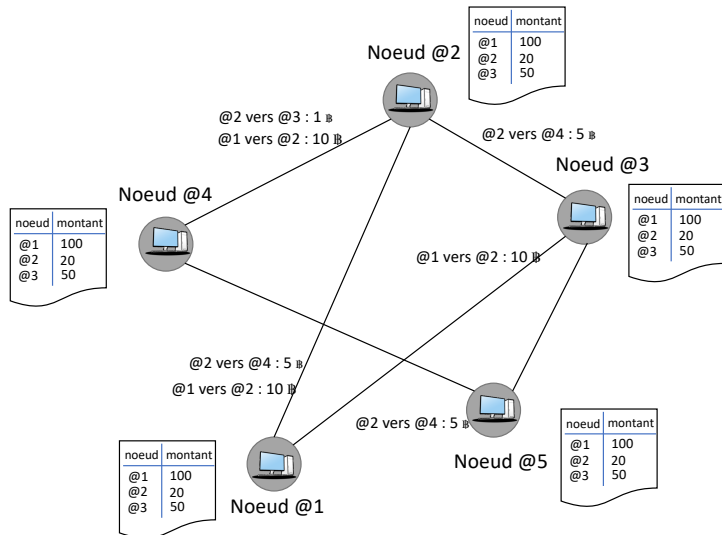
Exemples de réseaux P2P : KazaA, BitTorrent, eDonkey, eMule

Architecture Client-Serveur



Architecture Pair-à-Pair

La distinction entre client et serveur **disparaît** : il n'y a plus que des noeuds qui communiquent directement les uns avec les autres



Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- ▶ Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit **recevoir**, **envoyer** et **relayer** les données.

Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- ▶ Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit **recevoir**, **envoyer** et **relayer** les données.
- ▶ **Réduction** des coûts, **simplicité** d'utilisation, **rapidité** d'installation.

Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- ▶ Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit **recevoir**, **envoyer** et **relayer** les données.
- ▶ **Réduction** des coûts, **simplicité** d'utilisation, **rapidité** d'installation.
- ▶ **Résistant** aux pannes : si des machines tombent en panne, le réseau continue de fonctionner (redondance).

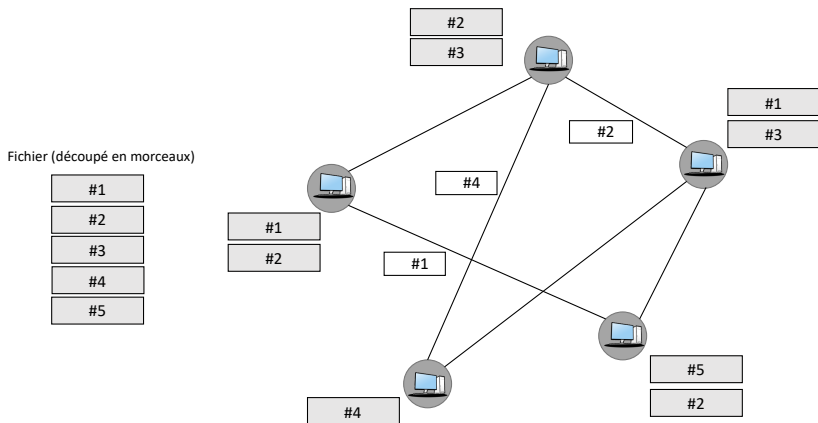
Avantages des réseaux P2P (vs. Client/Serveur)

Les avantages des réseaux P2P (en comparaison des réseaux Client/Serveur) :

- ▶ Aucune hiérarchie centrale (réseau maillé) : chaque nœud doit **recevoir**, **envoyer** et **relayer** les données.
- ▶ **Réduction** des coûts, **simplicité** d'utilisation, **rapidité** d'installation.
- ▶ **Résistant** aux pannes : si des machines tombent en panne, le réseau continue de fonctionner (redondance).
- ▶ **Augmentation** des performances : plus il y a de machines, plus le réseau est performant

Exemple de réseau P2P : le partage de fichiers

Une des utilisations des réseaux P2P est le partage de fichiers (téléchargement de films ou musiques).



Les fichiers sont **découpés en morceaux** et les morceaux sont envoyés/récupérés dans un **ordre quelconque** par les machines.

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la **sécurité** :

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la **sécurité** :

- ▶ **Intégrité** de l'information

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la **sécurité** :

- ▶ **Intégrité** de l'information
- ▶ **Confidentialité**, protection des données

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la **sécurité** :

- ▶ **Intégrité** de l'information
- ▶ **Confidentialité**, protection des données
- ▶ **Authentification** des échanges

Inconvénients des réseaux P2P

Les réseaux P2P souffrent principalement de problèmes liés à la **sécurité** :

- ▶ **Intégrité** de l'information
- ▶ **Confidentialité**, protection des données
- ▶ **Authentification** des échanges
- ▶ Etc.

- ▶ **Découverte** des autres pairs, gestion des déconnexions, pannes, etc.
- ▶ **Localisation** des données sur le réseau
- ▶ **Routage** des messages
- ▶ **Récupération** sur faute, panne

Cela nécessite de mettre en place de **protocoles spécifiques** entre les pairs.

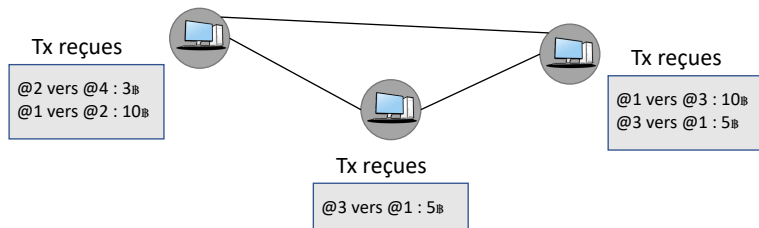
Voir par exemple :

G. Feltin, G. Doyen, O. Festor. **Les protocoles peer-to-peer, leur utilisation et leur détection.**

<https://hal.inria.fr/inria-00099498/document>

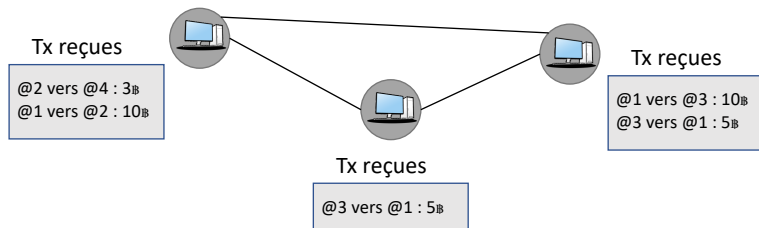
Cohérence dans un réseau P2P

Un réseau P2P ne peut garantir que les nœuds reçoivent les transactions **en même temps** et dans le **même ordre**



Cohérence dans un réseau P2P

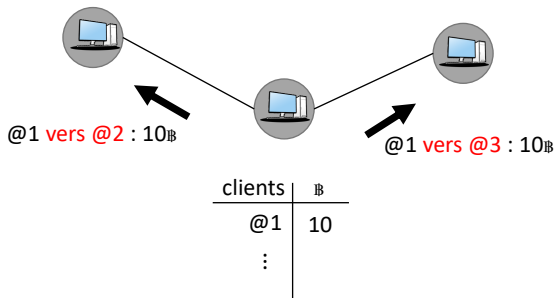
Un réseau P2P ne peut garantir que les nœuds reçoivent les transactions **en même temps** et dans le **même ordre**



Comment faire dans ce cas pour que tous les nœuds aient la même version (vision) du livre de comptes ?

Le problème de la double dépense

Dans le cas de la Blockchain, l'incohérence du réseau P2P peut permettre à un client de dépenser plus qu'il ne possède.



Par exemple, il peut émettre (au moins) **deux transactions différentes** pour dépenser l'intégralité de son compte et les envoyer à des nœuds différents du réseau.

Pour éviter le problème de la double dépense, et plus généralement pour que tous les nœuds d'un réseau aient la même vision du registre, il faut mettre en place un **consensus distribué**.

La blockchain met en œuvre un **protocole de consensus** afin que tous les nœuds du réseau s'accordent sur la liste des transactions à valider (i.e. à mettre) dans le prochain bloc de la chaîne.

Le problème du consensus distribué

Soit un réseau constitué de N clients. Chaque client i choisit (au plus) une valeur v_i (au moins un client choisit une valeur).

Le problème du consensus distribué consiste à faire exécuter par les clients un même **protocole**, de sorte qu'ils communiquent entre eux en s'échangeant des messages et qu'à la fin du protocole ils s'accordent sur une même valeur v (qui doit être l'une des valeurs v_i).

Le protocole de consensus doit fonctionner même si certains clients **tombent en panne** de manière

- ▶ **permanente** : défaillance des machines ou des liens de communication
- ▶ **temporaire** : des messages peuvent être perdus, les clients peuvent être beaucoup trop lents à communiquer, etc.
- ▶ **byzantine** : les clients ne respectent pas les règles du protocole (volontairement ou non).

⇒ Les clients qui ne sont pas en panne doivent **obligatoirement** terminer.

Primitives cryptographiques

L'algorithme de consensus d'une Blockchain, mais également l'intégrité de la chaîne de blocs, reposent principalement sur l'utilisation de deux primitives cryptographiques :

- ▶ **SHA-2** : famille d'algorithmes de hachage sécurisés (ex. SHA-256 ou SHA-512)
- ▶ **DSA** : algorithme de signature numérique

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

- `sha256(m)` doit être calculé **très rapidement**

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

- ▶ `sha256(m)` doit être calculé **très rapidement**
- ▶ `sha256` est à **sens unique**, c'est-à-dire qu'il est très difficile de calculer m à partir de r

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

- ▶ `sha256(m)` doit être calculé **très rapidement**
- ▶ `sha256` est à **sens unique**, c'est-à-dire qu'il est très difficile de calculer m à partir de r
- ▶ le risque de collisions, c'est-à-dire de messages différents m_1 et m_2 tels que `sha256(m1) = sha256(m2)`, doit être extrêmement faible

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

- ▶ `sha256(m)` doit être calculé **très rapidement**
- ▶ `sha256` est à **sens unique**, c'est-à-dire qu'il est très difficile de calculer m à partir de r
- ▶ le risque de collisions, c'est-à-dire de messages différents m_1 et m_2 tels que `sha256(m1) = sha256(m2)`, doit être extrêmement faible
- ▶ l'image r d'un message m doit être très différente de l'image r' d'une perturbation très minime de m .

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

- ▶ `sha256(m)` doit être calculé **très rapidement**
- ▶ `sha256` est à **sens unique**, c'est-à-dire qu'il est très difficile de calculer m à partir de r
- ▶ le risque de collisions, c'est-à-dire de messages différents m_1 et m_2 tels que `sha256(m1) = sha256(m2)`, doit être extrêmement faible
- ▶ l'image r d'un message m doit être très différente de l'image r' d'une perturbation très minime de m .

SHA-256 produit un haché de **256 bits** avec un niveau de sécurité d'une collision pour 2^{128} opérations.

SHA-256

Il s'agit d'une fonction de hachage `sha256` qui transforme des messages m (ou valeurs) de longueur finie (mais quelconque) en des messages condensés r de **longueur fixe**.

- ▶ `sha256(m)` doit être calculé **très rapidement**
- ▶ `sha256` est à **sens unique**, c'est-à-dire qu'il est très difficile de calculer m à partir de r
- ▶ le risque de collisions, c'est-à-dire de messages différents m_1 et m_2 tels que $\text{sha256}(m_1) = \text{sha256}(m_2)$, doit être extrêmement faible
- ▶ l'image r d'un message m doit être très différente de l'image r' d'une perturbation très minime de m .

SHA-256 produit un haché de **256 bits** avec un niveau de sécurité d'une collision pour 2^{128} opérations.

C'est une version améliorée de SHA-1 (et MD5) qui résiste (pour le moment) aux attaques permettant de créer des collisions sans utiliser un algorithme par force brute (attaque des anniversaires).

DSA

Digital Signature Algorithm (DSA) est un algorithme de **signature numérique** basé sur la **cryptographie asymétrique** qui utilise une paire (pub, priv) de **clé publique** et **clé privée**.

La clé publique peut être donnée à tout le monde, mais la clé privée doit être gardée secrète.

Il s'agit d'une sorte de fonction de hachage qui hache un message m en utilisant la clé privée et qui permet de retrouver le message de départ en utilisant la clé publique (ou inversement).

La fonction de hachage **sign(n , priv)** calcule la signature d'un message m selon la clé privée $priv$.

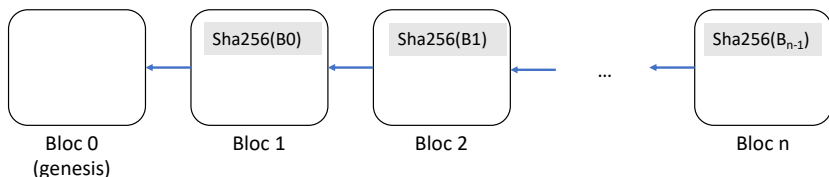
La fonction de hachage inverse **priv(s , pub)** permet de retrouver le message m à partir de la signature s et de la clé publique pub .

$$\text{verify}(\text{sign}(m, \text{priv}), \text{pub}) = m$$

Chaîne de blocs sécurisée (1/2)

On utilise la fonction de hachage SHA-256 pour garantir l'intégrité de la chaîne de blocs.

Pour cela, un bloc n contient toujours l'empreinte numérique du bloc $n - 1$.



De cette manière, il est impossible d'ajouter ou supprimer un bloc sans que cela ne se voit.

Chaîne de blocs sécurisée (1/2)

Est-ce que l'ajout d'une empreinte numérique suffit à sécuriser la chaîne de blocs ?

Chaîne de blocs sécurisée (1/2)

Est-ce que l'ajout d'une empreinte numérique suffit à sécuriser la chaîne de blocs ?

Une manière de modifier ou supprimer un bloc i est de **recalculer** l'empreinte numérique dans tous les blocs $k > i$.

Chaîne de blocs sécurisée (1/2)

Est-ce que l'ajout d'une empreinte numérique suffit à sécuriser la chaîne de blocs ?

Une manière de modifier ou supprimer un bloc i est de **recalculer** l'empreinte numérique dans tous les blocs $k > i$.

Comment peut-on **empêcher** ce calcul ?

Preuve de travail

Pour résoudre ce problème, certaines blockchains comme Bitcoin imposent une **forme particulière** à l'empreinte d'un bloc, par exemple qu'elle commence par n chiffres 0, où n est un entier choisi en fonction de la difficulté voulue pour calculer les empreintes :

$$\text{sha256}(b) = \underbrace{000\dots000}_n xxxx$$

Preuve de travail

Pour résoudre ce problème, certaines blockchains comme Bitcoin imposent une **forme particulière** à l'empreinte d'un bloc, par exemple qu'elle commence par n chiffres 0, où n est un entier choisi en fonction de la difficulté voulue pour calculer les empreintes :

$$\text{sha256}(b) = \underbrace{000\dots000}_n xxx$$

Pour résoudre ce problème, seul un nombre entier (appelé **nonce**) peut être ajouté au bloc.

Il n'existe pas de méthode connue autre que la **force brute** pour résoudre ce problème. Il faut donc dépenser beaucoup d'énergie (donc d'argent) si on souhaite tricher.

Cette technique de sécurisation de la blockchain est appelée **preuve de travail** (Proof-of-Work, POW)

Consensus par preuve de travail (1/3)

L'ajout d'un bloc (donc des transactions) dans une blockchain imposant la preuve de travail nécessite que des machines se chargent d'effectuer le (lourd) calcul de l'empreinte du bloc.

Ces machines sont appelées des mineurs.

Consensus par preuve de travail (1/3)

L'ajout d'un bloc (donc des transactions) dans une blockchain imposant la preuve de travail nécessite que des machines se chargent d'effectuer le (lourd) calcul de l'empreinte du bloc.

Ces machines sont appelées des **mineurs**.

Si plusieurs machines décident de participer à ce calcul, **la première qui trouve l'empreinte gagne**.

Elle propage alors le bloc qu'elle vient de miner à toutes les autres machines (mineur ou autres), qui peuvent alors vérifier (facilement) qu'elle a en effet trouvé la bonne empreinte.

Toutes les machines peuvent alors **ajouter** ce bloc à leur copie de la blockchain.

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “**même moment**”.

Comment **choisir** quel bloc ajouter à la blockchain ?

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Solution :

Supposons qu'une machine ait une copie de la blockchain avec n blocs et qu'elle reçoive un bloc dont l'indice est k .

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “même moment”.

Comment choisir quel bloc ajouter à la blockchain ?

Solution :

Supposons qu'une machine ait une copie de la blockchain avec n blocs et qu'elle reçoive un bloc dont l'indice est k .

- Si $k = n + 1$, alors elle ajoute ce bloc à sa (copie de la) blockchain, s'il est valide et que les transactions qu'il contient le sont également.

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “**même moment**”.

Comment **choisir** quel bloc ajouter à la blockchain ?

Solution :

Supposons qu'une machine ait une copie de la blockchain avec n blocs et qu'elle reçoive un bloc dont l'indice est k .

- ▶ Si $k = n + 1$, alors elle ajoute ce bloc à sa (copie de la) blockchain, s'il est valide et que les transactions qu'il contient le sont également.
- ▶ Si $k \leq n$, alors elle l'ignore

Consensus par preuve de travail (2/3)

Malgré la preuve de travail, il est possible que deux machines (ou plus) réussissent à miner un bloc au “**même moment**”.

Comment **choisir** quel bloc ajouter à la blockchain ?

Solution :

Supposons qu'une machine ait une copie de la blockchain avec n blocs et qu'elle reçoive un bloc dont l'indice est k .

- ▶ Si $k = n + 1$, alors elle ajoute ce bloc à sa (copie de la) blockchain, s'il est valide et que les transactions qu'il contient le sont également.
- ▶ Si $k \leq n$, alors elle l'ignore
- ▶ Si $k \geq n$, alors elle demande à la machine ayant miné ce bloc sa copie de la blockchain afin de remplacer la sienne.

De cette manière, chaque machine cherche toujours à avoir **la plus grande blockchain**, car elle représente une preuve d'un travail plus importante, donc un plus grande "richesse".

Quel intérêt une machine d'une blockchain a-t-elle à miner des blocs, puisque cela coûte très cher ?

Quel intérêt une machine d'une blockchain a-t-elle à miner des blocs, puisque cela coûte très cher ?

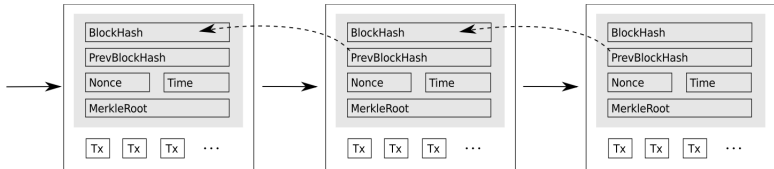
Réponse : Miner rapporte de l'argent.

BITCOIN

Bitcoin : chaines de blocs

(source :

F. Tschorsch et B. Scheuermann. *Bitcoin and Beyond : A Technical Survey on Decentralized Digital Currencies*)



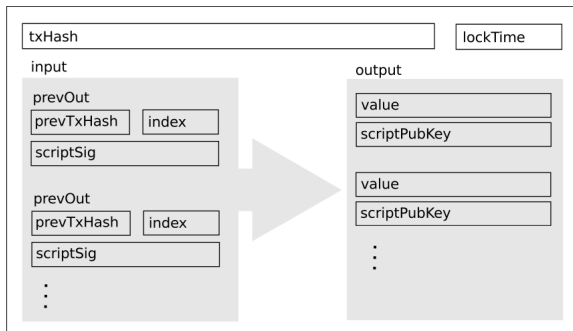
On génère une **adresse Bitcoin** B à partir d'une clé publique P en hachant tout d'abord H avec la fonction **SHA-256**, puis le résultat est de nouveau haché avec la fonction **RIPEMD-160**. Enfin, les adresses sont représentée en base 58, une transformation binaire vers ASCII qui utilise 58 caractères (sans caractères ambigus comme 0, O, l, 1, etc.)

Cet encodage des clés publiques permet à la fois de **gagner en espace** et d'**obfusquer les clés**.

Bitcoin : transactions

Dans Bitcoin, l'argent que possède un utilisateur est représenté par l'ensemble des transactions qu'il a reçues des autres utilisateurs (ou celui qu'il a gagné par minage ou frais de transactions) et qu'il n'a pas encore utilisé.

Pour dépenser de l'argent, il faut donc “consommer” des transactions non dépensées (UTXO : *Unspent Transaction Output*) et les relier à une **nouvelle** UTXO (qui peut avoir plusieurs sorties).



La machine Script de Bitcoin

Chaque transaction Bitcoin correspond à un **petit programme** écrit dans un langage appelé **Script**.

Ce langage permet de commander une **machine à pile** à l'aide d'un ensemble d'instructions (plus d'une centaine).

`https://en.bitcoin.it/wiki/Script`

Exemples d'instructions :

DUP

EQUAL

HASH160

EQUALVERIFY

CHECKSIG

CHECKMULTISIG

IF, ELSE, ENDIF

...

Bitcoin : scripts (1/2)

Il y a deux types de script Bitcoin : les scripts de “verrouillage” (Lock ou scriptPubKey) et ceux de “déverrouillage” (Unlock ou scriptSig). Les transactions Bitcoin utilisent essentiellement 4 familles de scripts.

P2PK

Lock : `<pk>CHECKSIG`

Unlock : `<sig>`

P2PKH :

Lock : `DUP HASH160 <Hpk> EQUALVERIFY CHECKSIG`

Unlock : `<sig> <pk>`

P2MS :

Lock : `<n> <pkh>...<pkh> <m> CHECKMULTISIG`

Unlock : `<sig>...<sig>`

P2SH :

Lock : `HASH160 <hashscript> EQUAL`

Unlock : `<sig>...<sig><serialized_script>`

Bitcoin : scripts (1/2)

Stack	Script
	sigBob pubKeyBob OP_DUP OP_HASH160 pubKeyBobHash OP_EQUALVERIFY OP_CHECKSIG
sigBob pubKeyBob	OP_DUP OP_HASH160 pubKeyBobHash OP_EQUALVERIFY OP_CHECKSIG
sigBob pubKeyBob pubKeyBob	OP_HASH160 pubKeyBobHash OP_EQUALVERIFY OP_CHECKSIG
sigBob pubKeyBob pubKeyBobHash	pubKeyBobHash OP_EQUALVERIFY OP_CHECKSIG
sigBob pubKeyBob pubKeyBobHash pubKeyBobHash	OP_EQUALVERIFY OP_CHECKSIG
sigBob pubKeyBob	OP_CHECKSIG
true	

`\begin{Practice}`

Bitcoin scripts

Exercice

En utilisant un explorateur de blocs pour Bitcoin, étudier les scripts de quelques transactions.

```
\end{Practice}
```

Client légers

Une blockchain est constituée de deux types de nœuds : les **miners** et les **wallets**.

Les wallets (ou portefeuilles) sont des **clients légers** qui ne fonctionnent pas comme un nœud du réseau P2P : **ils ne cherchent pas** à miner des blocs ni à télécharger l'ensemble de la blockchain.

Ils **suivent** simplement les transactions qu'ils effectuent en **interrogeant** un serveur de la blockchain.

La technologie sous-jacente à ces wallets est la vérification Simplifiée des Paiements (SPV) qui permet de vérifier qu'une transaction est bien dans une blockchain, sans pour autant télécharger toute la blockchain.

Les clients légers d'une blockchain comme Bitcoin s'appellent des **portefeuilles** (*wallet*, en anglais).

Contrairement aux nœuds du réseau P2P, les *wallets* sont utilisés par les clients de la blockchain pour **émettre des transactions et réaliser leur suivi**.

Pour effectuer ces opérations, un *wallet* se connecte à un nœud du réseau pour obtenir les **informations minimales** nécessaires pour vérifier qu'une transaction est validée, sans jamais télécharger la blockchain **complète**.

Cette fonctionnalité s'appelle la **vérification de paiement simple**.

Comment vérifier qu'une transaction est dans un bloc i , sans télécharger les transactions contenues dans ce bloc, mais **uniquement l'en-tête du bloc** ?

Liste d'empreintes

Une solution consiste à calculer l'**empreinte numérique** de la liste des transactions du bloc et la stocker dans l'en-tête du bloc.

Liste d'empreintes

Une solution consiste à calculer l'**empreinte numérique** de la liste des transactions du bloc et la stocker dans l'en-tête du bloc.

Étant donnée la liste $[H_0; H_1; \dots; H_k]$ des empreintes numériques des transactions, on peut par exemple calculer l'empreinte de la liste comme l'empreinte de la somme $H_0 + \dots + H_k$.

Liste d'empreintes

Une solution consiste à calculer l'**empreinte numérique** de la liste des transactions du bloc et la stocker dans l'en-tête du bloc.

Étant donnée la liste $[H_0; H_1; \dots; H_k]$ des empreintes numériques des transactions, on peut par exemple calculer l'empreinte de la liste comme l'empreinte de la somme $H_0 + \dots + H_k$.

Ainsi, pour vérifier qu'une transaction particulière v est dans un bloc b_i , il suffit de :

Liste d'empreintes

Une solution consiste à calculer l'**empreinte numérique** de la liste des transactions du bloc et la stocker dans l'en-tête du bloc.

Étant donnée la liste $[H_0; H_1; \dots; H_k]$ des empreintes numériques des transactions, on peut par exemple calculer l'empreinte de la liste comme l'empreinte de la somme $H_0 + \dots + H_k$.

Ainsi, pour vérifier qu'une transaction particulière v est dans un bloc b_i , il suffit de :

- demander (comme preuve) au nœud du réseau la **liste ℓ des empreintes** de toutes les transactions contenues dans b_i

Liste d'empreintes

Une solution consiste à calculer l'**empreinte numérique** de la liste des transactions du bloc et la stocker dans l'en-tête du bloc.

Étant donnée la liste $[H_0; H_1; \dots; H_k]$ des empreintes numériques des transactions, on peut par exemple calculer l'empreinte de la liste comme l'empreinte de la somme $H_0 + \dots + H_k$.

Ainsi, pour vérifier qu'une transaction particulière v est dans un bloc b_i , il suffit de :

- ▶ demander (comme preuve) au nœud du réseau la **liste ℓ des empreintes** de toutes les transactions contenues dans b_i
- ▶ **vérifier** que l'empreinte numérique de v est dans ℓ

Liste d'empreintes

Une solution consiste à calculer l'**empreinte numérique** de la liste des transactions du bloc et la stocker dans l'en-tête du bloc.

Étant donnée la liste $[H_0; H_1; \dots; H_k]$ des empreintes numériques des transactions, on peut par exemple calculer l'empreinte de la liste comme l'empreinte de la somme $H_0 + \dots + H_k$.

Ainsi, pour vérifier qu'une transaction particulière v est dans un bloc b_i , il suffit de :

- ▶ demander (comme preuve) au nœud du réseau la **liste ℓ des empreintes** de toutes les transactions contenues dans b_i
- ▶ **vérifier** que l'empreinte numérique de v est dans ℓ
- ▶ **re-calculer** l'empreinte numérique de ℓ et **vérifier** qu'elle correspond à l'empreinte des transactions stockée dans l'en-tête de b_i

Arbres de Merkle

Afin de minimiser la taille de ℓ , on va organiser la liste des (empreinte des) transactions comme un arbre.

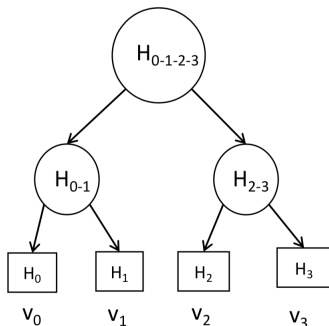
Cette structure en arbre s'appelle un **arbre de Merkle**. Elle a été inventée en 1979 par Ralk Merkle. D'abord déposée sous la forme d'un brevet, Merkle l'a ensuite publiée en 1987 dans l'article suivant :

R. Merkle. [A digital signature based on a conventional encryption function](https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf). <https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf>

Arbre de Merkle : principe

Il s'agit d'une structure d'**arbre binaire** dans laquelle :

- ▶ chaque **feuille** contient l'**empreinte d'une transaction**
- ▶ chaque **nœud** contient l'empreinte numérique de la **somme des empreintes** de ses deux fils.



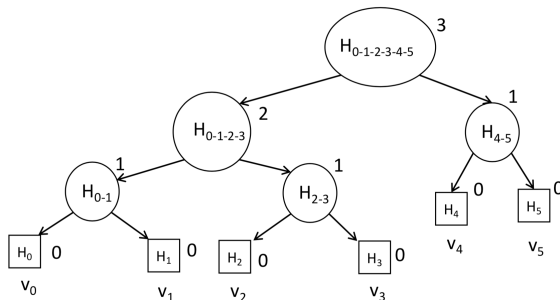
- ▶ Les feuilles $H_0, \dots, H_3$ contiennent respectivement les empreintes des transactions $v_0, \dots, v_3$
- ▶ Les nœuds internes H_{i-j} sont égaux à $\text{hash}(H_i + H_j)$
- ▶ Enfin, la racine $H_{0-1-2-3}$ est égale à $\text{hash}(H_{0-1} + H_{2-3})$

Arbres de Merkle : propriété

Les arbres de Merkle ont la propriété que les **sous-arbres gauches** sont toujours **complets**, c-à-d que tous les niveaux de ces arbres sont **remplis**.

Ainsi, un nœud de niveau n aura nécessairement 2^{n-1} feuilles dans son arbre gauche.

Exemple :



C'est **uniquement** l'empreinte stockée à la **racine** de l'arbre de Merkle qui est stockée dans l'en-tête des blocs d'une blockchain.

Seuls les nœuds du réseau P2P de la blockchain contiennent les transactions qui sont stockées, dans chaque bloc, sous la forme d'arbres de Merkle

Preuves de Merkle : principe

Pour vérifier qu'une transaction v est associée à un bloc (dont on a que l'en-tête), il suffit de renvoyer la **liste des empreintes** contenues dans les nœuds frères des nœuds parcourus depuis la racine de l'arbre jusqu'à la position de la feuille contenant l'empreinte de v .

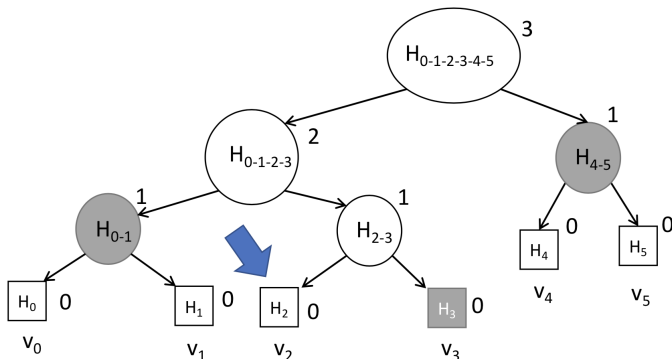
Cette liste, appelée **preuve de Merkle**, permet alors de refaire le calcul complet de la racine de l'arbre et vérifier qu'il est égal à la valeur stockée dans le bloc.

Preuves de Merkle : exemple

La preuve de Merkle pour la transaction v_2 de l'arbre ci-dessous est la liste $[H_{4-5}; H_{0-1}; H_3]$.

En effet, on a

$$H_{0-1-2-3-4-5} = \text{hash}(H_{4-5} + \text{hash}(H_{0-1} + \text{hash}(\text{hash}(v_2) + H_3)))$$



\begin{Practice}

Merkle trees

Exercice

En utilisant l'API de blockstream :

`https://github.com/Blockstream/esplora/blob/master/API.md`

et un outil comme `curl` ou une bibliothèque comme `requests` de Python :

Écrire un programme qui récupère la racine de l'arbre de Merkle d'un bloc et la preuve de Merkle de la présence d'une transaction.

Vérifier que la preuve de Merkle est correcte.

```
\end{Practice}
```

2-LAYERS

Passage à l'échelle de Bitcoin

Acutellement, le principal problème d'une blockchain comme Bitcoin est le **passage à l'échelle**

Ceci est généralement visible en mesurant le **nombre de transactions par seconde (TPS)** qu'une blockchain peut supporter.

Passage à l'échelle de Bitcoin

Actuellement, le principal problème d'une blockchain comme Bitcoin est le **passage à l'échelle**

Ceci est généralement visible en mesurant le **nombre de transactions par seconde (TPS)** qu'une blockchain peut supporter.

Bitcoin génère **1 bloc** toutes les **10 minutes** et chaque bloc peut contenir 1 Mega Octet (= 1048576 octets) de données. Par ailleurs, une transaction fait 380 octets en moyenne.

$$T_B(\text{Time Block}) = 600$$

$$B(\text{Block size}) = 1048576$$

$$A(\text{Average transaction size}) = 380$$

Passage à l'échelle de Bitcoin

Acutellement, le principal problème d'une blockchain comme Bitcoin est le **passage à l'échelle**

Ceci est généralement visible en mesurant le **nombre de transactions par seconde (TPS)** qu'une blockchain peut supporter.

Bitcoin génère **1 bloc** toutes les **10 minutes** et chaque bloc peut contenir 1 Mega Octet (= 1048576 octets) de données. Par ailleurs, une transaction fait 380 octets en moyenne.

$$T_B(\text{Time Block}) = 600$$

$$B(\text{Block size}) = 1048576$$

$$A(\text{Average transaction size}) = 380$$

Le nombre de transactions par bloc (TPB) et la valeur TPS est :

$$TPB = B/A = 1048576/380 \approx 2759$$

$$TPS = TPB/T_B \approx 2759/600 \approx 4.6$$

Comment augmenter la valeur TPS ?

Bitcoin permet donc **4.6 transactions par seconde** en moyenne. En comparaison, le réseau **Visa** permet des pics à **50000 TPS** et des centaines de millions par jour.

Pour obtenir un tel débit, un bloc de 11 Go devrait être miné sur Bitcoin toutes les 10 minutes, soit environ **600 petaoctets** par année !

Par ailleurs, en 2021, Bitcoin compte environ **10000 noeuds** et le temps moyen pour propager un bloc à ces noeuds est estimé à **14s**.

Comment augmenter la valeur TPS ?

Bitcoin permet donc **4.6 transactions par seconde** en moyenne. En comparaison, le réseau **Visa** permet des pics à **50000 TPS** et des centaines de millions par jour.

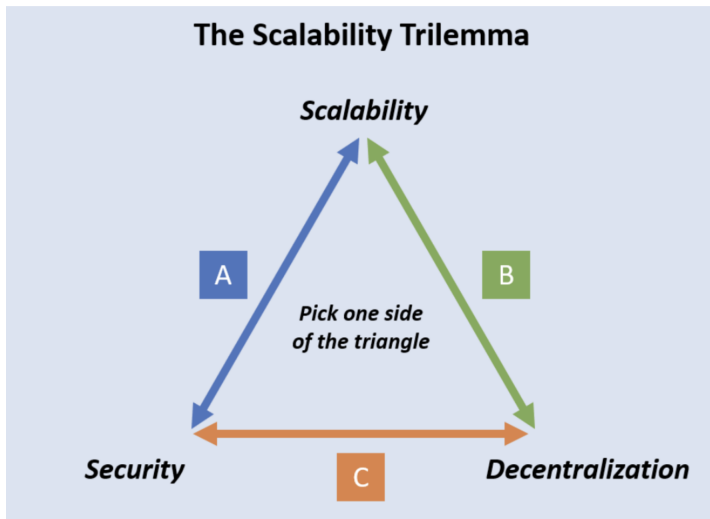
Pour obtenir un tel débit, un bloc de 11 Go devrait être miné sur Bitcoin toutes les 10 minutes, soit environ **600 petaoctets** par année !

Par ailleurs, en 2021, Bitcoin compte environ **10000 noeuds** et le temps moyen pour propager un bloc à ces noeuds est estimé à **14s**.

Augmenter la taille des blocs n'est pas une bonne idée :

- ▶ Seuls quelques noeuds pourraient supporter de stocker une telle quantité de données
- ▶ **Peu de miners** = **peu de sécurité**
- ▶ La **validation** de la blockchain serait **impossible**

Le trilemme de la Blockchain



Il est difficile pour une blockchain d'avoir les trois propriétés :
passage à l'échelle, **décentralisation**, **sécurité**.

Lightning Network

Une solution pour augmenter le passage à l'échelle de Bitcoin est de construire une couche secondaire, **par dessus Bitcoin**, dans laquelle les utilisateurs vont pouvoir réaliser des **micro-paiements** (presque) **instantanés** et avec **peu de frais** à travers des communications **hors chaîne**.

La blockchain Bitcoin va être utilisée pour **sécuriser** ces paiements,

La principale couche secondaire pour Bitcoin est le **Lightning Network**

Il s'agit d'un réseau de **canaux de micro-paiements**.

Canaux de micro-paiements

Un canal de micro-paiement permet à deux “agents” de faire des paiements très rapidement, sans passer par la blockchain, sauf à l'ouverture et fermeture du canal.

Canaux de micro-paiements

Un canal de micro-paiement permet à deux “agents” de faire des paiements très rapidement, sans passer par la blockchain, sauf à l'ouverture et fermeture du canal.

Supposons qu'Alice et Bob souhaitent s'échanger régulièrement de l'argent (dans les deux sens).

Canaux de micro-paiements

Un canal de micro-paiement permet à deux “agents” de faire des paiements très rapidement, sans passer par la blockchain, sauf à l’ouverture et fermeture du canal.

Supposons qu’Alice et Bob souhaitent s’échanger régulièrement de l’argent (dans les deux sens).

Pour cela, Alice et Bob souhaite commencer par ouvrir un compte commun (multisig) sur Bitcoin, en y déposant chacun une certaine somme, par exemple, Alice souhaite ouvrir le compte en déposant 1 BTC et Bob 0.5 BTC.

Comment faire pour que l’ouverture d’un tel compte se fasse en toute sécurité ?

Ouverture d'un canal

L'ouverture de ce compte (AB) nécessite deux transactions :

1. $A \xrightarrow{1} B$: 1 BTC de Alice vers AB (signée par Alice)
2. $B \xrightarrow{0.5} A$: 0.5 BTC de Bob vers AB (signée par Bob)

Ouverture d'un canal

L'ouverture de ce compte (AB) nécessite deux transactions :

1. $A \xrightarrow{1} B$: 1 BTC de Alice vers AB (signée par Alice)
2. $B \xrightarrow{0.5} A$: 0.5 BTC de Bob vers AB (signée par Bob)

Mais cela n'est pas suffisant car si Bob ne joue pas le jeu, alors Alice vient de perdre 1 BTC dans ce compte commun (et réciproquement) car elle ne pourra plus récupérer son argent.

Ouverture d'un canal

L'ouverture de ce compte (AB) nécessite deux transactions :

1. $A \xrightarrow{1} B$: 1 BTC de Alice vers AB (signée par Alice)
2. $B \xrightarrow{0.5} A$: 0.5 BTC de Bob vers AB (signée par Bob)

Mais cela n'est pas suffisant car si Bob ne joue pas le jeu, alors Alice vient de perdre 1 BTC dans ce compte commun (et réciproquement) car elle ne pourra plus récupérer son argent.

Pour cela, avant d'envoyer leurs transactions, Alice et Bob construisent et signent chacun la transaction à double sortie suivante qui représente l'état du canal de paiement.

$$AB \begin{array}{l} \xrightarrow{1} A \\ \xrightarrow{0.5} B \end{array}$$

Alice et Bob gardent cette transaction (d'engagement) et ils envoient seulement les deux premières sur Bitcoin. Cela permet d'ouvrir le canal de paiement en toute sécurité

Micro-paiements

Supposons qu'Alice souhaite donner **0.1 BTC** à Bob en utilisant leur compte commun. À l'issue du paiement, Alice aura **0.9 BTC** sur ce compte et Bob aura **0.6 BTC**.

Plutôt que de passer par Bitcoin pour réaliser ce transfère, Alice et Bob peuvent *simplement* “jeter” la transaction d'engagement représentant l'état du canal et la remplacer par une nouvelle transaction :

$$AB \begin{array}{l} \xrightarrow{0.9} A \\ \xrightarrow{0.6} B \end{array}$$

Comme la précédente, cette transaction n'a pas à être envoyée sur Bitcoin, simplement conservée précieusement par Alice et Bob.

Le simple fait qu'Alice et Bob soient **en possession** de cette transaction permet d'**acter le transfère**.

Timelock et Clé de révocation

Il reste un problème important :

Comment garantir que l'engagement $AB \xrightarrow{1} A$
 $\xrightarrow{0.5} B$ ne soit plus utilisée ?

Timelock et Clé de révocation

Il reste un problème important :

Comment garantir que l'engagement $AB \begin{matrix} \xrightarrow{1} A \\ \xrightarrow{0.5} B \end{matrix}$ ne soit plus utilisée ?

Pour cela, on va définir des scripts qui permettent de verrouiller temporairement et avec des clés de révocation les transactions sur ces canaux de paiement.

Timelock et Clé de révocation

Il reste un problème important :

Comment garantir que l'engagement $AB \xrightarrow{1} A$
 $\xrightarrow{0.5} B$ ne soit plus utilisée ?

Pour cela, on va définir des scripts qui permettent de verrouiller temporairement et avec des clés de révocation les transactions sur ces canaux de paiement.

Un script **timelock** bloque l'application d'une (sortie de) transaction pendant un nombre de blocs donné.

Un script avec **clé de révocation** bloque l'application d'une (sortie de) transaction si le signataire de la transaction ne connaît pas la clé de révocation.

Transactions d'engagement sécurisées

Les engagements construits par Alice et Bob sont définis sous la forme de contrats avec Timelock et clé de révocation. Celui d'Alice a la forme suivante :

$$\begin{array}{l} \xrightarrow{1} A \quad \text{timelock } N \\ AB \xrightarrow{1} B \quad \text{clé de révocation} = (HS_A, HS_B) \\ \xrightarrow{0.5} B \end{array}$$

où HS_A (resp. HS_B) est la valeur de hash d'un secret détenu seulement par A (resp. B). Si cette transaction est envoyée sur Bitcoin, alors

Transactions d'engagement sécurisées

Les engagements construits par Alice et Bob sont définis sous la forme de contrats avec Timelock et clé de révocation. Celui d'Alice a la forme suivante :

$$\begin{array}{ll} \xrightarrow{1} A & \text{timelock } N \\ AB \xrightarrow{1} B & \text{clé de révocation} = (HS_A, HS_B) \\ \xrightarrow{0.5} B & \end{array}$$

où HS_A (resp. HS_B) est la valeur de hash d'un secret détenu seulement par A (resp. B). Si cette transaction est envoyée sur Bitcoin, alors

- ▶ L'effet de la sortie $AB \xrightarrow{1} A$ est **décalée de N blocs**. Ainsi A ne pourra récupérer son argent que dans N blocs.
- ▶ L'effet de la sortie $AB \xrightarrow{1} B$ est possible uniquement si le signataire peut fournir deux secrets S_A et S_B tels que $hash(S_A) = HS_A$ et $hash(S_B) = HS_B$. Ainsi, B peut être crédité de 1 BTC seulement s'il **connait le secret de A**.
- ▶ B peut **immédiatement** avoir 0.5 BTC

Avant de créer un nouvel engagement, Alice et Bob créent chacun un nouveau secret S'_A et S'_B .

Protocole sur un canal

Avant de créer un nouvel engagement, Alice et Bob créent chacun un nouveau secret S'_A et S'_B .

Ils **s'échangent** également les valeurs de hash HS'_A et HS'_B de ces secrets, ainsi que les secrets S_A et S_B de l'engagement précédent.

Avant de créer un nouvel engagement, Alice et Bob créent chacun un nouveau secret S'_A et S'_B .

Ils **s'échangent** également les valeurs de hash HS'_A et HS'_B de ces secrets, ainsi que les secrets S_A et S_B de l'engagement précédent.

De cette manière, si A veut tricher :

- ▶ B va **immédiatement** toucher l'argent de l'état précédent ;
- ▶ B a le temps de vider le reste du solde du compte commun car il connaît le secret de A, et que ce dernier est **retardé** par le timelock pour “voler” l'argent de l'état précédent.

Fermeture d'un canal

Pour fermer un canal d'une **manière coopérative**, Alice et Bob doivent construire une dernière transaction, la **transaction de fermeture**, de la forme suivante :

$$AB \begin{array}{l} \xrightarrow{x} A \\ \xrightarrow{y} B \end{array}$$

où x_A et x_B représentent les soldes de Alice et Bob sur le compte commun, respectivement.

Alice et Bob doivent également se mettre d'accord sur les **frais de transaction** qu'ils sont prêts à donner pour récupérer leur argent dans un **délai raisonnable** (plus les frais sont élevés, plus une transaction a de chances de passer).

Cela permet une fermeture **rapide** et à moindre **frais** (contrairement à une fermeture forcée en utilisant la dernière transaction d'engagement).

LN : Un réseau de canaux

Il est **impossible** pour un utilisateur d'ouvrir un canal avec chaque autre utilisateur (trop coûteux), cela serait trop coûteux et le nombre de canaux serait bien trop grand.

Comment Alice et Bob peuvent-ils alors s'échanger de l'argent s'ils n'ont pas un canal commun ?

LN : Un réseau de canaux

Il est **impossible** pour un utilisateur d'ouvrir un canal avec chaque autre utilisateur (trop coûteux), cela serait trop coûteux et le nombre de canaux serait bien trop grand.

Comment Alice et Bob peuvent-ils alors s'échanger de l'argent s'ils n'ont pas un canal commun ?

Le Lightning Network est un **graphe de canaux de paiement**, chaque canal reliant deux utilisateurs.

Les **noeuds** de ce graphe représentent les utilisateurs, les **arêtes** sont les canaux.

Un utilisateur peut avoir ouvert des canaux avec plusieurs utilisateurs, par conséquent un noeud peut avoir plusieurs voisins.

LN : Exemple

Pour comprendre le fonctionnement de ce graphe, supposons qu'Alice et Charlie aient un canal en commun, et que Bob et Charlie aient également un canal.

$$A(x_A) \xleftrightarrow{x} (x_C)C(y_C) \xleftrightarrow{y} (y_B)B$$

où x est le nom du canal entre Alice et Charlie, et x_A (resp. x_C) représente l'argent dont Alice (resp. Charlie) dispose sur ce canal.

LN : Paiement

Supposons qu'Alice souhaite envoyer une somme d à Bob. Pour cela, elle doit d'abord **envoyer d à Charlie**, puis ce dernier doit **envoyer d à Bob**. Si cela fonctionne, on se retrouve dans la situation suivante :

$$A(x_A - d) \xleftrightarrow{x} (x_C + d)C(y_C - d) \xleftrightarrow{y} (y_B + d)B$$

LN : Paiement

Supposons qu'Alice souhaite envoyer une somme d à Bob. Pour cela, elle doit d'abord **envoyer d à Charlie**, puis ce dernier doit **envoyer d à Bob**. Si cela fonctionne, on se retrouve dans la situation suivante :

$$A(x_A - d) \xleftarrow{x} (x_C + d) C(y_C - d) \xleftarrow{y} (y_B + d) B$$

On voit que cette opération est **neutre** pour Charlie puisqu'il possède autant d'argent.

Supposons qu'Alice souhaite envoyer une somme d à Bob. Pour cela, elle doit d'abord **envoyer d à Charlie**, puis ce dernier doit **envoyer d à Bob**. Si cela fonctionne, on se retrouve dans la situation suivante :

$$A(x_A - d) \xrightarrow{x} (x_C + d)C(y_C - d) \xrightarrow{y} (y_B + d)B$$

On voit que cette opération est **neutre** pour Charlie puisqu'il possède autant d'argent.

Mais cet échange peut **échouer** si $y_C < d$. Dans ce cas, Alice doit :

- ▶ chercher un **autre chemin** dans le graphe en espérant que tous les noeuds aient suffisamment d'argent pour acheminer son transfère
- ▶ ou transférer la somme en **plusieurs fois**, via **différents chemins**.

Supposons qu'Alice souhaite envoyer une somme d à Bob. Pour cela, elle doit d'abord **envoyer d à Charlie**, puis ce dernier doit **envoyer d à Bob**. Si cela fonctionne, on se retrouve dans la situation suivante :

$$A(x_A - d) \xleftarrow{x} (x_C + d)C(y_C - d) \xleftarrow{y} (y_B + d)B$$

On voit que cette opération est **neutre** pour Charlie puisqu'il possède autant d'argent.

Mais cet échange peut **échouer** si $y_C < d$. Dans ce cas, Alice doit :

- ▶ chercher un **autre chemin** dans le graphe en espérant que tous les noeuds aient suffisamment d'argent pour acheminer son transfère
- ▶ ou transférer la somme en **plusieurs fois**, via **différents chemins**.

Problème : seules les **capacités initiales** des canaux sont connues (il suffit de lire la Blockchain), mais pas les **capacités réelles**.

LN : Frais de transfère

LN possède également des **frais de transfère**. Cela permet de rétribuer les **noeuds intermédiaires** qui font transiter l'argent.

Pour un même canal, les frais peuvent être différents selon le **sens de transfère**.

Dans l'exemple précédent, Charlie peut imposer des frais ϵ pour transférer l'argent à Bob à travers son canal y . Dans ce cas, Alice doit prévoir de transférer $d + \epsilon$ vers Charlie pour payer ces frais.

LN : Frais de transfère

LN possède également des **frais de transfère**. Cela permet de rétribuer les **noeuds intermédiaires** qui font transiter l'argent.

Pour un même canal, les frais peuvent être différents selon le **sens de transfère**.

Dans l'exemple précédent, Charlie peut imposer des frais ϵ pour transférer l'argent à Bob à travers son canal y . Dans ce cas, Alice doit prévoir de transférer $d + \epsilon$ vers Charlie pour payer ces frais.

On obtient alors la situation suivante après le transfère :

$$A(x_A - (d + \epsilon)) \xleftrightarrow{x} (x_C + d + \epsilon)C(y_C - d) \xleftrightarrow{y} (y_B + d)B$$

LN : Choix de la route

Les frais lié à un transfère dépendent de la **route empruntée** (puisque les frais sont différents selon les noeuds)

Contrairement à un réseau classique d'échange de données où chaque noeud (ou routeur) du réseau **choisit** vers quel voisin transférer un paquet, dans LN ce sont les **noeuds de départ** (comme Alice) qui choisissent l'**intégralité** de la route à emprunter, par exemple, en fonction :

- ▶ des frais qu'ils sont prêts à payer pour leur paiement
- ▶ des capacités des canaux à emprunter pour maximiser leur chance de succès
- ▶ etc.

L'application LN doit maintenir à jour la **carte du réseau** pour que chaque utilisateur puisse choisir ses chemins.

Dans le paiement d'Alice vers Bob, que se passe-t-il si Charlie **ne fait pas** le transfère vers Bob ?

Pour sécuriser les transfères, on utilise des scripts de paiement particulier appelés **HTLC** (Hashed Time Locked Contract).

Dans le paiement d'Alice vers Bob, que se passe-t-il si Charlie **ne fait pas** le transfère vers Bob ?

Pour sécuriser les transfères, on utilise des scripts de paiement particulier appelés **HTLC** (Hashed Time Locked Contract).

Les contrats de ces paiements sont :

- ▶ **Conditionnels** : un paiement est **finalisé** uniquement si le destinataire réussit à fournir un certain **secret**
- ▶ **Temporels** : un paiement **expire dans le temps** si le secret n'est pas révélé

Protocole de transfère avec HTLC

1. Pour qu'Alice puisse envoyer d'une manière sécurisée d BTC à Bob via Charlie, Alice commence par demander à Bob de lui envoyer $H_s = \text{hash}(s)$, le hash d'un secret s connu **uniquement de Bob**.

Protocole de transfère avec HTLC

1. Pour qu'Alice puisse envoyer d'une manière sécurisée d BTC à Bob via Charlie, Alice commence par demander à Bob de lui envoyer $H_s = \text{hash}(s)$, le hash d'un secret s connu **uniquement de Bob**.
2. Alice envoie ensuite un contrat **HTLC**(d, H_s, N) à Charlie. Ce contrat ne permet à Charlie de gagner d BTC que s'il peut **livrer** le secret s . Si Charlie ne révèle pas s avant le bloc **N**, les fonds **ne sont pas bougés**.

Protocole de transfère avec HTLC

1. Pour qu'Alice puisse envoyer d'une manière sécurisée d BTC à Bob via Charlie, Alice commence par demander à Bob de lui envoyer $H_s = \text{hash}(s)$, le hash d'un secret s connu **uniquement de Bob**.
2. Alice envoie ensuite un contrat $\text{HTLC}(d, H_s, N)$ à Charlie. Ce contrat ne permet à Charlie de gagner d BTC que s'il peut **livrer** le secret s . Si Charlie ne révèle pas s avant le bloc N , les fonds **ne sont pas bougés**.
3. Charlie envoie $\text{HTLC}(d, H_s, N - i)$ à Bob (avec $1 \leq i$).

Protocole de transfère avec HTLC

1. Pour qu'Alice puisse envoyer d'une manière sécurisée d BTC à Bob via Charlie, Alice commence par demander à Bob de lui envoyer $H_s = \text{hash}(s)$, le hash d'un secret s connu **uniquement de Bob**.
2. Alice envoie ensuite un contrat $\text{HTLC}(d, H_s, N)$ à Charlie. Ce contrat ne permet à Charlie de gagner d BTC que s'il peut **livrer** le secret s . Si Charlie ne révèle pas s avant le bloc N , les fonds **ne sont pas bougés**.
3. Charlie envoie $\text{HTLC}(d, H_s, N - i)$ à Bob (avec $1 \leq i$).
4. Bob **déverrouille** le contrat en montrant qu'il connaît s ; En voyant ce secret, Charlie peut à son tour **déverrouiller** le contrat et le paiement est finalisé.

Protocole de transfère avec HTLC

1. Pour qu'Alice puisse envoyer d'une manière sécurisée d BTC à Bob via Charlie, Alice commence par demander à Bob de lui envoyer $H_s = \text{hash}(s)$, le hash d'un secret s connu **uniquement de Bob**.
2. Alice envoie ensuite un contrat $\text{HTLC}(d, H_s, N)$ à Charlie. Ce contrat ne permet à Charlie de gagner d BTC que s'il peut **livrer** le secret s . Si Charlie ne révèle pas s avant le bloc N , les fonds **ne sont pas bougés**.
3. Charlie envoie $\text{HTLC}(d, H_s, N - i)$ à Bob (avec $1 \leq i$).
4. Si Bob **ne déverrouille pas** le contrat, dans ce cas le bloc $N - i$ finit par être miné et le contract expire. Puisque Charlie ne voit pas le secret s , son HTLC finit également par expirer. Le transfère est **annulé**.

Protocole de transfère avec HTLC

1. Pour qu'Alice puisse envoyer d'une manière sécurisée d BTC à Bob via Charlie, Alice commence par demander à Bob de lui envoyer $H_s = \text{hash}(s)$, le hash d'un secret s connu **uniquement de Bob**.
2. Alice envoie ensuite un contrat $\text{HTLC}(d, H_s, N)$ à Charlie. Ce contrat ne permet à Charlie de gagner d BTC que s'il peut **livrer** le secret s . Si Charlie ne révèle pas s avant le bloc N , les fonds **ne sont pas bougés**.
3. Charlie envoie $\text{HTLC}(d, H_s, N - i)$ à Bob (avec $1 \leq i$).
4. Si Bob **ne déverrouille pas** le contrat, dans ce cas le bloc $N - i$ finit par être miné et le contract expire. Puisque Charlie ne voit pas le secret s , son HTLC finit également par expirer. Le transfère est **annulé**.

Pourquoi le deuxième HTLC a-t-il un temps d'expiration à $N - i$?

HTLC et transaction d'engagement (1/2)

Les HTLC sont représentés par des transactions d'engagement.

Supposons que le canal entre Alice et Charlie soit le suivant :

$$A(1) \longleftrightarrow (2)C$$

et qu'Alice souhaite envoyer HTLC(0.1, H_s , N) à Charlie.

Un HTLC est représenté par les transactions d'engagement :

Alice	Charlie
$\xRightarrow{0.9} A(\text{TL} + \text{Revoc})$	$\xRightarrow{0.9} A$
$AC \xrightarrow{2} C$	$AC \xRightarrow{2} C(\text{TL} + \text{Revoc})$
$\xrightarrow{0.1} \text{HTLC}_{\text{OUT}}$	$\xrightarrow{0.1} \text{HTLC}_{\text{IN}}$

Les sorties HTLC (OUT et IN) sont protégées par H_s (pour Charlie) et le timelock N (pour Alice)

HTLC et transaction d'engagement (2/2)

Si le secret est **dévoilé avant** l'expiration du timelock N, Alice et Bob génèrent les transactions d'engagement correspondant à l'état **après** l'acceptation du HTLC.

Alice	Charlie
$AC \xrightarrow{0.9} A(TL + Revoc)$ $\xrightarrow{2.1} C$	$AC \xrightarrow{0.9} A$ $\xrightarrow{2.1} C(TL + Revoc)$

Avantages et inconvénients

Quels sont les **avantages** et **inconvénients** du Lightning Network ?