Blockchain

Sylvain Conchon

Laboratoire Méthodes Formelles

Université Paris-Saclay, CNRS, ENS Paris-Saclay

sylvain.conchon@universite-paris-saclay.fr

BITCOIN

Bitcoin : adresses

Une adresse Bitcoin permet de désigner un “compte”. C’est l’équivalent d’un numéro IBAN pour un compte en banque.

Une adresse Bitcoin est une chaîne de 27 à 34 caractères en base 58 (pas de caractères ambigus).

```
  1 2 3 4 5 6 7 8 9
A B C D E F G H   J K L M N   P Q R S T U V W X Y Z
a b c d e f g h i j k   m n o p q r s t u v w x y z
```

Les adresses Bitcoin peuvent également être représentées par des QR-code.

Ces adresses commencent par les chiffres 1 (Pay to Pubkey Hash – P2PKH) ou 3 (Pay to Script Hash – P2SH).

L’adresse `1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa` serait celle de Satoshi (elle a reçu les 50 BTC pour le minage de Genesis).

Bitcoin : génération d'une adresse

On génère une adresse Bitcoin B à partir d'une clé publique P.

Pour cela, en hachant tout d'abord P avec la fonction SHA-256, puis le résultat est de nouveau haché avec la fonction RIPEMD-160.

La clé finale est générée en ajoutant 4 octets de checksum, un préfixe (0x00), et en appliquant un encodage base58.

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

Portefeuille

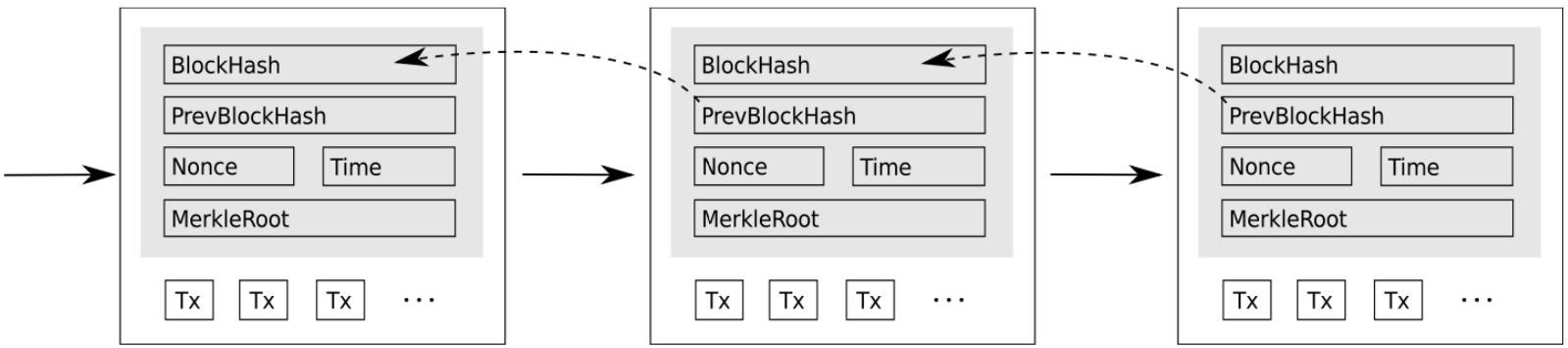
Un porte-monnaie (Wallet) pour Bitcoin est une application ou un dispositif physique qui permet de stocker, envoyer ou recevoir des crypt-monnaies.

Les clés publiques et privées d'un utilisateur sont stockées dans son wallet.

Bitcoin : chaines de blocs

(source :

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



UTXO

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 utilisées.

Ces transactions non dépensées s'appellent des UTXO (*Unspent Transaction Output*).

Les nœuds Bitcoin ne tiennent pas à jour de "comptes", ils se contentent de gérer un ensemble d'UTXOs.

Transactions Bitcoin

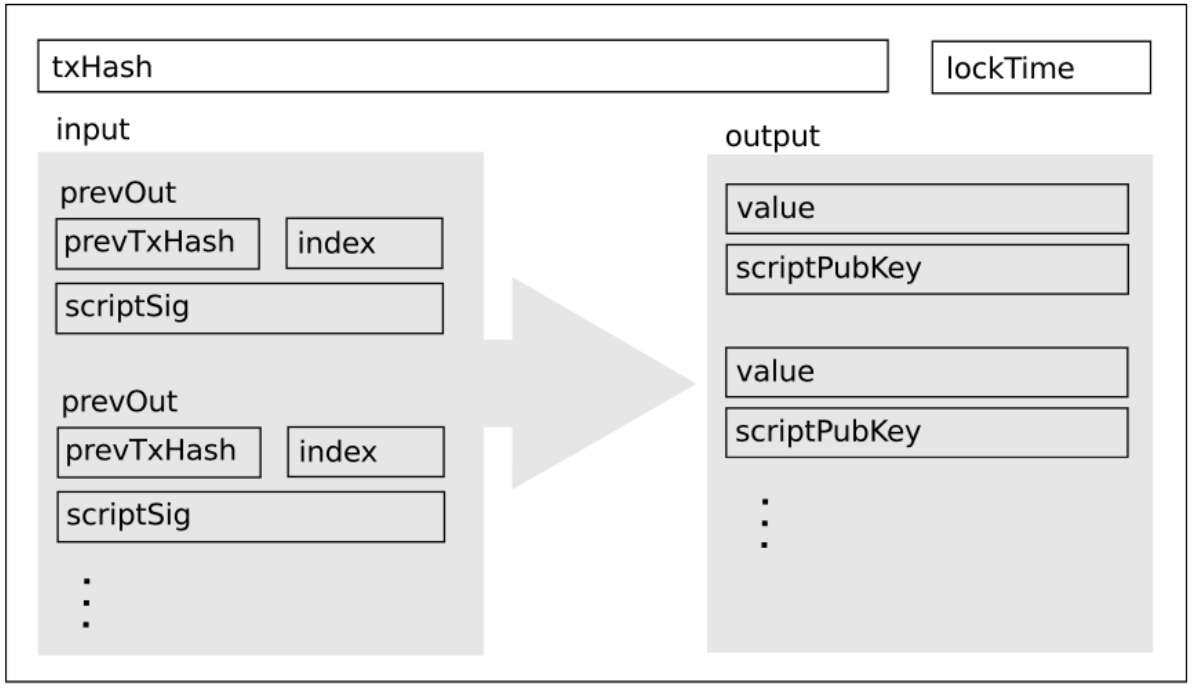
Pour construire une transaction Bitcoin, il faut créer un message qui relie des entrées et des sorties.

- Les entrées sont des valeurs de hash de transactions UTXO qui indiquent des adresses contenues dans votre wallet.
- Les sorties contiennent les adresses de destination et les valeurs envoyées à ces adresses.

Une transaction consiste donc “consommer” des transactions UTXO et à en produire de nouvelles.

En plus de ces informations, chaque transaction possède son hash et un éventuel verrou d’utilisation temporel (locktime, voir diapositives sur HTLC).

Scriptsig et ScriptPubKey



L'utilisation d'une transaction UTXO est verrouillée par un petit programme (script) appelé scriptPubKey ou Lock.

Pour la déverrouiller, il faut exécuter ce programme en lui donnant des arguments appelés Scriptsig ou Unlock.

La machine `Script` de Bitcoin

Les programmes des transactions Bitcoin sont écrits dans un langage appelé `Script`.

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

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

Exemples d'instructions :

OP_DUP	Duplicates the top stack item
OP_EQUAL	Returns 1 if the inputs are exactly equal, 0 otherwise
OP_HASH160	The input is hashed twice (SHA-256, RIPEMD-160)
OP_EQUALVERIFY	OP_EQUAL, but terminates the script in failure if false.
OP_CHECKSIG	Verifies a signature
CHECKMULTISIG	Verifies the signatures of of M-of-N multisig
IF, ELSE, ENDIF	
...	

Bitcoin : scripts (1/2)

Les transactions Bitcoin utilisent essentiellement les 4 familles de scripts suivantes :

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`	

Exercice 4 : interpréteur Script

Étendre votre moteur Bitcoin avec un interpréteur du langage script et échanger des transactions dans ce langage.

CLIENTS LÉGERS

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.

Portefeuilles

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.

Problématique

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

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

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

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

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

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.

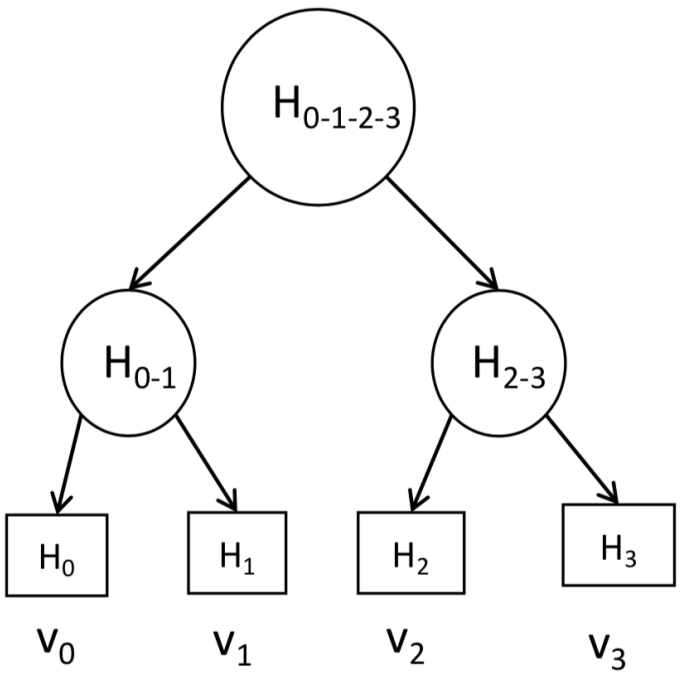
Cette structure en arbre s'appelle un arbre de Merkle. Elle a été inventée en 1979 par Ralk Merkle. D'abord déposée sous la forme d'un brevet, Merkle l'a ensuite publiée en 1987 dans l'article suivant :

R. Merkle. A digital signature based on a conventional encryption function. `https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf`

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.



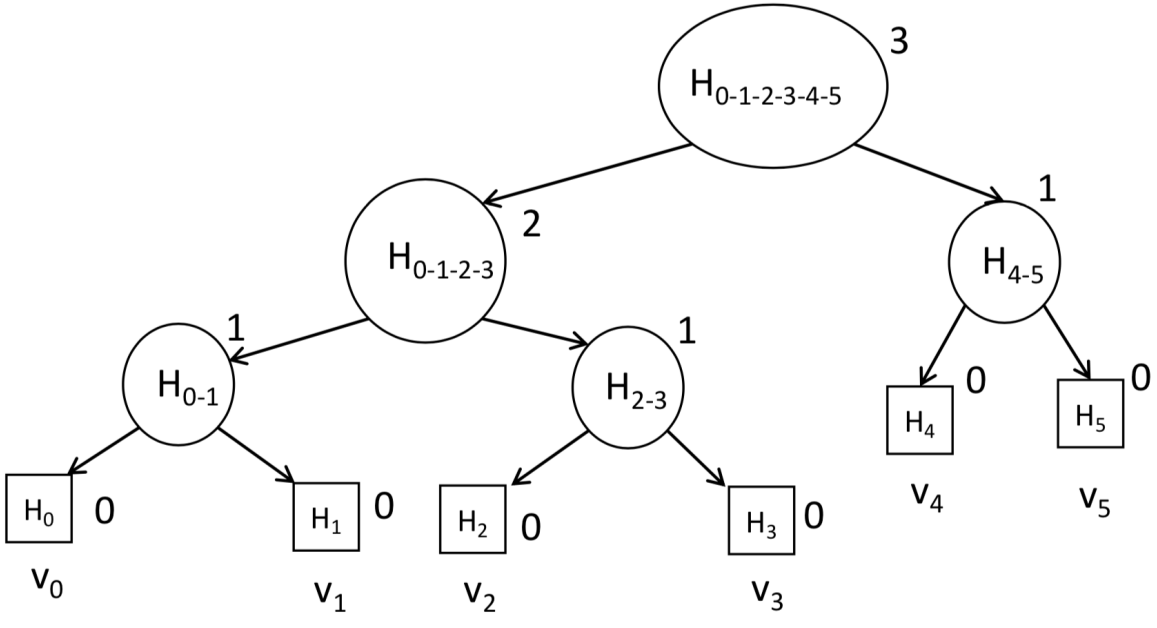
- Les feuilles $H_0, \ldots, H_3$ contiennent respectivement les empreintes des transactions $v_0, \ldots, v_3$
- Les nœuds internes H_{i-j} sont égaux à $\texttt{hash}(H_i + H_j)$
- Enfin, la racine $H_{0-1-2-3}$ est égale à $\texttt{hash}(H_{0-1} + H_{2-3})$

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 :



Racine de Merkle

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`.

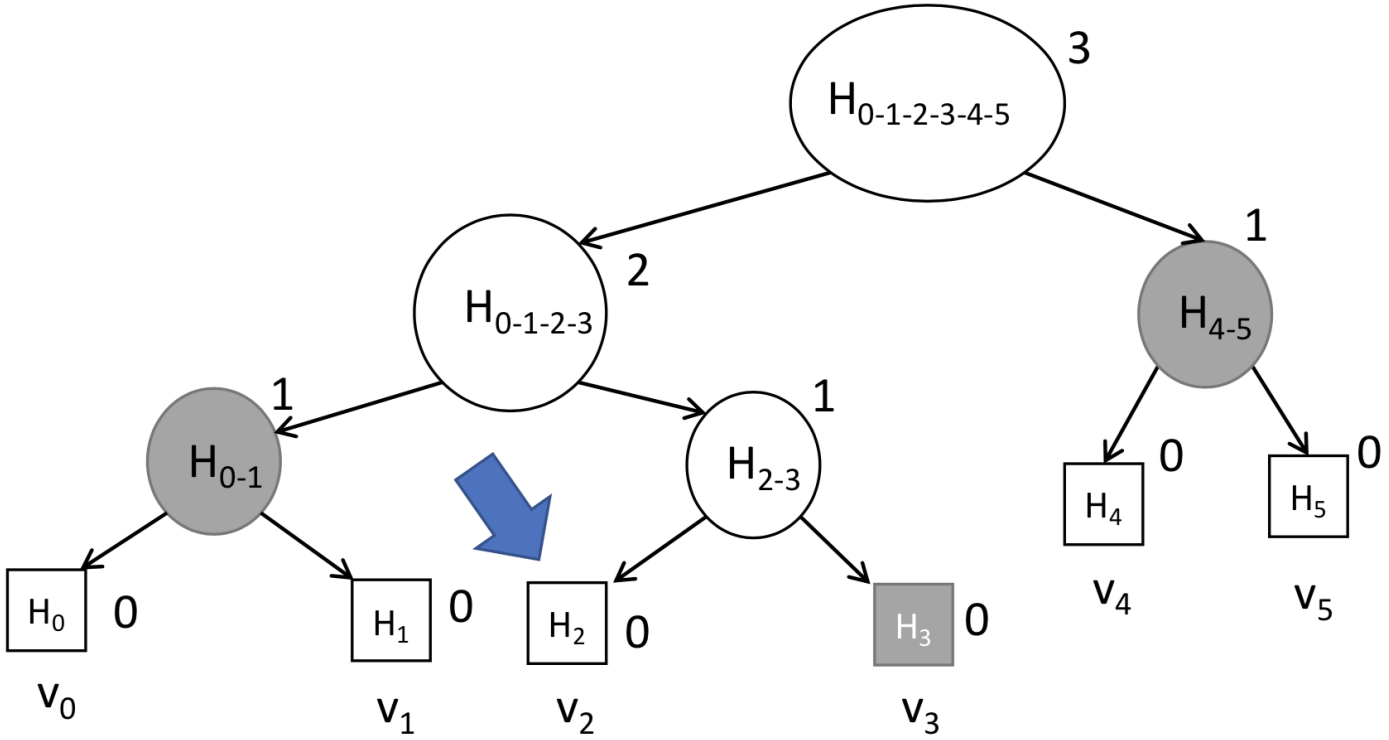
Cette liste, appelée preuve de Merkle, permet alors de refaire le calcul complet de la racine de l'arbre et vérifier qu'il est égal à la valeur stockée dans le bloc.

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

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



Exercice 5 : Clients légers

Proposer une implémentation des arbres de Merkle en Python.

Utiliser cette structure de données pour implémenter un client léger pour Bitcoin.

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

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.

$$\mathrm{T_B(Time\ Block)} = 600$$
$$\mathrm{B(Block\ size)} = 1048576$$
$$\mathrm{A(Average\ transaction\ size)} = 380$$

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.

$$\begin{aligned} &\mathsf{T_B}(\text{Time Block}) = 600 \\ &\mathsf{B}(\text{Block size}) = 1048576 \\ &\mathsf{A}(\text{Average transaction size}) = 380 \end{aligned}$$

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

$$\begin{aligned} \mathsf{TPB} &= \mathsf{B}/\mathsf{A} = 1048576/380 \approx 2759 \\ \mathsf{TPS} &= \mathsf{TPB}/\mathsf{T_B} \approx 2759/600 \approx 4.6 \end{aligned}$$

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.

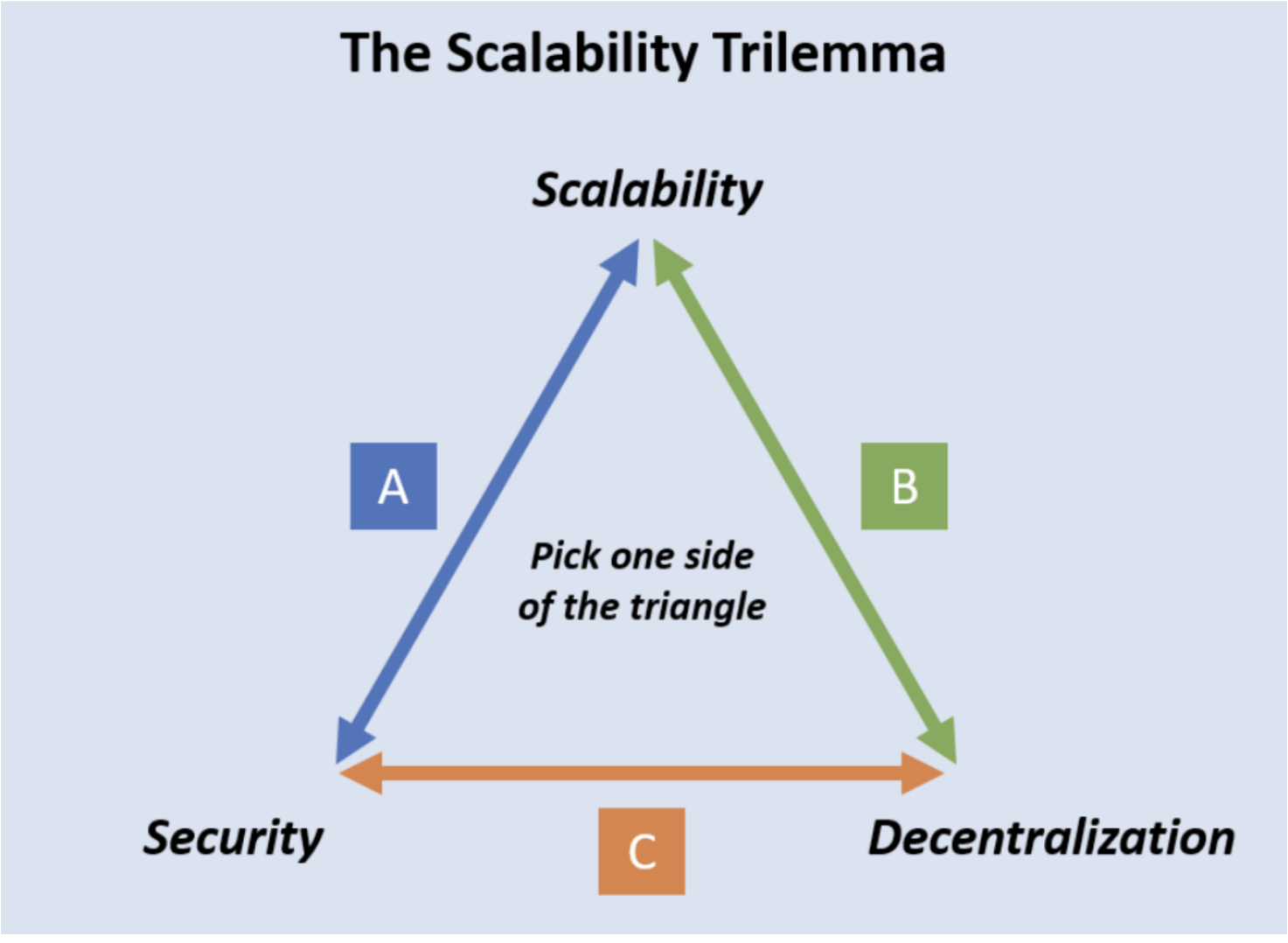
Pour obtenir un tel débit, un bloc de 11 Go devrait être miné sur Bitcoin toutes les 10 minutes, soit environ 600 petaoctets par année !

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 chaine.

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 $\overset{1}{\rightarrow}$ B : 1 BTC de Alice vers AB (signée par Alice)
2. B $\overset{0.5}{\rightarrow}$ 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 $\overset{1}{\rightarrow}$ B : 1 BTC de Alice vers AB (signée par Alice)
2. B $\overset{0.5}{\rightarrow}$ 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 $\overset{1}{\rightarrow}$ B : 1 BTC de Alice vers AB (signée par Alice)
2. B $\overset{0.5}{\rightarrow}$ 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.

$$\text{AB} \begin{array}{l} \overset{1}{\rightarrow} \text{A} \\ \overset{0.5}{\rightarrow} \text{B} \end{array}$$

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.

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 :

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

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 :

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

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 $\begin{matrix} \xrightarrow{1} \text{A} \\ \xrightarrow{0.5} \text{B} \end{matrix}$ ne soit plus utilisé ?

Timelock et Clé de révocation

Il reste un problème important :

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

Pour cela, on va définir des scripts qui permettent de vérouiller 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 $\begin{matrix}\xrightarrow{1} A \\ \xrightarrow{0.5} B\end{matrix}$ ne soit plus utilisé ?

Pour cela, on va définir des scripts qui permettent de vérouiller 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 connait pas la clé de révocation.

Transactions d'engagement sécurisées

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

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

où $\mathsf{HS_A}$ (resp. $\mathsf{HS_B}$) 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 constuits par Alice et Bob sont définis sous la forme de contrats avec Timelock et clé de révocation. Celui d'Alice a la forme suivante :

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

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

Protocole sur un canal

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.

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.

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 connait 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 :

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

où x_{A} et x_{B} représentent les soldes de Alice et Bob sur le compte commun, respectivement.

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 :

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

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 (LN) 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.

$$\mathsf{A}(x_{\mathsf{A}}) \overset{x}{\longleftrightarrow} (x_{\mathsf{C}})\mathsf{C}(y_{\mathsf{C}}) \overset{y}{\longleftrightarrow} (y_{\mathsf{B}})\mathsf{B}$$

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 :

$$\mathsf{A}(x_{\mathsf{A}} - d) \overset{x}{\longleftrightarrow} (x_{\mathsf{C}} + d)\mathsf{C}(y_{\mathsf{C}} - d) \overset{y}{\longleftrightarrow} (y_{\mathsf{B}} + d)\mathsf{B}$$

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 :

$$\mathsf{A}(x_{\mathsf{A}} - d) \overset{x}{\longleftrightarrow} (x_{\mathsf{C}} + d)\mathsf{C}(y_{\mathsf{C}} - d) \overset{y}{\longleftrightarrow} (y_{\mathsf{B}} + d)\mathsf{B}$$

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

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 :

$$\mathsf{A}(x_{\mathsf{A}} - d) \overset{x}{\longleftrightarrow} (x_{\mathsf{C}} + d)\mathsf{C}(y_{\mathsf{C}} - d) \overset{y}{\longleftrightarrow} (y_{\mathsf{B}} + d)\mathsf{B}$$

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

Mais cet échange peut échouer si $y_{\mathsf{C}} < d$. Dans ce cas, Alice doit :

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

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 :

$$\mathsf{A}(x_{\mathsf{A}} - d) \overset{x}{\longleftrightarrow} (x_{\mathsf{C}} + d)\mathsf{C}(y_{\mathsf{C}} - d) \overset{y}{\longleftrightarrow} (y_{\mathsf{B}} + d)\mathsf{B}$$

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

Mais cet échange peut échouer si $y_{\mathsf{C}} < d$. Dans ce cas, Alice doit :

- chercher un autre chemin dans le graphe en espérant que tous les noeuds aient suffisament 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 :

$$\mathsf{A}(x_{\mathsf{A}} - (d + \epsilon)) \overset{x}{\longleftrightarrow} (x_{\mathsf{C}} + d + \epsilon)\mathsf{C}(y_{\mathsf{C}} - d) \overset{y}{\longleftrightarrow} (y_{\mathsf{B}} + d)\mathsf{B}$$

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.

HTLC

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).

HTLC

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 $\mathsf{H}_s = hash(s)$, 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 $\mathsf{H}_s = hash(s)$, le hash d'un secret s connu uniquement de Bob.

2. Alice envoie ensuite un contrat $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N})$ 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 $\mathsf{H}_s = hash(s)$, le hash d'un secret s connu uniquement de Bob.

2. Alice envoie ensuite un contrat $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N})$ 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 $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N} - i)$ 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 $\mathsf{H}_s = hash(s)$, le hash d'un secret s connu uniquement de Bob.

2. Alice envoie ensuite un contrat $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N})$ 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 $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N} - i)$ à Bob (avec $1 \leq i$).

4. Bob déverrouille le contrat en montrant qu'il connait 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 $\mathsf{H}_s = hash(s)$, le hash d'un secret s connu uniquement de Bob.

2. Alice envoie ensuite un contrat $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N})$ 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 $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N} - i)$ à Bob (avec $1 \leq i$).

4. Si Bob ne déverrouille pas le contrat, dans ce cas le bloc $\mathsf{N} - i$ 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 $\mathsf{H}_s = hash(s)$, le hash d'un secret s connu uniquement de Bob.

2. Alice envoie ensuite un contrat $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N})$ 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 $\mathsf{HTLC}(d, \mathsf{H}_s, \mathsf{N} - i)$ à Bob (avec $1 \leq i$).

4. Si Bob ne déverrouille pas le contrat, dans ce cas le bloc $\mathsf{N} - i$ 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 $\mathsf{N} - i$?

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 :

$$\mathsf{A}(1) \longleftrightarrow (2)\mathsf{C}$$

et qu'Alice souhaite envoyer $\mathsf{HTLC}(0.1, \mathsf{H}_s, \mathsf{N})$ à Charlie.

Un HTLC est représenté par les transactions d'engagement :

Alice	Charlie
$\overset{0.9}{\Rightarrow}$ A(TL + Revoc)	$\overset{0.9}{\rightarrow}$ A
AC $\overset{2}{\rightarrow}$ C	AC $\overset{2}{\Rightarrow}$ C(TL + Revoc)
$\overset{0.1}{\rightarrow}$ $\mathsf{HTLC}_{\mathsf{OUT}}$	$\overset{0.1}{\rightarrow}$ $\mathsf{HTLC}_{\mathsf{IN}}$

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 $\overset{0.9}{\Rightarrow}$ A(TL + Revoc) AC $\overset{2.1}{\rightarrow}$ C	AC $\overset{0.9}{\rightarrow}$ A AC $\overset{2.1}{\Rightarrow}$ C(TL + Revoc)

Avantages et inconvénients

Quels sont les avantages et inconvénients du Lightning Network ?