Premiere Edition

Comprendre Bitcoin

W. Foteping

Ce livre est destiné aux développeurs, mais les deux premiers chapitres discutent du bitcoin à un niveau également accessible aux non-programmeurs. Toute personne ayant une compréhension de base de la technologie peut lire les deux premiers chapitres et bien comprendre le bitcoin.

Bitcoin offre un moyen efficace de transférer de l'argent sur Internet et est contrôlé par un réseau décentralisé avec un ensemble de règles transparent, présentant ainsi une alternative à la monnaie fiduciaire contrôlée par la banque centrale.

WILLIAM FOTEPING

Comprendre Bitcoin

First edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Table des matières

1

A propos de l'auteur

William Foteping est un technologue et un entrepreneur en série qui a une expertise et compréhension simple du blockchain. En tant que conférencier, éducateur et écrivain engageant, William rend les sujets complexes accessibles et faciles à comprendre. En tant que conseiller, il aide les startups à reconnaître, évaluer et gérer les risques de sécurité et d'entreprise.

Pour la majeure partie de cette décennie, William a développé et fournit des logiciels d'entreprise, y compris des applications Web et mobiles basées sur le cloud.

La recherche actuelle de William porte sur la construction des applications à partir de la blockchain en Afrique. Ses principaux intérêts de recherche comprennent la construction des applications décentralisées (DApp), d' Organisations Autonomes Décentralisées (DAO), la sécurité et l'évolution d'un consensus sur le blockchain. William est l'un des acteurs qui reconnaît que la blockchain en Afrique va résoudre une meilleure transparence des flux financiers, va développer une technologie de gouvernance éthique et démocratique, pourra être un outil de meilleure allocation de fonds.

2

Introduction

Qu'est-ce que Bitcoin ?

Bitcoin[1] est un ensemble de concepts et de technologies qui forment la base d'un écosystème de monnaie numérique. Les unités de devise appelées bitcoin sont utilisées pour stocker et transmettre de la valeur entre les participants au réseau bitcoin. Les utilisateurs de Bitcoin communiquent entre eux en utilisant le protocole Bitcoin principalement via Internet, bien que d'autres réseaux de transport puissent également être utilisés. La pile de protocoles bitcoin, disponible en tant que logiciel open source , peut être exécutée sur une large gamme d'appareils informatiques, y compris les ordinateurs portables et les smartphones, ce qui rend la technologie facilement accessible.

[1] "Bitcoin : A Peer-to-Peer Electronic Cash System," Satoshi Nakamoto (https ://bitcoin.org/bitcoin.pdf).

Les utilisateurs peuvent transférer des bitcoins sur le réseau pour faire à peu près tout ce qui peut être fait avec les devises conventionnelles, y compris l'achat et la vente de biens, l'envoi d'argent à des personnes ou des organisations ou l'octroi de crédit. Le Bitcoin peut être acheté, vendu et échangé contre d'autres devises dans des bureaux de change spécialisés. Bitcoin en un sens est la forme d'argent parfaite pour Internet car il est rapide, sécurisé et sans frontières.

Contrairement aux monnaies traditionnelles, le bitcoin est entièrement virtuel. Il n'y a pas de pièces physiques ni même de pièces numériques en soi. Les pièces sont impliquées dans les transactions qui transfèrent la valeur de l'expéditeur au destinataire. Les utilisateurs de bitcoin possèdent des clés qui leur permettent de prouver la propriété de bitcoin dans le réseau bitcoin. Avec ces clés, ils peuvent signer des transactions pour débloquer la valeur et la dépenser en la transférant à un nouveau propriétaire. Les clés sont souvent stockées dans un portefeuille numérique sur l'ordinateur ou le smartphone de chaque utilisateur. La possession de la clé permettant de signer une transaction est la seule condition préalable à la dépense de bitcoin, mettant le contrôle entièrement entre les mains de chaque utilisateur.

Bitcoin est un système distribué peer-to-peer. En tant que tel, il n'y a pas de serveur ou de point de contrôle « central ». Les bitcoins, c'est -à- dire les unités de bitcoin, sont créés par un processus appelé « extraction », qui consiste à se concurrencer pour trouver des solutions à un problème mathématique lors du traitement des transactions bitcoin. Tout participant au réseau Bitcoin (c'est-à-dire toute personne utilisant un

appareil exécutant la pile de protocoles Bitcoin complète) peut fonctionner en tant que mineur, en utilisant la puissance de traitement de son ordinateur pour vérifier et enregistrer les transactions. Toutes les 10 minutes, en moyenne, un mineur de bitcoin peut valider les transactions des 10 dernières minutes et est récompensé par un tout nouveau bitcoin. Essentiellement, l'extraction de bitcoins décentralise les fonctions d'émission de devises et de compensation d'une banque centrale et remplace le besoin de toute banque centrale.

Le protocole bitcoin comprend des algorithmes intégrés qui régulent la fonction d'extraction sur le réseau. La difficulté de la tâche de traitement que les mineurs doivent effectuer est ajustée de manière dynamique afin que, en moyenne, quelqu'un réussisse toutes les 10 minutes, quel que soit le nombre de mineurs (et la quantité de traitement) en concurrence à tout moment. Le protocole réduit également de moitié le taux de création de nouveaux bitcoins tous les 4 ans et limite le nombre total de bitcoins qui seront créés à un total fixe juste en dessous de 21 millions de pièces. Le résultat est que le nombre de bitcoins en circulation suit de près une courbe facilement prévisible qui approche les 21 millions d'ici 2140. En raison de la diminution du taux d'émission du bitcoin, sur le long terme, la monnaie bitcoin est déflationniste. De plus, le bitcoin ne peut pas être gonflé en "imprimant" de la nouvelle monnaie au-delà du taux d'émission attendu.

Dans les coulisses, bitcoin est également le nom du protocole, un réseau peer-to-peer et une innovation informatique distribuée. La monnaie bitcoin n'est en réalité que la première application de cette invention. Bitcoin représente l'aboutissement

de décennies de recherche dans la cryptographie et les systèmes distribués et comprend quatre innovations clés réunies dans une combinaison unique et puissante. Bitcoin se compose de :

- Un réseau peer-to-peer décentralisé (le protocole Bitcoin)
- Un registre public des transactions (la blockchain)
- Un ensemble de règles pour la validation indépendante des transactions et l'émission de devises (règles de consensus)
- Un mécanisme pour atteindre un consensus décentralisé mondial sur la blockchain valide (algorithme de preuve de travail)

En tant que développeur, je vois le bitcoin comme l'internet de l'argent, un réseau pour propager de la valeur et sécuriser la propriété d'actifs numériques via le calcul distribué. Il y a beaucoup plus dans le bitcoin que ce que l'on voit à première vue.

Dans ce chapitre, nous commencerons par expliquer certains des principaux concepts et termes, obtenir le logiciel nécessaire et utiliser Bitcoin pour des transactions simples. Dans les chapitres suivants, nous commencerons à développer les couches de technologie qui rendent le bitcoin possible et examinerons le fonctionnement interne du réseau et du protocole Bitcoin.

Monnaies numériques avant Bitcoin

L'émergence d'une monnaie numérique viable est étroitement liée aux développements de la cryptographie. Cela n'est pas

surprenant si l'on considère les défis fondamentaux liés à l'utilisation de bits pour représenter la valeur qui peut être échangée contre des biens et des services. Trois questions de base pour toute personne acceptant de l'argent numérique sont :

1. Puis-je être sûr que l'argent est authentique et non contrefait ?
2. Puis-je être sûr que l'argent numérique ne peut être dépensé qu'une seule fois (ce que l'on appelle le problème de la « double dépense ») ?
3. Puis-je être sûr que personne d'autre ne peut prétendre que cet argent leur appartient et pas à moi ?

Les émetteurs de papier-monnaie luttent constamment contre le problème de la contrefaçon en utilisant des papiers et des technologies d'impression de plus en plus sophistiqués. L'argent physique résout facilement le problème de la double dépense car le même billet papier ne peut pas être à deux endroits à la fois. Bien entendu, la monnaie conventionnelle est également souvent stockée et transmise numériquement. Dans ces cas, les problèmes de contrefaçon et de double dépense sont traités en compensant toutes les transactions électroniques par le biais d'autorités centrales qui ont une vision globale de la monnaie en circulation. Pour la monnaie numérique, qui ne peut pas tirer parti des encres ésotériques ou des bandes holographiques, la cryptographie fournit la base pour faire confiance à la légitimité de la revendication de valeur d'un utilisateur. Plus précisément, les signatures numériques cryptographiques permettent à un utilisateur de signer un actif numérique ou une transaction prouvant la propriété de cet actif. Avec l'architecture appropriée, les signatures numériques

peuvent également être utilisées pour résoudre le problème de la double dépense.

Lorsque la cryptographie a commencé à être plus largement disponible et comprise à la fin des années 1980, de nombreux chercheurs ont commencé à essayer d'utiliser la cryptographie pour créer des monnaies numériques. Ces premiers projets de monnaie numérique ont émis de la monnaie numérique, généralement soutenue par une monnaie nationale ou un métal précieux comme l'or.

Bien que ces anciennes monnaies numériques aient fonctionné, elles étaient centralisées et, par conséquent, étaient faciles à attaquer par les gouvernements et les pirates. Les premières monnaies numériques utilisaient une chambre de compensation centrale pour régler toutes les transactions à intervalles réguliers, tout comme un système bancaire traditionnel. Malheureusement, dans la plupart des cas, ces monnaies numériques naissantes ont été ciblées par des gouvernements inquiets et ont finalement été condamnées à leur disparition. Certains ont échoué dans des crashs spectaculaires lorsque la société mère a liquidé brutalement. Pour résister à l'intervention d'antagonistes, qu'il s'agisse de gouvernements légitimes ou d'éléments criminels, une monnaie numérique *décentralisée* était nécessaire pour éviter un point d'attaque unique. Bitcoin est un tel système, décentralisé par conception, et libre de toute autorité centrale ou point de contrôle qui peut être attaqué ou corrompu.

Histoire du Bitcoin

Le Bitcoin a été inventé en 2008 avec la publication d'un article intitulé « Bitcoin : un système de paiement électronique peer-to-peer » [1] écrit sous le pseudonyme de Satoshi Nakamoto (voir [satoshi_whitepaper]). Nakamoto a combiné plusieurs inventions antérieures telles que la b-money et HashCash pour créer un système de trésorerie électronique complètement décentralisé qui ne repose pas sur une autorité centrale pour l'émission de devises ou le règlement et la validation des transactions. La principale innovation a été d'utiliser un système de calcul distribué (appelé algorithme de « preuve de travail ») pour mener une « élection » globale toutes les 10 minutes, permettant au réseau décentralisé d'arriver à un *consensus* sur l'état des transactions. Cela résout élégamment le problème de la double dépense où une seule unité monétaire peut être dépensée deux fois. Auparavant, le problème de la double dépense était une faiblesse de la monnaie numérique et était résolu en compensant toutes les transactions via une chambre de compensation centrale.

Le réseau bitcoin a commencé en 2009, sur la base d'une implémentation de référence publiée par Nakamoto et révisée depuis par de nombreux autres programmeurs. La mise en œuvre de l'algorithme Proof-of-Work (exploitation minière) qui fournit la sécurité et la résilience du bitcoin a augmenté de manière exponentielle en puissance et dépasse désormais la puissance de traitement combinée des meilleurs supercalculateurs du monde. La valeur marchande totale de Bitcoin a parfois dépassé 1000 milliards de dollars américains, en fonction du taux de change

bitcoin en dollar. La plus grosse transaction traitée à ce jour par le réseau était de 1,1 milliard de dollars américains, transmise instantanément et traitée pour un montant de seulement 0,68 $.

Satoshi Nakamoto s'est retiré du public en avril 2011, laissant la responsabilité de développer le code et le réseau à un groupe de bénévoles prospère. L'identité de la ou des personnes derrière Bitcoin est encore inconnue. Cependant, ni Satoshi Nakamoto ni personne d'autre n'exerce de contrôle individuel sur le système Bitcoin, qui fonctionne sur la base de principes mathématiques totalement transparents, d' un code open source et d'un consensus entre les participants. L'invention elle-même est révolutionnaire et a déjà engendré une nouvelle science dans les domaines de l'informatique distribuée, de l'économie et de l'économétrie.

Une solution à un problème de calcul distribué

L'invention de Satoshi Nakamoto est également une solution pratique et novatrice à un problème de l'informatique distribuée, connu sous le nom de « problème des généraux byzantins ». En bref, le problème consiste à essayer de se mettre d'accord sur une ligne de conduite ou sur l'état d'un système en échangeant des informations sur un réseau peu fiable et potentiellement compromis. La solution de Satoshi Nakamoto, qui utilise le concept de preuve de travail pour parvenir à un consensus *sans autorité centrale de confiance* , représente une percée dans l'informatique distribuée et a une large applicabilité

au-delà de la monnaie. Il peut être utilisé pour parvenir à un consensus sur les réseaux décentralisés afin de prouver l'équité des élections, des loteries, des registres d'actifs, de la notarisation numérique, etc.

Utilisations de Bitcoin, les utilisateurs et leurs histoires

Bitcoin est une innovation dans l'ancienne technologie de l'argent. À la base, l'argent facilite simplement l'échange de valeur entre les personnes. Par conséquent, afin de bien comprendre le bitcoin et ses utilisations, nous l'examinerons du point de vue des personnes qui l'utilisent. Chacune des personnes et leurs histoires, telles qu'énumérées ici, illustrent un ou plusieurs cas d'utilisation spécifiques. Nous les verrons tout au long du livre :

Commerce de détail de faible valeur en Amérique du Nord

Alice vit dans la région de la baie de Californie du Nord. Elle a entendu parler du bitcoin par ses amis technophiles et souhaite commencer à l'utiliser. Nous suivrons son histoire au fur et à mesure qu'elle apprend le bitcoin, en acquiert, puis passe une partie de son bitcoin pour acheter une tasse de café au Bob's Cafe à Palo Alto. Cette histoire nous présentera le logiciel, les échanges et les transactions de base du point de vue d'un consommateur de détail.

Commerce de détail de grande valeur en Amérique du Nord

Carol est propriétaire d'une galerie d'art à San Francisco. Elle vend des peintures coûteuses pour Bitcoin. Cette histoire présentera les risques d'une attaque consensuelle à « 51% » pour les détaillants d'articles de grande valeur.

Services contractuels offshore

Bob, le propriétaire du café à Palo Alto, est en train de créer un nouveau site Web. Il a signé un contrat avec un développeur Web indien, Gopesh , qui vit à Bangalore, en Inde. Gopesh a accepté d'être payé en bitcoin. Cette histoire examinera l'utilisation du bitcoin pour l'externalisation, les services contractuels et les virements électroniques internationaux.

magasin en ligne

Gabriel est un jeune adolescent entreprenant de Rio de Janeiro, qui dirige une petite boutique en ligne qui vend des t-shirts, des tasses à café et des autocollants de marque Bitcoin. Gabriel est trop jeune pour avoir un compte bancaire, mais ses parents encouragent son esprit d'entreprise.

Dons de bienfaisance

Eugenia est la directrice d'une organisation caritative pour enfants aux Philippines. Récemment, elle a découvert le bitcoin et souhaite l'utiliser pour atteindre un tout nouveau groupe de donateurs étrangers et nationaux afin de collecter des fonds pour son organisation caritative. Elle étudie également des moyens d'utiliser le bitcoin pour distribuer rapidement des fonds dans les domaines qui en ont besoin. Cette histoire montrera l'utilisation du bitcoin pour la collecte de fonds mondiale à travers les devises et les frontières et l'utilisation d'un registre ouvert pour la transparence dans les organisations caritatives.

Importer / Exporter

Mohammed est un importateur d'électronique à Dubaï. Il essaie d'utiliser Bitcoin pour acheter des produits électroniques

aux États-Unis et en Chine pour les importer aux EAU afin d'accélérer le processus de paiement des importations. Cette histoire montrera comment le bitcoin peut être utilisé pour les grands paiements internationaux d'entreprise à entreprise liés à des biens physiques.

Exploitation minière pour Bitcoin

Jing est étudiant en génie informatique à Shanghai. Il a construit une plate-forme « minière » pour exploiter le bitcoin en utilisant ses compétences en ingénierie pour compléter ses revenus. Cette histoire examinera la base « industrielle » du bitcoin : l'équipement spécialisé utilisé pour sécuriser le réseau bitcoin et émettre de la nouvelle monnaie.

Chacune de ces histoires est basée sur les vraies personnes et les vraies industries utilisant actuellement Bitcoin pour créer de nouveaux marchés, de nouvelles industries et des solutions innovantes aux problèmes économiques mondiaux.

Début de Test

Bitcoin est un protocole accessible à l'aide d'une application cliente qui parle le protocole. Un « portefeuille bitcoin » est l'interface utilisateur la plus courante du système bitcoin, tout comme un navigateur Web est l'interface utilisateur la plus courante pour le protocole HTTP. Il existe de nombreuses implémentations et marques de portefeuilles Bitcoin, tout comme il existe de nombreuses marques de navigateurs Web (par exemple, Chrome, Safari, Firefox et Internet Explorer). Et tout comme nous avons tous nos navigateurs préférés (Mozilla

Firefox, Yay !) Et nos méchants (Internet Explorer, Beurk !), Les portefeuilles Bitcoin varient en qualité, performances, sécurité, confidentialité et fiabilité. Il existe également une implémentation de référence du protocole bitcoin qui comprend un portefeuille, connu sous le nom de « Satoshi Client » ou « Bitcoin Core », qui est dérivé de l'implémentation originale écrite par Satoshi Nakamoto.

Choisir un portefeuille Bitcoin

Les portefeuilles Bitcoin sont l'une des applications les plus développées de l'écosystème Bitcoin. Il y a une concurrence intense, et alors qu'un nouveau portefeuille est probablement en cours de développement, plusieurs portefeuilles de l'année dernière ne sont plus activement maintenus. De nombreux portefeuilles se concentrent sur des plates-formes spécifiques ou des utilisations spécifiques et certains sont plus adaptés aux débutants tandis que d'autres sont remplis de fonctionnalités pour les utilisateurs avancés. Le choix d'un portefeuille est très subjectif et dépend de l'utilisation et de l'expertise de l'utilisateur. Par conséquent , il serait inutile de recommander une marque ou un portefeuille spécifique. Cependant, nous pouvons classer les portefeuilles Bitcoin en fonction de leur plate-forme et de leur fonction et fournir une certaine clarté sur tous les différents types de portefeuilles existants. Mieux encore, déplacer des clés ou des graines entre des portefeuilles Bitcoin est relativement facile, il vaut donc la peine d'essayer plusieurs portefeuilles différents jusqu'à ce que vous en trouviez un qui répond à vos besoins.

Les portefeuilles Bitcoin peuvent être classés comme suit, selon

la plate-forme :

Portefeuille sur desktop

Un portefeuille de bureau était le premier type de portefeuille Bitcoin créé comme implémentation de référence et de nombreux utilisateurs exécutent des portefeuilles de bureau pour les fonctionnalités, l'autonomie et le contrôle qu'ils offrent. L'exécution sur des systèmes d'exploitation à usage général tels que Windows et Mac OS présente certains inconvénients en matière de sécurité, car ces plates-formes sont souvent non sécurisées et mal configurées.

Portefeuille mobile

Un portefeuille mobile est le type de portefeuille Bitcoin le plus courant. Fonctionnant sur des systèmes d'exploitation pour téléphones intelligents tels qu'Apple iOS et Android, ces portefeuilles sont souvent un excellent choix pour les nouveaux utilisateurs. Beaucoup sont conçus pour la simplicité et la facilité d'utilisation, mais il existe également des portefeuilles mobiles complets pour les utilisateurs expérimentés.

Portefeuille Web

Les portefeuilles Web sont accessibles via un navigateur Web et stockent le portefeuille de l'utilisateur sur un serveur appartenant à un tiers. Ceci est similaire au webmail en ce sens qu'il repose entièrement sur un serveur tiers. Certains de ces services fonctionnent à l'aide de code côté client s'exécutant dans le navigateur de l'utilisateur, qui garde le contrôle des clés Bitcoin entre les mains de l'utilisateur. La plupart, cependant, présentent un compromis en prenant le contrôle des clés bitcoin des utilisateurs en échange d'une facilité d'utilisation. Il est

déconseillé de stocker de grandes quantités de bitcoin sur des systèmes tiers.

Portefeuille matériel

Les portefeuilles matériels sont des appareils qui exploitent un portefeuille Bitcoin autonome sécurisé sur du matériel spécial. Ils se connectent généralement à un ordinateur de bureau ou à un appareil mobile via un câble USB ou une communication en champ proche (NFC) et fonctionnent avec un navigateur Web ou un logiciel associé. En gérant toutes les opérations liées au bitcoin sur le matériel spécialisé, ces portefeuilles sont considérés comme très sécurisés et adaptés au stockage de grandes quantités de bitcoin.

Portefeuille en papier

Les clés contrôlant le bitcoin peuvent également être imprimées pour un stockage à long terme. Ceux-ci sont connus sous le nom de portefeuilles en papier même si d'autres matériaux (bois, métal, etc.) peuvent être utilisés. Les portefeuilles en papier offrent un moyen peu technologique mais hautement sécurisé de stocker des bitcoins à long terme. Le stockage hors ligne est également souvent appelé *stockage à froid* .

Une autre façon de catégoriser les portefeuilles Bitcoin est leur degré d'autonomie et la façon dont ils interagissent avec le réseau Bitcoin :

Client nœud complet

Un client complet, ou « nœud complet », est un client qui stocke l'historique complet des transactions Bitcoin (chaque transaction de chaque utilisateur, jamais), gère les portefeuilles

des utilisateurs et peut initier des transactions directement sur le réseau Bitcoin. Un nœud complet gère tous les aspects du protocole et peut valider indépendamment l'ensemble de la blockchain et toute transaction. Un client full-node consomme d'importantes ressources informatiques (par exemple, plus de 125 Go de disque, 2 Go de RAM) mais offre une autonomie complète et une vérification indépendante des transactions.

Client léger

Un client léger, également connu sous le nom de client de vérification de paiement simplifié (SPV), se connecte aux nœuds complets Bitcoin (mentionnés précédemment) pour accéder aux informations de transaction Bitcoin, mais stocke le portefeuille de l'utilisateur localement et crée, valide et transmet les transactions de manière indépendante. . Les clients légers interagissent directement avec le réseau bitcoin, sans intermédiaire.

Client API tiers

Un client API tiers est un client qui interagit avec Bitcoin via un système tiers d'interfaces de programmation d'application (API), plutôt qu'en se connectant directement au réseau Bitcoin. Le portefeuille peut être stocké par l'utilisateur ou par des serveurs tiers, mais toutes les transactions passent par un tiers.

En combinant ces catégorisations, de nombreux portefeuilles Bitcoin appartiennent à quelques groupes, les trois plus courants étant le client complet de bureau, le portefeuille mobile léger et le portefeuille tiers Web. Les lignes entre les différentes catégories sont souvent floues, car de nombreux portefeuilles fonctionnent sur plusieurs plates-formes et peuvent interagir

avec le réseau de différentes manières.

Pour les besoins de ce livre, nous démontrerons l'utilisation d'une variété de clients bitcoin téléchargeables, de l'implémentation de référence (Bitcoin Core) aux portefeuilles mobiles et Web. Certains des exemples nécessiteront l'utilisation de Bitcoin Core, qui, en plus d'être un client complet, expose également les API aux services de portefeuille, de réseau et de transaction. Si vous prévoyez d'explorer les interfaces programmatiques dans le système Bitcoin, vous devrez exécuter Bitcoin Core, ou l'un des clients alternatifs.

Démarrage rapide

Alice, que nous avons présentée plus haut, n'est pas une utilisatrice technique et n'a entendu parler que récemment du bitcoin par son ami Joe. Lors d'une fête, Joe explique à nouveau avec enthousiasme le bitcoin à tout le monde et propose une démonstration. Intriguée, Alice demande comment elle peut commencer avec Bitcoin. Joe dit qu'un portefeuille mobile est le meilleur pour les nouveaux utilisateurs et il recommande quelques-uns de ses portefeuilles préférés. Alice télécharge " Bluewallet " pour Android et l'installe sur son téléphone.

Quand Alice exécute Bluewallet pour la première fois, elle choisit l'option de créer un nouveau portefeuille Bitcoin et prend un moment **à Joe et à toutes les autres parties** pour écrire une phrase mnémonique secrète *dans l'ordre* sur un morceau de papier. Comme expliqué par le portefeuille mobile et par Joe plus tôt, la phrase mnémotechnique permet à Alice de restaurer son portefeuille au cas où elle perdrait son appareil mobile et lui

donnerait accès à ses fonds sur un autre appareil. Après avoir créé son portefeuille et sécurisé sa phrase mnémotechnique, Alice peut taper sur son portefeuille pour voir son montant en bitcoins, son historique de transactions, ainsi que deux boutons qui lui permettent de *recevoir* ou d' *envoyer des* bitcoins, indiqués dans le *Bluewallet Mobile Wallet.*

Mots mnémotechniques

Un portefeuille Bitcoin moderne fournira une *phrase mnémotechnique* (parfois également appelée « graine » ou « phrase d'amorce ») à sauvegarder pour Alice. La phrase mnémonique se compose de 12 à 24 mots anglais, sélectionnés au hasard par le logiciel, et utilisés comme base pour les clés générées par le portefeuille. La phrase mnémotechnique peut être utilisée par Alice pour restaurer toutes les transactions et les fonds dans son portefeuille en cas d'événement tel qu'un appareil mobile perdu, un bug logiciel ou une corruption de la mémoire.

> *Conseil*
>
> *Le terme correct pour ces mots de sauvegarde est « phrase mnémotechnique ». Nous évitons d'utiliser le terme « graine » pour désigner une phrase mnémotechnique, car même si son utilisation est courante, elle est incorrecte.*

Stocker le Mnemonic en toute sécurité

Alice doit faire attention à stocker la phrase mnémotechnique

d'une manière qui équilibre la nécessité d'éviter le vol et la perte accidentelle. Si elle ne la protège pas suffisamment, son mnémonique risque d'être volé. Si elle le protège trop, son mnémonique risque d'être définitivement perdu. La manière recommandée pour équilibrer correctement ces risques est d'écrire deux copies de la phrase mnémotechnique sur papier, chacun des mots étant numéroté selon l'ordre.

Une fois qu'Alice a enregistré la phrase mnémotechnique, elle devrait prévoir de stocker chaque copie dans un endroit sûr séparé, comme un tiroir de bureau verrouillé ou un coffre-fort ignifuge.

Avertissement

N'essayez jamais un schéma de sécurité « DIY » qui s'écarte de quelque manière que ce soit de la recommandation des meilleures pratiques en matière de stockage du Mnemonic *en toute sécurité . Ne coupez pas votre mnémonique en deux, ne faites pas de captures d'écran, ne stockez pas sur des clés USB, des e - mails ou des lecteurs cloud, ne les cryptez pas ou n'essayez aucune autre méthode non standard. Vous ferez pencher la balance de manière à risquer une perte permanente ou un vol. De nombreuses personnes ont perdu des fonds, non pas par vol, mais parce qu'elles ont essayé une solution non standard sans avoir l'expertise nécessaire pour équilibrer les risques encourus. La recommandation de bonnes pratiques est soigneusement équilibrée par des experts et convient à la grande majorité des utilisateurs.*

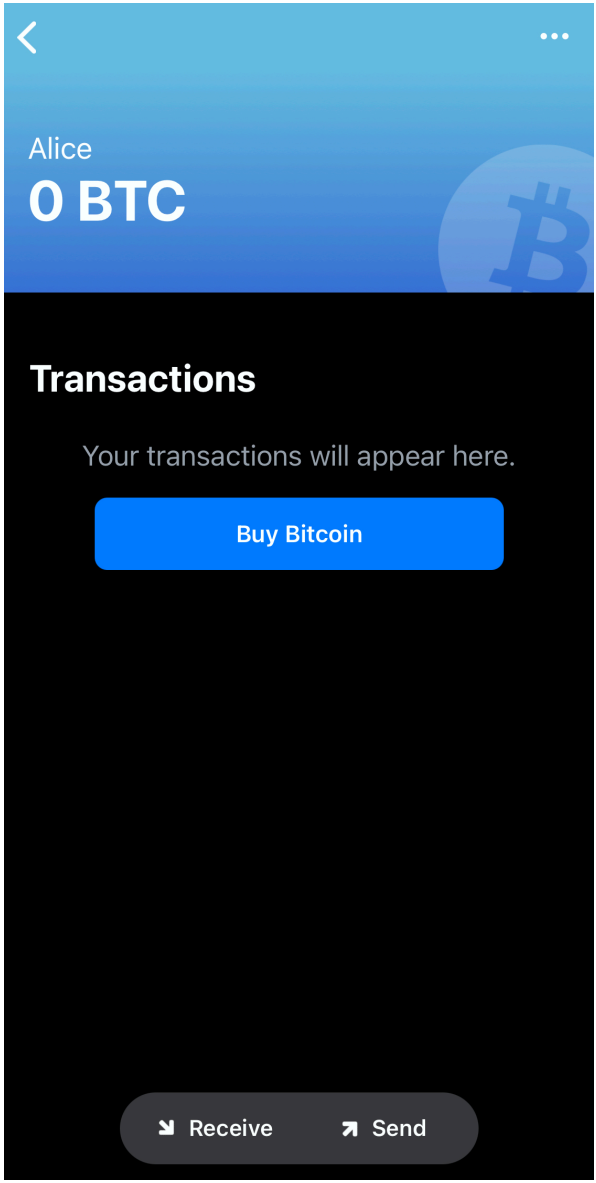


Figure 1. Un portefeuille mobile

La vue principale du portefeuille affiche le montant du bitcoin, l'historique des transactions et les boutons *Recevoir* et *Envoyer* . En outre, de nombreux portefeuilles offrent la possibilité d'acheter Bitcoin directement via un échange ou un service similaire où vous pouvez offrir de l'argent fiduciaire en échange de crypto-monnaie, ce qui se fait en trouvant le prix actuel du Bitcoin et en vendant à l'utilisateur du portefeuille à egal ou au-dessus de ce prix. . Le bouton *Acheter Bitcoin* permettrait à Alice d'acheter Bitcoin de cette manière.

Alice est maintenant prête à commencer à utiliser son nouveau portefeuille Bitcoin. Son application de portefeuille a généré de manière aléatoire une clé privée (décrite plus en détail dans un cours prochain) qui sera utilisée pour dériver des adresses bitcoin qui se dirigent vers son portefeuille. À ce stade, ses adresses Bitcoin ne sont pas connues du réseau Bitcoin ou "enregistrées" avec une partie quelconque du système Bitcoin. Ses adresses bitcoin sont simplement des nombres aléatoires qui correspondent à sa clé privée qu'elle peut utiliser pour contrôler l'accès aux fonds. Les adresses sont générées indépendamment par son portefeuille sans référence ni inscription à aucun service. En fait, dans la plupart des portefeuilles, il n'y a aucune association entre une adresse bitcoin et des informations identifiables de l'extérieur, y compris l'identité de l'utilisateur. Jusqu'au moment où une adresse est référencée en tant que destinataire de valeur dans une transaction publiée sur le registre bitcoin, l'adresse bitcoin fait simplement partie du grand nombre d'adresses possibles qui sont valides en bitcoin. Ce n'est qu'une fois qu'une adresse a été associée à une transaction qu'elle fait partie des adresses connues du réseau.

Alice utilise le bouton *Recevoir*, qui affiche un code QR avec une adresse bitcoin. Le code QR est le carré avec un motif de points noirs et blancs, servant de forme de code-barres contenant les mêmes informations dans un format pouvant être scanné par la caméra du smartphone de Joe. Dans la plupart des portefeuilles, appuyer sur le code QR l'agrandira également, de sorte qu'il puisse être plus facilement scanné. À côté du code QR du portefeuille se trouve l'adresse bitcoin qu'il code, et Alice peut choisir d'envoyer manuellement son adresse à Joe en la copiant dans son presse-papiers d'un simple toucher. Il est à noter que lors de la première réception de fonds sur un nouveau portefeuille mobile, de nombreux portefeuilles revérifieront souvent que vous avez bien sécurisé votre phrase mnémotechnique. Cela peut aller d'une simple invite à demander à l'utilisateur de ressaisir manuellement la phrase.

Conseil

Les adresses Bitcoin commencent par 1, 3 ou bc1. Comme les adresses e-mail, elles peuvent être partagées avec d'autres utilisateurs de bitcoin qui peuvent les utiliser pour envoyer des bitcoins directement dans votre portefeuille. Il n'y a rien de sensible, du point de vue de la sécurité, à l'adresse bitcoin. Il peut être posté n'importe où sans risquer la sécurité du compte. Contrairement aux adresses e-mail, vous pouvez créer de nouvelles adresses aussi souvent que vous le souhaitez, qui dirigeront toutes des fonds vers votre portefeuille. En fait, de nombreux portefeuilles modernes créent automatiquement une nouvelle adresse pour chaque transaction afin de maximiser

la confidentialité. Un portefeuille est simplement une collection d'adresses et de clés qui débloquent les fonds à l'intérieur.

Alice est maintenant prête à recevoir des fonds. Son application de portefeuille a généré de manière aléatoire une clé privée avec son adresse bitcoin correspondante. À ce stade, son adresse Bitcoin n'est pas connue du réseau Bitcoin ou "enregistrée" avec une partie quelconque du système Bitcoin. Son adresse bitcoin est simplement un nombre qui correspond à une clé qu'elle peut utiliser pour contrôler l'accès aux fonds. Il a été généré indépendamment par son portefeuille sans référence ni inscription à aucun service. En fait, dans la plupart des portefeuilles, il n'y a aucune association entre l'adresse bitcoin et des informations identifiables de l'extérieur, y compris l'identité de l'utilisateur. Jusqu'au moment où cette adresse est référencée en tant que destinataire de valeur dans une transaction publiée sur le registre bitcoin, l'adresse bitcoin fait simplement partie du grand nombre d'adresses possibles qui sont valides en bitcoin. Ce n'est qu'une fois qu'il a été associé à une transaction qu'il fait partie des adresses connues du réseau.

Obtenir son premier Bitcoin

Il existe plusieurs façons pour Alice d'acquérir des bitcoins :

- Elle peut échanger une partie de sa monnaie nationale (par exemple USD) dans un échange de crypto-monnaie
- Elle peut en acheter à un ami ou à une connaissance d'un Meetup Bitcoin, en échange d' argent liquide
- Elle peut trouver un *guichet automatique Bitcoin* dans sa

région, qui fait office de distributeur automatique, vendant du bitcoin contre de l' argent
- Elle peut proposer ses compétences ou un produit qu'elle vend et accepte le paiement en bitcoin
- Elle peut demander à son employeur ou à ses clients de la payer en bitcoin

Toutes ces méthodes présentent des degrés de difficulté variables, et beaucoup impliqueront le paiement de frais. Certaines institutions financières exigeront également qu'Alice fournisse des documents d'identification pour se conformer aux réglementations bancaires locales / aux pratiques anti-blanchiment d'argent (AML), un processus connu sous le nom de Know Your Customer (KYC). Cependant, avec toutes ces méthodes, Alice pourra recevoir des bitcoins.

Conseil

L'un des avantages du bitcoin par rapport aux autres systèmes de paiement est que, lorsqu'il est utilisé correctement, il offre aux utilisateurs beaucoup plus de confidentialité. Acquérir, détenir et dépenser des bitcoins ne vous oblige pas à divulguer des informations sensibles et personnellement identifiables à des tiers. Cependant, lorsque le bitcoin touche les systèmes traditionnels, tels que les échanges de devises, les réglementations nationales et internationales s'appliquent souvent. Afin d' échanger des bitcoins contre votre monnaie nationale, vous devrez souvent fournir une preuve d'identité et des informations bancaires. Les utilisateurs doivent savoir qu'une fois

> *qu'une adresse bitcoin est associée à une identité, toutes les transactions bitcoin associées sont également faciles à identifier et à suivre. C'est l'une des raisons pour lesquelles de nombreux utilisateurs choisissent de conserver des comptes d'échange dédiés non liés à leur portefeuille.*

Alice a été initiée au bitcoin par un ami , elle a donc un moyen facile d'acquérir son premier bitcoin. Ensuite, nous verrons comment elle achète du bitcoin à son ami Joe et comment Joe envoie le bitcoin dans son portefeuille.

Trouver le prix actuel du Bitcoin

Avant qu'Alice puisse acheter du bitcoin à Joe, ils doivent s'entendre sur le *taux de change* entre le bitcoin et le dollar américain. Cela soulève une question courante pour ceux qui découvrent le bitcoin : "Qui fixe le prix du bitcoin ?" La réponse courte est que le prix est fixé par les marchés.

Bitcoin, comme la plupart des autres devises, a un *taux de change flottant* . Cela signifie que la valeur du bitcoin par rapport à toute autre devise fluctue en fonction de l'offre et de la demande sur les différents marchés sur lesquels il est négocié. Par exemple, le « prix » du bitcoin en dollars américains est calculé sur chaque marché en fonction des échanges les plus récents de bitcoin et de dollars américains. En tant que tel, le prix a tendance à fluctuer minutieusement plusieurs fois par seconde. Un service de tarification agrégera les prix de plusieurs marchés et calculera une moyenne pondérée en fonction du volume représentant le taux de change général du marché d'une paire de devises (par exemple, BTC / USD).

Il existe des centaines d'applications et de sites Web qui peuvent fournir le taux actuel du marché. Voici quelques-uns des plus populaires :

Bitcoin Average

Un site qui fournit une vue simple de la moyenne pondérée en volume pour chaque devise.

CoinCap

Un service répertoriant la capitalisation boursière et les taux de change de centaines de crypto-monnaies , dont le bitcoin.

Chicago Mercantile Exchange Bitcoin Reference Rate

Un taux de référence qui peut être utilisé comme référence institutionnelle et contractuelle, fourni dans le cadre des flux de données d'investissement par le CME.

En plus de ces divers sites et applications, la plupart des portefeuilles Bitcoin convertiront automatiquement les montants entre Bitcoin et d'autres devises. Joe utilisera son portefeuille pour convertir automatiquement le prix avant d'envoyer du bitcoin à Alice.

Envoi et réception de Bitcoin

Alice a décidé d'échanger 10 dollars américains contre du bitcoin, afin de ne pas risquer trop d'argent sur cette nouvelle technologie. Elle donne à Joe 10 $ en espèces, ouvre son application de portefeuille mobile Bluewallet et sélectionne Recevoir. Cela affiche un code QR avec la première adresse bitcoin d'Alice.

Joe sélectionne ensuite Envoyer sur son portefeuille de smartphone Bluewallet et se voit présenter un écran contenant les entrées suivantes :

- Le montant à envoyer, en bitcoin (BTC) ou dans sa devise locale (USD)
- Une adresse Bitcoin de destination
- Une note de transaction (description)
- Des frais de transaction

Dans le champ de saisie de l'adresse bitcoin, il y a un petit bouton *Scan* . Cela permet à Joe de scanner le code QR avec la caméra de son smartphone afin de ne pas avoir à taper l'adresse bitcoin d'Alice, qui est assez longue et difficile à taper. Joe appuie sur le bouton *Scan* et il active la caméra du smartphone, scannant le code QR affiché sur le smartphone d'Alice.

Joe a maintenant l'adresse bitcoin d'Alice définie comme destinataire. Joe entre le montant de 10 dollars américains et son portefeuille le convertit en accédant au taux de change le plus récent à partir d'un service en ligne. Le taux de change à l'époque est de 100 dollars américains par bitcoin, donc 10 dollars américains valent 0,10 bitcoin (BTC), ou 100 millibitcoins (mBTC), comme le montre la capture d'écran du portefeuille de Joe (voir l' écran d'envoi du portefeuille bitcoin mobile Bluewallet plus bas).

Dans l'entrée note / description de la transaction, Joe entre "Alice". Il peut utiliser ce champ pour ajouter des informations concernant sa transaction pour référence future. Cette fonction est destinée à la tenue de ses archives uniquement. La note de

transaction sera stockée dans son portefeuille et seul Joe pourra la voir. Il ne sera pas envoyé à Alice, ni stocké sur la blockchain.

Il sélectionne également des frais de transaction pour sa transaction. Plus les frais de transaction sont élevés, plus sa transaction sera confirmée rapidement (incluse dans un bloc par un mineur). Il sélectionne les frais de transaction minimum possibles à ce moment-là (0 sat / b).

Conseil

Le prix du bitcoin a beaucoup changé au fil du temps, et une quantité incroyable depuis la rédaction de la première édition de ce livre. En mars 2021, une personne aurait besoin d'environ 54000 USD pour acheter un bitcoin entier. De nombreux exemples dans ce livre font référence à des transactions passées dans la vie réelle, lorsque le prix du bitcoin était beaucoup plus bas et que les transactions sans frais étaient encore possibles. Pensez à la générosité d'un ami que Joe aurait été s'il avait conclu le même accord avec Alice aujourd'hui !

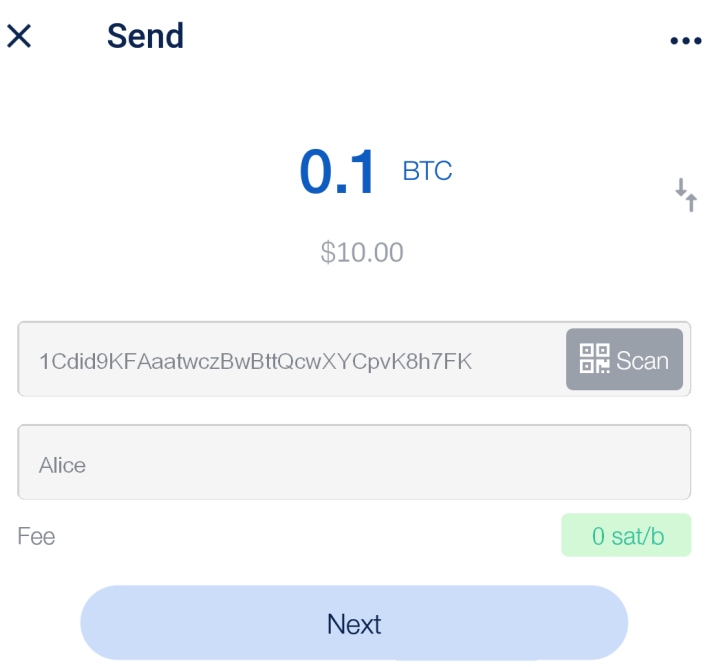


figure 2. Photo d'envoi dans le porte - monnaie Bitcoin

Joe vérifie ensuite soigneusement qu'il a saisi le montant correct, car il est sur le point de transmettre de l'argent et les erreurs sont irréversibles. Après avoir revérifié l'adresse et le montant, il appuie sur Envoyer pour transmettre la transaction. Le portefeuille bitcoin mobile de Joe construit une transaction qui attribue 0,10 BTC à l'adresse fournie par Alice, en tirant les fonds du portefeuille de Joe et en signant la transaction avec les clés privées de Joe. Cela indique au réseau Bitcoin que Joe a autorisé un transfert de valeur vers la nouvelle adresse

d'Alice. Lorsque la transaction est transmise via le protocole peer-to-peer, elle se propage rapidement sur le réseau bitcoin. En moins d'une seconde, la plupart des nœuds bien connectés du réseau reçoivent la transaction et voient l'adresse d'Alice pour la première fois.

Pendant ce temps, le portefeuille d'Alice « écoute » constamment les transactions publiées sur le réseau bitcoin, à la recherche de celles qui correspondent aux adresses qu'il contient. Quelques secondes après que le portefeuille de Joe ait transmis la transaction, le portefeuille d'Alice indiquera qu'il reçoit 0,10 BTC.

> *Conseil*
>
> *Chaque bitcoin peut être subdivisé en 100 millions d'unités, chacune appelée " satoshi " (singulier) ou " satoshis " (pluriel). Nommé d'après le créateur de Bitcoin, le Satoshi est la plus petite unité de bitcoin, équivalente à 0,00000001 BTC.*

Les confirmations

Au début, le portefeuille d'Alice affichera la transaction de Joe comme "Non confirmée". Cela signifie que la transaction a été propagée sur le réseau mais n'a pas encore été enregistrée dans le registre des transactions bitcoin, connu sous le nom de blockchain. Pour être confirmée, une transaction doit être incluse dans un bloc et ajoutée à la blockchain, ce qui se produit toutes les 10 minutes, en moyenne. En termes financiers traditionnels, on parle de *compensation* . Pour plus de détails sur

la propagation, la validation et la compensation (confirmation) des transactions Bitcoin, voir (le chapitre sur l'Extraction ou encore mining) .

Alice est maintenant l'heureuse propriétaire de 0,10 BTC qu'elle peut dépenser. Dans le chapitre suivant, nous examinerons son premier achat avec Bitcoin et examinerons plus en détail les technologies de transaction et de propagation sous-jacentes.

3

Comment fonctionne Bitcoin ?

Transactions, blocs, exploitation minière et blockchain

Le système Bitcoin, contrairement aux systèmes bancaires et de paiement traditionnels, est basé sur une confiance décentralisée. Au lieu d'une autorité centrale de confiance, dans Bitcoin, la confiance est obtenue en tant que propriété émergente des interactions de différents participants dans le système Bitcoin. Dans ce chapitre, nous examinerons le bitcoin à un niveau élevé en suivant une seule transaction à travers le système bitcoin et regarderons comme il devient « fiable » et accepté par le mécanisme bitcoin du consensus distribué et est finalement enregistré sur la blockchain, le grand livre distribué de toutes transactions. Les chapitres suivants se pencheront sur la technologie derrière les transactions, le réseau et l'exploitation minière.

Présentation du Bitcoin

Dans le diagramme de vue d'ensemble présenté dans *Bitcoin Overview*, nous voyons que le système Bitcoin se compose d'utilisateurs avec des portefeuilles contenant des clés, des transactions qui sont propagées sur le réseau et des mineurs qui produisent (grâce à un calcul compétitif) la blockchain de consensus, qui est le registre faisant autorité toutes transactions.

Chaque exemple de ce chapitre est basé sur une transaction réelle effectuée sur le réseau bitcoin, simulant les interactions entre les utilisateurs (Joe, Alice, Bob et Gopesh) en envoyant des fonds d'un portefeuille à un autre. Lors du suivi d'une transaction via le réseau bitcoin vers la blockchain, nous utiliserons un site *explorateur de blockchain* pour visualiser chaque étape. Un explorateur de blockchain est une application Web qui fonctionne comme un moteur de recherche Bitcoin, en ce sens qu'il vous permet de rechercher des adresses, des transactions et des blocs et de voir les relations et les flux entre eux.

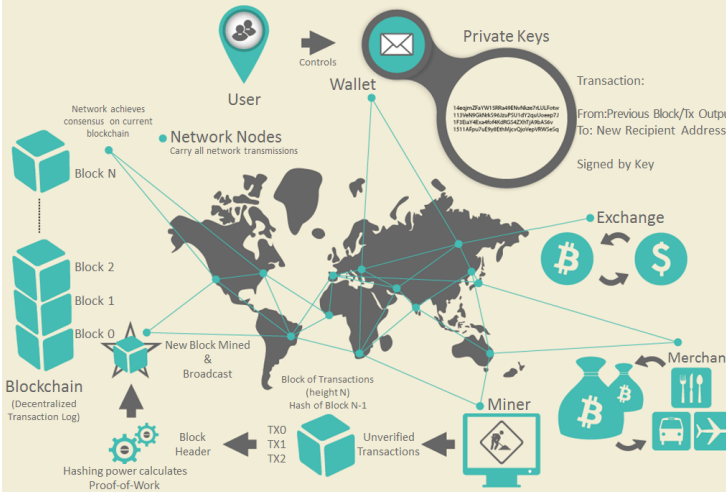


Figure 1. Présentation de Bitcoin

Les explorateurs de blockchain populaires incluent :

- Explorateur BlockCypher
- blockchain.info
- BitPay Insight
- Explorateur Blockstream

Chacun d'eux a une fonction de recherche qui peut prendre une adresse Bitcoin, un hachage de transaction, un numéro de bloc ou un hachage de bloc et récupérer les informations correspondantes du réseau Bitcoin. Pour chaque transaction ou exemple de bloc, nous vous fournirons une URL afin que vous puissiez la rechercher vous-même et l'étudier en détail.

Acheter une tasse de café

Alice, présentée dans le chapitre précédent, est une nouvelle utilisatrice qui vient d'acquérir son premier bitcoin. Dans le chapitre précédent , Alice a rencontré son ami Joe pour échanger de l'argent contre du bitcoin. La transaction créée par Joe a financé le portefeuille d'Alice avec 0,10 BTC. Maintenant, Alice va faire sa première transaction au détail, en achetant une tasse de café au café Bob's à Palo Alto, en Californie.

Bob's Cafe a récemment commencé à accepter les paiements Bitcoin en ajoutant une option Bitcoin à son système de point de vente. Les prix au Bob's Cafe sont indiqués dans la devise locale (dollars américains), mais à la caisse, les clients ont la possibilité de payer en dollars ou en bitcoins. Alice passe sa commande pour une tasse de café et Bob l'inscrit dans le registre, comme il le fait pour toutes les transactions. Le système de point de vente convertit automatiquement le prix total en dollars américains en bitcoin au taux du marché en vigueur et affiche le prix dans les deux devises :

```
Le total:
1,50 USD
0,015 BTC
```

Bob dit : "C'est un dollar cinquante ou quinze millibits ."

Le système de point de vente de Bob créera également automatiquement un code QR spécial contenant une *demande de paiement* (voir Code QR de demande de paiement ci-dessous).

Contrairement à un code QR qui contient simplement une adresse bitcoin de destination, une demande de paiement est une URL encodée en QR qui contient une adresse de destination, un montant de paiement et une description générique telle que « Bob's Cafe ». Cela permet à une application de portefeuille Bitcoin de pré-remplir les informations utilisées pour envoyer le paiement tout en affichant une description lisible par l'homme à l'utilisateur. Vous pouvez scanner le code QR avec une application de portefeuille Bitcoin pour voir ce qu'Alice verrait.

Figure 2. Code QR de la demande de paiement

Conseil

Essayez de scanner cela avec votre portefeuille pour voir l'adresse et le montant, mais N'ENVOYEZ PAS D'ARGENT.

Le code QR de la demande de paiement encode l'URL suivante, définie dans le BIP-21 :

```
bitcoin: 1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA?
montant = 0,015 &
label = Bob% 27s% 20Café &
message = Achat% 20at% 20Bob% 27s% 20Cafe

Composants de l'URL

Une adresse bitcoin:
"1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
Le montant du paiement: "0,015"
Une étiquette pour l'adresse du destinataire: "Bob's
Cafe"
Une description pour le paiement: "Achat au Bob's
Cafe"
```

Alice utilise son smartphone pour scanner le code-barres affiché. Son smartphone montre un paiement de 0,0150 BTC à Bob's Cafe et elle sélectionne Envoyer pour autoriser le paiement. En quelques secondes (à peu près le même laps de temps qu'une autorisation de carte de crédit), Bob voit la transaction sur le registre et termine la transaction.

Dans les sections suivantes, nous examinerons cette transaction

plus en détail. Nous verrons comment le portefeuille d'Alice l'a construit, comment il a été propagé sur le réseau, comment il a été vérifié et, enfin, comment Bob peut dépenser ce montant dans les transactions suivantes.

> *Noter*
>
> *Le réseau bitcoin peut effectuer des transactions en valeurs fractionnaires, par exemple, de millibitcoin (1 / 1000e de bitcoin) à 1/100 000 000e de bitcoin, qui est connu sous le nom de satoshi . Tout au long de ce livre, nous utiliserons le terme « bitcoin » pour désigner toute quantité de monnaie bitcoin, de la plus petite unité (1 satoshi) au nombre total (21 000 000) de tous les bitcoins qui seront jamais extraits.*

Vous pouvez examiner la transaction d'Alice avec Bob's Cafe sur la blockchain en utilisant un site d'explorateur de blocs (voir la transaction d'Alice sur blockchain.info ci- dessous) :

Exemple 1. Voir la transaction d'Alice sur blockchain.info

```
https://blockchain.info/tx/0627052b6f28912f2703066a912ea577f2ce4d
```

Transactions Bitcoin

En termes simples, une transaction indique au réseau que le propriétaire d'une certaine valeur de bitcoin a autorisé le transfert de cette valeur à un autre propriétaire. Le nouveau propriétaire peut désormais dépenser le bitcoin en créant une autre transaction qui autorise le transfert à un autre propriétaire, et ainsi de suite, dans une chaîne de propriété.

Entrées et sorties de transaction

Les transactions sont comme des lignes dans un grand livre de comptabilité en partie double. Chaque transaction contient une ou plusieurs « entrées », qui sont comme des débits sur un compte Bitcoin. De l'autre côté de la transaction, il y a une ou plusieurs « sorties », qui sont comme des crédits ajoutés à un compte bitcoin. Les entrées et sorties (débits et crédits) ne totalisent pas nécessairement le même montant. Au lieu de cela, les extrants totalisent un peu moins que les intrants et la différence représente *des frais de transaction* implicites , qui sont un petit paiement collecté par le mineur qui inclut la transaction dans le grand livre. Une transaction bitcoin est affichée comme une écriture comptable dans la transaction en tant que comptabilité en partie double .

La transaction contient également une preuve de propriété pour chaque montant de bitcoin (entrées) dont la valeur est dépensée, sous la forme d'une signature numérique du propriétaire, qui peut être validée indépendamment par n'importe qui. En termes de bitcoin, « dépenser » signifie signer une transaction qui

transfère la valeur d'une transaction précédente à un nouveau propriétaire identifié par une adresse bitcoin.

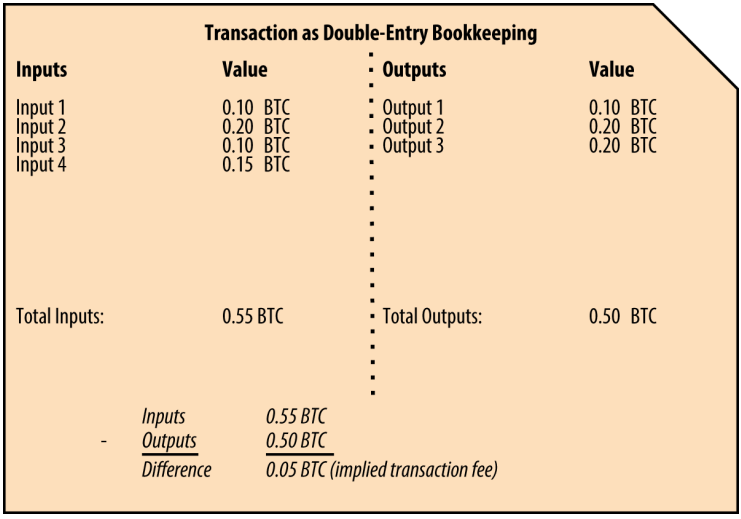


Figure 3. Transaction en tant que comptabilité en partie double

Chaînes de transaction

Le paiement d'Alice à Bob's Cafe utilise la sortie d'une transaction précédente comme entrée. Dans le chapitre précédent, Alice a reçu des bitcoins de son ami Joe en échange d'argent. Cette transaction a créé une valeur bitcoin verrouillée par la clé d'Alice. Sa nouvelle transaction avec Bob's Cafe fait référence à la transaction précédente en tant qu'entrée et crée de nouvelles sorties pour payer la tasse de café et recevoir la monnaie. Les transactions forment une chaîne, où les entrées de la

dernière transaction correspondent aux sorties des transactions précédentes. La clé d'Alice fournit la signature qui déverrouille ces sorties de transaction précédentes, prouvant ainsi au réseau bitcoin qu'elle possède les fonds. Elle attache le paiement du café à l'adresse de Bob, « encombrant » ainsi cette sortie de l'exigence que Bob produise une signature afin de dépenser ce montant. Cela représente un transfert de valeur entre Alice et Bob. Cette chaîne de transactions, de Joe à Alice en passant par Bob, est illustrée dans Une chaîne de transactions, où la sortie d'une transaction est l'entrée de la transaction suivante (voir schema ci-dessous)

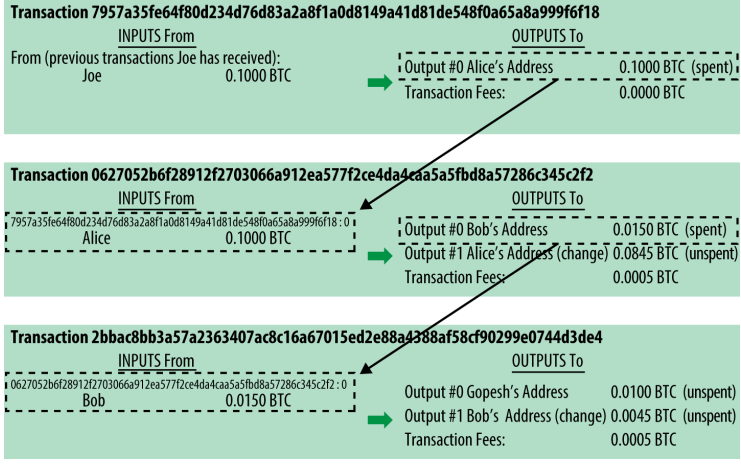


Figure 4. Une chaîne de transactions, où la sortie d'une transaction est l'entrée de la transaction suivante

Faire le changement

De nombreuses transactions Bitcoin comprendront des sorties qui référencent à la fois une adresse du nouveau propriétaire et une adresse du propriétaire actuel, appelée adresse de *changement* . En effet, les entrées de transaction, comme les billets de banque, ne peuvent pas être divisées. Si vous achetez un article de 5 $ US dans un magasin mais que vous utilisez un billet de 20 $ US pour payer l'article, vous prévoyez de recevoir 15 $ US en monnaie. Le même concept s'applique aux entrées de transaction Bitcoin. Si vous avez acheté un article qui coûte 5 bitcoins mais que vous n'aviez qu'une entrée de 20 bitcoins à utiliser, votre portefeuille créerait une seule transaction qui enverrait deux sorties, une sortie de 5 bitcoins au propriétaire du magasin et une sortie de 15 bitcoins à vous-même comme changement (moins les frais de transaction applicables). Il est important de noter que l'adresse de changement ne doit pas nécessairement être la même que celle de l'entrée et, pour des raisons de confidentialité, il s'agit souvent d'une nouvelle adresse du portefeuille du propriétaire.

Différents portefeuilles peuvent utiliser différentes stratégies lors de l'agrégation des entrées pour effectuer un paiement demandé par l'utilisateur. Ils peuvent regrouper de nombreux petits intrants ou en utiliser un égal ou supérieur au paiement souhaité. À moins que le portefeuille ne puisse regrouper les entrées de manière à correspondre exactement au paiement souhaité plus les frais de transaction, le portefeuille devra générer des changements. Ceci est très similaire à la façon dont les gens gèrent les espèces. Si vous utilisez toujours la plus grosse facture dans votre poche, vous vous retrouverez

avec une poche pleine de monnaie en vrac. Si vous n'utilisez que la monnaie en vrac, vous n'aurez toujours que de grosses factures. Les gens trouvent inconsciemment un équilibre entre ces deux extrêmes, et les développeurs de portefeuilles Bitcoin s'efforcent de programmer cet équilibre.

En résumé, les *transactions* déplacent la valeur des *entrées* de *transaction aux sorties de transaction* . Une entrée est une référence à la sortie d'une transaction précédente, indiquant d'où provient la valeur. Une transaction comprend généralement une sortie qui dirige une valeur spécifique vers l'adresse bitcoin d'un nouveau propriétaire et une sortie de modification vers le propriétaire d'origine. Les sorties d'une transaction peuvent être utilisées comme entrées dans une nouvelle transaction, créant ainsi une chaîne de propriété lorsque la valeur est déplacée du propriétaire au propriétaire (voir Une chaîne de transactions, où la sortie d'une transaction est l'entrée de la transaction suivante ci-dessus) .

Formulaires de transaction courants

La forme de transaction la plus courante est un simple paiement d'une adresse à une autre, qui comprend souvent une « modification » retournée au propriétaire d'origine. Ce type de transaction a une entrée et deux sorties et est affiché dans la transaction la plus courante (schema ci-dessous).

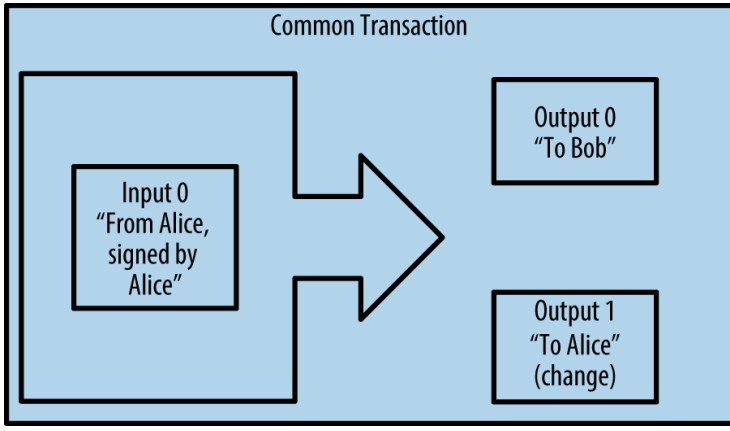


Figure 5. Transaction la plus courante

Une autre forme courante de transaction est celle qui regroupe plusieurs entrées en une seule sortie (voir Transaction regroupant des fonds ci-dessous). Cela représente l'équivalent réel de l' échange d'une pile de pièces de monnaie et de billets de banque contre un seul billet plus grand. Des transactions comme celles-ci sont parfois générées par des applications de portefeuille pour nettoyer de nombreux montants plus petits qui ont été reçus en échange de paiements.

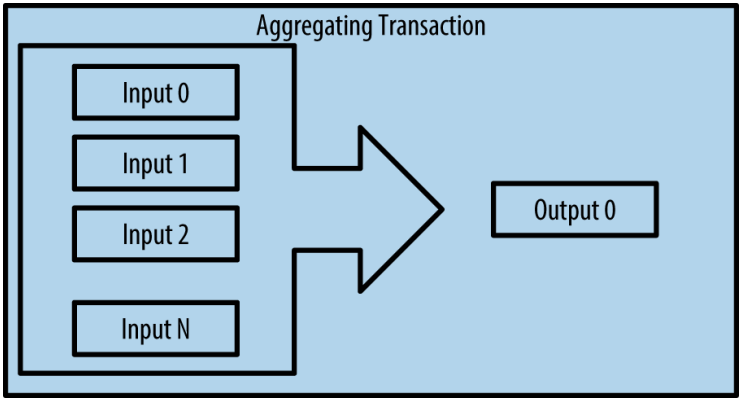


Figure 6. Agrégation des fonds de transaction

Enfin, une autre forme de transaction que l'on voit souvent sur le grand livre bitcoin est une transaction par lots, qui distribue une entrée à plusieurs sorties représentant plusieurs destinataires, une technique appelée « transaction par lots » (voir Transaction distribuant des fonds ci-dessous). Étant donné que ce type de transaction est utile pour économiser les frais de transaction, il est couramment utilisé par les entités commerciales pour distribuer des fonds, par exemple lorsqu'une entreprise traite les paiements de paie à plusieurs employés ou lorsqu'un échange de bitcoins traite les retraits de plusieurs clients en un seul transaction.

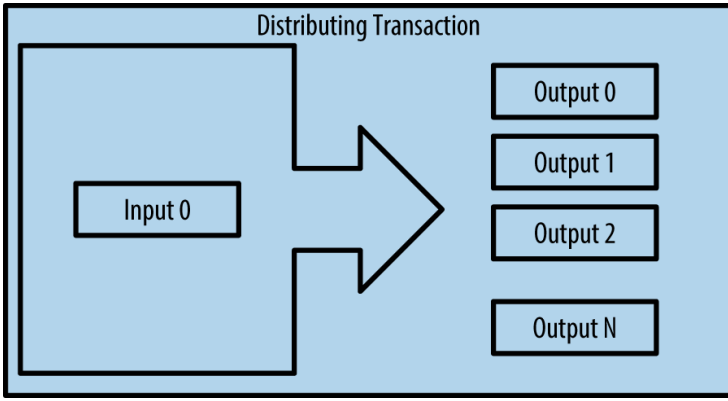


Figure 7. Transaction de distribution de fonds

Construire une transaction

L'application de portefeuille d'Alice contient toute la logique pour sélectionner les entrées et les sorties appropriées pour construire une transaction selon les spécifications d'Alice. Alice a seulement besoin de spécifier une destination et un montant, et le reste se passe dans l'application portefeuille sans qu'elle en voie les détails. Il est important de noter qu'une application de portefeuille peut créer des transactions même si elle est complètement hors ligne. Comme pour écrire un chèque à la maison et l'envoyer plus tard à la banque dans une enveloppe, la transaction n'a pas besoin d'être construite et signée lorsqu'elle est connectée au réseau Bitcoin.

Obtenir les bonnes entrées

L'application de portefeuille d'Alice devra d'abord trouver des entrées permettant de payer le montant qu'elle souhaite envoyer à Bob. La plupart des portefeuilles gardent une trace de toutes les sorties disponibles appartenant aux adresses du portefeuille. Par conséquent, le portefeuille d'Alice contiendrait une copie de la sortie de la transaction de la transaction de Joe, qui a été créée en échange d'argent liquide. Une application de portefeuille Bitcoin qui s'exécute en tant que client à nœud complet contient en fait une copie de chaque sortie non dépensée de chaque transaction dans la blockchain. Cela permet à un portefeuille de créer des entrées de transaction et de vérifier rapidement que les transactions entrantes ont des entrées correctes. Cependant, comme un client à nœud complet occupe beaucoup d'espace disque, la plupart des portefeuilles utilisateur exécutent des clients « légers » qui ne suivent que les sorties non dépensées de l'utilisateur.

Si l'application de portefeuille ne conserve pas une copie des sorties de transaction non dépensées, elle peut interroger le réseau Bitcoin pour récupérer ces informations à l'aide d'une variété d'API disponibles par différents fournisseurs ou en demandant à un nœud complet à l' aide d'un appel d'interface de programmation d'application (API). Rechercher toutes les sorties non dépensées pour l'adresse bitcoin d'Alice (schemas ci-dessus) montre une requête API, construite comme une commande HTTP GET vers une URL spécifique. Cette URL renverra toutes les sorties de transaction non dépensées pour une adresse, donnant à toute application les informations dont elle a besoin pour construire des entrées de transaction pour les dépenses. Nous utilisons le simple client HTTP en ligne de commande *cURL* pour récupérer la réponse.

Exemple 2. Recherchez toutes les sorties non dépensées pour l'adresse bitcoin d'Alice

```
$ curl https://blockchain.info/unspent?active=
1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK

 {

"unspent_outputs":[

       {

               "tx_hash":"186f9f998a5...2836dd734d2804fe65fa35779

                "tx_index":104810202,

                 "tx_output_n": 0,

         "script":"76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8

                 "value": 10000000,

                 "value_hex": "00989680",

                 "confirmations":0

        }
  ]
}
```

La réponse dans Rechercher toutes les sorties non dépensées pour l'adresse bitcoin d'Alice montre une sortie non dépensée (une qui n'a pas encore été utilisée) sous la propriété de l'adresse d'Alice 1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK. La ré-

ponse comprend la référence à la transaction dans laquelle cette sortie non dépensée est contenue (le paiement de Joe) et sa valeur en satoshis , à 10 millions, équivalent à 0,10 bitcoin. Avec ces informations, l'application de portefeuille d'Alice peut construire une transaction pour transférer cette valeur à de nouvelles adresses de propriétaire.

Conseil
Visualisez la transaction de Joe à Alice .

Comme vous pouvez le voir, le portefeuille d'Alice contient suffisamment de bitcoin dans une seule sortie non dépensée pour payer la tasse de café. Si cela n'avait pas été le cas, l'application de portefeuille d'Alice devra peut-être "fouiller" dans une pile de petites sorties non dépensées, comme ramasser des pièces dans un sac à main jusqu'à ce qu'elle puisse en trouver suffisamment pour payer le café. Dans les deux cas, il peut être nécessaire de récupérer des modifications, ce que nous verrons dans la section suivante, car l'application de portefeuille crée les sorties de transaction (paiements).

Création des sorties

Une sortie de transaction est créée sous la forme d'un script qui crée un engagement sur la valeur et ne peut être utilisée que par l'introduction d'une solution dans le script. En termes plus simples, la sortie de transaction d'Alice contiendra un script qui dit quelque chose comme, "Cette sortie est payable à quiconque peut présenter une signature de la clé correspondant

à l'adresse de Bob." Parce que seul Bob a le portefeuille avec les clés correspondant à cette adresse, seul le portefeuille de Bob peut présenter une telle signature pour utiliser cette sortie. Alice va donc "encombrer" la valeur de sortie avec une demande de signature de Bob.

Cette transaction comprendra également une deuxième sortie, car les fonds d'Alice se présentent sous la forme d'une sortie de 0,10 BTC, trop d'argent pour la tasse de café 0,015 BTC. Alice aura besoin de 0,085 BTC en changement. Le paiement de change d'Alice est créé par le portefeuille d'Alice en tant que sortie dans la même transaction que le paiement à Bob. Essentiellement, le portefeuille d'Alice divise ses fonds en deux paiements : un à Bob et un à elle-même. Elle peut ensuite utiliser (dépenser) la sortie de modification dans une transaction ultérieure.

Enfin, pour que la transaction soit traitée par le réseau en temps opportun, l'application de portefeuille d'Alice ajoutera une somme modique. Ce n'est pas explicite dans la transaction ; il est impliqué par la différence entre les entrées et les sorties. Si au lieu de prendre 0,085 en changement, Alice ne crée que 0,0845 comme deuxième sortie, il restera 0,0005 BTC (un demi-millibitcoin). Les 0,10 BTC de l'entrée ne sont pas entièrement dépensés avec les deux sorties, car elles totaliseront moins de 0,10. La différence qui en résulte est les *frais de transaction* qui sont perçus par le mineur en tant que frais de validation et d'inclusion de la transaction dans un bloc à enregistrer sur la blockchain.

La transaction qui en résulte peut être visualisée à l'aide d'une

application Web d'explorateur de chaînes de blocs, comme indiqué dans la transaction d'Alice à Bob's Cafe (schema sur la transaction plus haut).

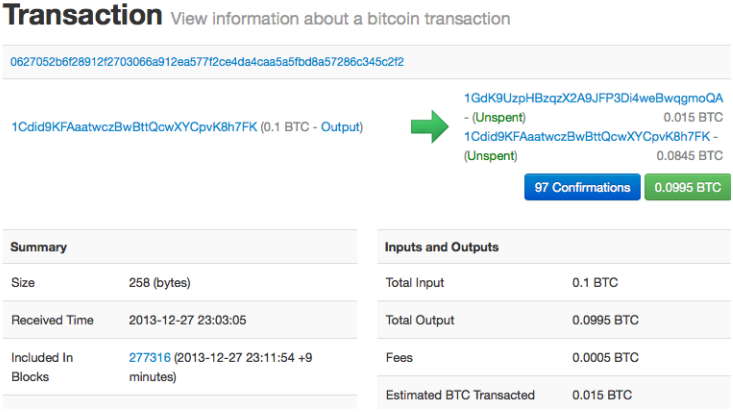


Figure 8. Transaction d'Alice à Bob's Cafe

> *Conseil*
>
> *Visualisez la transaction d'Alice dans le restaurant Bob's Cafe .*

Ajout de la transaction au Ledger

La transaction créée par l'application de portefeuille d'Alice est longue de 258 octets et contient tout le nécessaire pour confirmer la propriété des fonds et affecter de nouveaux propriétaires. Désormais, la transaction doit être transmise

au réseau bitcoin où elle fera partie de la blockchain. Dans la section suivante, nous verrons comment une transaction fait partie d'un nouveau bloc et comment le bloc est « extrait ». Enfin, nous verrons comment le nouveau bloc, une fois ajouté à la blockchain, est de plus en plus reconnu par le réseau à mesure que de nouveaux blocs sont ajoutés.

Transmettre la transaction

Étant donné que la transaction contient toutes les informations nécessaires à traiter, peu importe comment et où elle est transmise au réseau Bitcoin. Le réseau Bitcoin est un réseau peer-to-peer, chaque client Bitcoin participant en se connectant à plusieurs autres clients Bitcoin. Le but du réseau Bitcoin est de propager les transactions et les blocs à tous les participants.

Comment ça se propage

Tout système, tel qu'un serveur, une application de bureau ou un portefeuille, qui participe au réseau bitcoin en "parlant" le protocole bitcoin est appelé un *nœud bitcoin* . L'application de portefeuille d'Alice peut envoyer la nouvelle transaction à n'importe quel nœud Bitcoin auquel elle est connectée via n'importe quel type de connexion : filaire, WiFi , mobile, etc. Son portefeuille Bitcoin n'a pas besoin d'être connecté directement au portefeuille Bitcoin de Bob et elle n'a pas à le faire. utilisez la connexion Internet offerte par le café, bien que ces deux options soient également possibles. Tout nœud bitcoin qui reçoit une transaction valide qu'il n'a pas vue auparavant la transmettra immédiatement à tous les autres nœuds auxquels il est connecté, une technique de propagation connue sous le nom d' *inondation* . Ainsi, la transaction se propage rapidement à travers le réseau peer-to-peer, atteignant un grand pourcentage

des nœuds en quelques secondes.

Le point de vue de Bob

Si l'application de portefeuille bitcoin de Bob est directement connectée à l'application de portefeuille d'Alice, l'application de portefeuille de Bob pourrait être le premier nœud à recevoir la transaction. Cependant, même si le portefeuille d'Alice envoie la transaction via d'autres nœuds, il atteindra le portefeuille de Bob en quelques secondes. Le portefeuille de Bob identifiera immédiatement la transaction d'Alice comme un paiement entrant car il contient des sorties échangeables par les clés de Bob. L'application de portefeuille de Bob peut également vérifier indépendamment que la transaction est bien formée, utilise des sorties précédemment non dépensées et contient des frais de transaction suffisants pour être incluse dans le bloc suivant. À ce stade, Bob peut supposer, avec peu de risques, que la transaction sera bientôt incluse dans un bloc et confirmée.

Conseil

Une idée fausse commune sur les transactions Bitcoin est qu'elles doivent être « confirmées » en attendant 10 minutes pour un nouveau bloc, ou jusqu'à 60 minutes pour six confirmations complètes. Bien que les confirmations garantissent que la transaction a été acceptée par l'ensemble du réseau, un tel délai n'est pas nécessaire pour les articles de petite valeur comme une tasse de café. Un commerçant peut accepter une transaction valide de petite valeur sans confirmation, avec pas plus de risque qu'un paiement par carte de crédit effectué sans pièce d'identité ni signature, comme les commerçants acceptent

couramment aujourd'hui.

Extraction de Bitcoin

La transaction d'Alice est maintenant propagée sur le réseau bitcoin. Il ne fait pas partie de la *blockchain* tant qu'il n'a pas été vérifié et inclus dans un bloc par un processus appelé *minage* . Voir [Chapitre sur le Mining] pour une explication détaillée.

Le système de confiance Bitcoin est basé sur le calcul. Les transactions sont regroupées en *blocs* , ce qui nécessite une énorme quantité de calcul pour être prouvée , mais seulement une petite quantité de calcul pour vérifier comme prouvé. Le processus d'extraction a deux objectifs en bitcoin :

- Les nœuds de minage valident toutes les transactions en se référant aux *règles de consensus* de Bitcoin . Par conséquent, le minage assure la sécurité des transactions Bitcoin en rejetant les transactions invalides ou mal formées.

- L'exploitation minière crée de nouveaux bitcoins dans chaque bloc, presque comme une banque centrale imprimant de l'argent neuf. La quantité de bitcoin créée par bloc est limitée et diminue avec le temps, suivant un calendrier d'émission fixe.

L'exploitation minière réalise un juste équilibre entre le coût

et la récompense. L'exploitation minière utilise l'électricité pour résoudre un problème mathématique. Un mineur qui réussit recevra une *récompense* sous la forme de nouveaux frais de bitcoin et de transaction. Cependant, la récompense ne sera collectée que si le mineur a correctement validé toutes les transactions, à la satisfaction des règles du *consensus* . Cet équilibre délicat assure la sécurité du bitcoin sans autorité centrale.

Une bonne façon de décrire l'exploitation minière est comme un jeu de sudoku compétitif géant qui se réinitialise chaque fois que quelqu'un trouve une solution et dont la difficulté s'ajuste automatiquement de sorte qu'il faut environ 10 minutes pour trouver une solution. Imaginez un puzzle sudoku géant de plusieurs milliers de lignes et de colonnes. Si je vous montre un puzzle terminé, vous pouvez le vérifier assez rapidement. Cependant, si le puzzle a quelques carrés remplis et que les autres sont vides, il faut beaucoup de travail pour le résoudre ! La difficulté du sudoku peut être ajustée en changeant sa taille (plus ou moins de lignes et de colonnes), mais elle peut toujours être vérifiée assez facilement même si elle est très grande. Le "puzzle" utilisé dans le bitcoin est basé sur un hachage cryptographique et présente des caractéristiques similaires : il est asymétriquement difficile à résoudre mais facile à vérifier, et sa difficulté peut être ajustée.

Dans [L'histoire des utilisateurs, chapitre 1] , nous avons présenté Jing, un entrepreneur de Shanghai. Jing dirige une *ferme minière* , qui est une entreprise qui gère des milliers d'ordinateurs miniers spécialisés, en compétition pour la récompense. Toutes les 10 minutes environ, les ordinateurs miniers de

Jing rivalisent avec des milliers de systèmes similaires dans une course mondiale pour trouver une solution à un bloc de transactions. La recherche d'une telle solution, la soi-disant *Proof-of-Work* (PoW), nécessite des quadrillions d'opérations de hachage par seconde sur l'ensemble du réseau Bitcoin. L'algorithme de preuve de travail consiste à hacher à plusieurs reprises l'en-tête du bloc et un nombre aléatoire avec l'algorithme cryptographique SHA256 jusqu'à ce qu'une solution correspondant à un modèle prédéterminé émerge. Le premier mineur à trouver une telle solution remporte le tour de la concurrence et publie ce bloc dans la blockchain.

Jing a commencé l'exploitation minière en 2010 en utilisant un ordinateur de bureau très rapide pour trouver une preuve de travail appropriée pour les nouveaux blocs. Alors que de plus en plus de mineurs ont commencé à rejoindre le réseau Bitcoin, la difficulté du problème a augmenté rapidement. Bientôt, Jing et d'autres mineurs sont passés à du matériel plus spécialisé, avec des unités de traitement graphique (GPU) dédiées haut de gamme, souvent utilisées dans les ordinateurs de bureau ou les consoles de jeu. Au moment d'écrire ces lignes, la difficulté est si élevée qu'il n'est rentable d'exploiter qu'avec des circuits intégrés spécifiques à l'application (ASIC), essentiellement des centaines d'algorithmes de minage imprimés en matériel, fonctionnant en parallèle sur une seule puce de silicium. La société de Jing participe également à un *pool minier* , qui, tout comme un pool de loterie, permet à plusieurs participants de partager leurs efforts et leurs récompenses. La société de Jing gère désormais un entrepôt contenant des milliers de mineurs ASIC à exploiter pour Bitcoin 24 heures par jour. La société paie ses frais d'électricité en vendant le bitcoin qu'elle est capable

de générer à partir de l'exploitation minière, créant ainsi des revenus à partir des bénéfices.

Opérations minières en blocs

De nouvelles transactions affluent constamment dans le réseau à partir de portefeuilles d'utilisateurs et d'autres applications. Comme ceux-ci sont vus par les nœuds du réseau bitcoin, ils sont ajoutés à un pool temporaire de transactions non vérifiées maintenues par chaque nœud. Au fur et à mesure que les mineurs construisent un nouveau bloc, ils ajoutent des transactions non vérifiées de ce pool au nouveau bloc, puis tentent de prouver la validité de ce nouveau bloc, avec l'algorithme de minage (Proof-of-Work). Le processus d'exploitation minière est expliqué en détail dans [le chapitre sur l'exploitation minière] .

Les transactions sont ajoutées au nouveau bloc, priorisées par les transactions les plus coûteuses en premier et par quelques autres critères. Chaque mineur commence le processus d'extraction d'un nouveau bloc de transactions dès qu'il reçoit le bloc précédent du réseau, sachant qu'il a perdu ce tour de compétition précédent. Ils créent immédiatement un nouveau bloc, le remplissent avec les transactions et l'empreinte digitale du bloc précédent, et commencent à calculer la preuve de travail pour le nouveau bloc. Chaque mineur inclut une transaction spéciale dans son bloc, une qui paie à sa propre adresse bitcoin la récompense de bloc (actuellement 6,25 bitcoin nouvellement créé) plus la somme des frais de transaction

de toutes les transactions incluses dans le bloc. S'ils trouvent une solution qui rend ce bloc valide, ils « gagnent » cette récompense parce que leur bloc réussi est ajouté à la blockchain mondiale et la transaction de récompense qu'ils ont incluse devient dépensable. Jing, qui participe à un pool de minage, a mis en place son logiciel pour créer de nouveaux blocs qui attribuent la récompense à une adresse de pool. De là, une part de la récompense est distribuée à Jing et aux autres mineurs proportionnellement à la quantité de travail qu'ils ont contribué au dernier tour.

La transaction d'Alice a été récupérée par le réseau et incluse dans le pool de transactions non vérifiées. Une fois validé par le logiciel de minage, il a été inclus dans un nouveau bloc, appelé *bloc candidat* , généré par le pool de minage de Jing. Tous les mineurs participant à ce pool de minage commencent immédiatement à calculer la preuve de travail pour le bloc candidat. Environ cinq minutes après la première transmission de la transaction par le portefeuille d'Alice, l'un des mineurs ASIC de Jing a trouvé une solution pour le bloc candidat et l'a annoncé au réseau. Une fois que les autres mineurs ont validé le bloc gagnant , ils ont commencé la course pour générer le bloc suivant.

Le bloc gagnant de Jing est devenu une partie de la blockchain en tant que bloc # 277316, contenant 419 transactions, y compris la transaction d'Alice. Le bloc contenant la transaction d'Alice est compté comme une "confirmation" de cette transaction.

Conseil

Vous pouvez voir le bloc qui inclut la transaction d'Alice

.

Environ 20 minutes plus tard, un nouveau bloc, # 277317, est exploité par un autre mineur. Parce que ce nouveau bloc est construit au-dessus du bloc # 277316 qui contenait la transaction d'Alice, il a ajouté encore plus de calcul à la blockchain, renforçant ainsi la confiance dans ces transactions. Chaque bloc extrait au-dessus de celui contenant la transaction compte comme une confirmation supplémentaire pour la transaction d'Alice. Au fur et à mesure que les blocs s'empilent les uns sur les autres, il devient de plus en plus difficile d'inverser la transaction, ce qui la rend de plus en plus fiable par le réseau.

Dans le diagramme ci-dessous de la transaction d'Alice inclus dans le bloc # 277316 , nous pouvons voir le bloc # 277316, qui contient la transaction d'Alice. En dessous se trouvent 277316 blocs (y compris le bloc n ° 0), liés les uns aux autres dans une chaîne de blocs (blockchain) jusqu'au bloc n ° 0, connu sous le nom de *bloc de genèse* . Au fil du temps, à mesure que la "hauteur" des blocs augmente, la difficulté de calcul augmente pour chaque bloc et la chaîne dans son ensemble . Les blocs extraits après celui qui contient la transaction d'Alice agissent comme une assurance supplémentaire, car ils empilent plus de calculs dans une chaîne de plus en plus longue. Par convention, tout bloc avec plus de six confirmations est considéré comme irrévocable, car il faudrait une énorme quantité de calculs pour invalider et recalculer six blocs. Nous examinerons plus en détail le processus de minage et la façon dont il renforce la confiance dans [chapitre sur le minage] .

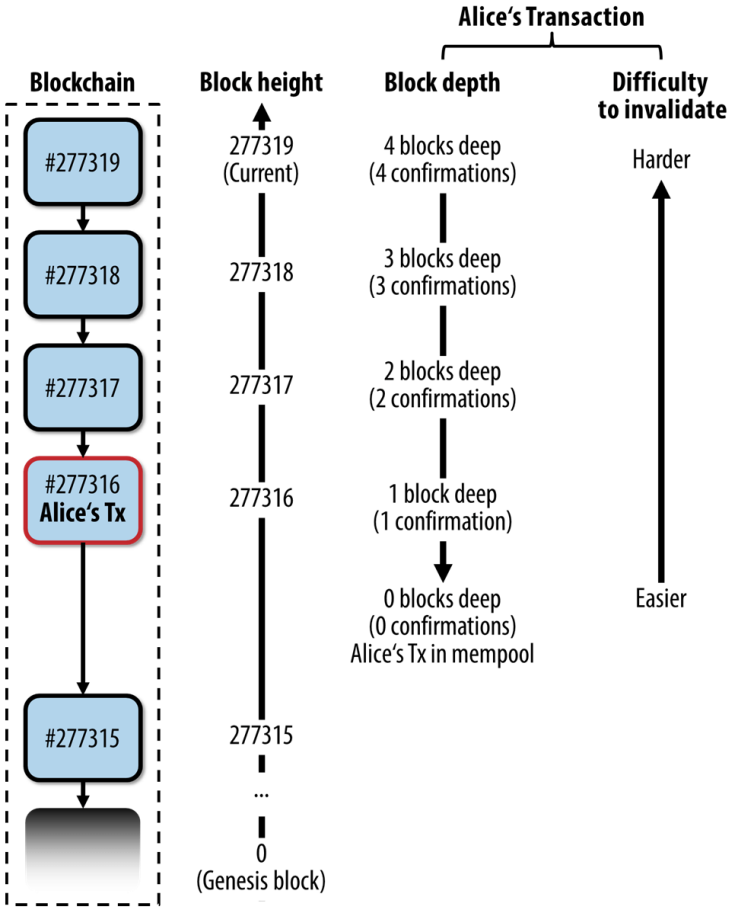


Figure 9. Transaction d'Alice incluse dans le bloc # 277316

Dépenser la transaction

Maintenant que la transaction d'Alice a été intégrée dans la blockchain dans le cadre d'un bloc, elle fait partie du grand livre distribué de bitcoin et visible par toutes les applications bitcoin. Chaque client Bitcoin peut vérifier indépendamment la transaction comme valide et utilisable. Les clients à nœud complet peuvent suivre la source des fonds à partir du moment où le bitcoin a été généré pour la première fois dans un bloc, de manière incrémentielle d'une transaction à l'autre, jusqu'à ce qu'ils atteignent l'adresse de Bob. Les clients légers peuvent faire ce qu'on appelle une vérification de paiement simplifiée en confirmant que la transaction est dans la blockchain et que plusieurs blocs sont extraits après elle, fournissant ainsi l'assurance que les mineurs l'ont acceptée comme valide.

Bob peut maintenant dépenser la sortie de cette transaction et d'autres transactions. Par exemple, Bob peut payer un entrepreneur ou un fournisseur en transférant la valeur du paiement de la tasse de café d'Alice à ces nouveaux propriétaires. Très probablement, le logiciel Bitcoin de Bob regroupera de nombreux petits paiements en un paiement plus important, concentrant peut-être tous les revenus Bitcoin de la journée en une seule transaction. Cela regrouperait les différents paiements en un seul produit (et une seule adresse). Pour obtenir un diagramme d'une transaction d'agrégation, voir Transaction d'agrégation de fonds (schema plus haut).

Au fur et à mesure que Bob dépense les paiements reçus d'Alice et d'autres clients, il étend la chaîne des transactions.

Allons supposer que Bob paie son concepteur web Gopesh à Bangalore pour une nouvelle page Web. Désormais, la chaîne de transactions ressemblera à la transaction d'Alice dans le cadre d'une chaîne de transactions de Joe à Gopesh , où la sortie d'une transaction est dépensée comme entrée de la transaction suivante (schema ci-dessous).

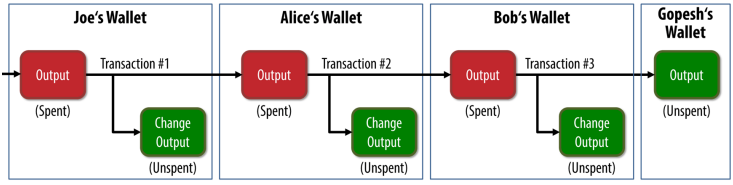


Figure 10. La transaction d'Alice dans le cadre d'une chaîne de transaction de Joe à Gopesh , où la sortie d'une transaction est dépensée comme entrée de la transaction suivante

Dans ce chapitre, nous avons vu comment les transactions construisent une chaîne qui déplace la valeur d'un propriétaire à l'autre. Nous avons également suivi la transaction d'Alice, à partir du moment où elle a été créée dans son portefeuille, via le réseau bitcoin et jusqu'aux mineurs qui l'ont enregistrée sur la blockchain. Dans le reste de ce livre, nous examinerons les technologies spécifiques derrière les portefeuilles, les adresses, les signatures, les transactions, le réseau et enfin l'exploitation minière.

4

Bitcoin Core : l'implémentation de référence

Bitcoin est un projet *open source* et le code source est disponible sous une licence ouverte (MIT), gratuit à télécharger et à utiliser dans n'importe quel but. L'open source signifie plus que la simple utilisation gratuite. Cela signifie également que le bitcoin est développé par une communauté ouverte de bénévoles. Au début, cette communauté se composait uniquement de Satoshi Nakamoto. En 2016, le code source de Bitcoin comptait plus de 400 contributeurs avec une douzaine de développeurs travaillant sur le code presque à plein temps et plusieurs dizaines d'autres à temps partiel. Tout le monde peut contribuer au code, y compris vous !

Lorsque le bitcoin a été créé par Satoshi Nakamoto, le logiciel était en fait terminé avant que le livre blanc reproduit dans [satoshi_whitepaper] ne soit écrit. Satoshi voulait s'assurer que cela fonctionnait avant d'écrire à ce sujet. Cette première implémentation, alors simplement connue sous le nom de « Bitcoin »

ou « client Satoshi », a été fortement modifiée et améliorée. Il a évolué pour devenir ce que l'on appelle *Bitcoin Core* , pour le différencier des autres implémentations compatibles. Bitcoin Core est l' *implémentation de référence* du système Bitcoin, ce qui signifie qu'il est la référence faisant autorité sur la façon dont chaque partie de la technologie doit être mise en œuvre. Bitcoin Core implémente tous les aspects du bitcoin, y compris les portefeuilles, un moteur de validation de transaction et de bloc et un nœud de réseau complet dans le réseau bitcoin peer-to-peer.

Avertissement

Même si Bitcoin Core inclut une implémentation de référence d'un portefeuille, celui-ci n'est pas destiné à être utilisé comme portefeuille de production pour les utilisateurs ou pour les applications. Les développeurs d'applications sont invités à construire des portefeuilles en utilisant des normes modernes, comme PIF-39 et 32-PIF. BIP signifie Bitcoin Improvement Proposal .

L'architecture Bitcoin Core (Source : Eric Lombrozo) montre l'architecture de Bitcoin Core.

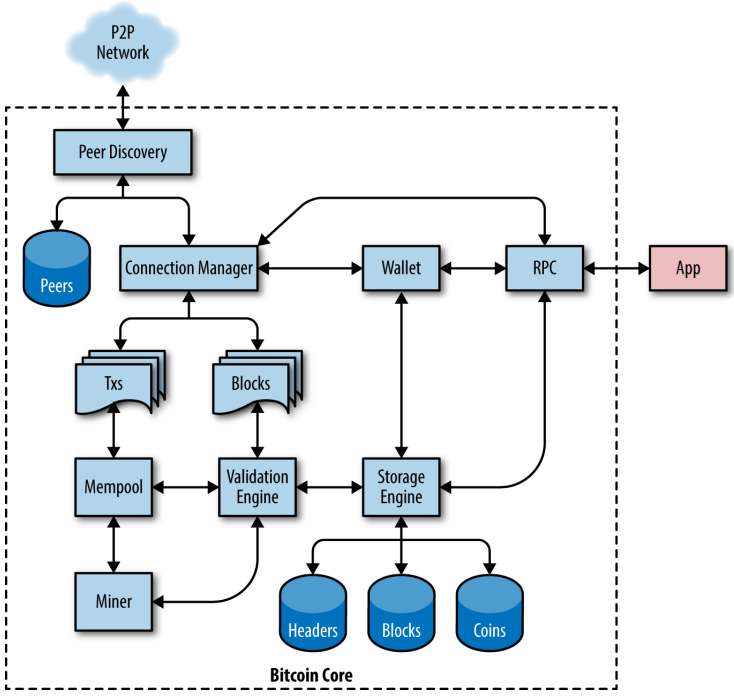


Figure 1. Architecture Bitcoin Core (Source : Eric Lombrozo)

Environnement de développement Bitcoin

Si vous êtes un développeur, vous voudrez mettre en place un environnement de développement avec tous les outils, bibliothèques et logiciels de support pour l'écriture d'applications Bitcoin. Dans ce chapitre hautement technique, nous allons parcourir ce processus étape par étape. Si le matériel devient trop dense (et que vous ne configurez pas réellement d'environnement de développement), n'hésitez pas à passer au chapitre suivant, qui est moins technique.

Compiler Bitcoin Core à partir du code source

Le code source de Bitcoin Core peut être téléchargé en tant qu'archive ou en clonant le référentiel source faisant autorité depuis GitHub. Sur la page de téléchargement de Bitcoin Core , sélectionnez la version la plus récente et téléchargez l'archive compressée du code source, par exemple bitcoin-0.15.0.2.tar.gz. Vous pouvez également utiliser la ligne de commande git pour créer une copie locale du code source à partir de la page bitcoin GitHub .

> *Conseil*
>
> *Dans de nombreux exemples de ce chapitre, nous utiliserons l'interface de ligne de commande du système d'exploitation (également appelée « shell »), accessible via une application « terminal ». Le shell affichera une invite ; vous tapez une commande ; et le shell répond avec du texte et une nouvelle invite pour votre prochaine commande. L'invite peut sembler différente sur votre système, mais dans les exemples suivants, elle est indiquée par un symbole $. Dans les exemples, lorsque vous voyez du texte après un symbole $, ne tapez pas le symbole $ mais tapez la commande immédiatement après, puis appuyez sur Entrée pour exécuter la commande. Dans les exemples, les lignes sous chaque commande sont les réponses du système d'exploitation à cette commande. Lorsque vous voyez le prochain préfixe $, vous saurez que c'est une nouvelle commande et vous devriez répéter le processus.*

Dans cet exemple, nous utilisons la commande git pour créer

une copie locale (“clone”) du code source :

```
$ git clone https://github.com/bitcoin/bitcoin.git
Clonage dans 'bitcoin' ...
remote: comptage d'objets: 102071, terminé.
remote: Compression d'objets: 100% (10/10), terminé.
Réception d'objets: 100% (102071/102071), 86,38 Mio |
730,00 Kio / s, terminé.
remote: Total 102071 (delta 4), réutilisé 5 (delta
1), pack-réutilisé 102060
Résolution des deltas: 100% (76168/76168), terminé.
Vérification de la connectivité ... terminé.
$
```

Conseil

Git est le système de contrôle de version distribué le plus largement utilisé, une partie essentielle de la boîte à outils de tout développeur de logiciel. Vous devrez peut-être installer la commande git, ou une interface utilisateur graphique pour git, sur votre système d’exploitation si vous ne l’avez pas déjà.

Une fois l’opération de clonage git terminée, vous aurez une copie locale complète du référentiel de code source dans le répertoire *bitcoin* . Accédez à ce répertoire en tapant ** cd bitcoin ** à l’invite :

```
$ cd bitcoin
```

Sélection d'une version Bitcoin Core

Par défaut, la copie locale sera synchronisée avec le code le plus récent, qui peut être une version instable ou bêta de Bitcoin. Avant de compiler le code, sélectionnez une version spécifique en extrayant une *balise de* version . Cela synchronisera la copie locale avec un instantané spécifique du référentiel de code identifié par une balise de mot-clé. Les balises sont utilisées par les développeurs pour marquer des versions spécifiques du code par numéro de version. Tout d'abord, pour trouver les balises disponibles, nous utilisons la commande git tag :

```
$ balise git
v0.1.5
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
v0.12.0rc1
v0.12.0rc2
...
```

La liste des balises montre toutes les versions publiées de Bitcoin. Par convention, les *versions candidates* , qui sont destinées à être testées, ont le suffixe « rc ». Les versions stables qui peuvent être exécutées sur des systèmes de production n'ont pas de suffixe. Dans la liste précédente, sélectionnez la version la plus élevée, qui au moment de la rédaction était v0.15.0. Pour synchroniser le code local avec cette version, utilisez la

commande git checkout :

```
$ git checkout v0.15.0
HEAD est maintenant à 3751912 ... Merge # 11295: doc:
Les anciens fee_estimates.dat sont supprimés par
0.15.0
```

Vous pouvez confirmer que vous avez la version souhaitée "extraite" en exécutant la commande git status :

```
$ git status
HEAD détaché à la v0.15.0
rien à valider, le répertoire de travail est nettoyé
```

Configuration de la version Bitcoin Core

Le code source comprend de la documentation, qui peut être trouvée dans un certain nombre de fichiers. Consultez la documentation principale située dans *README.md* dans le répertoire *bitcoin* en tapant ** more README.md ** à l'invite et en utilisant la barre d'espace pour passer à la page suivante. Dans ce chapitre, nous allons créer le client bitcoin en ligne de commande, également appelé bitcoind sous Linux. Passez en revue les instructions de compilation du client de ligne de commande bitcoind sur votre plate-forme en tapant ** more doc / build-unix.md **. Des instructions alternatives pour macOS et Windows peuvent être trouvées dans le répertoire

doc , comme *build-osx.md* ou *build-windows.md* , respectivement.

Examinez attentivement les conditions préalables à la génération, qui se trouvent dans la première partie de la documentation de génération. Ce sont des bibliothèques qui doivent être présentes sur votre système avant de pouvoir commencer à compiler Bitcoin. Si ces conditions préalables sont manquantes, le processus de génération échouera avec une erreur. Si cela se produit parce que vous avez manqué un prérequis, vous pouvez l'installer, puis reprendre le processus de génération là où vous vous étiez arrêté. En supposant que les prérequis sont installés, vous démarrez le processus de génération en générant un ensemble de scripts de génération à l'aide du script *autogen.sh* .

```
$ ./autogen.sh
...
glibtoolize: copie du fichier 'build-aux / m4 /
libtool.m4'
glibtoolize: copie du fichier 'build-aux / m4 /
ltoptions.m4'
glibtoolize: copie du fichier 'build-aux / m4 /
ltsugar.m4'
glibtoolize: copie du fichier 'build-aux / m4 /
ltversion.m4'
...
configure.ac:10: installation de 'build-aux / compile'
configure.ac:5: installation de 'build-aux /
config.guess'
configure.ac:5: installation de 'build-aux /
config.sub'
configure.ac:9: installation de 'build-aux /
```

```
install-sh'
configure.ac:9: installation de 'build-aux / missing'
Makefile.am: installation de 'build-aux / depcomp'
...
```

Le script *autogen.sh* crée un ensemble de scripts de configuration automatique qui interrogeront votre système pour découvrir les paramètres corrects et garantir que vous disposez de toutes les bibliothèques nécessaires pour compiler le code. Le plus important d'entre eux est le script de configuration qui offre un certain nombre d'options différentes pour personnaliser le processus de construction. Tapez **. / Configure —help ** pour voir les différentes options :

```
$ ./configure - aide
`configure 'configure Bitcoin Core 0.15.0 pour
s'adapter à de nombreux types de systèmes.

Utilisation: ./configure [OPTION] ... [VAR = VALUE]
...

...
Caractéristiques optionnelles:
 --disable-option-checking ignore unecognized
 --enable / - avec des options
 --disable-FEATURE n'inclut pas FEATURE (identique à
 --enable-FEATURE = no)
 --enable-FEATURE [= ARG] include FEATURE [ARG = yes]

 --enable-wallet enable wallet (la valeur par défaut
 est oui)
```

```
 --with-gui [= no | qt4 | qt5 | auto]
...
```

Le script de configuration vous permet d'activer ou de désactiver certaines fonctionnalités de bitcoind via l'utilisation des indicateurs —enable-FEATURE et —disable-FEATURE, où FEATURE est remplacé par le nom de la fonction, comme indiqué dans la sortie de l'aide. Dans ce chapitre, nous allons créer le client bitcoind avec toutes les fonctionnalités par défaut. Nous n'utiliserons pas les indicateurs de configuration, mais vous devriez les examiner pour comprendre quelles fonctionnalités facultatives font partie du client. Si vous êtes dans un milieu universitaire, les restrictions de laboratoire informatique peuvent vous obliger à installer des applications dans votre répertoire personnel (par exemple, en utilisant —prefix = $ HOME).

Voici quelques options utiles qui remplacent le comportement par défaut du script de configuration :

```
--prefix = $ HOME
```

Cela remplace l'emplacement d'installation par défaut (qui est */ usr / local /*) pour l'exécutable résultant. Utilisez $ HOME pour tout mettre dans votre répertoire personnel, ou dans un chemin différent.

```
--disable-wallet
```

Ceci est utilisé pour désactiver l'implémentation du portefeuille de référence.

```
--with-incompatible-bdb
```

Si vous créez un portefeuille, autorisez l'utilisation d'une version incompatible de la bibliothèque Berkeley DB.

```
--with-gui = non
```

Ne construisez pas l'interface utilisateur graphique, qui nécessite la bibliothèque Qt. Cela crée uniquement des bitcoins de serveur et de ligne de commande.

Ensuite, exécutez le script de configuration pour découvrir automatiquement toutes les bibliothèques nécessaires et créer un script de construction personnalisé pour votre système :

```
$ ./configure
vérification du type de système de construction ...
x86_64-unknown-linux-gnu
vérification du type de système hôte ...
x86_64-unknown-linux-gnu
recherche d'une installation compatible BSD ... / usr
/ bin / install -c
vérifier si l'environnement de construction est sain
... oui
```

```
recherche d'un mkdir -p ... / bin / mkdir -p sécurisé
pour les threads
recherche de gawk ... gawk
vérifier si faire des ensembles $ (MAKE) ... oui
...
[de nombreuses pages de tests de configuration
suivent]
...
$
```

Si tout s'est bien passé, la commande configure se terminera par la création des scripts de construction personnalisés qui nous permettront de compiler bitcoind. S'il manque des bibliothèques ou des erreurs, la commande configure se terminera par une erreur au lieu de créer les scripts de construction. Si une erreur se produit, elle est probablement due à une bibliothèque manquante ou incompatible. Consultez à nouveau la documentation de compilation et assurez-vous d'installer les prérequis manquants. Ensuite, exécutez à nouveau configure et voyez si cela corrige l'erreur.

Construire les exécutables Bitcoin Core

Ensuite, vous compilerez le code source, un processus qui peut prendre jusqu'à une heure, en fonction de la vitesse de votre CPU et de la mémoire disponible. Pendant le processus de compilation, vous devriez voir une sortie toutes les quelques secondes ou toutes les quelques minutes, ou une erreur si quelque chose ne va pas. Si une erreur se produit ou si le processus de compilation est interrompu, il peut être repris

à tout moment en tapant à nouveau make. Tapez ** make ** pour commencer la compilation de l'application exécutable :

```
$ make
Faire tout dans src
 CXX crypto / libbitcoinconsensus_la-hmac_sha512.lo
 CXX crypto / libbitcoinconsensus_la-ripemd160.lo
 CXX crypto / libbitcoinconsensus_la-sha1.lo
 CXX crypto / libbitcoinconsensus_la-sha256.lo
 CXX crypto / libbitcoinconsensus_la-sha512.lo
 CXX libbitcoinconsensus_la-hash.lo
 Primitives CXX /
 libbitcoinconsensus_la-transaction.lo
 CXX libbitcoinconsensus_la-pubkey.lo
 Script CXX /
 libbitcoinconsensus_la-bitcoinconsensus.lo
 Script CXX / libbitcoinconsensus_la-interpreter.lo

[... de nombreux autres messages de compilation
suivent ...]

$
```

Sur un système rapide avec plusieurs processeurs, vous souhaiterez peut-être définir le nombre de travaux de compilation parallèles. Par exemple, make -j 2 utilisera deux cœurs s'ils sont disponibles. Si tout se passe bien, Bitcoin Core est désormais compilé. Vous devez exécuter la suite de tests unitaires avec make check pour vous assurer que les bibliothèques liées ne sont pas endommagées de manière évidente. La dernière étape consiste à installer les différents exécutables sur votre système à l'aide de la commande make install. Vous pouvez être invité à

entrer votre mot de passe utilisateur, car cette étape nécessite des privilèges administratifs :

```
$ make check && sudo make install
Mot de passe:
Faire installer dans src
../build-aux/install-sh -c -d '/ usr / local / lib'
libtool: install: / usr / bin / install -c bitcoind /
usr / local / bin / bitcoind
libtool: installez: / usr / bin / install -c
bitcoin-cli / usr / local / bin / bitcoin-cli
libtool: installez: / usr / bin / install -c
bitcoin-tx / usr / local / bin / bitcoin-tx
...
$
```

L'installation par défaut de bitcoind le place dans */ usr / local / bin* . Vous pouvez confirmer que Bitcoin Core est correctement installé en demandant au système le chemin des exécutables, comme suit :

```
$ which bitcoind
/ usr / local / bin / bitcoind

$ which bitcoin-cli
/ usr / local / bin / bitcoin-cli
```

Exécuter un nœud Bitcoin Core

Le réseau peer-to-peer de Bitcoin est composé de « nœuds » de réseau, gérés principalement par des bénévoles et certaines des entreprises qui créent des applications Bitcoin. Les nœuds bitcoin exécutant ont une vue directe et faisant autorité de la blockchain bitcoin, avec une copie locale de toutes les transactions, validées indépendamment par leur propre système. En exécutant un nœud, vous n'avez pas besoin de compter sur un tiers pour valider une transaction. De plus, en exécutant un nœud Bitcoin, vous contribuez au réseau Bitcoin en le rendant plus robuste.

L'exécution d'un nœud, cependant, nécessite un système connecté en permanence avec suffisamment de ressources pour traiter toutes les transactions Bitcoin. Selon que vous choisissez d'indexer toutes les transactions et de conserver une copie complète de la blockchain, vous pouvez également avoir besoin de beaucoup d'espace disque et de RAM. Au début de 2021, un nœud d'index complet a besoin de 2 Go de RAM et d'un minimum de 360 Go d'espace disque (voir https ://blockchain.info/charts/blocks-size). Les nœuds Bitcoin transmettent et reçoivent également des transactions et des blocs Bitcoin, consommant de la bande passante Internet. Si votre connexion Internet est limitée, a un faible plafond de données ou est mesurée (facturée par le gigabit), vous ne devriez probablement pas exécuter un nœud bitcoin dessus, ou l'exécuter d'une manière qui limite sa bande passante (voir

Exemple de configuration d'un système aux ressources limitées
).

Conseil

Bitcoin Core conserve une copie complète de la blockchain par défaut, avec chaque transaction qui s'est produite sur le réseau Bitcoin depuis sa création en 2009. Cet ensemble de données fait des dizaines de gigaoctets et est téléchargé progressivement sur plusieurs jours ou semaines, selon le vitesse de votre CPU et connexion Internet. Bitcoin Core ne sera pas en mesure de traiter les transactions ou de mettre à jour les soldes de compte tant que l'ensemble de données de la blockchain n'aura pas été téléchargé. Assurez-vous que vous disposez de suffisamment d'espace disque, de bande passante et de temps pour terminer la synchronisation initiale. Vous pouvez configurer Bitcoin Core pour réduire la taille de la blockchain en supprimant les anciens blocs (voir Exemple de configuration d'un système à ressources limitées), mais il téléchargera toujours l'ensemble de données avant de supprimer les données.

Malgré ces besoins en ressources, des milliers de volontaires gèrent des nœuds Bitcoin. Certains fonctionnent sur des systèmes aussi simples qu'un Raspberry Pi (un ordinateur à 35 USD de la taille d'un paquet de cartes). De nombreux bénévoles exécutent également des nœuds Bitcoin sur des serveurs loués, généralement une variante de Linux. Une instance de *serveur privé virtuel* (VPS) ou de *serveur de cloud computing* peut être

utilisée pour exécuter un nœud bitcoin. Ces serveurs peuvent être loués pour 25 $ à 50 $ USD par mois auprès de divers fournisseurs.

Pourquoi voudriez-vous exécuter un nœud ? Voici quelques-unes des raisons les plus courantes :

- Si vous développez un logiciel Bitcoin et que vous devez vous fier à un nœud Bitcoin pour un accès programmable (API) au réseau et à la blockchain.
- Si vous créez des applications qui doivent valider les transactions selon les règles de consensus de Bitcoin. En règle générale, les éditeurs de logiciels Bitcoin exécutent plusieurs nœuds.
- Si vous souhaitez prendre en charge Bitcoin. L'exécution d'un nœud rend le réseau plus robuste et capable de servir plus de portefeuilles, plus d'utilisateurs et plus de transactions.
- Si vous ne souhaitez pas compter sur un tiers pour traiter ou valider vos transactions.

Si vous lisez ce livre et que vous souhaitez développer un logiciel Bitcoin, vous devriez exécuter votre propre nœud.

Configuration du nœud Bitcoin Core

Bitcoin Core recherchera un fichier de configuration dans son répertoire de données à chaque démarrage. Dans cette section,

nous examinerons les différentes options de configuration et créerons un fichier de configuration. Pour localiser le fichier de configuration, exécutez bitcoind -printtoconsole dans votre terminal et recherchez les deux premières lignes.

```
$ bitcoind -printtoconsole
Bitcoin version v0.15.0
Using the 'standard' SHA256 implementation
Using data directory /home/ubuntu/.bitcoin/
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
...
[a lot more debug output]
...
```

Vous pouvez appuyer sur Ctrl-C pour arrêter le nœud une fois que vous avez déterminé l'emplacement du fichier de configuration. En général, le fichier de configuration se trouve dans le *répertoire de* données *.bitcoin* sous le répertoire personnel de votre utilisateur. Il n'est pas créé automatiquement, mais vous pouvez créer un fichier de configuration de démarrage en copiant et en collant à partir de l' exemple de configuration d'un exemple de nœud d'index complet , ci-dessous. Vous pouvez créer ou modifier le fichier de configuration dans votre éditeur préféré.

Bitcoin Core propose plus de 100 options de configuration qui modifient le comportement du nœud de réseau, le stockage de la blockchain et de nombreux autres aspects de son fonction-

nement. Pour voir une liste de ces options, exécutez bitcoind —help :

```
$ bitcoind --help
Bitcoin Core Daemon version v0.15.0

Usage:
  bitcoind [options]                     Start
  Bitcoin Core Daemon

Options:

  -?
       Print this help message and exit

  -version
       Print version and exit

  -alertnotify=<cmd>
       Execute command when a relevant alert is
       received or we see a really
       long fork (%s in cmd is replaced by message)
...
[many more options]
...

  -rpcthreads=<n>
       Set the number of threads to service RPC calls
       (default: 4)
```

Voici quelques-unes des options les plus importantes que vous pouvez définir dans le fichier de configuration ou en tant que

paramètres de ligne de commande pour bitcoind :

alertnotify

Exécutez une commande ou un script spécifié pour envoyer des alertes d'urgence au propriétaire de ce nœud, généralement par e-mail.

conf

Un autre emplacement pour le fichier de configuration. Cela n'a de sens qu'en tant que paramètre de ligne de commande pour bitcoind, car il ne peut pas être dans le fichier de configuration auquel il se réfère.

datadir

Sélectionnez le répertoire et le système de fichiers dans lesquels placer toutes les données de la blockchain. Par défaut, il s'agit du sous-répertoire *.bitcoin* de votre répertoire personnel. Assurez-vous que ce système de fichiers dispose de plusieurs gigaoctets d'espace libre.

prune

Réduisez les besoins en espace disque à autant de mégaoctets en supprimant les anciens blocs. Utilisez-le sur un nœud aux ressources limitées qui ne peut pas s'adapter à la blockchain complète.

txindex

Maintenez un index de toutes les transactions. Cela signifie une copie complète de la blockchain qui vous permet de récupérer par programme toute transaction par ID.

dbcache

La taille du cache UTXO. La valeur par défaut est de 450 Mio. Augmentez cette valeur sur le matériel haut de gamme et réduisez la taille du matériel bas de gamme pour économiser de la mémoire au détriment de la lenteur des E / S de disque.

maxconnections

Définissez le nombre maximum de nœuds à partir desquels accepter les connexions. Réduire ce paramètre par défaut réduira votre consommation de bande passante. À utiliser si vous avez un plafond de données ou si vous payez au gigaoctet.

maxmempool

Limitez le pool de mémoire de transaction à ce nombre de mégaoctets. Utilisez-le pour réduire l'utilisation de la mémoire sur les nœuds à mémoire limitée.

maxreceivebuffer / maxsendbuffer

Limitez la mémoire tampon par connexion à ce nombre de multiples de 1000 octets. À utiliser sur les nœuds à mémoire limitée.

minrelaytxfee

Définissez le taux de frais minimum pour la transaction que vous relayez. En dessous de cette valeur, la transaction est traitée non standard, rejetée du pool de transactions et non relayée.

Index de la base de données des transactions et option txindex

Par défaut, Bitcoin Core crée une base de données contenant *uniquement* les transactions liées au portefeuille de l'utilisateur.

Si vous souhaitez pouvoir accéder à *n'importe quelle* transaction avec des commandes telles que getrawtransaction (voir Exploration et décodage des transactions), vous devez configurer Bitcoin Core pour créer un index de transaction complet, ce qui peut être réalisé avec l'option txindex. Définissez txindex = 1 dans le fichier de configuration Bitcoin Core. Si vous ne définissez pas cette option dans un premier temps et que vous la définissez ultérieurement sur l'indexation complète, vous devez redémarrer bitcoind avec l'option -reindex et attendre qu'il reconstruise l'index.

Un exemple de configuration d'un nœud d'index complet montre comment vous pouvez combiner les options précédentes, avec un nœud entièrement indexé, fonctionnant en tant que backend API pour une application Bitcoin.

Exemple 1. Exemple de configuration d'un nœud d'index complet

```
alertnotify=myemailscript.sh "Alert: %s"
datadir=/lotsofspace/bitcoin
txindex=1
```

Un exemple de configuration d'un système à ressources limitées montre un nœud à ressources limitées s'exécutant sur un serveur plus petit.

Exemple 2. Exemple de configuration d'un système à ressources limitées

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
dbcache=150
maxmempool=150
maxreceivebuffer=2500
maxsendbuffer=500
```

Une fois que vous avez modifié le fichier de configuration et défini les options qui représentent le mieux vos besoins, vous pouvez tester bitcoind avec cette configuration. Exécutez Bitcoin Core avec l'option printtoconsole pour s'exécuter au premier plan avec une sortie vers la console :

```
$ bitcoind -printtoconsole

Bitcoin version v0.15.0
InitParameterInteraction: parameter interaction:
-whitelistforcerelay=1 -> setting -whitelistrelay=1
Assuming ancestors of block
000000000000000003b9ce759c2a087d52abc4266f8f4ebd6d768b89defa50a
have valid signatures.
Using the 'standard' SHA256 implementation
Default data directory /home/ubuntu/.bitcoin
Using data directory /lotsofspace/.bitcoin
Using config file /home/ubuntu/.bitcoin/bitcoin.conf
Using at most 125 automatic connections (1048576 file
descriptors available)
Using 16 MiB out of 32/2 requested for signature
cache, able to store 524288 elements
Using 16 MiB out of 32/2 requested for script
execution cache, able to store 524288 elements
Using 2 threads for script verification
HTTP: creating work queue of depth 16
```

```
No rpcpassword set - using random cookie
authentication
Generated RPC authentication cookie
/lotsofspace/.bitcoin/.cookie
HTTP: starting 4 worker threads
init message: Verifying wallet(s)...
Using BerkeleyDB version Berkeley DB 4.8.30: (April
9, 2010)
Using wallet wallet.dat
CDBEnv::Open: LogDir=/lotsofspace/.bitcoin/database
ErrorFile=/lotsofspace/.bitcoin/db.log
scheduler thread start
Cache configuration:
* Using 250.0MiB for block index database
* Using 8.0MiB for chain state database
* Using 1742.0MiB for in-memory UTXO set (plus up to
286.1MiB of unused mempool space)
init message: Loading block index...
Opening LevelDB in /lotsofspace/.bitcoin/blocks/index
Opened LevelDB successfully

[... more startup messages ...]
```

Vous pouvez appuyer sur Ctrl-C pour interrompre le processus une fois que vous êtes convaincu qu'il charge les paramètres corrects et s'exécute comme vous le souhaitez.

Pour exécuter Bitcoin Core en arrière-plan en tant que processus, démarrez-le avec l'option daemon, comme bitcoind -daemon.

Pour surveiller la progression et l'état d'exécution de votre nœud bitcoin, utilisez la commande bitcoin-cli getblockchai-

ninfo :

```
$ bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 0,
  "headers": 83999,
  "bestblockhash":
  "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26
  "difficulty": 1,
  "mediantime": 1231006505,
  "verificationprogress": 3.783041623201835e-09,
  "chainwork":
  "0000000000000000000000000000000000000000000000000000000100010000
  "pruned": false,
  [...]
}
```

Cela montre un nœud avec une hauteur de blockchain de 0 blocs et 83999 en-têtes. Le nœud récupère actuellement les en-têtes de bloc de la meilleure chaîne et continue ensuite à télécharger les blocs complets.

Une fois que vous êtes satisfait des options de configuration que vous avez sélectionnées, vous devez ajouter du bitcoin aux scripts de démarrage de votre système d'exploitation, afin qu'il s'exécute en continu et redémarre lorsque le système d'exploitation redémarre. Vous trouverez un certain nombre d'exemples de scripts de démarrage pour divers systèmes d'exploitation dans le répertoire source de bitcoin sous *contrib / init* et un fichier *README.md* indiquant quel système utilise

quel script.

Interface de programmation d'applications Bitcoin Core (API)

Le client Bitcoin Core implémente une interface JSON-RPC qui est également accessible à l'aide de l'assistant de ligne de commande bitcoin-cli. La ligne de commande nous permet d'expérimenter de manière interactive les fonctionnalités qui sont également disponibles par programmation via l'API. Pour commencer, appelez la commande help pour voir une liste des commandes bitcoin RPC disponibles :

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] ( "account" )
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction [{"txid":"id","vout":n},...]
{"address":amount,...}
decoderawtransaction "hexstring"
...
...
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

Chacune de ces commandes peut prendre un certain nombre de paramètres. Pour obtenir une aide supplémentaire, une description détaillée et des informations sur les paramètres,

ajoutez le nom de la commande après l'aide. Par exemple, pour voir l'aide sur la commande getblockhash RPC :

```
$ bitcoin-cli help getblockhash
getblockhash height

Returns hash of block in best-block-chain at height
provided.

Arguments:
1. height         (numeric, required) The height index

Result:
"hash"         (string) The block hash

Examples:
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc":
"1.0", "id":"curltest", "method": "getblockhash",
"params": [1000] }' -H 'content-type: text/plain;'
http://127.0.0.1:8332/
```

À la fin des informations d'aide, vous verrez deux exemples de la commande RPC, utilisant l'assistant bitcoin-cli ou le client HTTP curl. Ces exemples montrent comment vous pouvez appeler la commande. Copiez le premier exemple et voyez le résultat :

```
$ bitcoin-cli getblockhash 1000
00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09
```

Le résultat est un hachage de bloc, qui est décrit plus en détail dans les chapitres suivants. Mais pour l'instant, cette commande devrait renvoyer le même résultat sur votre système, démontrant que votre nœud Bitcoin Core est en cours d'exécution, accepte les commandes et a des informations sur le bloc 1000 à vous renvoyer.

Dans les sections suivantes, nous montrerons quelques commandes RPC très utiles et leur résultat attendu.

Obtenir des informations sur l'état du client Bitcoin Core

Bitcoin Core fournit des rapports d'état sur différents modules via l'interface JSON-RPC. Les commandes les plus importantes incluent getblockchaininfo, getmempoolinfo, getnetworkinfo et getwalletinfo.

La commande RPC getblockchaininfo de Bitcoin a été introduite plus tôt. La commande getnetworkinfo affiche des informations de base sur l'état du nœud de réseau bitcoin. Utilisez bitcoin-cli pour l'exécuter :

```
$ bitcoin-cli getnetworkinfo
```

```
"version": 150000,
  "subversion": "/Satoshi:0.15.0/",
  "protocolversion": 70015,
  "localservices": "000000000000000d",
  "localrelay": true,
```

```
  "timeoffset": 0,
  "networkactive": true,
  "connections": 8,
  "networks": [
    ...detailed information about all networks (ipv4,
    ipv6 or onion)...
  ],
  "relayfee": 0.00001000,
  "incrementalfee": 0.00001000,
  "localaddresses": [
  ],
  "warnings": ""
}
```

Les données sont renvoyées en JavaScript Object Notation (JSON), un format qui peut facilement être « consommé » par tous les langages de programmation, mais qui est également assez lisible par l'homme. Parmi ces données, nous voyons les numéros de version du client logiciel bitcoin (150000) et du protocole bitcoin (70015). Nous voyons le nombre actuel de connexions (8) et diverses informations sur le réseau bitcoin et les paramètres liés à ce client.

Conseil

Il faudra un certain temps, peut-être plus d'un jour, pour que le client bitcoind « rattrape » la hauteur actuelle de la blockchain lorsqu'il télécharge des blocs à partir d'autres clients Bitcoin. Vous pouvez vérifier sa progression en utilisant getblockchaininfo pour voir le nombre de blocs connus.

Exploration et décodage des transactions

Commandes : getrawtransaction, decoderawtransaction

Dans le chapitre precedent , Alice a acheté une tasse de café au Bob's Cafe. Sa transaction a été enregistrée sur la blockchain avec l'identifiant de transaction (txid) 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2. Utilisons l'API pour récupérer et examiner cette transaction en passant l'ID de transaction en tant que paramètre :

```
$ bitcoin-cli getrawtransaction
0627052↵b6f28912f2703066a912ea577f2ce4da4caa5a
5fbd8a57286c345c2f2

0100000001186↵f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804
000008↵b483045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1c
ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09c
10484↵ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe42
89↵d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0
0001976↵a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

Conseil

Un ID de transaction ne fait pas autorité tant qu'une transaction n'a pas été confirmée. L'absence de hachage de transaction dans la blockchain ne signifie pas que la transaction n'a pas été traitée. Ceci est connu sous le nom de « malléabilité de transaction », car les hachages de transaction peuvent être modifiés avant la confirmation dans un bloc. Après confirmation, le txid est immuable et

fait autorité.

La commande getrawtransaction renvoie une transaction sérialisée en notation hexadécimale. Pour décoder cela, nous utilisons la commande decoderawtransaction, en passant les données hexadécimales en tant que paramètre. Vous pouvez copier l'hex retourné par getrawtransaction et le coller en tant que paramètre pour decoderawtransaction :

```
$ bitcoin-cli decoderawtransaction
0100000001186↩┘f9f998a5aa6f048e51dd8419a14d8↩┘
a0f1a8a2836dd734d2804fe65fa35779000000008b483045022100884d142d8665
6↩┘ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9
cad530a863ea8f53982c09db8f6e381301410484ecc0d46f1918b30928fa0e4ed9
e0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5
336↩┘a8d752adfffffffff0260e31600000000001976a914ab68025513c3dbd2f7
d50f654e788acd0ef8000000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53
88ac00000000
```

```
{
  "txid":
  "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f
  "size": 258,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid":
      "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f1
      "vout": 0,
      "scriptSig": {
```

```
        "asm":"3045022100884d142d86652a3f47ba4746ec719bbfbd040a57
        "hex":"483045022100884d142d86652a3f47ba4746ec719bbfbd040a
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ab68...5f654e7
        OP_EQUALVERIFY OP_CHECKSIG",
        "hex":
        "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
        ]
      }
    },
    {
      "value": 0.08450000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 7f9b1a...025a8
        OP_EQUALVERIFY OP_CHECKSIG",
        "hex":
        "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
```

```
    ]
}
```

Le décodage de transaction montre tous les composants de cette transaction, y compris les entrées et sorties de transaction. Dans ce cas, nous voyons que la transaction qui a crédité notre nouvelle adresse de 15 millibits a utilisé une entrée et généré deux sorties. L'entrée de cette transaction était la sortie d'une transaction précédemment confirmée (représentée par le vin txid commençant par 7957a35fe). Les deux sorties correspondent au crédit de 15 millibits et à une sortie avec retour à l'expéditeur.

Nous pouvons explorer davantage la blockchain en examinant la transaction précédente référencée par son txid dans cette transaction en utilisant les mêmes commandes (par exemple, getrawtransaction). En passant d'une transaction à l'autre, nous pouvons suivre une chaîne de transactions car les pièces sont transmises de l'adresse du propriétaire à l'adresse du propriétaire.

Explorer les blocs

Commandes : getblock, getblockhash

L'exploration des blocs est similaire à l'exploration des transactions. Cependant, les blocs peuvent être référencés soit par la *hauteur* du bloc, soit par le *hachage* du bloc . Tout d'abord, trouvons un bloc par sa hauteur. Dans le chapitre precedent, nous avons vu que la transaction d'Alice était incluse dans le

bloc 277316.

Nous utilisons la commande getblockhash, qui prend la hauteur du bloc comme paramètre et renvoie le hachage de bloc pour ce bloc :

```
$ bitcoin-cli getblockhash 277316
000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

Maintenant que nous savons dans quel bloc la transaction d'Alice était incluse, nous pouvons interroger ce bloc. Nous utilisons la commande getblock avec le hachage de bloc comme paramètre :

```
$ bitcoin-cli getblock
0000000000000001↵b6b9a13b095e96db41c4a928b97ef2d944a9b3
1b2cc7bdc4
```

```
{
  "hash":
  "000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bc
  "confirmations": 37371,
  "size": 218629,
  "height": 277316,
  "version": 2,
  "merkleroot":
  "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7c
  "tx": [
```

```
    "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afb
    "b268b45c59b39d7596147577l8b9918caf0ba9d97c56f3b91956ff877c503
    "04905ff987ddd4cfe603b03cfb7ca50ee81d89d1f8f5f265c38f763eea4a2
    "32467aab5d04f51940075055c2f20bbd1195727c961431bf0aff8443f9710
    "561c5216944e21fa29dd12aaa1a45e3397f9c0d888359cb05e1f79fe73da3
[... hundreds of transactions ...]
    "78b300b2a1d2d9449b58db7bc71c3884d6e0579617e0da4991b9734cef7ab
    "6c87130ec283ab4c2c493b190c20de4b28ff3caf72d16ffa1ce3e96f2069a
    "6f423dbc3636ef193fd8898dfdf7621dcade1bbe509e963ffbff91f696d81
    "802ba8b2adabc5796a9471f25b02ae6aeee2439c679a5c33c4bbcee97e081
    "eaaf6a048588d9ad4d1c092539bd571dd8af30635c152a3b0e8b611e67d1a
    "e67abc6bd5e2cac169821afc51b207127f42b92a841e976f9b752157879ba
    "d38985a6a1bfd35037cb7776b2dc86797abbb7a06630f5d03df2785d50d5a
    "45ea0a3f6016d2bb90ab92c34a7aac9767671a8a84b9bcce6c019e60197c1
    "c098445d748ced5f178ef2ff96f2758cbec9eb32cb0fc65db313bcac1d3bc
  ],
  "time": 1388185914,
  "mediantime": 1388183675,
  "nonce": 924591752,
  "bits": "1903a30c",
  "difficulty": 1180923195.258026,
  "chainwork":
  "0000000000000000000000000000000000000000000000934695e92aaf53afa1
  "previousblockhash":
  "0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f056
  "nextblockhash":
  "0000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba
}
```

Le bloc contient 419 transactions et la 64e transaction répertoriée (0627052b …) est le paiement du café d'Alice. L'entrée de hauteur nous indique qu'il s'agit du 277316e bloc de la blockchain.

Utilisation de l'interface programmatique de Bitcoin Core

L'assistant bitcoin-cli est très utile pour explorer l'API Bitcoin Core et tester les fonctions. Mais tout l'intérêt d'une interface de programmation d'application est d'accéder aux fonctions par programmation. Dans cette section, nous montrerons comment accéder à Bitcoin Core à partir d'un autre programme.

L'API de Bitcoin Core est une interface JSON-RPC. JSON signifie JavaScript Object Notation et c'est un moyen très pratique de présenter des données que les humains et les programmes peuvent facilement lire. RPC signifie Remote Procedure Call, ce qui signifie que nous appelons des procédures (fonctions) distantes (sur le nœud Bitcoin Core) via un protocole réseau. Dans ce cas, le protocole réseau est HTTP ou HTTPS (pour les connexions cryptées).

Lorsque nous avons utilisé la commande bitcoin-cli pour obtenir de l'aide sur une commande, cela nous a montré un exemple d'utilisation de curl, le client HTTP de ligne de commande polyvalent pour construire l'un de ces appels JSON-RPC :

```
$ curl --user myusername --data-binary '{"jsonrpc":
"1.0", "id":"curltest", "method":
"getblockchaininfo", "params": [] }' -H
'content-type: text/plain;' http://127.0.0.1:8332/
```

Cette commande montre que curl soumet une requête HTTP à

l'hôte local (127.0.0.1), se connectant au port bitcoin par défaut (8332) et soumettant une requête jsonrpc pour la méthode getblockchaininfo en utilisant un encodage text / plain.

Vous remarquerez peut-être que curl demandera l'envoi d'informations d'identification avec la demande. Bitcoin Core créera un mot de passe aléatoire à chaque démarrage et le placera dans le répertoire de données sous le nom .cookie. L'assistant bitcoin-cli peut lire ce fichier de mot de passe étant donné le répertoire de données. De même, vous pouvez copier le mot de passe et le transmettre à curl (ou à tout autre wrapper Bitcoin Core RPC de niveau supérieur). Vous pouvez également créer un mot de passe statique avec le script d'assistance fourni dans *./share/rpcauth/rpcauth.py* dans le répertoire source de Bitcoin Core.

Si vous implémentez un appel JSON-RPC dans votre propre programme, vous pouvez utiliser une bibliothèque HTTP générique pour construire l'appel, similaire à ce qui est montré dans l'exemple curl précédent.

Cependant, il existe des bibliothèques dans la plupart des langages de programmation qui « enveloppent » l'API Bitcoin Core d'une manière qui rend cela beaucoup plus simple. Nous utiliserons la bibliothèque python-bitcoinlib pour simplifier l'accès aux API. N'oubliez pas que cela nécessite que vous ayez une instance Bitcoin Core en cours d'exécution, qui sera utilisée pour effectuer des appels JSON-RPC.

Le script Python dans Exécution de getblockchaininfo via l'API JSON-RPC de Bitcoin Core effectue un simple appel

getblockchaininfo et imprime le paramètre de blocs à partir des données renvoyées par Bitcoin Core (nœud complet requis).

Exemple 3. Exécution de getblockchaininfo via l'API JSON-RPC de Bitcoin Core

```
link:code/rpc_example.py[]
```

L'exécuter nous donne le résultat suivant :

```
$ python rpc_example.py
394075
```

Cela nous dit que notre nœud Bitcoin Core local a 394075 blocs dans sa blockchain. Ce n'est pas un résultat spectaculaire, mais cela démontre l'utilisation de base de la bibliothèque en tant qu'interface simplifiée pour l'API JSON-RPC de Bitcoin Core.

Ensuite, utilisons les appels getrawtransaction et decodetransaction pour récupérer les détails du paiement du café d'Alice. Dans Récupération d'une transaction et itération de ses sorties , nous récupérons la transaction d'Alice et listons les sorties de la transaction. Pour chaque sortie, nous montrons l'adresse et la valeur du destinataire. Pour rappel, la transaction d'Alice avait une sortie payant Bob's Cafe et une sortie pour changement à Alice.

Exemple 4. Récupération d'une transaction et itération de ses

sorties

```
link:code/rpc_transaction.py[]
```

En exécutant ce code, nous obtenons :

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA'],
Decimal('0.01500000'))
([u'1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK'],
Decimal('0.08450000'))
```

Les deux exemples précédents sont assez simples. Vous n'avez pas vraiment besoin d'un programme pour les exécuter ; vous pouvez tout aussi facilement utiliser l'assistant bitcoin-cli. L'exemple suivant, cependant, nécessite plusieurs centaines d'appels RPC et démontre plus clairement l'utilisation d'une interface de programmation.

Lors de la récupération d'un bloc et de l'ajout de toutes les sorties de transaction , nous récupérons d'abord le bloc 277316, puis nous récupérons chacune des 419 transactions en référence à chaque ID de transaction. Ensuite, nous parcourons chacune des sorties de la transaction et additionnons la valeur.

Exemple 5. Récupération d'un bloc et ajout de toutes les sorties de transaction

```
link:code/rpc_block.py[]
```

En exécutant ce code, nous obtenons :

```
$ python rpc_block.py

('Total value in block: ', Decimal('10322.07722534'))
```

Notre exemple de code calcule que la valeur totale des transactions dans ce bloc est de 10322,07722534 BTC (y compris 25 BTC de récompense et 0,0909 BTC de frais). Comparez cela au montant indiqué par un site d'exploration de blocs en recherchant le hachage ou la hauteur du bloc. Certains explorateurs de blocs indiquent la valeur totale en excluant la récompense et en excluant les frais. Voyez si vous pouvez repérer la différence.

Clients, bibliothèques et boîtes à outils alternatifs

Il existe de nombreux clients alternatifs, bibliothèques, boîtes à outils et même implémentations de nœuds complets dans l'écosystème Bitcoin. Ceux-ci sont implémentés dans une variété de langages de programmation, offrant aux programmeurs des interfaces natives dans leur langage préféré.

Les sections suivantes répertorient certaines des meilleures bibliothèques, clients et boîtes à outils, organisés par langages de programmation.

C / C ++

Bitcoin Core
L'implémentation de référence de Bitcoin

libbitcoin
Boîte à outils de développement C ++ multiplateforme, nœud et bibliothèque de consensus

explorateur de bitcoin
L'outil de ligne de commande de Libbitcoin

picocoin
Bibliothèque client légère en langage AC pour Bitcoin par Jeff Garzik

JavaScript

bcoin
Une implémentation de nœud complet modulaire et évolutive avec API

Bitcore
Nœud complet, API et bibliothèque par Bitpay

BitcoinJS
Une bibliothèque Bitcoin JavaScript pure pour node.js et les navigateurs

Java

bitcoinj
Une bibliothèque client Java full-node

PHP

bitwasp / bitcoin
Une bibliothèque PHP Bitcoin et des projets associés

Python

python-bitcoinlib
Une bibliothèque Bitcoin Python, une bibliothèque de consensus et un nœud par Peter Todd

pycoin
Une bibliothèque Bitcoin Python par Richard Kiss

pybitcointools
Un fork archivé de la bibliothèque bitcoin Python par Vitalik Buterin

Ruby

client bitcoin
Un wrapper de bibliothèque Ruby pour l'API JSON-RPC

Go

btcd
Un client Bitcoin à nœud complet en langage Go

Rust

rust-bitcoin
Bibliothèque Bitcoin Rust pour la sérialisation, l'analyse et les appels d'API

C #

NBitcoin
Bibliothèque Bitcoin complète pour le framework .NET

Objectif c

CoreBitcoin

Boîte à outils Bitcoin pour ObjC et Swift

Beaucoup plus de bibliothèques existent dans une variété d'autres langages de programmation et d'autres sont créées tout le temps.

5

Clés, Adresses

Vous avez peut-être entendu dire que le bitcoin est basé sur la *cryptographie* , qui est une branche des mathématiques largement utilisée dans la sécurité informatique. La cryptographie signifie « écriture secrète » en grec, mais la science de la cryptographie englobe plus que la simple écriture secrète, appelée cryptage. La cryptographie peut également être utilisée pour prouver la connaissance d'un secret sans révéler ce secret (signature numérique), ou prouver l'authenticité des données (empreinte digitale numérique). Ces types de preuves cryptographiques sont les outils mathématiques essentiels à Bitcoin et largement utilisés dans les applications Bitcoin. Ironiquement, le cryptage n'est pas une partie importante du bitcoin, car ses communications et ses données de transaction ne sont pas cryptées et n'ont pas besoin d'être cryptées pour protéger les fonds. Dans ce chapitre, nous présenterons une partie de la cryptographie utilisée dans Bitcoin pour contrôler la propriété des fonds, sous la forme de clés, d'adresses et de portefeuilles.

Introduction

La propriété du bitcoin est établie par *le* biais *de clés numériques* , d' *adresses bitcoin* et *de signatures numériques* . Les clés numériques ne sont pas réellement stockées sur le réseau, mais sont plutôt créées et stockées par les utilisateurs dans un fichier, ou une simple base de données, appelée *portefeuille* . Les clés numériques dans le portefeuille d'un utilisateur sont complètement indépendantes du protocole bitcoin et peuvent être générées et gérées par le logiciel de portefeuille de l'utilisateur sans référence à la blockchain ou accès à Internet. Les clés permettent de nombreuses propriétés intéressantes du bitcoin, notamment la confiance et le contrôle décentralisés, l'attestation de propriété et le modèle de sécurité à l'épreuve de la cryptographie.

La plupart des transactions Bitcoin nécessitent une signature numérique valide pour être incluse dans la blockchain, qui ne peut être générée qu'avec une clé secrète ; par conséquent, toute personne possédant une copie de cette clé a le contrôle du bitcoin. La signature numérique utilisée pour dépenser des fonds est également appelée *témoin* , un terme utilisé en cryptographie. Les données des témoins dans une transaction Bitcoin témoignent de la véritable propriété des fonds dépensés.

Les clés se présentent par paires constituées d'une clé privée (secrète) et d'une clé publique. Considérez la clé publique comme similaire à un numéro de compte bancaire et la clé privée comme similaire au code PIN secret, ou à la signature sur un chèque, qui permet de contrôler le compte. Ces clés

numériques sont très rarement vues par les utilisateurs de bitcoin. Pour la plupart, ils sont stockés dans le fichier du portefeuille et gérés par le logiciel du portefeuille Bitcoin.

Dans la partie paiement d'une transaction bitcoin, la clé publique du destinataire est représentée par son empreinte digitale numérique, appelée *adresse bitcoin* , qui est utilisée de la même manière que le nom du bénéficiaire sur un chèque (c'est-à-dire « Payer à l'ordre de ») . Dans la plupart des cas, une adresse bitcoin est générée à partir d'une clé publique et correspond à celle-ci. Cependant, toutes les adresses Bitcoin ne représentent pas des clés publiques ; ils peuvent aussi représenter d'autres bénéficiaires tels que des scripts, comme nous le verrons plus loin dans ce chapitre. De cette façon, les adresses bitcoin permettent d'abstraire le destinataire des fonds, rendant les destinations de transaction flexibles, similaires aux chèques papier : un instrument de paiement unique qui peut être utilisé pour payer sur les comptes des personnes, payer sur les comptes de l'entreprise, payer des factures ou payer en espèces. L'adresse bitcoin est la seule représentation des clés que les utilisateurs verront régulièrement, car c'est la partie dont ils ont besoin pour partager avec le monde.

Tout d'abord, nous présenterons la cryptographie et expliquerons les mathématiques utilisées dans le bitcoin. Ensuite, nous verrons comment les clés sont générées, stockées et gérées. Nous passerons en revue les différents formats de codage utilisés pour représenter les clés, adresses et adresses de script privées et publiques. Enfin, nous examinerons l'utilisation avancée des clés et des adresses : adresses personnalisées, multi-signatures et de script et portefeuilles papier.

Cryptographie à clé publique et crypto-monnaie

La cryptographie à clé publique a été inventée dans les années 1970 et est une base mathématique pour la sécurité informatique et de l'information.

Depuis l'invention de la cryptographie à clé publique, plusieurs fonctions mathématiques appropriées, telles que l'exponentiation des nombres premiers et la multiplication de la courbe elliptique, ont été découvertes. Ces fonctions mathématiques sont pratiquement irréversibles, ce qui signifie qu'elles sont faciles à calculer dans une direction et impossibles à calculer dans la direction opposée. Sur la base de ces fonctions mathématiques, la cryptographie permet la création de secrets numériques et de signatures numériques infalsifiables. Bitcoin utilise la multiplication de la courbe elliptique comme base de sa cryptographie.

Dans Bitcoin, nous utilisons la cryptographie à clé publique pour créer une paire de clés qui contrôle l'accès à Bitcoin. La paire de clés se compose d'une clé privée et, dérivée de celle-ci, d'une clé publique unique. La clé publique est utilisée pour recevoir des fonds et la clé privée est utilisée pour signer des transactions pour dépenser les fonds.

Il existe une relation mathématique entre la clé publique et la clé privée qui permet à la clé privée d'être utilisée pour générer des signatures sur les messages. Ces signatures peuvent être validées par rapport à la clé publique sans révéler la clé privée.

Lorsqu'il dépense du bitcoin, le propriétaire actuel du bitcoin

présente sa clé publique et une signature (différente à chaque fois, mais créée à partir de la même clé privée) dans une transaction pour dépenser ces bitcoins. Grâce à la présentation de la clé publique et de la signature, tout le monde dans le réseau bitcoin peut vérifier et accepter la transaction comme valide, confirmant que la personne transférant le bitcoin en était propriétaire au moment du transfert.

Conseil

Dans la plupart des implémentations de portefeuille, les clés privées et publiques sont stockées ensemble sous forme de paire de clés *pour plus de commodité. Cependant, la clé publique peut être calculée à partir de la clé privée, il est donc également possible de stocker uniquement la clé privée.*

Clés privées et publiques

Un portefeuille Bitcoin contient une collection de paires de clés, chacune composée d'une clé privée et d'une clé publique. La clé privée (k) est un nombre, généralement choisi au hasard. À partir de la clé privée, nous utilisons la multiplication de courbe elliptique, une fonction cryptographique à sens unique, pour générer une clé publique (K). À partir de la clé publique (K), nous utilisons une fonction de hachage cryptographique unidirectionnelle pour générer une adresse bitcoin (A). Dans cette section, nous commencerons par générer la clé privée, examinerons la courbe mathématique elliptique utilisée pour la transformer en clé publique, et enfin, générer une adresse

bitcoin à partir de la clé publique. La relation entre la clé privée, la clé publique et l'adresse bitcoin est indiquée dans la clé privée, la clé publique et l'adresse bitcoin .

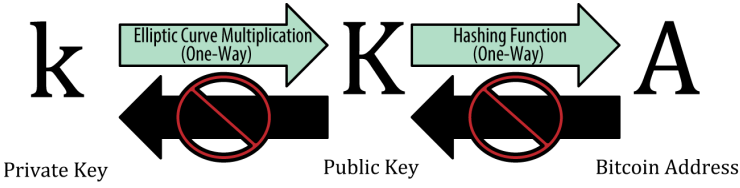


Figure 1. Clé privée, clé publique et adresse bitcoin

Pourquoi utiliser la cryptographie asymétrique (clés publiques / privées) ?

Pourquoi la cryptographie asymétrique est-elle utilisée dans Bitcoin ? Il n'est pas utilisé pour « crypter » (rendre secret) les transactions. Au contraire, la propriété utile de la cryptographie asymétrique est la capacité à générer *des signatures numériques* . Une clé privée peut être appliquée à l'empreinte numérique d'une transaction pour produire une signature numérique. Cette signature ne peut être produite que par une personne connaissant la clé privée. Cependant, toute personne ayant accès à la clé publique et à l'empreinte digitale de la transaction peut les utiliser pour *vérifier* la signature. Cette propriété utile de la cryptographie asymétrique permet à quiconque de vérifier chaque signature sur chaque transaction, tout en garantissant que seuls les propriétaires de clés privées peuvent produire des signatures valides.

Clés privées

Une clé privée est simplement un nombre, choisi au hasard. La propriété et le contrôle de la clé privée sont la racine du contrôle de l'utilisateur sur tous les fonds associés à l' adresse bitcoin correspondante . La clé privée est utilisée pour créer des signatures nécessaires pour dépenser des bitcoins en prouvant la propriété des fonds utilisés dans une transaction. La clé privée doit rester secrète à tout moment, car la révéler à des tiers équivaut à leur donner le contrôle du bitcoin sécurisé par cette clé. La clé privée doit également être sauvegardée et protégée contre toute perte accidentelle, car si elle est perdue, elle ne peut pas être récupérée et les fonds qu'elle garantit sont également perdus à jamais.

Conseil

La clé privée Bitcoin n'est qu'un nombre. Vous pouvez choisir vos clés privées au hasard en utilisant juste une pièce de monnaie, un crayon et du papier : lancez une pièce 256 fois et vous avez les chiffres binaires d'une clé privée aléatoire que vous pouvez utiliser dans un portefeuille Bitcoin. La clé publique peut alors être générée à partir de la clé privée.

Générer une clé privée à partir d'un nombre aléatoire

La première et la plus importante étape dans la génération de clés est de trouver une source sécurisée d'entropie ou d'aléatoire. La création d'une clé bitcoin est essentiellement la même chose que « Choisissez un nombre entre 1 et 2 256 ». La méthode exacte que vous utilisez pour choisir ce nombre n'a

pas d'importance tant qu'elle n'est pas prévisible ou répétable. Le logiciel Bitcoin utilise les générateurs de nombres aléatoires du système d'exploitation sous-jacent pour produire 256 bits d'entropie (caractère aléatoire). Habituellement, le générateur de nombres aléatoires du système d'exploitation est initialisé par une source humaine d'aléa, c'est pourquoi il vous sera peut-être demandé de faire bouger votre souris pendant quelques secondes.

Plus précisément, la clé privée peut être un nombre compris entre 0 et n - 1 inclus, où n est une constante (n = 1,1578 * 10 77 , un peu moins de 2 256) définie en tant que l'ordre de la courbe elliptique utilisé dans Bitcoin (voir Cryptographie à courbe elliptique expliquée). Pour créer une telle clé, nous choisissons au hasard un nombre de 256 bits et vérifions qu'il est inférieur à n. En termes de programmation, cela est généralement réalisé en alimentant une plus grande chaîne de bits aléatoires, collectés à partir d'une source d'aléatoire cryptographiquement sécurisée, dans l'algorithme de hachage SHA256, qui produira commodément un nombre de 256 bits. Si le résultat est inférieur à n, nous avons une clé privée appropriée. Sinon, nous réessayons simplement avec un autre nombre aléatoire.

Avertissement

N'écrivez pas votre propre code pour créer un nombre aléatoire ou n'utilisez pas un générateur de nombres aléatoires « simple » offert par votre langage de programmation. Utilisez un générateur de nombres pseudo-aléatoires cryptographiquement sécurisé (CSPRNG) avec

une graine provenant d'une source d'entropie suffisante. Étudiez la documentation de la bibliothèque de générateur de nombres aléatoires que vous choisissez pour vous assurer qu'elle est sécurisée par cryptographie. Une mise en œuvre correcte du CSPRNG est essentielle à la sécurité des clés.

Ce qui suit est une clé privée (k) générée aléatoirement affichée au format hexadécimal (256 bits affichés sous forme de 64 chiffres hexadécimaux, chacun de 4 bits) :

```
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
```

Tip

La taille de l'espace de clé privée de Bitcoin (2^{256}) est un nombre insondable. Il est d'environ 10^{77} en décimal. A titre de comparaison, on estime que l'univers visible contient 10^{80} atomes.

Pour générer une nouvelle clé avec le client Bitcoin Core (voir le chapitre 4), utilisez la commande getnewaddress. Pour des raisons de sécurité, il affiche uniquement la clé publique, pas la clé privée. Pour demander à bitcoind d'exposer la clé privée, utilisez la commande dumpprivkey. La commande dumpprivkey affiche la clé privée dans un format encodé en somme de contrôle Base58 appelé *Wallet Import Format* (WIF), que nous examinerons plus en détail dans les formats de clé

privée . Voici un exemple de génération et d'affichage d'une clé privée à l'aide de ces deux commandes :

```
$ bitcoin-cli getnewaddress
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
$ bitcoin-cli dumpprivkey
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

La commande dumpprivkey ouvre le portefeuille et extrait la clé privée qui a été générée par la commande getnewaddress. Il n'est pas possible pour bitcoind de connaître la clé privée à partir de la clé publique à moins qu'elles ne soient toutes deux stockées dans le portefeuille.

> *Conseil*
>
> *La commande dumpprivkey ne génère pas de clé privée à partir d'une clé publique, car cela est impossible. La commande révèle simplement la clé privée déjà connue du portefeuille et qui a été générée par la commande getnewaddress.*

Vous pouvez également utiliser l'outil de ligne de commande Bitcoin Explorer pour générer et afficher des clés privées avec les commandes seed, ec-new et ec-to-wif :

```
$ bx seed | bx ec-new | bx ec-to-wif
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Clés publiques

La clé publique est calculée à partir de la clé privée en utilisant la multiplication de la courbe elliptique, qui est irréversible : $K = k * G$, où k est la clé privée, G est un point constant appelé le *point générateur* et K est la clé publique résultante. L'opération inverse, connue sous le nom de « trouver le logarithme discret » - calculer k si vous connaissez K - est aussi difficile que d'essayer toutes les valeurs possibles de k , c'est-à-dire une recherche par force brute. Avant de montrer comment générer une clé publique à partir d'une clé privée, examinons un peu plus en détail la cryptographie à courbe elliptique.

Conseil

La multiplication de la courbe elliptique est un type de fonction que les cryptographes appellent une fonction "unidirectionnelle" : elle est facile à faire dans un sens (multiplication) et impossible à faire dans le sens inverse ("division", ou trouver le logarithme discret). Le propriétaire de la clé privée peut facilement créer la clé publique, puis la partager avec le monde entier en sachant que personne ne peut inverser la fonction et calculer la clé privée à partir de la clé publique. Cette astuce mathématique devient la base de signatures numériques infalsifiables et sécurisées qui prouvent la propriété de

fonds Bitcoin.

Explication de la cryptographie à courbe elliptique

La cryptographie à courbe elliptique est un type de cryptographie asymétrique ou à clé publique basée sur le problème du logarithme discret exprimé par addition et multiplication sur les points d'une courbe elliptique.

Une courbe elliptique est un exemple de courbe elliptique, similaire à celle utilisée par Bitcoin.

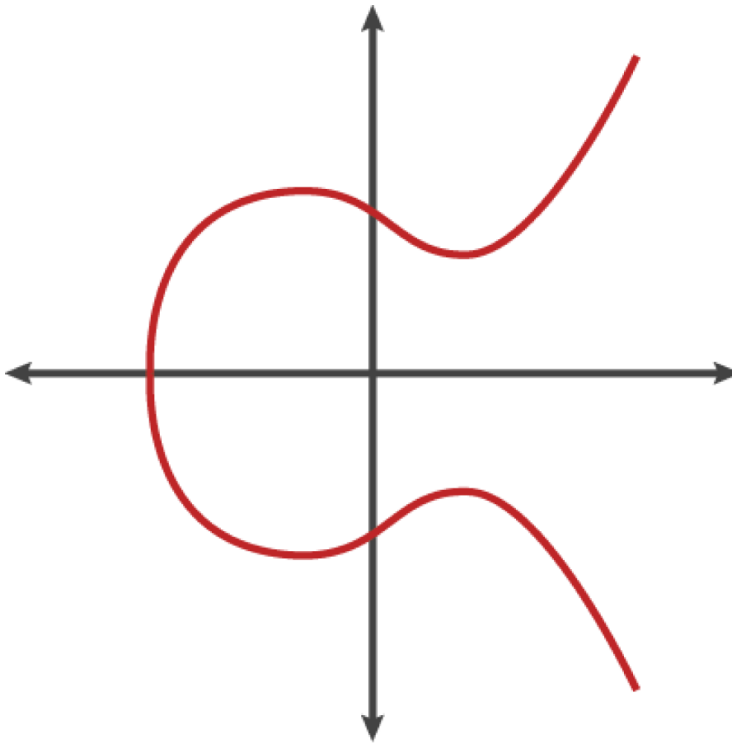


Figure 2. Une courbe elliptique

Bitcoin utilise une courbe elliptique spécifique et un ensemble de constantes mathématiques, tels que définis dans une norme appelée secp256k1, établie par le National Institute of Standards and Technology (NIST). La courbe secp256k1 est définie par la fonction suivante, qui produit une courbe elliptique :

\ [\ begin {équation} {y ^ 2 = (x ^ 3 + 7)} ~ \ text {over} ~ (\ mathbb {F} _p) \ end {équation} \]

ou alors

\ [\ begin {équation} {y ^ 2 \ mod p = (x ^ 3 + 7) \ mod p} \ end {équation} \]

Le *mod p* (nombre premier modulo p) indique que cette courbe est sur un corps fini d'ordre premier *p* , également écrit \ (\ (\ mathbb {F} _p \) \), où p = 2 256 - 2 32 - 2 9 - 2 8 - 2 7 - 2 6 - 2 4 - 1, un très grand nombre premier.

Parce que cette courbe est définie sur un champ fini d'ordre premier plutôt que sur les nombres réels, elle ressemble à un motif de points dispersés en deux dimensions, ce qui la rend difficile à visualiser. Cependant, le calcul est identique à celui d'une courbe elliptique sur des nombres réels. À titre d'exemple, la cryptographie à courbe elliptique : la visualisation d'une courbe elliptique sur F (p), avec p = 17 montre la même courbe elliptique sur un champ fini beaucoup plus petit d'ordre premier 17, montrant un motif de points sur une grille. La courbe elliptique bitcoin secp256k1 peut être considérée comme un motif beaucoup plus complexe de points sur une grille d'une dimension insondable.

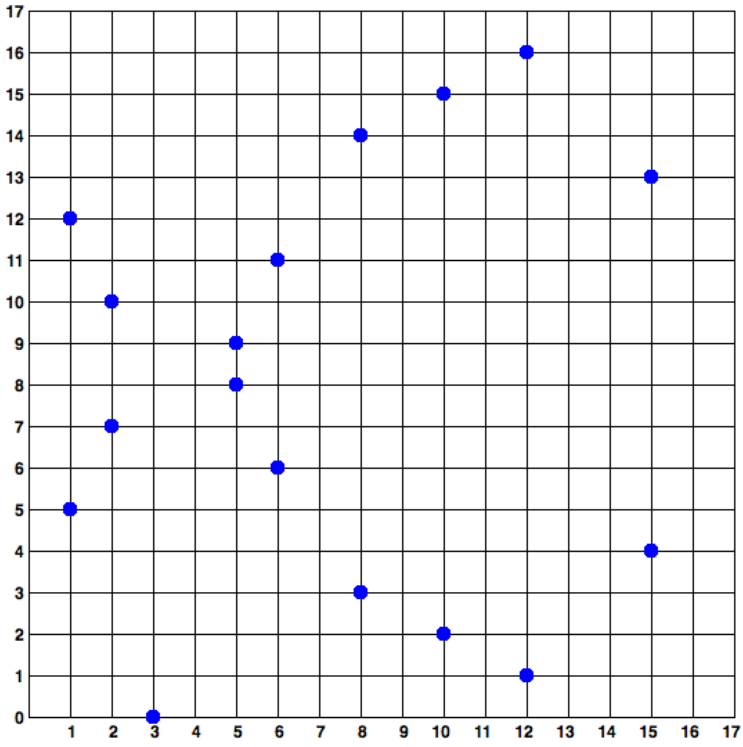


Figure 3. Cryptographie à courbe elliptique : visualisation d'une courbe elliptique sur F (p), avec p = 17

Ainsi, par exemple, ce qui suit est un point P avec des coordonnées (x, y) qui est un point sur la courbe secp256k1 :

```
P =
(55066263022277343669578718895168534326250603453777594175500187360
3267051002075881697808308513050704318447127338065924327593890433
```

Utiliser Python pour confirmer que ce point est sur la courbe elliptique montre comment vous pouvez le vérifier vous-même en utilisant Python :

Exemple 1. Utilisation de Python pour confirmer que ce point est sur la courbe elliptique

```
Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1
(clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> p =
1157920892373161954235709850086879078532699846656405640394575840077
>>> x =
55066263022277343669578718895168534326250603453777594175500187360389
>>> y =
32670510020758816978083085130507043184471273380659243275938904335757
>>> (x ** 3 + 7 - y**2) % p
0
```

En mathématiques de courbe elliptique, il y a un point appelé le « point à l'infini », qui correspond à peu près au rôle de zéro en plus. Sur les ordinateurs, il est parfois représenté par x = y = 0 (ce qui ne satisfait pas l'équation de la courbe elliptique, mais c'est un cas distinct facile qui peut être vérifié).

Il existe également un opérateur +, appelé « addition », qui possède des propriétés similaires à l'addition traditionnelle de nombres réels que les élèves apprennent. Étant donné deux points P 1 et P 2 sur la courbe elliptique, il y a un troisième

point P 3 = P 1 + P 2 , également sur la courbe elliptique.

Géométriquement, ce troisième point P 3 est calculé en traçant une ligne entre P 1 et P 2 . Cette ligne coupera la courbe elliptique à exactement un endroit supplémentaire. Appelez ce point P 3 '= (x, y). Réfléchissez ensuite sur l'axe des x pour obtenir P 3 = (x, –y).

Il existe quelques cas particuliers qui expliquent la nécessité du « point à l'infini ».

Si P 1 et P 2 sont le même point, la ligne "entre" P 1 et P 2 doit s'étendre pour être la tangente sur la courbe en ce point P 1 . Cette tangente coupera la courbe en exactement un nouveau point. Vous pouvez utiliser des techniques de calcul pour déterminer la pente de la ligne tangente. Ces techniques fonctionnent curieusement, même si nous limitons notre intérêt aux points de la courbe avec deux coordonnées entières !

Dans certains cas (c'est-à-dire si P 1 et P 2 ont les mêmes valeurs x mais des valeurs y différentes), la ligne entre P 1 et P 2 sera exactement verticale, auquel cas P 3 = "point à l'infini".

Si P 1 est le "point à l'infini", alors P 1 + P 2 = P 2 . De même, si P 2 est le point à l'infini, alors P 1 + P 2 = P 1 . Cela montre comment le point à l'infini joue le rôle de zéro.

Il s'avère que + est associatif, ce qui signifie que (A + B) + C = A + (B + C). Cela signifie que nous pouvons écrire A + B + C sans parenthèses et sans ambiguïté.

Maintenant que nous avons défini l'addition, nous pouvons définir la multiplication de la manière standard qui étend l'addition. Pour un point P sur la courbe elliptique, si k est un nombre entier, alors kP = P + P + P +… + P (k fois). Notez que k est parfois appelé "exposant" de manière déroutante dans ce cas.

Générer une clé publique

En partant d'une clé privée sous la forme d'un nombre *k* généré aléatoirement , on la multiplie par un point prédéterminé sur la courbe appelé *point générateur G* pour produire un autre point ailleurs sur la courbe, qui est la clé publique *K* correspondante . Le point générateur est spécifié dans le cadre de la norme secp256k1 et est toujours le même pour toutes les clés en bitcoin :

\ [\ begin {équation} {K = k * G} \ end {équation} \]

où *k* est la clé privée, *G* est le point générateur et *K* est la clé publique résultante, un point sur la courbe. Du fait que le point de générateur est toujours le même pour tous les utilisateurs de Bitcoin, une clé privée *k* multipliée par *G* entraîne toujours la même clé publique *K* . La relation entre *k* et *K* est fixe, mais ne peut être calculée dans un sens, à partir de *k* de *K* . C'est pourquoi une adresse bitcoin (dérivée de *K*) peut être partagée avec n'importe qui et ne révèle pas la clé privée (*k*) de l'utilisateur.

Conseil

Une clé privée peut être convertie en clé publique, mais une clé publique ne peut pas être reconvertie en clé privée

car le calcul ne fonctionne que dans un seul sens.

En implémentant la multiplication de la courbe elliptique, nous prenons la clé privée *k* générée précédemment et la multiplions par le point générateur G pour trouver la clé publique *K* :

```
K =
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
* G
```

En implémentant la multiplication de la courbe elliptique, nous prenons la clé privée *k* générée précédemment et la multiplions par le point générateur G pour trouver la clé publique *K* :

```
K = (x, y)

where,

x =
F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y =
07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Pour visualiser la multiplication d'un point par un entier, nous utiliserons la courbe elliptique plus simple sur des nombres réels - rappelez-vous, le calcul est le même. Notre objectif est de trouver le multiple kG du point générateur G , ce qui revient à ajouter G à lui-même, k fois de suite. Dans les courbes elliptiques, l'ajout d'un point à lui-même équivaut à dessiner une ligne tangente sur le point et à trouver à nouveau son

intersection avec la courbe, puis à refléter ce point sur l'axe des x.

Cryptographie à courbe elliptique : la visualisation de la multiplication d'un point G par un entier k sur une courbe elliptique montre le processus de dérivation de *G* , *2G* , *4G* et *8G* comme une opération géométrique sur la courbe.

Conseil

Bitcoin utilise la bibliothèque C optimisée secp256k1 pour faire le calcul de la courbe elliptique.

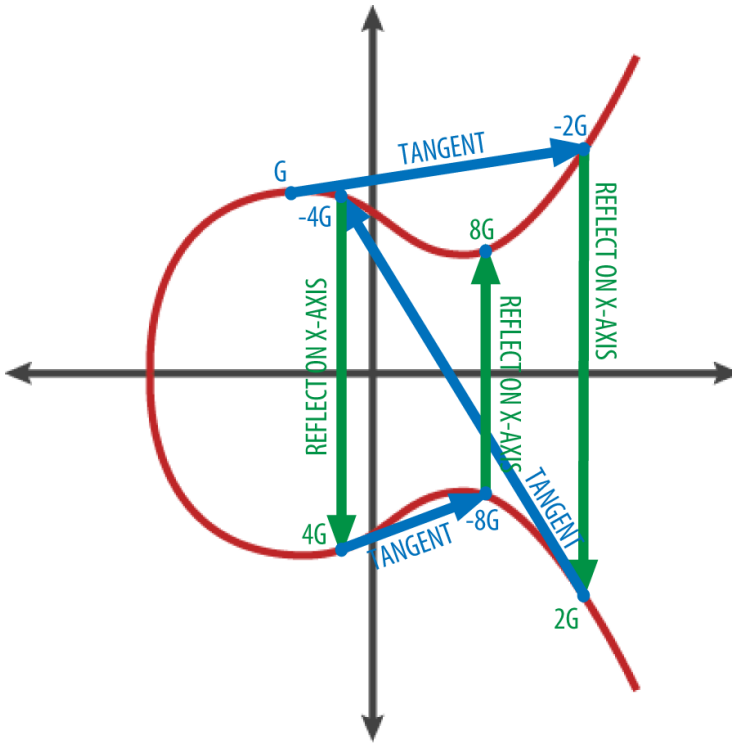


Figure 4. Cryptographie à courbe elliptique : visualisation de la multiplication d'un point G par un entier k sur une courbe elliptique

Adresses Bitcoin

Une adresse bitcoin est une chaîne de chiffres et de caractères qui peut être partagée avec toute personne souhaitant vous envoyer de l'argent. Les adresses produites à partir de clés publiques se composent d'une chaîne de chiffres et de lettres

commençant par le chiffre « 1 ». Voici un exemple d'adresse bitcoin :

```
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

L'adresse bitcoin est ce qui apparaît le plus souvent dans une transaction en tant que « destinataire » des fonds. Si nous comparons une transaction Bitcoin à un chèque papier, l'adresse Bitcoin est le bénéficiaire, c'est ce que nous écrivons sur la ligne après "Payer à l'ordre de". Sur un chèque papier, ce bénéficiaire peut parfois être le nom d'un titulaire de compte bancaire, mais peut également inclure des sociétés, des institutions ou même des espèces. Étant donné que les chèques papier n'ont pas besoin de spécifier un compte, mais plutôt d'utiliser un nom abstrait en tant que destinataire des fonds, ils sont des instruments de paiement très flexibles. Les transactions Bitcoin utilisent une abstraction similaire, l'adresse bitcoin, pour les rendre très flexibles. Une adresse bitcoin peut représenter le propriétaire d'une paire de clés privée / publique, ou elle peut représenter autre chose, comme un script de paiement, comme nous le verrons dans [p2sh] . Pour l'instant, examinons le cas simple, une adresse bitcoin qui représente et est dérivée d'une clé publique.

L'adresse bitcoin est dérivée de la clé publique grâce à l'utilisation d'un hachage cryptographique à sens unique. Un « algorithme de hachage » ou simplement « algorithme de hachage » est une fonction unidirectionnelle qui produit une empreinte digitale ou un « hachage » d'une entrée de taille arbitraire.

Les fonctions de hachage cryptographiques sont largement utilisées dans bitcoin : dans les adresses bitcoin, dans les adresses de script et dans l'algorithme de preuve de travail de minage. Les algorithmes utilisés pour créer une adresse bitcoin à partir d'une clé publique sont le Secure Hash Algorithm (SHA) et le RACE Integrity Primitives Evaluation Message Digest (RIPEMD), en particulier SHA256 et RIPEMD160.

En commençant par la clé publique *K* , nous calculons le hachage SHA256, puis nous calculons le hachage RIPEMD160 du résultat, produisant un nombre de 160 bits (20 octets) :

\ [\ begin {équation} {A = RIPEMD160 (SHA256 (K))} \ end {équation} \]

où *K* est la clé publique et *A* est l'adresse bitcoin résultante.

> *Conseil*
>
> *Une adresse bitcoin n'est* pas *la même chose qu'une clé publique. Les adresses Bitcoin sont dérivées d'une clé publique à l'aide d'une fonction unidirectionnelle.*

Les adresses Bitcoin sont presque toujours codées en tant que « Base58Check » (voir Encodage Base58 et Base58Check), qui utilise 58 caractères (un système de numérotation Base58) et une somme de contrôle pour améliorer la lisibilité humaine, éviter toute ambiguïté et se protéger contre les erreurs de transcription et de saisie d'adresses. Base58Check est également utilisé de nombreuses autres manières dans Bitcoin, chaque fois qu'un utilisateur a besoin de lire et de transcrire correctement

un nombre, tel qu'une adresse bitcoin, une clé privée, une clé cryptée ou un hachage de script. Dans la section suivante, nous examinerons la mécanique de l'encodage et du décodage Base58Check et les représentations qui en résultent. Clé publique en adresse bitcoin : la conversion d'une clé publique en adresse bitcoin illustre la conversion d'une clé publique en adresse bitcoin.

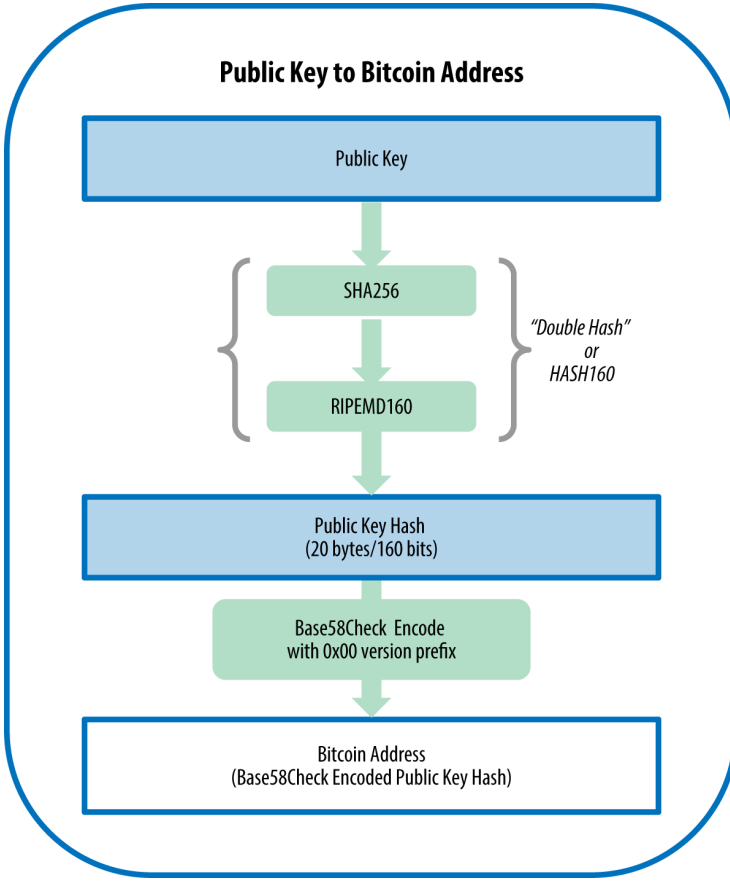


Figure 5. Clé publique en adresse bitcoin : conversion d'une clé publique en adresse bitcoin

Encodage Base58 et Base58Check

Afin de représenter des nombres longs de manière compacte, en utilisant moins de symboles, de nombreux systèmes infor-

matiques utilisent des représentations alphanumériques mixtes avec une base (ou base) supérieure à 10. Par exemple, alors que le système décimal traditionnel utilise les 10 chiffres 0 à 9, le système hexadécimal utilise 16, avec les lettres A à F comme six symboles supplémentaires. Un nombre représenté au format hexadécimal est plus court que la représentation décimale équivalente. Encore plus compacte, la représentation Base64 utilise 26 lettres minuscules, 26 lettres majuscules, 10 chiffres et 2 autres caractères tels que « ‘ ‘& # x201d ; et “/” pour transmettre des données binaires sur des supports textuels tels que le courrier électronique. Base64 est le plus couramment utilisé pour ajouter des pièces jointes binaires à un e-mail. Base58 est un format d’encodage binaire basé sur du texte développé pour être utilisé dans Bitcoin et utilisé dans de nombreuses autres crypto-monnaies. Il offre un équilibre entre représentation compacte, lisibilité et détection et prévention des erreurs. Base58 est un sous-ensemble de Base64, utilisant des lettres et des chiffres majuscules et minuscules, mais omettant certains caractères qui sont souvent confondus et peuvent apparaître identiques lorsqu’ils sont affichés dans certaines polices. Plus précisément, Base58 est Base64 sans le 0 (numéro zéro), O (majuscule o), l (L inférieur), I (majuscule i) et les symboles & # x201c ; “ ”et” / “. Ou, plus simplement, il s’agit d’un ensemble de lettres minuscules et majuscules et de chiffres sans les quatre (0, O, l, I) qui viennent d’être mentionnés. L’alphabet Base58 de Bitcoin montre l’alphabet Base58 complet.

Exemple 2. Alphabet Base58 de Bitcoin

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
```

Pour ajouter une sécurité supplémentaire contre les fautes de frappe ou les erreurs de transcription, Base58Check est un format d'encodage Base58, fréquemment utilisé dans Bitcoin, qui possède un code de vérification des erreurs intégré. La somme de contrôle correspond à quatre octets supplémentaires ajoutés à la fin des données en cours de codage. La somme de contrôle est dérivée du hachage des données codées et peut donc être utilisée pour détecter et prévenir les erreurs de transcription et de frappe. Lorsqu'il est présenté avec le code Base58Check, le logiciel de décodage calculera la somme de contrôle des données et la comparera à la somme de contrôle incluse dans le code. Si les deux ne correspondent pas, une erreur a été introduite et les données Base58Check ne sont pas valides. Cela empêche une adresse bitcoin mal saisie d'être acceptée par le logiciel de portefeuille comme destination valide, une erreur qui entraînerait autrement une perte de fonds.

Pour convertir des données (un nombre) dans un format Base58Check, nous ajoutons d'abord un préfixe aux données, appelé « octet de version », qui sert à identifier facilement le type de données encodées. Par exemple, dans le cas d'une adresse bitcoin, le préfixe est zéro (0x00 en hexadécimal), alors que le préfixe utilisé lors de l'encodage d'une clé privée est 128 (0x80 en hexadécimal). Une liste des préfixes de version courants est affichée dans le préfixe de version Base58Check et les exemples de résultats codés .

Ensuite, nous calculons la somme de contrôle "double-SHA", ce qui signifie que nous appliquons deux fois l'algorithme de hachage SHA256 sur le résultat précédent (préfixe et données) :

```
checksum = SHA256(SHA256(prefix+data))
```

À partir du hachage de 32 octets résultant (hachage d'un hachage), nous ne prenons que les quatre premiers octets. Ces quatre octets servent de code de vérification d'erreur ou de somme de contrôle. La somme de contrôle est concaténée (ajoutée) à la fin.

Le résultat est composé de trois éléments : un préfixe, les données et une somme de contrôle. Ce résultat est encodé à l'aide de l'alphabet Base58 décrit précédemment. Encodage Base58Check : un format Base58, versionné et total de contrôle pour encoder sans ambiguïté les données Bitcoin illustre le processus d'encodage Base58Check.

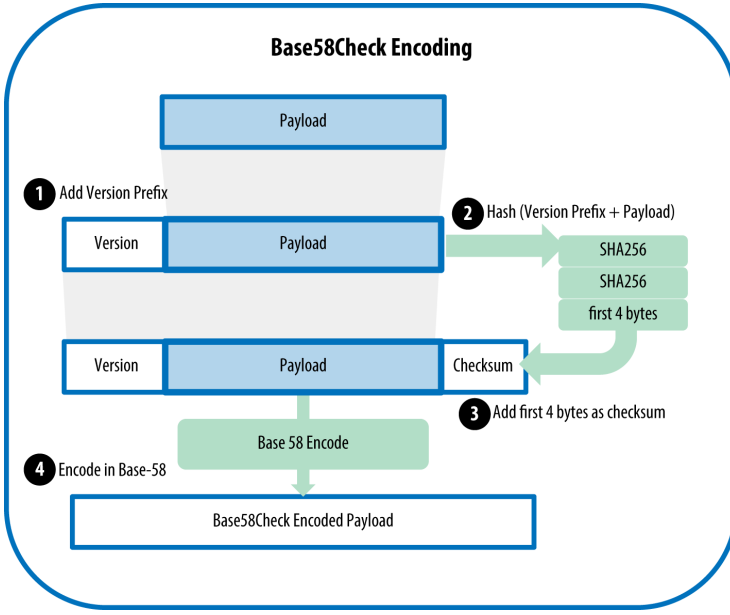


Figure 6. Encodage Base58Check : un format Base58, versionné et total de contrôle pour encoder sans ambiguïté des données Bitcoin

Dans Bitcoin, la plupart des données présentées à l'utilisateur sont encodées en Base58Check pour le rendre compact, facile à lire et facile à détecter les erreurs. Le préfixe de version dans le codage Base58Check est utilisé pour créer des formats facilement reconnaissables qui, lorsqu'ils sont codés en Base58, contiennent des caractères spécifiques au début de la charge utile codée en Base58Check. Ces caractères permettent aux humains d'identifier facilement le type de données encodées et comment les utiliser. C'est ce qui différencie, par exemple, une adresse bitcoin encodée en Base58Check qui commence par un 1 d'une clé privée WIF encodée en Base58Check qui

commence par un 5. Certains exemples de préfixes de version et les caractères Base58 résultants sont affichés dans le préfixe de version Base58Check et encodés exemples de résultats .

Tableau 1. Préfixe de version de Base58Check et exemples de résultats codés

Type	Version prefix (hex)	Base58 result prefix
Bitcoin Address	0x00	1
Pay-to-Script-Hash Address	0x05	3
Bitcoin Testnet Address	0x6F	m or n
Private Key WIF	0x80	5, K, or L
BIP-38 Encrypted Private Key	0x0142	6P
BIP-32 Extended Public Key	0x0488B21E	xpub

Formats clés

Les clés privées et publiques peuvent être représentées dans un certain nombre de formats différents. Ces représentations codent toutes le même nombre, même si elles semblent différentes. Ces formats sont principalement utilisés pour faciliter la lecture et la transcription des clés sans introduire d'erreurs.

Formats de clé privée

La clé privée peut être représentée dans un certain nombre de formats différents, qui correspondent tous au même nombre de 256 bits. Les représentations de clé privée (formats de codage) montrent trois formats courants utilisés pour représenter les clés privées. Différents formats sont utilisés dans différentes circonstances. Les formats binaires hexadécimaux et bruts sont utilisés en interne dans les logiciels et rarement montrés aux utilisateurs. Le WIF est utilisé pour l'importation / exportation de clés entre les portefeuilles et souvent utilisé dans les représentations de code QR (code-barres) de clés privées.

Tableau 2. Représentations des clés privées (formats de codage)

Type	Prefix	Description
Raw	None	32 bytes
Hex	None	64 hexadecimal digits
WIF	5	Base58Check encoding: Base58 with version prefix of 0x80 and 4-byte checksum
WIF-compressed	K or L	As above, with added suffix 0x01 before encoding

Exemple : Même clé, différents formats affiche la clé privée générée dans ces trois formats.

Tableau 3. Exemple : même clé, formats différents

Format	Private key
Hex	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526a
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Toutes ces représentations sont différentes manières d'afficher le même numéro, la même clé privée. Ils ont un aspect différent, mais n'importe quel format peut facilement être converti en n'importe quel autre format. Notez que le "binaire brut" n'est pas montré dans l' exemple : Même clé, formats différents comme tout encodage pour l'affichage ici ne seraient, par définition, pas des données binaires brutes.

Nous utilisons la commande wif -to-ec de Bitcoin Explorer (voir [appdx_bx]) pour montrer que les deux clés WIF représentent la même clé privée :

```
$ bx wif-to-ec
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```

Décoder à partir de Base58Check

Les commandes de Bitcoin Explorer (voir [appdx_bx]) facilitent l'écriture de scripts shell et de "tubes" de ligne de commande qui manipulent les clés, les adresses et les transactions Bitcoin. Vous pouvez utiliser Bitcoin Explorer pour décoder le format Base58Check sur la ligne de commande.

Nous utilisons la commande base58check-decode pour décoder la clé non compressée :

```
$ bx base58check-decode
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
wrapper
{
    checksum 4286807748
    payload
    1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526a
    version 128
}
```

Le résultat contient la clé comme charge utile, le préfixe de version WIF 128 et une somme de contrôle.

Notez que la "charge utile" de la clé compressée est ajoutée avec le suffixe 01, signalant que la clé publique dérivée doit être compressée :

```
$ bx base58check-decode
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
wrapper
{
    checksum 2339607926
    payload
```

```
    1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526ae
    version 128
}
```

Encoder de hex à Base58Check

Pour encoder dans Base58Check (le contraire de la commande précédente), nous utilisons la commande base58check-encode de Bitcoin Explorer (voir [appdx_bx]) et fournissons la clé privée hexadécimale, suivie du préfixe de version WIF 128 :

```
bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
--version 128
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
```

Encoder de hex (clé compressée) à Base58Check

Pour encoder dans Base58Check en tant que clé privée « compressée » (voir Clés privées compressées), nous ajoutons le suffixe 01 à la clé hexadécimale puis encodons comme dans la section précédente :

```
$ bx base58check-encode
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
--version 128
KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
```

Le format compressé WIF résultant commence par un “K.” Cela

indique que la clé privée à l'intérieur a un suffixe de «01» et sera utilisée pour produire des clés publiques compressées uniquement (voir Clés publiques compressées).

Formats de clé publique

Les clés publiques sont également présentées de différentes manières, généralement sous forme de clés publiques *compressées* ou *non compressées* .

Comme nous l'avons vu précédemment, la clé publique est un point sur la courbe elliptique constitué d'une paire de coordonnées (x, y). Il est généralement présenté avec le préfixe 04 suivi de deux nombres de 256 bits : l'un pour la coordonnée *x* du point, l'autre pour la coordonnée *y* . Le préfixe 04 est utilisé pour distinguer les clés publiques non compressées des clés publiques compressées commençant par 02 ou 03.

Voici la clé publique générée par la clé privée que nous avons créée précédemment, représentée par les coordonnées x et y :

```
x =
F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
y =
07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Voici la même clé publique affichée sous forme de nombre de 520 bits (130 chiffres hexadécimaux) avec le préfixe 04 suivi de x puis de coordonnées y, comme 04 xy :

```
K =
04↵F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC34
07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
```

Clés publiques compressées

Les clés publiques compressées ont été introduites dans le bitcoin pour réduire la taille des transactions et conserver de l'espace disque sur les nœuds qui stockent la base de données de la blockchain bitcoin. La plupart des transactions incluent la clé publique, qui est nécessaire pour valider les informations d'identification du propriétaire et dépenser le bitcoin. Chaque clé publique nécessite 520 bits (préfixe + x + y), qui, multipliés par plusieurs centaines de transactions par bloc, soit des dizaines de milliers de transactions par jour, ajoutent une quantité importante de données à la blockchain.

Comme nous l'avons vu dans la section Clés publiques , une clé publique est un point (x, y) sur une courbe elliptique. Parce que la courbe exprime une fonction mathématique, un point sur la courbe représente une solution à l'équation et, par conséquent, si nous connaissons la coordonnée *x* , nous pouvons calculer la coordonnée *y* en résolvant l'équation y 2 mod p = (x 3 + 7) mod p. Cela nous permet de stocker uniquement la coordonnée *x* du point de clé publique, en omettant la coordonnée *y* et en réduisant la taille de la clé et l'espace requis pour la stocker de 256 bits. Une réduction de près de 50% de la taille de chaque transaction s'ajoute à beaucoup de données sauvegardées au fil du temps !

Alors que les clés publiques non compressées ont un préfixe 04, les clés publiques compressées commencent par un préfixe 02 ou 03. Voyons pourquoi il y a deux préfixes possibles : parce que le côté gauche de l'équation est y^2, la solution pour y est une racine carrée, qui peut avoir une valeur positive ou négative. Visuellement, cela signifie que la coordonnée y résultante peut être au-dessus ou au-dessous de l'axe des x. Comme vous pouvez le voir sur le graphique de la courbe elliptique dans Une courbe elliptique, la courbe est symétrique, ce qui signifie qu'elle est réfléchie comme un miroir par l'axe des x. Ainsi, bien que nous puissions omettre la coordonnée y, nous devons stocker le *signe* de y (positif ou négatif) ; ou en d'autres termes, nous devons nous rappeler s'il était au-dessus ou en dessous de l'axe des x car chacune de ces options représente un point différent et une clé publique différente. Lors du calcul de la courbe elliptique en arithmétique binaire sur le corps fini d'ordre premier p, la coordonnée y est paire ou impaire, ce qui correspond au signe positif / négatif comme expliqué précédemment. Par conséquent, pour distinguer les deux valeurs possibles de y, nous stockons une clé publique compressée avec le préfixe 02 si le y est pair, et 03 s'il est impair, permettant au logiciel de déduire correctement la coordonnée y de la coordonnée x et de décompresser la clé publique des coordonnées complètes du point. La compression de clé publique est illustrée dans la compression de clé publique .

Voici la même clé publique générée précédemment, représentée comme une clé publique compressée stockée sur 264 bits (66 chiffres hexadécimaux) avec le préfixe 03 indiquant que la coordonnée y est impaire :

```
K =
03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
```

Cette clé publique compressée correspond à la même clé privée, c'est-à-dire qu'elle est générée à partir de la même clé privée. Cependant, il semble différent de la clé publique non compressée. Plus important encore, si nous convertissons cette clé publique compressée en une adresse bitcoin à l'aide de la fonction de double hachage (RIPEMD160 (SHA256 (K))), cela produira une adresse bitcoin *différente* . Cela peut prêter à confusion, car cela signifie qu'une seule clé privée peut produire une clé publique exprimée dans deux formats différents (compressé et non compressé) qui produisent deux adresses bitcoin différentes. Cependant, la clé privée est identique pour les deux adresses bitcoin.

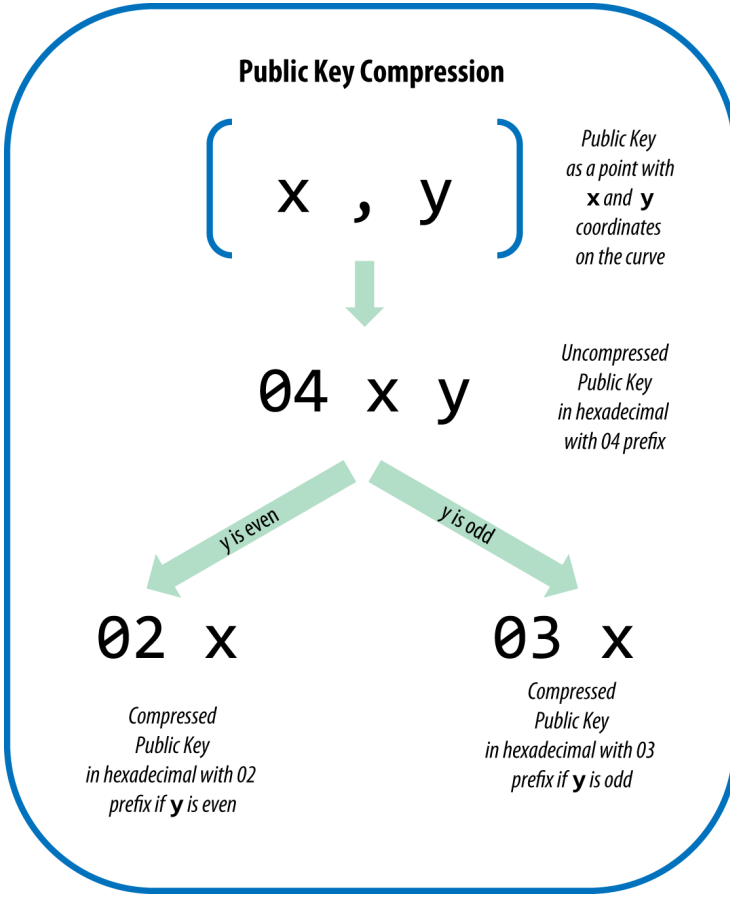


Figure 7. Compression de clé publique

Les clés publiques compressées deviennent progressivement la valeur par défaut des clients Bitcoin, ce qui a un impact significatif sur la réduction de la taille des transactions et donc de la blockchain. Cependant, tous les clients ne prennent pas encore en charge les clés publiques compressées. Les clients plus

récents qui prennent en charge les clés publiques compressées doivent tenir compte des transactions des clients plus anciens qui ne prennent pas en charge les clés publiques compressées. Ceci est particulièrement important lorsqu'une application de portefeuille importe des clés privées à partir d'une autre application de portefeuille Bitcoin, car le nouveau portefeuille doit analyser la blockchain pour trouver les transactions correspondant à ces clés importées. Quelles adresses Bitcoin le portefeuille Bitcoin doit-il rechercher ? Les adresses bitcoin produites par des clés publiques non compressées, ou les adresses bitcoin produites par des clés publiques compressées ? Les deux sont des adresses bitcoin valides et peuvent être signées par la clé privée, mais ce sont des adresses différentes !

Pour résoudre ce problème, lorsque les clés privées sont exportées à partir d'un portefeuille, le WIF utilisé pour les représenter est implémenté différemment dans les nouveaux portefeuilles Bitcoin, pour indiquer que ces clés privées ont été utilisées pour produire des clés publiques *compressées* et donc *des* adresses Bitcoin *compressées* . Cela permet au portefeuille importateur de faire la distinction entre les clés privées provenant de portefeuilles plus anciens ou plus récents et de rechercher dans la blockchain des transactions avec des adresses bitcoin correspondant respectivement aux clés publiques non compressées ou compressées. Voyons comment cela fonctionne plus en détail, dans la section suivante.

Clés privées compressées

Ironiquement, le terme « clé privée compressée » est un abus de langage, car lorsqu'une clé privée est exportée sous forme de

compression WIF, elle est en fait un octet de *plus* qu'une clé privée « non compressée ». En effet, la clé privée a un suffixe d'un octet ajouté (affiché comme 01 en hexadécimal dans Exemple : même clé, formats différents), ce qui signifie que la clé privée provient d'un portefeuille plus récent et ne doit être utilisée que pour produire des clés publiques compressées . Les clés privées ne sont pas elles-mêmes compressées et ne peuvent pas être compressées. Le terme « clé privée compressée » signifie en réalité « clé privée à partir de laquelle seules les clés publiques compressées doivent être dérivées », alors que « clé privée non compressée » signifie en réalité « clé privée à partir de laquelle seules les clés publiques non compressées doivent être dérivées ». Vous devez uniquement faire référence au format d'exportation comme « compressé WIF » ou « WIF » et ne pas faire référence à la clé privée elle-même comme « compressée » pour éviter toute confusion supplémentaire.

Exemple : même clé, différents formats affiche la même clé, codée aux formats compressés WIF et WIF.

Tableau 4. Exemple : même clé, formats différents

Format	Private key
Hex	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
WIF	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Hex-compressed	1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD01
WIF-compressed	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ

Figure 8. Un exemple de portefeuille en papier simple

Notez que le format de clé privée compressée en hexadécimal a un octet supplémentaire à la fin (01 en hexadécimal). Alors que le préfixe de version Base58Check est le même (0x80) pour les formats compressés WIF et WIF, l'ajout d'un octet à la fin du nombre fait passer le premier caractère de l'encodage Base58 d'un 5 à un *K* ou *L* . Considérez cela comme l'équivalent Base58 de la différence de codage décimal entre le nombre 100 et le nombre 99. Alors que 100 est un chiffre plus long que 99, il a également un préfixe de 1 au lieu d'un préfixe de 9. Lorsque la longueur change, il affecte le préfixe. En Base58, le préfixe 5 se transforme en *K* ou *L* lorsque la longueur du nombre augmente d'un octet.

N'oubliez pas que ces formats *ne* sont *pas* utilisés de manière interchangeable. Dans un portefeuille plus récent qui implémente des clés publiques compressées, les clés privées ne seront jamais exportées que sous forme de compression WIF (avec un préfixe *K* ou *L*). Si le portefeuille est une implémentation plus ancienne et n'utilise pas de clés publiques compressées, les clés privées ne seront exportées qu'au format WIF (avec un préfixe 5). Le but ici est de signaler au portefeuille qui importe ces clés privées s'il doit rechercher dans la blockchain des clés et des adresses publiques compressées ou non.

Si un portefeuille Bitcoin est capable d'implémenter des clés publiques compressées, il les utilisera dans toutes les transactions. Les clés privées du portefeuille seront utilisées pour dériver les points de clé publique sur la courbe, qui seront compressés. Les clés publiques compressées seront utilisées pour produire des adresses bitcoin et celles-ci seront utilisées dans les transactions. Lors de l'exportation de clés privées à partir d'un nouveau

portefeuille qui implémente des clés publiques compressées, le WIF est modifié, avec l'ajout d'un suffixe d'un octet 01 à la clé privée. La clé privée encodée en Base58Check qui en résulte est appelée « WIF compressé » et commence par la lettre *K* ou *L* , au lieu de commencer par « 5 » comme c'est le cas avec les clés encodées WIF (non compressées) des anciens portefeuilles.

> *Conseil*
>
> *"Clés privées compressées" est un abus de langage ! Ils ne sont pas compressés ; plutôt, WIF-compressé signifie que les clés ne doivent être utilisées que pour dériver des clés publiques compressées et leurs adresses bitcoin correspondantes. Ironiquement, une clé privée encodée "WIF-compressée" est un octet de plus car elle a le suffixe 01 ajouté pour la distinguer d'une clé "non compressée".*

Implémentation de clés et d'adresses en C ++

Regardons le processus complet de création d'une adresse bitcoin, d'une clé privée, à une clé publique (un point sur la courbe elliptique), à une adresse à double hachage, et enfin, l'encodage Base58Check. Le code C ++ dans Création d'une adresse bitcoin encodée en Base58Check à partir d'une clé privée montre le processus étape par étape complet, de la clé privée à l'adresse bitcoin encodée en Base58Check. L'exemple de code utilise la bibliothèque libbitcoin introduite dans [alt_libraries] pour certaines fonctions d'assistance.

Exemple 3. Création d'une adresse bitcoin encodée en

Base58Check à partir d'une clé privée

```
link:code/addr.cpp[]
```

Le code utilise une clé privée prédéfinie pour produire la même adresse bitcoin à chaque fois qu'il est exécuté, comme indiqué dans Compilation et exécution du code addr .

Exemple 4. Compilation et exécution du code addr

```
# Compile the addr.cpp code
$ g++ -o addr addr.cpp -std=c++11 $(pkg-config
--cflags --libs libbitcoin)# Run the addr executable
$ ./addr
Public key:
0202a406624211f2abbdc68da3df929f938c3399dd79fac1b51b0e4ad1d26a47aa
Address: 1PRTTaJesdNovgne6Ehcdu1fpEdX7913CK
```

Conseil

Le code dans la compilation et l'exécution du code addr produit une adresse bitcoin (1PRTT ...) à partir d'une clé publique compressée *(voir Clés publiques compressées). Si vous utilisiez la clé publique non compressée à la place, cela produirait une adresse bitcoin différente (14K1y ...).*

Implémentation de clés et d'adresses en Python

La bibliothèque Bitcoin la plus complète en Python est pybitcointools de Vitalik Buterin. Dans la génération et le formatage de clés et d'adresses avec la bibliothèque pybitcointools , nous utilisons la bibliothèque pybitcointools (importée en tant que "bitcoin") pour générer et afficher des clés et des adresses dans différents formats.

Exemple 5. Génération et formatage de clés et d'adresses avec la bibliothèque pybitcointools

```
link:code/key-to-address-ecc-example.py[]
```

L'exécution de key-to-address-ecc-example.py montre la sortie de l'exécution de ce code.

Exemple 6. Exécution de key-to-address-ecc-example.py

```
$ python key-to-address-ecc-example.py
Private Key (hex) is:
 3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (decimal) is:
 26563230048437957592232553826663696440606756685920117476832299
Private Key (WIF) is:
 5JG9hT3beGTJuUAmCQEmNaxAuMacCTfXuw1R3FCXig23RQHMr4K
Private Key Compressed (hex) is:
 3aba4162c7251c891207b747840551a71939b0de081f85c4e44cf7c13e41daa6
Private Key (WIF-Compressed) is:
```

```
 KyBsPXxTuVD82av65KZkrGrWi5qLMah5SdNq6uftawDbgKa2wv6S
Public Key (x,y) coordinates is:
 (41637322786646325214887832269588396900663353932545912953362782457
 16388935128781238405526710466724741593761085120864331449066658622
Public Key (hex) is:
 045↵c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b0
243bcefdd4347074d44bd7356d6a53c495737dd96295e2a9374bf5f02ebfc176
Compressed Public Key (hex) is:
 025c0de3b9c8ab18dd04e3511243ec2952002dbfadc864b9628910169d9b9b00e
Bitcoin Address (b58check) is:
 1thMirt546nngXqyPEz532S8fLwbozud8
Compressed Bitcoin Address (b58check) is:
 14cxpo3MBCYYWCgF74SWTdcmxipnGUsPw3
```

Un script illustrant les mathématiques de courbes elliptiques utilisées pour les clés bitcoin est un autre exemple, utilisant la bibliothèque Python ECDSA pour les mathématiques de courbes elliptiques et sans utiliser de bibliothèques Bitcoin spécialisées.

Exemple 7. Un script démontrant les mathématiques de courbe elliptique utilisées pour les clés Bitcoin

```
link:code/ec-math.py[]
```

L'installation de la bibliothèque Python ECDSA et l'exécution du script ec_math.py affiche la sortie produite par l'exécution de ce script.

Avertissement

Un script démontrant la courbe mathématique elliptique utilisée pour les clés bitcoin utilise os.urandom, qui reflète un générateur de nombres aléatoires cryptographiquement sécurisé (CSRNG) fourni par le système d'exploitation sous-jacent. Attention : selon le système d'exploitation, os.urandom peut ne pas *être implémenté avec une sécurité suffisante ou correctement amorcé et peut* ne pas *être approprié pour générer des clés bitcoin de qualité production.*

Exemple 8. Installation de la bibliothèque Python ECDSA et exécution du script ec_math.py

```
# Install Python PIP package manager
$ sudo apt-get install python-pip
# Install the Python ECDSA library
$ sudo pip install ecdsa
# Run the script
$ python ec-math.py
Secret:
380908350159543588624811326288874439059062049959123782780601687 03
EC point:
(700488535318671794898577504976069662723825834713229354546245955 4
105262206478686743191060800263479589329920209527285803935736021 68
BTC public key:
029ade3effb0a67d5c8609850d797366af428f4a0d5194cb221d807770a15228 7
```

Clés et adresses avancées

Dans les sections suivantes, nous examinerons les formes avancées de clés et d'adresses, telles que les clés privées cryptées, les adresses de script et multi-signatures, les adresses personnalisées et les portefeuilles papier.

Hash Pay-to-Script (P2SH) et adresses multisig

Comme nous le savons, les adresses bitcoin traditionnelles commencent par le chiffre « 1 » et sont dérivées de la clé publique, qui est dérivée de la clé privée. Bien que n'importe qui puisse envoyer du bitcoin à une adresse « 1 », ce bitcoin ne peut être dépensé qu'en présentant la signature de clé privée et la clé publique correspondantes.

Les adresses Bitcoin qui commencent par le chiffre « 3 » sont des adresses de hachage pay-to-script (P2SH), parfois appelées à tort adresses multisignatures ou multisig. Ils désignent le bénéficiaire d'une transaction bitcoin comme hachage d'un script, au lieu du propriétaire d'une clé publique. La fonctionnalité a été introduite en janvier 2012 avec BIP-16 (voir [appdxbitcoinimpproposals]) et est largement adoptée car elle offre la possibilité d'ajouter des fonctionnalités à l'adresse elle-même. Contrairement aux transactions qui « envoient » des fonds à des adresses bitcoin traditionnelles « 1 », également appelées hachage de clé publique (P2PKH), les fonds envoyés à des adresses « 3 » nécessitent plus que la présentation d'une

clé publique et une signature de clé privée comme preuve de propriété. Les exigences sont désignées au moment de la création de l'adresse, dans le script, et toutes les entrées de cette adresse seront encombrées des mêmes exigences.

Une adresse P2SH est créée à partir d'un script de transaction, qui définit qui peut dépenser une sortie de transaction (pour plus de détails, voir [p2sh]). Le codage d'une adresse P2SH implique l'utilisation de la même fonction de double-hachage que celle utilisée lors de la création d'une adresse bitcoin, appliquée uniquement sur le script au lieu de la clé publique :

```
script hash = RIPEMD160(SHA256(script))
```

Le "hachage de script" qui en résulte est codé avec Base58Check avec un préfixe de version de 5, ce qui donne une adresse codée commençant par 3. Un exemple d'adresse P2SH est 3F6i6kwke vjR7AsAd4te2YB2zZyASEm1HM, qui peut être dérivée à l'aide des commandes de Bitcoin Explorer script-encode, sha256, ripemd160 et base58check-encode (voir [appdx_bx]) comme suit :

```
$ echo \
'DUP HASH160
[89abcdefabbaabbaabbaabbaabbaabbaabbaabba]
EQUALVERIFY CHECKSIG' > script
$ bx script-encode < script | bx sha256 | bx
ripemd160 \
```

```
| bx base58check-encode --version 5
3F6i6kwkevjR7AsAd4te2YB2zZyASEm1HM
```

Conseil

P2SH n'est pas nécessairement identique à une transaction standard multisignature. Une adresse P2SH représente le plus souvent *un script multi-signature, mais elle peut également représenter un script encodant d'autres types de transactions.*

Adresses multisignatures et P2SH

Actuellement, l'implémentation la plus courante de la fonction P2SH est le script d'adresse multi-signature. Comme son nom l'indique, le script sous-jacent nécessite un nombre minimum de signatures pour prouver la propriété et donc dépenser des fonds. La fonction multi-signature bitcoin est conçue pour exiger des signatures M (également appelées « seuil ») à partir d'un total de N clés, appelées multisig M-of-N, où M est égal ou inférieur à N. Par exemple , Bob le propriétaire du café de [chapitre 1] pourrait utiliser une adresse multisignature nécessitant des signatures 1 sur 2 provenant d'une clé lui appartenant et d'une clé appartenant à son conjoint, garantissant que l'un d'eux pourrait signer pour dépenser une sortie de transaction verrouillée à cela adresse. Cela serait similaire à un « compte joint » tel que mis en œuvre dans les banques traditionnelles où l'un ou l'autre des conjoints peut dépenser avec une seule signature. Ou Gopesh, le concepteur Web payé par Bob pour créer un site Web, peut avoir une adresse multisignature 2

sur 3 pour son entreprise qui garantit qu'aucun fonds ne peut être dépensé à moins qu'au moins deux des partenaires commerciaux ne signent une transaction.

Nous explorerons comment créer des transactions qui dépensent des fonds à partir d'adresses P2SH (et multi-signatures) dans [transactions] .

Adresses personnalisées

Les adresses personnalisées sont des adresses bitcoin valides qui contiennent des messages lisibles par l'homme. Par exemple, 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 est une adresse valide qui contient les lettres formant le mot « Love » comme les quatre premières lettres Base58. Les adresses personnalisées nécessitent de générer et de tester des milliards de clés privées candidates, jusqu'à ce qu'une adresse bitcoin avec le modèle souhaité soit trouvée. Bien qu'il y ait quelques optimisations dans l'algorithme de génération de vanité, le processus consiste essentiellement à choisir une clé privée au hasard, à dériver la clé publique, à dériver l'adresse bitcoin et à vérifier si elle correspond au modèle de vanité souhaité, répétant des milliards de fois jusqu'à ce qu'un correspondance est trouvée.

Une fois qu'une adresse personnalisée correspondant au modèle souhaité est trouvée, la clé privée à partir de laquelle elle a été dérivée peut être utilisée par le propriétaire pour dépenser des bitcoins exactement de la même manière que toute autre adresse. Les adresses personnalisées ne sont ni moins ni plus sûres que toute autre adresse. Ils dépendent de la même

cryptographie à courbe elliptique (ECC) et de la même SHA que toute autre adresse. Vous ne pouvez pas trouver plus facilement la clé privée d'une adresse commençant par un modèle de vanité que vous ne pouvez le faire pour toute autre adresse.

Dans [chapitre 1] , nous avons présenté Eugenia, une directrice caritative pour enfants opérant aux Philippines. Disons qu'Eugenia organise une collecte de fonds Bitcoin et souhaite utiliser une adresse Bitcoin personnalisée pour faire connaître la collecte de fonds. Eugenia créera une adresse de courtoisie commençant par « 1Kids » pour promouvoir la collecte de fonds de bienfaisance pour les enfants. Voyons comment cette adresse de vanité sera créée et ce que cela signifie pour la sécurité de l'association caritative d'Eugenia.

Générer des adresses personnalisées

Il est important de réaliser qu'une adresse bitcoin est simplement un nombre représenté par des symboles dans l'alphabet Base58. La recherche d'un modèle tel que "1Kids" peut être considérée comme la recherche d'une adresse comprise entre 1Kids11111111111111111111111111111 et 1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzz. Il y a environ 58 29 adresses (environ 1,4 * 10 51) dans cette plage, toutes commençant par « 1Kids ». La plage d'adresses personnalisées commençant par "1Kids" montre la plage d'adresses qui ont le préfixe 1Kids.

Tableau 5. La plage d'adresses personnalisées commençant par "1Kids"

From	1Kids1111111111111111111111111111
	1Kids1111111111111111111111111112
	1Kids1111111111111111111111111113
	...
To	1Kidszzzzzzzzzzzzzzzzzzzzzzzzzzzz

Regardons le modèle “1Kids” comme un nombre et voyons à quelle fréquence nous pourrions trouver ce modèle dans une adresse bitcoin (voir La fréquence d’un modèle de vanité (1KidsCharity) et le temps de recherche moyen sur un PC de bureau). Un ordinateur de bureau moyen, sans matériel spécialisé, peut rechercher environ 100 000 clés par seconde.

Tableau 6. Fréquence d’un modèle de vanité (1KidsCharity) et temps de recherche moyen sur un ordinateur de bureau

Length	Pattern	Frequency	Average search time
1	1K	1 in 58 keys	< 1 milliseconds
2	1Ki	1 in 3,364	50 milliseconds
3	1Kid	1 in 195,000	< 2 seconds
4	1Kids	1 in 11 million	1 minute
5	1KidsC	1 in 656 million	1 hour
6	1KidsCh	1 in 38 billion	2 days
7	1KidsCha	1 in 2.2 trillion	3–4 months
8	1KidsChar	1 in 128 trillion	13–18 years
9	1KidsChari	1 in 7 quadrillion	800 years
10	1KidsCharit	1 in 400 quadrillion	46,000 years
11	1KidsCharity	1 in 23 quintillion	2.5 million years

Comme vous pouvez le voir, Eugenia ne créera pas de sitôt l'adresse personnalisée "1KidsCharity", même si elle avait accès à plusieurs milliers d'ordinateurs. Chaque caractère supplémentaire augmente la difficulté d'un facteur de 58. Les modèles de plus de sept caractères sont généralement trouvés par du matériel spécialisé, tel que des postes de travail personnalisés avec plusieurs GPU. Il s'agit souvent de « plates-formes » d'extraction de bitcoins réutilisées qui ne sont plus rentables pour l'extraction de bitcoins, mais peuvent être utilisées pour

trouver des adresses personnalisées. Les recherches de vanité sur les systèmes GPU sont de plusieurs ordres de grandeur plus rapides que sur un processeur à usage général.

Une autre façon de trouver une adresse de vanité est d'externaliser le travail à un pool de mineurs de vanité, comme la piscine de Vanity Pool . Un pool de ce type est un service qui permet à ceux qui possèdent du matériel GPU de gagner des bitcoins en recherchant des adresses personnalisées pour d'autres. Pour un petit paiement (0,01 bitcoin ou environ 5 $ au moment de la rédaction de cet article), Eugenia peut externaliser la recherche d'une adresse de vanité de modèle à sept caractères et obtenir des résultats en quelques heures au lieu d'avoir à exécuter une recherche de processeur pendant des mois.

Générer une adresse personnalisée est un exercice de force brute : essayez une clé aléatoire, vérifiez l'adresse résultante pour voir si elle correspond au modèle souhaité, répétez jusqu'à ce que vous réussissiez. Vanity Address Miner montre un exemple de "vanity miner", un programme conçu pour trouver des adresses personnalisées, écrites en C ++. L'exemple utilise la bibliothèque libbitcoin, que nous avons introduite dans [alt_libraries] .

Exemple 9. Mineur d'adresse personnelle

```
link:code/vanity-miner.cpp[]
```

Noter

> *Le mineur d'adresses de vanité utilise std : : random_device. En fonction de l'implémentation, il peut refléter un CSRNG fourni par le système d'exploitation sous-jacent. Dans le cas d'un système d'exploitation de type Unix tel que Linux, il s'inspire de / dev / urandom. Le générateur de nombres aléatoires utilisé ici est à des fins de démonstration, et il n'est* pas *approprié pour générer des clés bitcoin de qualité production car il n'est pas implémenté avec une sécurité suffisante.*

L'exemple de code doit être compilé à l'aide d'un compilateur C ++ et lié à la bibliothèque libbitcoin (qui doit d'abord être installée sur ce système). Pour exécuter l'exemple, exécutez l'exécutable vanity-miner sans paramètres (voir Compilation et exécution de l'exemple vanity-miner) et il tentera de trouver une adresse personnalisée commençant par « 1kid ».

Exemple 10. Compilation et exécution de l'exemple vanity-miner

```
# Compile the code with g++
$ g++ -o vanity-miner vanity-miner.cpp $(pkg-config
--cflags --libs libbitcoin)# Run the example
$ ./vanity-miner
Found vanity address!
1KiDzkG4MxmovZryZRj8tK81oQRhbZ46YT
Secret:
57cc268a05f83a23ac9d930bc8565bac4e277055f4794cbd1a39e5e71c038f3f
# Run it again for a different result
$ ./vanity-miner
Found vanity address!
```

```
1Kidxr3wsmMzzouwXibKfwTYs5Pau8TUFn
Secret:
7f65bbbbe6d8caae74a0c6a0d2d7b5c6663d71b60337299a1a2cf34c04b2a623
# Use "time" to see how long it takes to find a result
$ time ./vanity-miner
Found vanity address!
1KidPWhKgGRQWD5PP5TAnGfDyfWp5yceXM
Secret:
2a802e7a53d8aa237cd059377b616d2bfcfa4b0140bc85fa008f2d3d4b225349

real0m8.868s
user0m8.828s
sys0m0.035s
```

L'exemple de code prendra quelques secondes pour trouver une correspondance pour le modèle à trois caractères "kid", comme nous pouvons le voir lorsque nous utilisons la commande time Unix pour mesurer le temps d'exécution. Changez le modèle de recherche dans le code source et voyez combien de temps il faut pour les modèles à quatre ou cinq caractères !

Sécurité des adresses Vanity

Les adresses Vanity peuvent être utilisées pour améliorer *et* contrecarrer les mesures de sécurité ; ils sont vraiment une épée à double tranchant. Utilisée pour améliorer la sécurité, une adresse distincte rend plus difficile pour les adversaires de remplacer leur propre adresse et de tromper vos clients en les payant à votre place. Malheureusement, les adresses Vanity permettent également à quiconque de créer une adresse qui *ressemble à* n'importe quelle adresse aléatoire, ou même à une

autre adresse Vanity, trompant ainsi vos clients.

Eugenia pourrait annoncer une adresse générée aléatoirement (par exemple, 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy) à laquelle les gens peuvent envoyer leurs dons. Ou, elle pourrait générer une adresse personnalisée qui commence par 1Kids, pour la rendre plus distinctive.

Dans les deux cas, l'un des risques liés à l'utilisation d'une seule adresse fixe (plutôt qu'une adresse dynamique distincte par donateur) est qu'un voleur puisse infiltrer votre site Web et le remplacer par sa propre adresse, détournant ainsi les dons vers lui-même. Si vous avez annoncé votre adresse de don à différents endroits, vos utilisateurs peuvent inspecter visuellement l'adresse avant d'effectuer un paiement pour s'assurer qu'elle est la même qu'ils ont vue sur votre site Web, sur votre e-mail et sur votre flyer. Dans le cas d'une adresse aléatoire comme 1J7mdg5rbQyUHENYdx39WVW K7fsLpEoXZy, l'utilisateur moyen inspectera peut-être les premiers caractères "1J7mdg" et sera convaincu que l'adresse correspond. À l'aide d'un générateur d'adresses personnalisées, une personne ayant l'intention de voler en remplaçant une adresse d'apparence similaire peut rapidement générer des adresses qui correspondent aux premiers caractères, comme indiqué dans Génération d'adresses Vanity pour correspondre à une adresse aléatoire .

Tableau 7. Génération d'adresses personnalisées pour correspondre à une adresse aléatoire

Original Random Address	1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
Vanity (4-character match)	1J7md1QqU4LpctBetHS2ZoyLV5d6dShhEy
Vanity (5-character match)	1J7mdgYqyNd4ya3UEcq31Q7sqRMXw2XZ6n
Vanity (6-character match)	1J7mdg5WxGENmwyJP9xuGhG5KRzu99BBCX

Alors, une adresse personnalisée augmente-t-elle la sécurité ? Si Eugenia génère l'adresse personnalisée 1Kids33q44erFfp eXrmDSz7zEqG2FesZEN, les utilisateurs sont susceptibles de regarder le mot de modèle de vanité *et quelques caractères au - delà* , par exemple en remarquant la partie "1Kids33" de l'adresse. Cela forcerait un attaquant à générer une adresse personnalisée correspondant à au moins six caractères (deux de plus), dépensant un effort 3364 fois (58 × 58) supérieur à l' effort dépensé par Eugenia pour sa vanité à 4 caractères. Essentiellement, l'effort dépensé par Eugenia (ou pour lequel il paie un pool de vanité) "pousse" l'attaquant à devoir produire une vanité de modèle plus longue. Si Eugenia paie un pool pour générer une adresse de vanité à 8 caractères, l'attaquant serait poussé dans le royaume des 10 personnages, ce qui est impossible sur un ordinateur personnel et coûteux même avec une plate-forme d'extraction de vanité personnalisée ou un pool de vanité. Ce qui est abordable pour Eugenia devient inabordable pour l'attaquant, surtout si la récompense potentielle de la fraude n'est pas suffisamment élevée pour couvrir le coût de la génération de l'adresse Vanity.

Portefeuilles en papier

Les portefeuilles papier sont des clés privées bitcoin imprimées sur papier. Souvent, le portefeuille papier comprend également l'adresse bitcoin correspondante pour plus de commodité, mais ce n'est pas nécessaire car elle peut être dérivée de la clé privée.

Avertissement

Les portefeuilles en papier sont une technologie OBSOLETE et sont dangereux pour la plupart des utilisateurs. Il existe de nombreux pièges subtils impliqués dans leur génération, notamment la possibilité que le code générateur soit compromis par une « porte dérobée ». Des centaines de bitcoins ont été volés de cette façon. Les portefeuilles en papier sont présentés ici à titre informatif uniquement et ne doivent pas être utilisés pour stocker des bitcoins. Utilisez une phrase mnémotechnique BIP-39 pour sauvegarder vos clés. Utilisez un portefeuille matériel pour stocker les clés et signer les transactions. N'UTILISEZ PAS DE PORTEFEUILLES EN PAPIER.

Les portefeuilles en papier se présentent sous de nombreuses formes, tailles et designs, mais à un niveau très basique, ils ne sont qu'une clé et une adresse imprimées sur papier. Forme la plus simple d'un portefeuille papier : une impression de l'adresse bitcoin et de la clé privée montre la forme la plus simple d'un portefeuille papier.

Tableau 8. Forme la plus simple d'un portefeuille papier - une impression de l'adresse Bitcoin et de la clé privée

Public address	Private key (WIF)
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn

Les portefeuilles en papier sont disponibles dans de nombreux modèles et tailles, avec de nombreuses fonctionnalités différentes. Un exemple de portefeuille en papier simple montre un exemple de portefeuille en papier.

Figure 8. Un exemple de portefeuille en papier simple

Certains sont destinés à être offerts en cadeau et ont des thèmes saisonniers, tels que les thèmes de Noël et du Nouvel An. D'autres sont conçus pour être stockés dans un coffre-fort de banque ou un coffre-fort avec la clé privée cachée d'une manière ou d'une autre, soit avec des autocollants à gratter opaques, soit pliés et scellés avec une feuille adhésive inviolable.

D'autres conceptions comportent des copies supplémentaires de la clé et de l'adresse, sous la forme de talons détachables similaires aux talons de billets, vous permettant de stocker plusieurs copies pour vous protéger contre les incendies, les inondations ou d'autres catastrophes naturelles.

Figure 9. Exemple de portefeuille papier avec des copies supplémentaires des clés sur un « stub » de sauvegarde

6

Portefeuilles

Le mot « portefeuille » est utilisé pour décrire différentes choses dans Bitcoin.

À un niveau élevé, un portefeuille est une application qui sert d'interface utilisateur principale. Le portefeuille contrôle l'accès à l'argent d'un utilisateur, la gestion des clés et des adresses, le suivi du solde et la création et la signature des transactions.

Plus étroitement, du point de vue d'un programmeur, le mot « portefeuille » fait référence à la structure de données utilisée pour stocker et gérer les clés d'un utilisateur.

Dans ce chapitre, nous examinerons la deuxième signification, où les portefeuilles sont des conteneurs pour les clés privées, généralement implémentées sous forme de fichiers structurés

ou de bases de données simples.

Présentation de la technologie de portefeuille

Dans cette section, nous résumons les différentes technologies utilisées pour construire des portefeuilles Bitcoin conviviaux, sécurisés et flexibles.

Une idée fausse courante à propos du bitcoin est que les portefeuilles bitcoin contiennent du bitcoin. En fait, le portefeuille ne contient que des clés. Les « pièces » sont enregistrées dans la blockchain sur le réseau bitcoin. Les utilisateurs contrôlent les pièces sur le réseau en signant des transactions avec les clés de leur portefeuille. Dans un sens, un portefeuille Bitcoin est un *porte - clés* .

> *Conseil*
>
> *Les portefeuilles Bitcoin contiennent des clés, pas des pièces. Chaque utilisateur dispose d'un portefeuille contenant des clés. Les portefeuilles sont en réalité des porte-clés contenant des paires de clés privées / publiques. Les utilisateurs signent les transactions avec les clés, prouvant ainsi qu'ils sont propriétaires des sorties de transaction (leurs pièces). Les pièces sont stockées sur la blockchain sous la forme de sorties de transaction (souvent notées vout ou txout).*

Il existe deux principaux types de portefeuilles, qui se dis-

tinguent par le fait que les clés qu'ils contiennent sont liées les unes aux autres ou non.

Le premier type est un *portefeuille non déterministe* , où chaque clé est générée indépendamment à partir d'un nombre aléatoire. Les clés ne sont pas liées les unes aux autres. Ce type de portefeuille est également connu sous le nom de portefeuille JBOK de l'expression « Just a Bunch Of Keys ».

Le deuxième type de portefeuille est un *portefeuille déterministe* , où toutes les clés sont dérivées d'une seule clé principale, connue sous le nom de *graine* . Toutes les clés de ce type de portefeuille sont liées les unes aux autres et peuvent être générées à nouveau si l'on a la graine d'origine. Il existe un certain nombre de méthodes de *dérivation de clés* différentes utilisées dans les portefeuilles déterministes. La méthode de dérivation la plus couramment utilisée utilise une structure arborescente et est connue sous le nom de portefeuille *déterministe hiérarchique* ou *HD* .

Les portefeuilles déterministes sont initialisés à partir d'une séquence aléatoire (entropie). Pour faciliter leur utilisation, les séquences aléatoires sont codées sous forme de mots anglais, également appelés *mots de code mnémotechniques* .

Les prochaines sections présentent chacune de ces technologies à un niveau élevé.

Portefeuilles non déterministes (aléatoires)

Dans le premier portefeuille Bitcoin (maintenant appelé Bitcoin

Core), les portefeuilles étaient des collections de clés privées générées aléatoirement. Par exemple, le client Bitcoin Core d'origine prégénère 100 clés privées aléatoires lors du premier démarrage et génère plus de clés si nécessaire, en n'utilisant chaque clé qu'une seule fois. Ces portefeuilles sont remplacés par des portefeuilles déterministes car ils sont difficiles à gérer, à sauvegarder et à importer. L'inconvénient des clés aléatoires est que si vous en générez beaucoup, vous devez en conserver des copies, ce qui signifie que le portefeuille doit être sauvegardé fréquemment. Chaque clé doit être sauvegardée, ou les fonds qu'elle contrôle sont irrévocablement perdus si le portefeuille devient inaccessible. Cela entre directement en conflit avec le principe d'éviter la réutilisation d'adresses, en utilisant chaque adresse bitcoin pour une seule transaction. La réutilisation des adresses réduit la confidentialité en associant plusieurs transactions et adresses entre elles. Un portefeuille non déterministe de type 0 est un mauvais choix de portefeuille, surtout si vous souhaitez éviter la réutilisation d'adresses, car cela signifie gérer de nombreuses clés, ce qui crée le besoin de sauvegardes fréquentes. Bien que le client Bitcoin Core comprenne un portefeuille de type 0, l'utilisation de ce portefeuille est déconseillée par les développeurs de Bitcoin Core. Portefeuille non déterministe (aléatoire) de type 0 : une collection de clés générées aléatoirement (fig1) montre un portefeuille non déterministe, contenant une collection lâche de clés aléatoires.

Conseil

L'utilisation de portefeuilles non déterministes est déconseillée pour autre chose que de simples tests. Ils sont tout

simplement trop lourds à sauvegarder et à utiliser. Au lieu de cela, utilisez un portefeuille HD *standard avec une séquence aléatoire* mnémonique *(entropie ou « graine initiale ») pour la sauvegarde.*

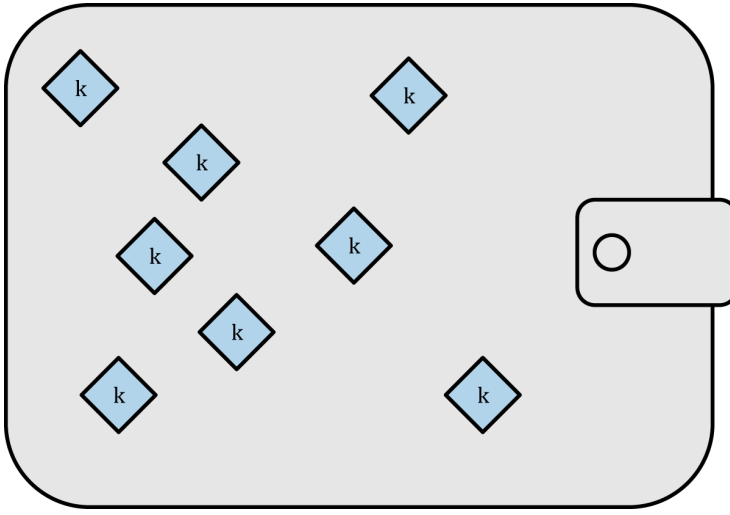


Figure 1. Portefeuille non déterministe (aléatoire) de type 0 : une collection de clés générées aléatoirement

Portefeuilles déterministes (prédéfinis)

Les portefeuilles déterministes, ou « prédéfinis », sont des portefeuilles qui contiennent des clés privées qui sont toutes dérivées d'une graine commune, grâce à l'utilisation d'une fonction de hachage unidirectionnelle. La graine est un nombre généré

aléatoirement qui est combiné avec d'autres données, comme un numéro d'index ou un « code de chaîne » (voir HD Wallets (BIP-32 / BIP-44) voir figure 3) pour dériver les clés privées. Dans un portefeuille déterministe, la graine est suffisante pour récupérer toutes les clés dérivées, et par conséquent, une seule sauvegarde au moment de la création est suffisante. La graine est également suffisante pour une exportation ou une importation de portefeuille, permettant une migration facile de toutes les clés de l'utilisateur entre différentes implémentations de portefeuille. Portefeuille déterministe de type 1 (ensemencé) : une séquence déterministe de clés dérivées d'une graine (fig 2) montre un diagramme logique d'un portefeuille déterministe.

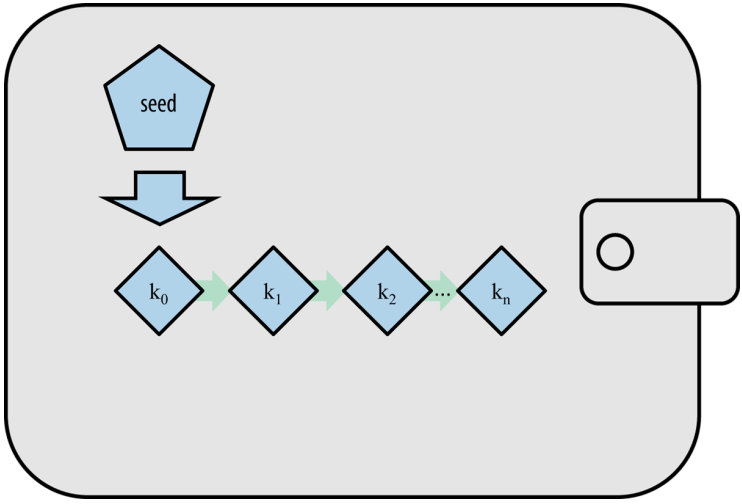


Figure 2. Portefeuille déterministe de type 1 (ensemencé) : une séquence déterministe de clés dérivées d'une graine

Portefeuilles HD (BIP-32 / BIP-44)

Des portefeuilles déterministes ont été développés pour faciliter la dérivation de nombreuses clés à partir d'une seule « graine ». La forme la plus avancée de portefeuilles déterministes est le portefeuille HD défini par la norme BIP-32. Les portefeuilles HD contiennent des clés dérivées dans une structure arborescente, de sorte qu'une clé parent peut dériver une séquence de clés enfants, chacune pouvant dériver une séquence de clés petits-enfants, et ainsi de suite, à une profondeur infinie. Cette arborescence est illustrée dans le portefeuille HD Type-2 : un arbre de clés généré à partir d'une seule graine . (fig 3)

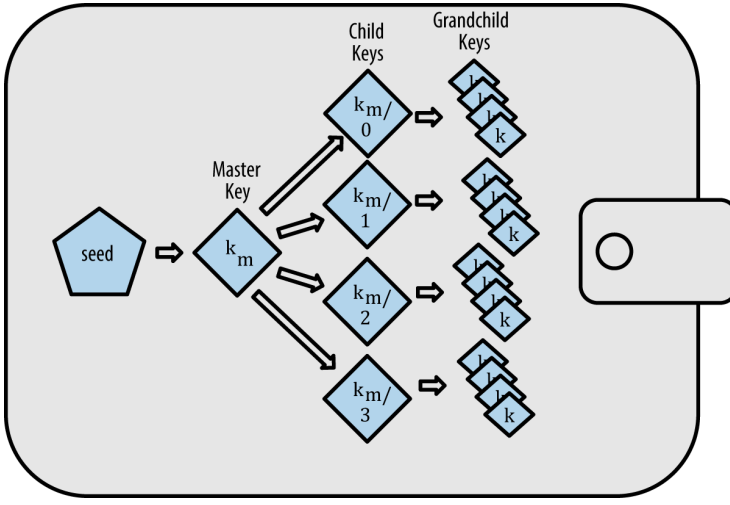


Figure 3. Portefeuille HD de type 2 : un arbre de clés généré à partir d'une seule graine

Les portefeuilles HD offrent deux avantages majeurs par rapport aux clés aléatoires (non déterministes). Premièrement, la structure arborescente peut être utilisée pour exprimer une signification organisationnelle supplémentaire, par exemple lorsqu'une branche spécifique de sous-clés est utilisée pour recevoir des paiements entrants et une branche différente est utilisée pour recevoir la modification des paiements sortants. Les branches de clés peuvent également être utilisées dans les paramètres de l'entreprise, en attribuant différentes branches à des départements, des filiales, des fonctions spécifiques ou des catégories comptables.

Le deuxième avantage des portefeuilles HD est que les utilisateurs peuvent créer une séquence de clés publiques sans avoir accès aux clés privées correspondantes. Cela permet aux portefeuilles HD d'être utilisés sur un serveur non sécurisé ou dans une capacité de réception uniquement, en émettant une clé publique différente pour chaque transaction. Les clés publiques n'ont pas besoin d'être préchargées ou dérivées à l'avance, mais le serveur n'a pas les clés privées qui peuvent dépenser les fonds.

Graines et codes mnémotechniques (BIP-39)

Les portefeuilles HD sont un mécanisme très puissant pour gérer de nombreuses clés et adresses. Ils sont encore plus utiles s'ils sont combinés avec une manière standardisée de créer des graines à partir d'une séquence de mots anglais faciles à transcrire, exporter et importer dans des portefeuilles. Ceci est connu comme un *mnémonique* et la norme est définie par BIP-39. Aujourd'hui, la plupart des portefeuilles Bitcoin (ainsi que des portefeuilles pour d'autres crypto-monnaies) utilisent

cette norme et peuvent importer et exporter des graines pour la sauvegarde et la récupération à l'aide de mnémoniques interopérables.

Regardons cela d'un point de vue pratique. Laquelle des graines suivantes est plus facile à transcrire, enregistrer sur papier, lire sans erreur, exporter et importer dans un autre portefeuille ?

Une graine pour un portefeuille déterministe, en hexadécimal

```
0C1E24E5917779D297E14D45F14E1A1A
```

Une graine pour un portefeuille déterministe, à partir d'un mnémonique de 12 mots

```
army van defense carry jealous true
garbage claim echo media make crunch
```

Bonnes pratiques relatives au portefeuille

Au fur et à mesure que la technologie des portefeuilles Bitcoin a mûri, certaines normes industrielles communes ont émergé qui rendent les portefeuilles Bitcoin largement interopérables, faciles à utiliser, sécurisés et flexibles. Ces normes communes sont :

- Mots de code mnémotechniques, basés sur BIP-39
- Portefeuilles HD, basés sur BIP-32

- Structure de portefeuille HD polyvalente, basée sur BIP-43
- Portefeuilles multi-devises et multi-comptes, basés sur BIP-44

Ces normes peuvent changer ou devenir obsolètes par les développements futurs, mais pour l'instant elles forment un ensemble de technologies imbriquées qui sont devenues la norme de portefeuille de facto pour Bitcoin.

Les normes ont été adoptées par une large gamme de portefeuilles bitcoin logiciels et matériels, rendant tous ces portefeuilles interopérables. Un utilisateur peut exporter un mnémonique généré sur l'un de ces portefeuilles et l'importer dans un autre portefeuille, en récupérant toutes les transactions, clés et adresses.

Certains exemples de portefeuilles logiciels prenant en charge ces normes incluent (classés par ordre alphabétique) Bluewallet, Breadwallet, Copay et Multibit HD. Des exemples de portefeuilles matériels prenant en charge ces normes incluent (classés par ordre alphabétique) KeepKey, Ledger et Trezor.

Les sections suivantes examinent chacune de ces technologies en détail.

Conseil

Si vous implémentez un portefeuille Bitcoin, il doit être construit comme un portefeuille HD, avec une graine dérivée et encodée en tant que code mnémonique pour la sauvegarde, suivant les BIP-32, BIP-39, BIP-43 et BIP-

44 normes, comme décrit dans les sections suivantes.

Utiliser un portefeuille Bitcoin

Dans le chapitre 3, nous avons présenté Gabriel, un jeune adolescent entreprenant de Rio de Janeiro, qui dirige une boutique en ligne simple qui vend des t-shirts, des tasses à café et des autocollants de marque Bitcoin.

Gabriel utilise un portefeuille matériel Trezor bitcoin (Un appareil Trezor : un portefeuille Bitcoin HD dans le matériel [fig4]) pour gérer en toute sécurité son bitcoin. Le Trezor est un simple périphérique USB avec deux boutons qui stocke les clés (sous la forme d'un portefeuille HD) et signe les transactions. Les portefeuilles Trezor implémentent toutes les normes de l'industrie décrites dans ce chapitre, donc Gabriel ne dépend d'aucune technologie propriétaire ou d'une solution de fournisseur unique.

Figure 4. Un appareil Trezor : un portefeuille Bitcoin HD dans le matériel

Lorsque Gabriel a utilisé le Trezor pour la première fois, l'appareil a généré une séquence aléatoire (entropie), le mnémonique associé et a dérivé une graine d'un générateur de nombres aléatoires matériel intégré. Lors de cette phase d'initialisation, le portefeuille a affiché une séquence numérotée de mots, un par un, à l'écran (voir Trezor affichant l'un des mots mnémotechniques[fig5]).

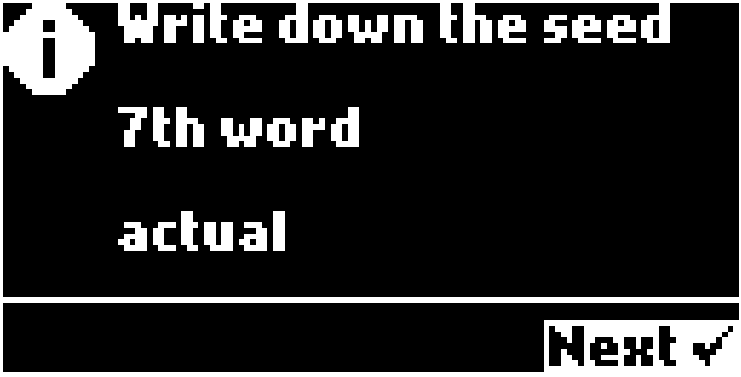


Figure 5. Trezor affichant l'un des mots mnémotechniques

En écrivant ce mnémonique, Gabriel a créé une sauvegarde (voir la sauvegarde papier de Gabriel du mnémonique [table 1]) qui peut être utilisée pour la récupération en cas de perte ou d'endommagement de l'appareil Trezor. Ce mnémonique peut être utilisé pour la récupération dans un nouveau Trezor ou dans l'un des nombreux portefeuilles logiciels ou matériels compatibles. Notez que la séquence des mots est importante, donc les sauvegardes papier mnémotechniques ont des espaces numérotés pour chaque mot. Gabriel a dû enregistrer soigneusement chaque mot dans l'espace numéroté pour conserver la séquence correcte.

1.	*army*	7.	*garbage*
2.	*van*	8.	*claim*
3.	*defense*	9.	*echo*
4.	*carry*	10.	*media*
5.	*jealous*	11.	*make*
6.	*true*	12.	*crunch*

Tableau 1. Sauvegarde papier de Gabriel du mnémonique

Note

Un mnémonique de 12 mots est montré dans la sauvegarde papier de Gabriel du mnémonique , pour plus de

simplicité. En fait, la plupart des portefeuilles matériels génèrent un mnémonique de 24 mots plus sécurisé. Le mnémonique est utilisé exactement de la même manière, quelle que soit sa longueur.

Pour la première implémentation de sa boutique en ligne, Gabriel utilise une seule adresse bitcoin, générée sur son appareil Trezor. Cette adresse unique est utilisée par tous les clients pour toutes les commandes. Comme nous le verrons, cette approche présente certains inconvénients et peut être améliorée avec un portefeuille HD.

Détails de la technologie du portefeuille

Examinons maintenant en détail chacune des normes industrielles importantes utilisées par de nombreux portefeuilles Bitcoin.

Mots de code mnémotechnique (BIP-39)

Les mots de code mnémotechniques sont des séquences de mots qui représentent (codent) un nombre aléatoire utilisé comme graine pour dériver un portefeuille déterministe. La séquence de mots est suffisante pour recréer la graine et à partir de là recréer le portefeuille et toutes les clés dérivées. Une application de portefeuille qui implémente des portefeuilles déterministes avec des mots mnémotechniques montrera à l'utilisateur une séquence de 12 à 24 mots lors de la création initiale d'un portefeuille. Cette séquence de mots est la sauvegarde du

portefeuille et peut être utilisée pour récupérer et recréer toutes les clés dans la même application de portefeuille ou dans toute autre application de portefeuille compatible. Les mots mnémoniques facilitent la sauvegarde des portefeuilles par les utilisateurs, car ils sont faciles à lire et à transcrire correctement, par rapport à une séquence aléatoire de nombres.

Conseil

Les mots mnémotechniques sont souvent confondus avec les « portefeuilles intellectuels ». Ils ne sont pas les mêmes. La principale différence est qu'un brainwallet se compose de mots choisis par l'utilisateur, tandis que les mots mnémotechniques sont créés au hasard par le portefeuille et présentés à l'utilisateur. Cette différence importante rend les mots mnémotechniques beaucoup plus sûrs, car les humains sont de très mauvaises sources d'aléatoire.

Les codes mnémotechniques sont définis dans le BIP-39. Notez que BIP-39 est une implémentation d'une norme de code mnémonique. Il existe une norme différente, avec un ensemble de mots différent, utilisé par le portefeuille Electrum et antérieur à BIP-39. Le BIP-39 a été proposé par la société derrière le portefeuille matériel Trezor et est incompatible avec la mise en œuvre d'Electrum. Cependant, le BIP-39 a maintenant obtenu un large soutien de l'industrie à travers des dizaines d'implémentations interopérables et devrait être considéré comme la norme de facto de l'industrie.

BIP-39 définit la création d'un code mnémonique et d'une graine, que nous décrivons ici en neuf étapes. Pour plus de clarté, le processus est divisé en deux parties : les étapes 1 à 6 sont illustrées dans Génération de mots mnémotechniques(fig6) et les étapes 7 à 9 sont illustrées dans Du mnémonique à la graine .

Générer des mots mnémotechniques

Les mots mnémotechniques sont générés automatiquement par le portefeuille en utilisant le processus standardisé défini dans BIP-39. Le portefeuille part d'une source d'entropie, ajoute une somme de contrôle, puis mappe l'entropie à une liste de mots :

1. Créez une séquence aléatoire (entropie) de 128 à 256 bits.
2. Créez une somme de contrôle de la séquence aléatoire en prenant les premiers bits (longueur d'entropie / 32) de son hachage SHA256.
3. Ajoutez la somme de contrôle à la fin de la séquence aléatoire.
4. Divisez le résultat en segments de 11 bits.
5. Mappez chaque valeur de 11 bits à un mot du dictionnaire prédéfini de 2048 mots.
6. Le code mnémonique est la séquence de mots.

La génération d'entropie et le codage sous forme de mots mnémotechniques (fig6) montre comment l'entropie est utilisée pour générer des mots mnémotechniques.

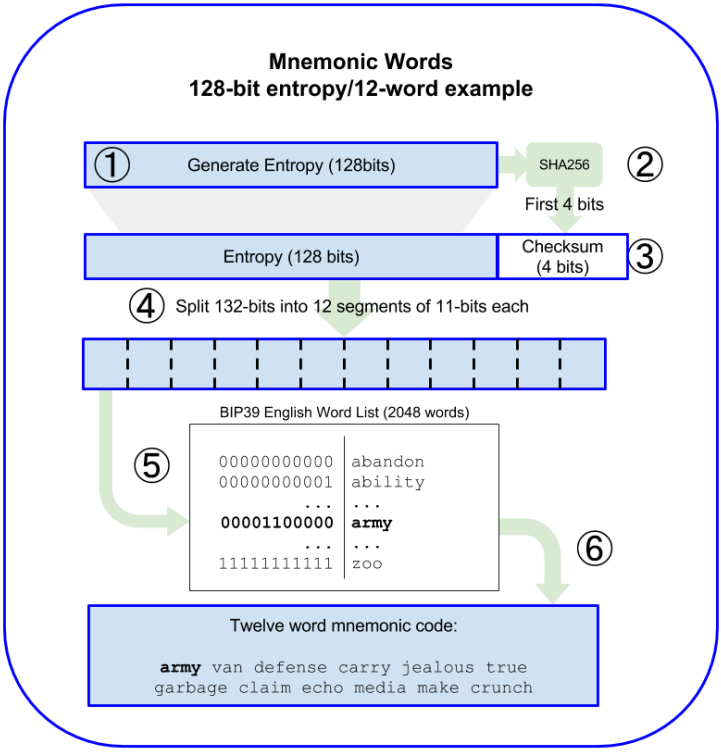


Figure 6. Génération d'entropie et codage sous forme de mots mnémotechniques

Codes mnémotechniques : l'entropie et la longueur des mots montrent la relation entre la taille des données d'entropie et la longueur des codes mnémotechniques dans les mots.

Entropy (bits)	Checksum (bits)	Entropy + checksum (bits)	Mnemonic length (words)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

Tableau 2. Codes mnémotechniques : entropie et longueur de mot

Du mnémonique à la graine

Les mots mnémotechniques représentent l'entropie d'une longueur de 128 à 256 bits. L'entropie est ensuite utilisée pour dériver une graine plus longue (512 bits) grâce à l'utilisation de la fonction d'étirement de clé PBKDF2. La graine produite est ensuite utilisée pour construire un portefeuille déterministe et dériver ses clés.

La fonction d'étirement des touches prend deux paramètres : le mnémonique et un *sel* . Le but d'un sel dans une fonction d'étirement des touches est de rendre difficile la création d'une table de recherche permettant une attaque par force brute. Dans la norme BIP-39, le sel a un autre but : il permet l'introduction d'une phrase de passe qui sert de facteur de

sécurité supplémentaire protégeant la graine, comme nous le décrirons plus en détail dans la phrase de passe facultative dans BIP-39 .

Le processus décrit aux étapes 7 à 9 continue du processus décrit précédemment dans Génération de mots mnémotechniques :

7.Le premier paramètre de la fonction d'étirement des touches PBKDF2 est le *mnémonique* produit à partir de l'étape 6.

8.Le deuxième paramètre de la fonction d'étirement des touches PBKDF2 est un *sel* . Le sel est composé de la constante de chaîne « mnémonique » concaténée avec une chaîne de phrase de passe facultative fournie par l'utilisateur.

9.PBKDF2 étend les paramètres mnémotechniques et salt en utilisant 2048 cycles de hachage avec l'algorithme HMAC-SHA512, produisant une valeur de 512 bits comme sortie finale. Cette valeur de 512 bits est la graine.

De mnémonique à graine montre comment un mnémonique est utilisé pour générer une graine.

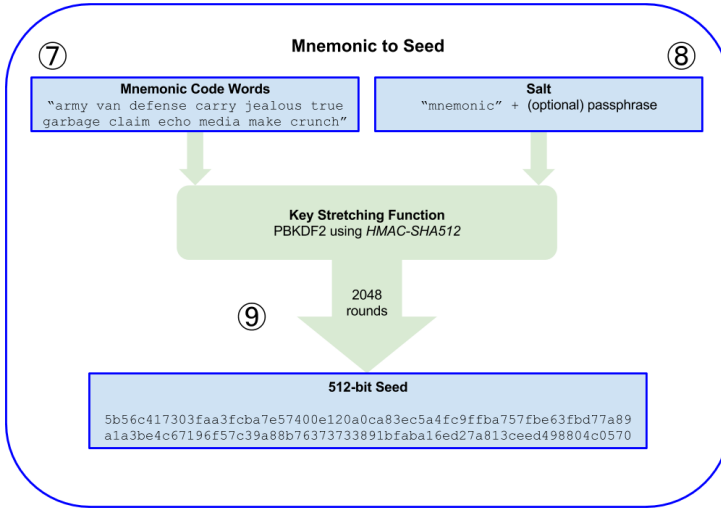


Figure 7. Du mnémonique à la graine

> *Note : La fonction d'étirement des touches, avec ses 2048 tours de hachage, est une protection très efficace contre les attaques par force brute contre le mnémonique ou la phrase de passe. Il est extrêmement coûteux (en calcul) d'essayer plus de quelques milliers de combinaisons de phrases de passe et de mnémoniques, alors que le nombre de graines dérivées possibles est vaste (2^{512}).*

Les tableaux *# mnemonic_128_no_pass* , *# mnemonic_128_w_pass* et *# mnemonic_256_no_pass* montrent quelques exemples de codes mnémoniques et les graines qu'ils produisent (avec ou sans phrase de passe).

Entropy input (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonic (12 words)	army van defense carry jealous true garbage claim echo media make crunch
Passphrase	(none)
Seed (512 bits)	5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39 a88b76373733891bfaba16ed27a813ceed498804c0570

Tableau 3. Code mnémonique d'entropie 128 bits, pas de phrase de passe, valeur de départ résultante

Entropy input (128 bits)	0c1e24e5917779d297e14d45f14e1a1a
Mnemonic (12 words)	army van defense carry jealous true garbage claim echo media make crunch
Passphrase	SuperDuperSecret
Seed (512 bits)	3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0 715861dc8a18358f80b79d49acf64142ae57037d1d54

Tableau 4. Code mnémonique d'entropie 128 bits, avec phrase de passe, valeur de départ résultante

Entropy input (256 bits)	2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemonic (24 words)	cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Passphrase	(none)
Seed (512 bits)	3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e5 5f1e0deaa082df8d487381379df848a6ad7e98798404

Tableau 5. Code mnémonique d'entropie de 256 bits, pas de phrase de passe, valeur de départ résultante

Conseil

De nombreux portefeuilles ne permettent pas la création de portefeuilles avec plus d'une phrase mnémotechnique de 12 mots. Vous remarquerez dans les tableaux ci-dessus que malgré les longueurs uniques d'entrée d'entropie, la taille de graine reste la même (512 bits). Du point de vue de la sécurité, la quantité d'entropie réellement utilisée pour la production de portefeuilles HD est d'environ 128 bits, ce qui équivaut à 12 mots. Fournir plus de 12 mots produit une entropie supplémentaire qui est inutile, et cette entropie inutilisée *n'est pas utilisée pour la dérivation de la graine de la manière que l'on pourrait initialement soupçonner. Du point de vue de la convivialité, 12 mots sont également plus faciles à écrire, à sauvegarder et à stocker.*

Phrase secrète facultative dans BIP-39

La norme BIP-39 permet l'utilisation d'une phrase de passe facultative dans la dérivation de la graine. Si aucune phrase de passe n'est utilisée, le mnémonique est étiré avec un sel constitué de la chaîne constante « mnémonique », produisant une graine spécifique de 512 bits à partir de n'importe quel mnémonique donné. Si une phrase de passe est utilisée, la fonction d'étirement produit une valeur de départ *différente* de ce même mnémonique. En fait, étant donné un seul mnémonique, chaque phrase de passe possible conduit à une graine différente. Essentiellement, il n'y a pas de « mauvaise » phrase de passe. Toutes les phrases de passe sont valides et elles mènent toutes à des graines différentes, formant un vaste ensemble de portefeuilles non initialisés possibles. L'ensemble

des portefeuilles possibles est si grand (2 512) qu'il n'y a aucune possibilité pratique de forcer brutalement ou de deviner accidentellement celui qui est en cours d'utilisation.

Conseil

Il n'y a pas de « mauvaises » phrases de passe dans BIP-39. Chaque phrase de passe mène à un portefeuille qui, à moins d'être utilisé précédemment, sera vide.

La phrase secrète facultative crée deux fonctionnalités importantes :

- Un deuxième facteur (quelque chose de mémorisé) qui rend un mnémonique inutile à lui seul, protégeant les sauvegardes mnémotechniques d'un compromis par un voleur.

- Une forme de déni plausible ou « portefeuille de contrainte », où une phrase secrète choisie mène à un portefeuille avec une petite quantité de fonds utilisée pour détourner un attaquant du « vrai » portefeuille qui contient la majorité des fonds.

Cependant, il est important de noter que l'utilisation d'une phrase de passe introduit également un risque de perte :

- Si le propriétaire du portefeuille est frappé d'incapacité ou est décédé et que personne d'autre ne connaît la phrase

de passe, le code mnémonique est inutile et tous les fonds stockés dans le portefeuille sont perdus à jamais.

- Inversement, si le propriétaire sauvegarde la phrase de passe au même endroit que le code mnémonique, cela va à l'encontre de l'objectif d'un deuxième facteur.

Bien que les phrases de passe soient très utiles, elles ne doivent être utilisées qu'en combinaison avec un processus soigneusement planifié de sauvegarde et de récupération, en tenant compte de la possibilité de survivre au propriétaire et de permettre à sa famille de récupérer le domaine de la cryptomonnaie.

Travailler avec des codes mnémotechniques

BIP-39 est implémenté en tant que bibliothèque dans de nombreux langages de programmation différents :

python-mnémonique
L'implémentation de référence du standard par l'équipe SatoshiLabs qui a proposé BIP-39, en Python

bitcoinjs / bip39
Une implémentation de BIP-39, dans le cadre du framework BitcoinJS populaire, en JavaScript

libbitcoin / mnémonique
Une implémentation de BIP-39, dans le cadre du framework

Libbitcoin populaire, en C ++

Créer un portefeuille HD à partir de la graine

Les portefeuilles HD sont créés à partir d'une seule *graine racine* , qui est un nombre aléatoire de 128, 256 ou 512 bits. Le plus souvent, cette graine est générée à partir d'un *mnémonique* comme détaillé dans la section précédente.

Chaque clé du portefeuille HD est dérivée de manière déterministe de cette graine racine, ce qui permet de recréer l'intégralité du portefeuille HD à partir de cette graine dans n'importe quel portefeuille HD compatible. Cela facilite la sauvegarde, la restauration, l'exportation et l'importation de portefeuilles HD contenant des milliers, voire des millions de clés, en transférant simplement uniquement le mnémonique dont la racine est dérivée.

Le processus de création des clés principales et du code de chaîne principal pour un portefeuille HD est illustré dans Création de clés principales et de code de chaîne à partir d'une graine racine .

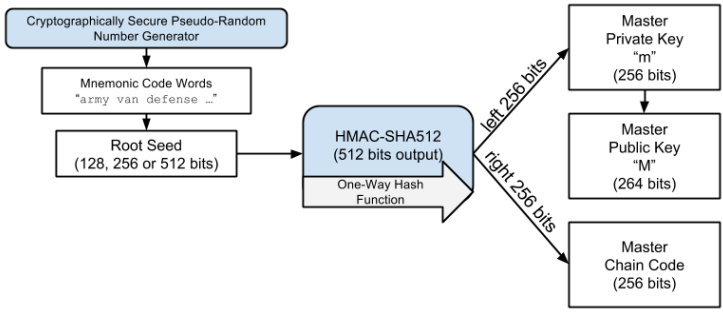


Figure 8. Création de clés principales et de code de chaîne à partir d'une graine racine

La graine racine est entrée dans l'algorithme HMAC-SHA512 et le hachage résultant est utilisé pour créer une *clé privée principale* (m) et un *code de chaîne principal* (c).

La clé privée principale (m) génère alors une clé publique principale correspondante (M) en utilisant le processus normal de multiplication de la courbe elliptique m * G.

Le code chaîne (c) est utilisé pour introduire l'entropie dans la fonction qui crée les clés enfants à partir des clés parents, comme nous le verrons dans la section suivante.

Dérivation de clé enfant privée

Les portefeuilles HD utilisent une fonction de *dérivation de clé enfant* (CKD) pour dériver les clés enfants des clés parent.

Les fonctions de dérivation de clé enfant sont basées sur une

fonction de hachage unidirectionnelle qui combine :

- Une clé privée ou publique parente (clé compressée ECDSA)
- Une graine appelée code de chaîne (256 bits)
- Un numéro d'index (32 bits)

Le code de chaîne est utilisé pour introduire des données aléatoires déterministes dans le processus, de sorte que la connaissance de l'index et d'une clé enfant ne soit pas suffisante pour dériver d'autres clés enfants. Connaître une clé enfant ne permet pas de retrouver ses frères, sauf si vous avez également le code de la chaîne. La graine de code de chaîne initiale (à la racine de l'arbre) est faite à partir de la graine, tandis que les codes de chaîne enfants suivants sont dérivés de chaque code de chaîne parent.

Ces trois éléments (clé parent, code de chaîne et index) sont combinés et hachés pour générer des clés enfants, comme suit.

La clé publique parente, le code de chaîne et le numéro d'index sont combinés et hachés avec l'algorithme HMAC-SHA512 pour produire un hachage de 512 bits. Ce hachage de 512 bits est divisé en deux moitiés de 256 bits. Les 256 bits de la moitié droite de la sortie de hachage deviennent le code de chaîne pour l'enfant. Les 256 bits de la moitié gauche du hachage sont ajoutés à la clé parente pour produire la clé privée enfant. Dans Extension d'une clé privée parente pour créer une clé privée enfant , nous le voyons illustré avec l'index mis à 0 pour produire l'enfant « zéro » (d'abord par index) du parent.

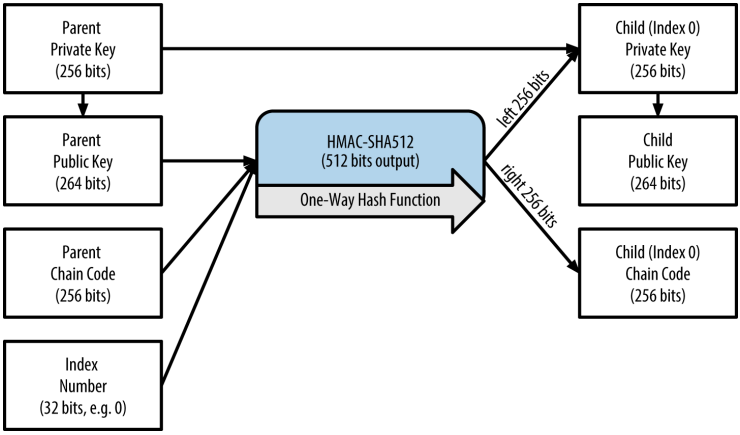


Figure 9. Extension d'une clé privée parente pour créer une clé privée enfant

La modification de l'index nous permet d'étendre le parent et de créer les autres enfants dans la séquence, par exemple, Enfant 0, Enfant 1, Enfant 2, etc. Chaque clé parent peut avoir 2 147 483 647 (2 31) enfants (2 31 est la moitié de l'ensemble 2 32 plage disponible car l'autre moitié est réservée à un type particulier de dérivation dont nous parlerons plus loin dans ce chapitre).

En répétant le processus un niveau plus bas dans l'arbre, chaque enfant peut à son tour devenir parent et créer ses propres enfants, en un nombre infini de générations.

Utilisation de clés enfants dérivées

Les clés privées enfants ne peuvent pas être distinguées des clés non déterministes (aléatoires). Étant donné que la fonction de dérivation est une fonction à sens unique, la clé enfant ne peut

pas être utilisée pour rechercher la clé parent. La clé enfant ne peut pas non plus être utilisée pour rechercher des frères et sœurs. Si vous avez le nième enfant, vous ne pouvez pas trouver ses frères et sœurs, tels que l'enfant n – 1 ou l'enfant n + 1, ou tout autre enfant faisant partie de la séquence. Seuls la clé parent et le code de chaîne peuvent dériver tous les enfants. Sans le code de chaîne enfant, la clé enfant ne peut pas non plus être utilisée pour dériver des petits-enfants. Vous avez besoin à la fois de la clé privée enfant et du code de chaîne enfant pour démarrer une nouvelle branche et dériver des petits-enfants.

Alors, à quoi la clé privée enfant peut-elle être utilisée seule ? Il peut être utilisé pour créer une clé publique et une adresse bitcoin. Ensuite, il peut être utilisé pour signer des transactions pour dépenser tout ce qui est payé à cette adresse.

Conseil

Une clé privée enfant, la clé publique correspondante et l'adresse bitcoin sont toutes indiscernables des clés et des adresses créées de manière aléatoire. Le fait qu'ils fassent partie d'une séquence n'est pas visible en dehors de la fonction de portefeuille HD qui les a créés. Une fois créées, elles fonctionnent exactement comme des touches « normales ».

Clés étendues

Comme nous l'avons vu précédemment, la fonction de dérivation de clé peut être utilisée pour créer des enfants à n'importe quel niveau de l'arborescence, en fonction des trois entrées :

une clé, un code de chaîne et l'index de l'enfant souhaité. Les deux ingrédients essentiels sont le code de la clé et de la chaîne, et combinés, ils sont appelés une *clé étendue* . Le terme « clé étendue » pourrait également être considéré comme « clé extensible » car une telle clé peut être utilisée pour dériver des enfants.

Les clés étendues sont stockées et représentées simplement comme la concaténation de la clé de 256 bits et du code de chaîne de 256 bits dans une séquence de 512 bits. Il existe deux types de clés étendues. Une clé privée étendue est la combinaison d'une clé privée et d'un code de chaîne et peut être utilisée pour dériver des clés privées enfants (et à partir d'elles, des clés publiques enfants). Une clé publique étendue est une clé publique et un code de chaîne, qui peuvent être utilisés pour créer des clés publiques enfants (*publiques uniquement*) .

Considérez une clé étendue comme la racine d'une branche dans l'arborescence du portefeuille HD. Avec la racine de la branche, vous pouvez dériver le reste de la branche. La clé privée étendue peut créer une branche complète, tandis que la clé publique étendue ne *peut* créer qu'une branche de clés publiques.

Conseil

Une clé étendue se compose d'une clé privée ou publique et d'un code de chaîne. Une clé étendue peut créer des enfants, générant sa propre branche dans l'arborescence. Le partage d'une clé étendue donne accès à l'ensemble de la branche.

Les clés étendues sont encodées à l'aide de Base58Check, pour exporter et importer facilement entre différents portefeuilles compatibles BIP-32. Le codage Base58Check pour les clés étendues utilise un numéro de version spécial qui se traduit par le préfixe « xprv » et « xpub » lorsqu'il est codé en caractères Base58 pour les rendre facilement reconnaissables. Étant donné que la clé étendue est de 512 ou 513 bits, elle est également beaucoup plus longue que les autres chaînes encodées en Base58Check que nous avons vues précédemment.

Voici un exemple de clé *privée* étendue , codée en Base58Check :

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5
```

Voici la clé *publique* étendue correspondante , codée en Base58Check :

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR
```

Dérivation de clé publique enfant

Comme mentionné précédemment, une caractéristique très utile des portefeuilles HD est la possibilité de dériver des clés enfants publiques à partir de clés parentales publiques, *sans* avoir les clés privées. Cela nous donne deux façons de dériver une clé publique enfant : soit à partir de la clé privée enfant, soit directement à partir de la clé publique parent.

Une clé publique étendue peut donc être utilisée pour dériver toutes les clés *publiques* (et uniquement les clés publiques) dans cette branche de la structure du portefeuille HD.

Ce raccourci peut être utilisé pour créer des déploiements à clé publique très sécurisés uniquement dans lesquels un serveur ou une application possède une copie d'une clé publique étendue et aucune clé privée. Ce type de déploiement peut produire un nombre infini de clés publiques et d'adresses bitcoin, mais ne peut pas dépenser l'argent envoyé à ces adresses. Pendant ce temps, sur un autre serveur plus sécurisé, la clé privée étendue peut dériver toutes les clés privées correspondantes pour signer des transactions et dépenser de l'argent.

Une application courante de cette solution consiste à installer une clé publique étendue sur un serveur Web qui sert une application de commerce électronique. Le serveur Web peut utiliser la fonction de dérivation de clé publique pour créer une nouvelle adresse bitcoin pour chaque transaction (par exemple, pour un panier client). Le serveur Web n'aura aucune clé privée qui serait vulnérable au vol. Sans les portefeuilles HD, le seul moyen de le faire est de générer des milliers d'adresses bitcoin sur un serveur sécurisé séparé, puis de les précharger sur le serveur de commerce électronique. Cette approche est lourde et nécessite une maintenance constante pour garantir que le serveur de commerce électronique ne « manque » pas d'adresses.

Une autre application courante de cette solution concerne les portefeuilles de stockage frigorifique ou matériels. Dans ce scénario, la clé privée étendue peut être stockée sur un

portefeuille papier ou un périphérique matériel (tel qu'un portefeuille matériel Trezor), tandis que la clé publique étendue peut être conservée en ligne. L'utilisateur peut créer des adresses de "réception" à volonté, tandis que les clés privées sont stockées hors ligne en toute sécurité. Pour dépenser les fonds, l'utilisateur peut utiliser la clé privée étendue sur un client bitcoin de signature hors ligne ou signer des transactions sur le périphérique de portefeuille matériel (par exemple, Trezor). L'extension d'une clé publique parent pour créer une clé publique enfant illustre le mécanisme permettant d'étendre une clé publique parent afin de dériver des clés publiques enfants.

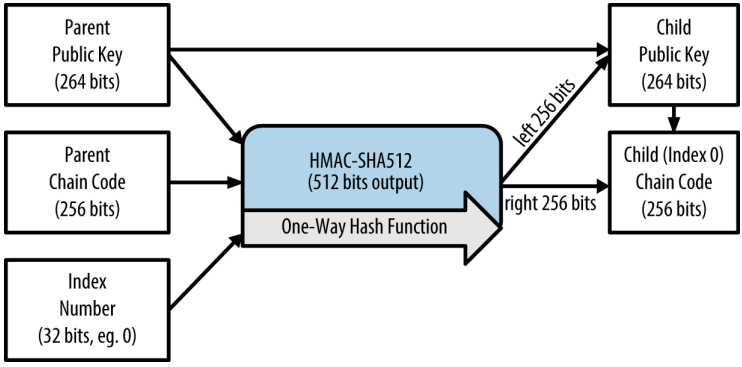


Figure 10. Extension d'une clé publique parente pour créer une clé publique enfant

Dérivation de clé enfant renforcée

La possibilité de dériver une branche de clés publiques à partir d'un xpub est très utile, mais elle comporte un risque

potentiel. L'accès à un xpub ne donne pas accès aux clés privées enfants. Cependant, étant donné que le xpub contient le code de chaîne, si une clé privée enfant est connue, ou en quelque sorte divulguée, elle peut être utilisée avec le code de chaîne pour dériver toutes les autres clés privées enfants . Une seule clé privée enfant divulguée, associée à un code de chaîne parent, révèle toutes les clés privées de tous les enfants. Pire encore, la clé privée enfant ainsi qu'un code de chaîne parent peuvent être utilisés pour déduire la clé privée parent.

Pour contrer ce risque, les portefeuilles HD utilisent une fonction de dérivation alternative appelée *dérivation renforcée* , qui « rompt » la relation entre la clé publique parente et le code de la chaîne enfant. La fonction de dérivation renforcée utilise la clé privée parente pour dériver le code de la chaîne enfant, au lieu de la clé publique parente. Cela crée un « pare-feu » dans la séquence parent / enfant, avec un code de chaîne qui ne peut pas être utilisé pour compromettre une clé privée parent ou sœur. La fonction de dérivation renforcée semble presque identique à la dérivation de clé privée enfant normale, sauf que la clé privée parente est utilisée comme entrée de la fonction de hachage, au lieu de la clé publique parente, comme indiqué dans le diagramme dans Dérivation renforcée d'une clé enfant ; omet la clé publique parente .

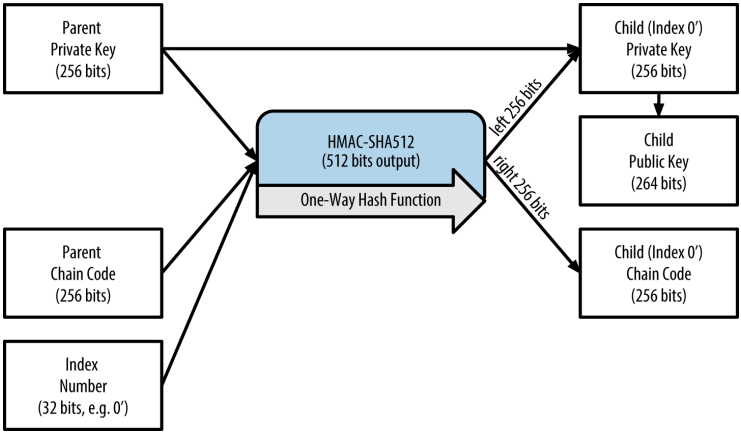


Figure 11. Dérivation renforcée d'une clé enfant ; omet la clé publique parente

Lorsque la fonction de dérivation privée renforcée est utilisée, la clé privée enfant et le code de chaîne résultants sont complètement différents de ce qui résulterait de la fonction de dérivation normale. La "branche" de clés qui en résulte peut être utilisée pour produire des clés publiques étendues qui ne sont pas vulnérables, car le code de chaîne qu'elles contiennent ne peut pas être exploité pour révéler des clés privées. La dérivation renforcée est donc utilisée pour créer un « espace » dans l'arborescence au-dessus du niveau où les clés publiques étendues sont utilisées.

En termes simples, si vous souhaitez utiliser la commodité d'un xpub pour dériver des branches de clés publiques, sans vous exposer au risque d'une fuite de code de chaîne, vous devez le dériver d'une clé parente renforcée, plutôt que d'une clé normale (non- durci) clé parent. En tant que meilleure

pratique, les enfants de niveau 1 des clés principales sont toujours dérivés par le biais de la dérivation renforcée, pour éviter toute compromission des clés principales.

Numéros d'index pour la dérivation normale et durcie

Le numéro d'index utilisé dans la fonction de dérivation est un entier de 32 bits. Pour distinguer facilement les clés dérivées par la fonction de dérivation normale des clés dérivées par dérivation renforcée, ce numéro d'index est divisé en deux plages. Les numéros d'index entre 0 et 2 31 –1 (0x0 à 0x7FFFFFFF) sont utilisés *uniquement* pour la dérivation normale. Les numéros d'index entre 2 31 et 2 32 –1 (0x80000000 à 0xFFFFFFFF) sont utilisés *uniquement* pour la dérivation renforcée. Par conséquent, si le numéro d'index est inférieur à 2 31 , l'enfant est normal, tandis que si le numéro d'index est égal ou supérieur à 2 31 , l'enfant est durci.

Pour faciliter la lecture et l'affichage du numéro d'index, le numéro d'index des enfants renforcés est affiché à partir de zéro, mais avec un symbole premier. La première clé enfant normale est donc affichée comme 0, tandis que le premier enfant durci (index 0x80000000) est affiché comme 0 & # x27 ;. Dans l'ordre, la deuxième clé renforcée aurait alors l'index 0x80000001 et serait affichée comme 1 & # x27 ;, et ainsi de suite. Lorsque vous voyez un index de portefeuille HD i & # x27 ;, cela signifie 2 31 + i.

Identifiant de clé de portefeuille HD (chemin)

Les clés d'un portefeuille HD sont identifiées à l'aide d'une

convention de dénomination de « chemin », chaque niveau de l'arborescence étant séparé par un caractère barre oblique (/) (voir les exemples de chemin de portefeuille HD). Les clés privées dérivées de la clé privée principale commencent par « m ». Les clés publiques dérivées de la clé publique principale commencent par "M." Par conséquent, la première clé privée enfant de la clé privée principale est m / 0. La première clé publique enfant est M / 0. Le deuxième petit-enfant du premier enfant est m / 0/1, et ainsi de suite.

L '« ascendance » d'une clé est lue de droite à gauche, jusqu'à ce que vous atteigniez la clé principale dont elle est dérivée. Par exemple, l'identificateur m / x / y / z décrit la clé privée qui est le z-ième enfant de la clé privée parente m / x / y, qui est le y-ième enfant de la clé privée parente m / x, qui est le x-ième enfant de la clé privée principale parent m.

HD path	Key described
m/0	The first (0) child private key from the master private key (m)
m/0/0	The first (0) child private key from the first child (m/0)
m/0'/0	The first (0) normal child from the first *hardened* child (m/0')
m/1/0	The first (0) child private key from the second child (m/1)
M/23/17/0/0	The first (0) child public key from the first child (M/23/17/0) from the 18th child (M/23/17) from the 24th child (M/23)

Tableau 6. Exemples de chemin de portefeuille HD

Naviguer dans l'arborescence du portefeuille HD

La structure arborescente du portefeuille HD offre une flexibilité incroyable. Chaque clé étendue parentale peut avoir 4 milliards d'enfants : 2 milliards d'enfants normaux et 2 milliards d'enfants endurcis. Chacun de ces enfants peut avoir 4 milliards d'enfants supplémentaires, et ainsi de suite. L'arbre peut être aussi profond que vous le souhaitez, avec un nombre infini de générations. Avec toute cette flexibilité, cependant, il devient assez difficile de naviguer dans cet arbre infini. Il est particulièrement difficile de transférer des portefeuilles HD entre les implémentations, car les possibilités d'organisation interne en succursales et sous-succursales sont infinies.

Deux BIP offrent une solution à cette complexité en créant des normes proposées pour la structure des arborescences de portefeuilles HD. Le BIP-43 propose l'utilisation du premier index enfant durci comme identifiant spécial qui signifie le « but » de la structure arborescente. Basé sur BIP-43, un portefeuille HD ne doit utiliser qu'une seule branche de niveau 1 de l'arborescence, le numéro d'index identifiant la structure et l'espace de noms du reste de l'arborescence en définissant son objectif. Par exemple, un portefeuille HD utilisant uniquement la branche m / i & # x27 ; / est destiné à signifier un objectif spécifique et cet objectif est identifié par le numéro d'index « i ».

Prolongeant cette spécification, le BIP-44 propose une structure multi-comptes en tant que numéro « objet » 44 ‘sous le BIP-43. Tous les portefeuilles HD suivant la structure BIP-44 sont identifiés par le fait qu'ils n'utilisaient qu'une seule branche de l'arbre : m / 44 ‘/.

Le BIP-44 spécifie que la structure se compose de cinq niveaux d'arborescence prédéfinis :

```
m / purpose' / coin_type' / account' / change /
address_index
```

Le "but" de premier niveau est toujours fixé à 44 '. Le "coin_type" de deuxième niveau spécifie le type de pièce de monnaie crypto, permettant des portefeuilles HD multidevises où chaque devise a son propre sous-arbre sous le deuxième niveau. Trois devises sont définies pour l'instant : Bitcoin est m / 44 '/ 0', Bitcoin Testnet est m / 44 & # x27 ; / 1 & # x27 ;, et Litecoin est m / 44 & # x27 ; / 2 & # x27 ;.

Le troisième niveau de l'arborescence est « compte », qui permet aux utilisateurs de subdiviser leurs portefeuilles en sous-comptes logiques séparés, à des fins comptables ou organisationnelles. Par exemple, un portefeuille HD peut contenir deux "comptes" bitcoin : m / 44 & # x27 ; / 0 & # x27 ; / 0 & # x27 ; et m / 44 & # x27 ; / 0 & # x27 ; / 1 & # x27 ;. Chaque compte est la racine de son propre sous-arbre.

Au quatrième niveau, « change », un portefeuille HD a deux sous-arborescences, un pour créer des adresses de réception et un pour créer des adresses de changement. Notez que, alors que les niveaux précédents utilisaient une dérivation renforcée, ce niveau utilise une dérivation normale. Cela permet à ce niveau de l'arborescence d'exporter des clés publiques étendues pour une utilisation dans un environnement non sécurisé. Les

adresses utilisables sont dérivées par le portefeuille HD en tant qu'enfants du quatrième niveau, faisant du cinquième niveau de l'arborescence le "address_index". Par exemple, la troisième adresse de réception des paiements Bitcoin dans le compte principal serait M / 44 & # x27 ; / 0 & # x27 ; / 0 & # x27 ; / 0/2. Les exemples de structure de portefeuille BIP-44 HD montrent quelques autres exemples.

HD path	Key described
M/44'/0'/0'/0/2	The third receiving public key for the primary bitcoin account
M/44'/0'/3'/1/14	The fifteenth change-address public key for the fourth bitcoin account
m/44'/2'/0'/0/1	The second private key in the Litecoin main account, for signing transactions

Tableau 7. Exemples de structure de portefeuille BIP-44 HD

Utilisation d'une clé publique étendue sur un magasin Web

Voyons comment les portefeuilles HD sont utilisés en poursuivant notre histoire avec la boutique en ligne de Gabriel.

Gabriel a d'abord créé sa boutique en ligne comme un passe-temps, basé sur une simple page Wordpress hébergée. Son magasin était assez basique avec seulement quelques pages et un bon de commande avec une seule adresse bitcoin.

Gabriel a utilisé la première adresse Bitcoin générée par son appareil Trezor comme adresse Bitcoin principale pour son magasin. De cette façon, tous les paiements entrants seraient payés à une adresse contrôlée par son portefeuille matériel Trezor.

Les clients soumettraient une commande en utilisant le formulaire et enverraient le paiement à l'adresse Bitcoin publiée par Gabriel, déclenchant un e-mail avec les détails de la commande que Gabriel traiterait. Avec seulement quelques commandes par semaine, ce système fonctionnait assez bien.

Cependant, la petite boutique en ligne a connu un grand succès et a attiré de nombreuses commandes de la communauté locale. Bientôt, Gabriel a été submergé. Toutes les commandes payant la même adresse, il est devenu difficile de faire correspondre correctement les commandes et les transactions, en particulier lorsque plusieurs commandes pour le même montant se sont rapprochées.

Le portefeuille HD de Gabriel offre une bien meilleure solution grâce à la possibilité de dériver des clés enfants publiques sans connaître les clés privées. Gabriel peut charger une clé publique étendue (xpub) sur son site Web, qui peut être utilisée pour dériver une adresse unique pour chaque commande client. Gabriel peut dépenser les fonds de son Trezor, mais l'Xpub chargé sur le site Web ne peut que générer des adresses et recevoir des fonds. Cette fonctionnalité des portefeuilles HD est une excellente fonctionnalité de sécurité. Le site Web de Gabriel ne contient aucune clé privée et n'a donc pas besoin de niveaux de sécurité élevés.

Pour exporter le xpub, Gabriel utilise le logiciel Web en conjonction avec le portefeuille matériel Trezor. Le périphérique Trezor doit être branché pour que les clés publiques soient exportées. Notez que les portefeuilles matériels n'exporteront jamais les clés privées - celles-ci restent toujours sur l'appareil. L'exportation d'un xpub à partir d'un portefeuille matériel Trezor montre l'interface Web que Gabriel utilise pour exporter le xpub.

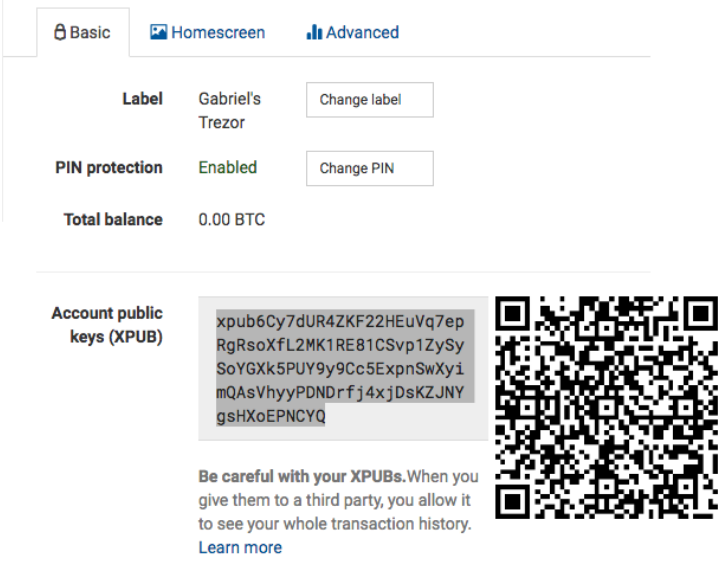


Figure 12. Exportation d'un xpub à partir d'un portefeuille matériel Trezor

Gabriel copie l'Xpub dans le logiciel de boutique Bitcoin de

sa boutique en ligne. Il utilise *BTCPay Server* , qui est une boutique en ligne open source pour une variété de plates-formes d'hébergement et de contenu Web. BTCPay Server utilise xpub pour générer une adresse unique pour chaque achat.

Découverte et gestion de compte

L'entreprise de Gabriel est florissante. Il a fourni sa clé publique étendue (xpub) au *serveur BTCPay* , qui génère des adresses uniques pour les clients sur son site Web. Chaque fois qu'un client sur le site Web de Gabriel clique sur le bouton « Commander » avec une modalité de paiement spécifiée (dans ce cas, bitcoin), *BTCPay Server* génère une nouvelle adresse pour ce client. Plus spécifiquement, le *serveur BTCPay* itère sur l' arborescence *address_index* pour créer une nouvelle adresse à afficher pour le client, comme défini par BIP-44. Si le client décide de changer de mode de paiement ou d'abandonner complètement la transaction, cette adresse bitcoin reste inutilisée et ne sera pas utilisée pour un autre client tout de suite.

À un moment donné, le site Web de Gabriel peut avoir un grand nombre d'adresses en suspens pour les clients effectuant des achats, dont certaines peuvent rester inutilisées et finir par expirer. Une fois ces adresses *expirées* , *BTCPay Server* reviendra pour réutiliser ces adresses pour combler le vide dans *address_index* , mais il devient clair comment il peut y avoir des écarts entre les feuilles *address_index* de l'arbre déterministe hiérarchique où se trouve réellement l'argent.

Disons que Gabriel est intéressé à voir son montant total de bitcoins gagné sur un portefeuille de montre uniquement (un

qui vous permet de voir l'historique des transactions, mais pas de dépenser des fonds) qui est distinct du serveur BTCPay mais également conforme à la norme BIP-44 . Comment ce portefeuille séparé devrait-il chercher des fonds dans ce vaste arbre héréditaire et quand devrait-il cesser de chercher ? La plupart des portefeuilles suivront généralement un processus itératif qui utilise la *limite d'espace* , une limite prédéfinie selon laquelle s'il n'y a pas d'adresses utilisées dans une rangée au-delà de ce nombre limite, le portefeuille arrêtera de rechercher la chaîne d'adresses. La limite d'écart par défaut est généralement fixée à 20. Ceci est détaillé dans BIP-44 .

Conseil

Les limites de l'écart expliquent le phénomène par lequel l'importation d'un portefeuille peut montrer un solde incorrect ou nul. Les fonds ne sont pas perdus, mais la fonction d'importation de portefeuille n'a pas traversé suffisamment de feuilles pour détecter complètement les fonds. De nombreux portefeuilles permettent de modifier cette limite d'espace par défaut, et Gabriel devra peut-être augmenter cette limite pour permettre à son portefeuille d'importer entièrement son historique de transactions.

7

Transactions

Introduction

Les transactions sont la partie la plus importante du système Bitcoin. Tout le reste dans Bitcoin est conçu pour garantir que les transactions peuvent être créées, propagées sur le réseau, validées et finalement ajoutées au grand livre mondial des transactions (la blockchain). Les transactions sont des structures de données qui codent le transfert de valeur entre les participants au système Bitcoin. Chaque transaction est une entrée publique dans la blockchain de Bitcoin, le grand livre de comptabilité mondial à double entrée.

Dans ce chapitre, nous examinerons toutes les différentes formes de transactions, ce qu'elles contiennent, comment les créer, comment elles sont vérifiées et comment elles deviennent partie intégrante de l'enregistrement permanent de toutes les transactions. Lorsque nous utilisons le terme « portefeuille »

dans ce chapitre, nous faisons référence au logiciel qui construit les transactions, pas seulement à la base de données des clés.

Transactions en détail

Dans le chapitre 3 , nous avons examiné la transaction qu'Alice utilisait pour payer le café au café de Bob en utilisant un explorateur de blocs (la transaction d'Alice à Bob's Cafe [fig1]).

L'application de l'explorateur de blocs montre une transaction de « l'adresse » d'Alice à « l'adresse » de Bob. Il s'agit d'une vue très simplifiée de ce qui est contenu dans une transaction. En fait, comme nous le verrons dans ce chapitre, la plupart des informations affichées sont construites par l'explorateur de blocs et ne sont pas réellement dans la transaction.

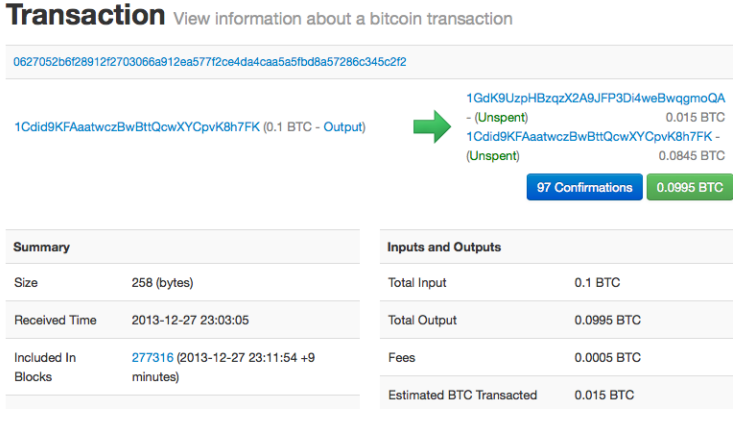


Figure 1. Transaction d'Alice dans le cafe de Bob

Transactions - dans les coulisses

Dans les coulisses, une transaction réelle est très différente d'une transaction fournie par un explorateur de blocs typique. En fait, la plupart des constructions de haut niveau que nous voyons dans les différentes interfaces utilisateur de l'application Bitcoin *n'existent pas réellement* dans le système Bitcoin.

Nous pouvons utiliser l'interface de ligne de commande de Bitcoin Core (getrawtransaction et decoderawtransaction) pour récupérer la transaction "brute" d'Alice, la décoder et voir ce qu'elle contient. Le résultat ressemble à ceci :

La transaction d'Alice décodée

```
{
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid":
      "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999
      "vout": 0,
      "scriptSig" :
      "3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb
      0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "scriptPubKey": "OP_DUP OP_HASH160
      ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

```
      OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value": 0.08450000,
      "scriptPubKey": "OP_DUP OP_HASH160
      7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
      OP_EQUALVERIFY OP_CHECKSIG",
    }
  ]
}
```

Vous remarquerez peut-être certaines choses à propos de cette transaction, principalement les choses qui manquent ! Où est l'adresse d'Alice ? Où est l'adresse de Bob ? Où est l'entrée 0.1 "envoyée" par Alice ? Dans Bitcoin, il n'y a pas de pièces, pas d'expéditeurs, pas de destinataires, pas de soldes, pas de comptes et pas d'adresses. Toutes ces choses sont construites à un niveau supérieur pour le bénéfice de l'utilisateur, pour rendre les choses plus faciles à comprendre.

Vous pouvez également remarquer beaucoup de champs étranges et indéchiffrables et de chaînes hexadécimales. Ne vous inquiétez pas, nous expliquerons chaque champ présenté ici en détail dans ce chapitre.

Sorties et entrées de transaction

Le bloc de construction fondamental d'une transaction Bitcoin est une *sortie de transaction* . Les sorties de transaction sont

des morceaux indivisibles de monnaie bitcoin, enregistrés sur la blockchain et reconnus comme valides par l'ensemble du réseau. Les nœuds complets Bitcoin suivent toutes les sorties disponibles et dépensables, appelées *sorties de transaction non dépensées* ou *UTXO* . La collection de tous les UTXO est connue sous le *nom de jeu UTXO* et compte actuellement des millions d'UTXO. L'ensemble UTXO grandit au fur et à mesure de la création d'un nouvel UTXO et se réduit lorsque l'UTXO est consommé. Chaque transaction représente un changement (transition d'état) dans l'ensemble UTXO.

Lorsque nous disons que le portefeuille d'un utilisateur a "reçu" du bitcoin, ce que nous voulons dire, c'est que le portefeuille a détecté un UTXO qui peut être dépensé avec l'une des clés contrôlées par ce portefeuille. Ainsi, le « solde » bitcoin d'un utilisateur est la somme de tous les UTXO que le portefeuille de l'utilisateur peut dépenser et qui peuvent être dispersés parmi des centaines de transactions et des centaines de blocs. Le concept de solde est créé par l'application portefeuille. Le portefeuille calcule le solde de l'utilisateur en scannant la blockchain et en agrégeant la valeur de tout UTXO que le portefeuille peut dépenser avec les clés qu'il contrôle. La plupart des portefeuilles maintiennent une base de données ou utilisent un service de base de données pour stocker un ensemble de référence rapide de tous les UTXO qu'ils peuvent dépenser avec les clés qu'ils contrôlent.

Une sortie de transaction peut avoir une valeur arbitraire (entière) dénommée comme un multiple de satoshis. Tout comme les dollars peuvent être divisés à deux décimales sous forme de centimes, le bitcoin peut être divisé en huit

décimales sous forme de satoshis. Bien qu'une sortie puisse avoir n'importe quelle valeur arbitraire, une fois créée, elle est indivisible. C'est une caractéristique importante des sorties qui doit être soulignée : les sorties sont *des* unités de valeur *discrètes* et *indivisibles* , libellées en satoshis entiers. Une sortie non dépensée ne peut être consommée dans son intégralité que par une transaction.

Si un UTXO est supérieur à la valeur souhaitée d'une transaction, il doit toujours être consommé dans son intégralité et le changement doit être généré dans la transaction. En d'autres termes, si vous avez un UTXO d'une valeur de 20 bitcoin et que vous voulez payer seulement 1 bitcoin, votre transaction doit consommer l'intégralité de l'UTXO de 20 bitcoins et produire deux sorties : une payant 1 bitcoin à votre destinataire souhaité et une autre payant 19 bitcoin en échange. retour à votre portefeuille. En raison de la nature indivisible des sorties de transaction, la plupart des transactions Bitcoin devront générer des changements.

Imaginez un acheteur achetant une boisson à 1,50 $, cherchant dans son portefeuille et essayant de trouver une combinaison de pièces de monnaie et de billets de banque pour couvrir le coût de 1,50 $. L'acheteur choisira la monnaie exacte si elle est disponible, par exemple un billet d'un dollar et deux quarts (un quart vaut 0,25 $), ou une combinaison de coupures plus petites (six quarts), ou si nécessaire, une unité plus grande comme un billet de 5 $. Si elle remet trop d'argent, disons 5 $, au propriétaire de la boutique, elle s'attendra à 3,50 $ de monnaie, qu'elle remettra dans son portefeuille et disposera pour de futures transactions.

De même, une transaction bitcoin doit être créée à partir de l'UTXO d'un utilisateur dans toutes les dénominations disponibles pour l'utilisateur. Les utilisateurs ne peuvent pas réduire de moitié un UTXO, pas plus qu'ils ne peuvent couper un billet d'un dollar de moitié et l'utiliser comme monnaie. L'application de portefeuille de l'utilisateur choisira généralement parmi l'UTXO disponible de l'utilisateur pour composer un montant supérieur ou égal au montant de transaction souhaité.

Comme dans la vraie vie, l'application Bitcoin peut utiliser plusieurs stratégies pour satisfaire le montant de l'achat : combiner plusieurs unités plus petites, trouver le changement exact ou utiliser une seule unité plus grande que la valeur de la transaction et effectuer le changement. Tout cet assemblage complexe d'UTXO utilisable est effectué automatiquement par le portefeuille de l'utilisateur et est invisible pour les utilisateurs. Cela n'est pertinent que si vous construisez par programme des transactions brutes à partir d'UTXO.

Une transaction consomme des sorties de transaction non dépensées précédemment enregistrées et crée de nouvelles sorties de transaction qui peuvent être consommées par une transaction future. De cette façon, des morceaux de valeur bitcoin passent de propriétaire à propriétaire dans une chaîne de transactions consommant et créant UTXO.

L'exception à la chaîne de sortie et d'entrée est un type spécial de transaction appelé transaction *coinbase* , qui est la première transaction de chaque bloc. Cette transaction est placée là par le mineur « gagnant » et crée un tout nouveau bitcoin payable à ce mineur en récompense de l'exploitation minière. Cette

transaction spéciale coinbase ne consomme pas d'UTXO ; au lieu de cela, il a un type d'entrée spécial appelé « coinbase ». C'est ainsi que la masse monétaire de Bitcoin est créée pendant le processus d'extraction, comme nous le verrons dans [le chapitre sur l'extraction] .

Conseil

Qu'est-ce qui vient en premier ? Entrées ou sorties, la poule ou l'œuf ? À proprement parler, les sorties viennent en premier parce que les transactions coinbase, qui génèrent de nouveaux bitcoins, n'ont aucune entrée et créent des sorties à partir de rien.

Sorties de transaction

Chaque transaction Bitcoin crée des sorties, qui sont enregistrées dans le registre Bitcoin. Presque toutes ces sorties, à une exception près, créent des morceaux de bitcoin utilisables appelés UTXO, qui sont ensuite reconnus par l'ensemble du réseau et disponibles pour que le propriétaire les dépense dans une future transaction.

UTXO sont suivis par chaque client bitcoin à nœud complet dans l'ensemble UTXO. Les nouvelles transactions consomment (dépensent) une ou plusieurs de ces sorties de l'ensemble UTXO.

Les sorties de transaction se composent de deux parties :

- Une quantité de bitcoin, libellée en *satoshis* , la plus petite

unité de bitcoin
- Un puzzle cryptographique qui détermine les conditions requises pour dépenser la sortie

Le casse - tête cryptographique est également connu comme un *script de verrouillage* , un *script témoin* , ou un scriptPubKey.

Le langage de script de transaction, utilisé dans le script de verrouillage mentionné précédemment, est décrit en détail dans Scripts de transaction et langage de script .

Maintenant, regardons la transaction d'Alice (montrée précédemment dans Transactions - Dans les coulisses) et voyons si nous pouvons identifier les sorties. Dans l'encodage JSON, les sorties sont dans un tableau (liste) nommé vout :

```
"vout": [
  {
    "value": 0.01500000,
    "scriptPubKey": "OP_DUP OP_HASH160
    ab68025513c3dbd2f7b92a94e0581f5d50f654e7
    OP_EQUALVERIFY    OP_CHECKSIG"
  },
  {
    "value": 0.08450000,
    "scriptPubKey": "OP_DUP OP_HASH160
    7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
    OP_EQUALVERIFY OP_CHECKSIG",
  }
]
```

Comme vous pouvez le voir, la transaction contient deux

sorties. Chaque sortie est définie par une valeur et un puzzle cryptographique. Dans l'encodage indiqué par Bitcoin Core, la valeur est affichée en bitcoin, mais dans la transaction elle-même, elle est enregistrée sous la forme d'un entier libellé en satoshis. La deuxième partie de chaque sortie est le puzzle cryptographique qui définit les conditions de dépenses. Bitcoin Core le montre comme scriptPubKey et nous montre une représentation lisible par l'homme du script.

Le sujet du verrouillage et du déverrouillage de UTXO sera abordé plus tard, dans Script de Construction (Lock + Unlock) . Le langage de script utilisé pour le script dans scriptPubKey est présenté dans Scripts de transaction et langage de script . Mais avant de nous plonger dans ces sujets, nous devons comprendre la structure globale des entrées et sorties des transactions.

Sérialisation des transactions - sorties

Lorsque les transactions sont transmises sur le réseau ou échangées entre applications, elles sont *sérialisées* . La sérialisation est le processus de conversion de la représentation interne d'une structure de données dans un format qui peut être transmis un octet à la fois, également appelé flux d'octets. La sérialisation est le plus couramment utilisée pour coder des structures de données pour la transmission sur un réseau ou pour le stockage dans un fichier. Le format de sérialisation d'une sortie de transaction est indiqué dans Sérialisation de sortie de transaction .

Size	Field	Description
8 bytes (little-endian)	Amount	Bitcoin value in satoshis (10^{-8} bitcoin)
1–9 bytes (VarInt)	Locking-Script Size	Locking-Script length in bytes, to follow
Variable	Locking-Script	A script defining the conditions needed to spend the output

Tableau 1. Sérialisation de sortie de transaction

La plupart des bibliothèques et des frameworks Bitcoin ne stockent pas les transactions en interne sous forme de flux d'octets, car cela nécessiterait une analyse complexe chaque fois que vous auriez besoin d'accéder à un seul champ. Pour plus de commodité et de lisibilité, les bibliothèques Bitcoin stockent les transactions en interne dans des structures de données (généralement des structures orientées objet).

Le processus de conversion de la représentation par flux d'octets d'une transaction à la structure de données de représentation interne d'une bibliothèque est appelé *désérialisation* ou *analyse de transaction* . Le processus de reconversion en flux d'octets pour la transmission sur le réseau, pour le hachage ou pour le stockage sur disque s'appelle la *sérialisation* . La plupart des bibliothèques Bitcoin ont des fonctions intégrées pour la sérialisation et la désérialisation des transactions.

Voyez si vous pouvez décoder manuellement la transaction d'Alice à partir de la forme hexadécimale sérialisée, en trouvant

certains des éléments que nous avons vus précédemment. La section contenant les deux sorties est mise en évidence dans la transaction d'Alice, sérialisée et présentée en notation hexadécimale pour vous aider :

Exemple 1. Transaction d'Alice, sérialisée et présentée en notation hexadécimale

0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2 836dd73 4d2804fe65fa35779000000008b483045022100884d 142d86652a3f47 ba4746ec719bbfbd040a570b1deccbb6498c7 5c4ae24cb02204b9f039 ff08df09cbe9f6addac960298cad530a8 63ea8f53982c09db8f6e3813 01410484ecc0d46f1918b30928f a0e4ed99f16a0fb4fde0735e7ade84 16ab9fe423cc541233637 6789d172787ec3457eee41c04f4938de5cc1 7b4a10fa336a8d 752adfffffffff02 **60e31600000000001976a914ab6 8025513c3 dbd2f7b92a94e0581f5d50f654e788acd0ef800000000000 1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac 00000000**

Voici quelques conseils :

- Il y a deux sorties dans la section en surbrillance, chacune sérialisée comme indiqué dans la sérialisation de sortie de transaction .
- La valeur de 0,015 bitcoin est de 1 500 000 satoshis. C'est 16 e3 60 en hexadécimal.
- Dans la transaction sérialisée, la valeur 16 e3 60 est codée dans l'ordre des octets petit boutiste (octet le moins significatif en premier), elle ressemble donc à 60 e3 16.
- La longueur de scriptPubKey est de 25 octets, soit 19 en hexadécimal.

Entrées de transaction

Les entrées de transaction identifient (par référence) quel UTXO sera consommé et fournissent une preuve de propriété via un script de déverrouillage.

Pour construire une transaction, un portefeuille sélectionne parmi l'UTXO qu'il contrôle, UTXO avec une valeur suffisante pour effectuer le paiement demandé. Parfois, un UTXO suffit, d'autres fois plus d'un est nécessaire. Pour chaque UTXO qui sera consommé pour effectuer ce paiement, le portefeuille crée une entrée pointant vers l'UTXO et le déverrouille avec un script de déverrouillage.

Examinons plus en détail les composants d'une entrée. La première partie d'une entrée est un pointeur vers un UTXO par référence au hachage de la transaction et un index de sortie, qui identifie l'UTXO spécifique dans cette transaction. La deuxième partie est un script de déverrouillage, que le portefeuille construit afin de satisfaire les conditions de dépenses définies dans l'UTXO. Le plus souvent, le script de déverrouillage est une signature numérique et une clé publique prouvant la propriété du bitcoin. Cependant, tous les scripts de déverrouillage ne contiennent pas de signatures. La troisième partie est un numéro de séquence, qui sera discuté plus tard.

Prenons l'exemple de Transactions — Dans les coulisses . Les entrées de transaction sont un tableau (liste) appelé vin :

Les entrées de transaction dans la transaction d'Alice

```
"vin": [
  {
    "txid":
    "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f
    "vout": 0,
    "scriptSig" :
    "3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6
    0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe4
    "sequence": 4294967295
  }
]
```

Comme vous pouvez le voir, il n'y a qu'une seule entrée dans la liste (car un UTXO contenait une valeur suffisante pour effectuer ce paiement). L'entrée contient quatre éléments :

- Un ID de transaction, référençant la transaction qui contient l'UTXO dépensé
- Un index de sortie (vout), identifiant quel UTXO de cette transaction est référencé (le premier est zéro)
- Un scriptSig, qui satisfait les conditions placées sur l'UTXO, le débloquant pour les dépenses
- Un numéro de séquence (à discuter plus tard)

Dans la transaction d'Alice, l'entrée pointe vers l'ID de transaction :

```
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18
```

et l'indice de sortie 0 (c'est-à-dire le premier UTXO créé par

cette transaction). Le script de déverrouillage est construit par le portefeuille d'Alice en récupérant d'abord l'UTXO référencé, en examinant son script de verrouillage, puis en l'utilisant pour créer le script de déverrouillage nécessaire pour le satisfaire.

En regardant simplement l'entrée, vous avez peut-être remarqué que nous ne savons rien de cet UTXO, à part une référence à la transaction parente qui le contient. Nous ne connaissons pas sa valeur (montant en satoshi), et nous ne connaissons pas le script de verrouillage qui définit les conditions de sa dépense. Pour trouver ces informations, nous devons récupérer l'UTXO référencé en récupérant la transaction parente qui le contient. Notez que comme la valeur de l'entrée n'est pas explicitement indiquée, nous devons également utiliser l'UTXO référencé afin de calculer les frais qui seront payés dans cette transaction (voir Frais de transaction).

Ce n'est pas seulement le portefeuille d'Alice qui a besoin de récupérer UTXO référencé dans les entrées. Une fois cette transaction diffusée sur le réseau, chaque nœud de validation devra également récupérer l'UTXO référencé dans les entrées de transaction afin de valider la transaction.

Les transactions en elles-mêmes semblent incomplètes car elles manquent de contexte. Ils référencent UTXO dans leurs entrées mais sans récupérer cet UTXO nous ne pouvons pas connaître la valeur des entrées ou leurs conditions de verrouillage. Lors de l'écriture d'un logiciel bitcoin, chaque fois que vous décodez une transaction dans l'intention de la valider ou de compter les frais ou de vérifier le script de déverrouillage, votre code devra d'abord récupérer l'UTXO référencé de la blockchain

afin de construire le contexte implicite mais non présent dans les références UTXO des entrées. Par exemple, pour calculer le montant payé en frais, vous devez connaître la somme des valeurs des entrées et des sorties. Mais sans récupérer l'UTXO référencé dans les entrées, vous ne connaissez pas leur valeur. Ainsi, une opération apparemment simple comme le comptage des frais dans une seule transaction implique en fait plusieurs étapes et des données provenant de plusieurs transactions.

Nous pouvons utiliser la même séquence de commandes avec Bitcoin Core que celle utilisée lors de la récupération de la transaction d'Alice (getrawtransaction et decoderawtransacti on). Avec cela, nous pouvons obtenir l'UTXO référencé dans l'entrée de la transaction d'Alice et jeter un œil :

UTXO de la transaction précédente, référencé dans l'entrée de la transaction d'Alice

```
"vout": [
   {
     "value": 0.10000000,
     "scriptPubKey": "OP_DUP OP_HASH160
     7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
     OP_EQUALVERIFY OP_CHECKSIG"
```

On voit que cet UTXO a une valeur de 0,1 BTC et qu'il a un script de verrouillage (scriptPubKey) qui contient "OP_DUP OP_HASH160…".

Conseil

Pour bien comprendre la transaction d'Alice, nous avons dû récupérer la transaction précédente référencée comme entrée. Une fonction qui récupère les transactions précédentes et les sorties de transactions non dépensées est très courante et existe dans presque toutes les bibliothèques et API Bitcoin.

Sérialisation des transactions - entrées

Lorsque les transactions sont sérialisées pour la transmission sur le réseau, leurs entrées sont codées dans un flux d'octets comme indiqué dans la sérialisation des entrées de transaction .

Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent; first one is 0
1–9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking script
4 bytes	Sequence Number	Used for locktime or disabled (0xFFFFFFFF)

Tableau 2. Sérialisation des entrées de transaction

Comme pour les sorties, voyons si nous pouvons trouver les entrées de la transaction d'Alice au format sérialisé. Tout d'abord, les entrées décodées :

```
"vin": [
  {
    "txid":
    "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f
    "vout": 0,
    "scriptSig" :
    "3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6
    0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe4
    "sequence": 4294967295
  }
],
```

Voyons maintenant si nous pouvons identifier ces champs dans le codage hexadécimal sérialisé de la transaction d'Alice, sérialisé et présenté en notation hexadécimale :

Exemple 2. Transaction d'Alice, sérialisée et présentée en notation hexadécimale

010000000**1186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd73 4d2804fe65fa35779000000008b48304502210088 4d142d86652a3f47 ba4746ec719bbfbd040a570b1deccbb6 498c75c4ae24cb02204b9f039 ff08df09cbe9f6addac960298 cad530a863ea8f53982c09db8f6e3813 01410484ecc0d46f1 918b30928fa0e4ed99f16a0fb4fde0735e7ade84 16ab9fe42**

3cc5412336376789d172787ec3457eee41c04f4938de5cc1 7b4a10fa336a8d752adfffffffff0260e31600000000001976a91 4ab6 8025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef800 000000000 1976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc 025a888ac00000 000

Astuces :

- L'ID de transaction est sérialisé dans l'ordre des octets inversés, il commence donc par (hexadécimal) 18 et se termine par 79
- L'index de sortie est un groupe de zéros de 4 octets, facile à identifier
- La longueur du scriptSig est de 139 octets, soit 8b en hexadécimal
- Le numéro de séquence est réglé sur FFFFFFFF, encore une fois facile à identifier

ScriptSig est un type spécifique de script de déverrouillage qui, lorsqu'il est sérialisé pour la transmission sur le réseau, les entrées sont codées dans un flux d'octets comme indiqué dans la sérialisation d'entrée ScriptSig . La sérialisation du champ de signature est détaillée dans Sérialisation des signatures (DER) . Le champ de signature comprend également un type de hachage de signature (SIGHASH), qui est détaillé dans Types de hachage de signature (SIGHASH) .

Size	Field	Description
1–9 bytes (VarInt)	Signature Size	Signature length in bytes, to follow
Variable	Signature	A signature that is produced by the user's wallet from his or her private key, which includes a SIGHASH
1–9 bytes (VarInt)	Public Key Size	Public key length in bytes, to follow
Variable	Public Key	The public key, unhashed

Tableau 3. Sérialisation d'entrée ScriptSig

Les frais de transaction

La plupart des transactions incluent des frais de transaction, qui compensent les mineurs de bitcoins pour la sécurisation du réseau. Les frais servent également de mécanisme de sécurité eux-mêmes, en rendant économiquement impossible pour les attaquants d'inonder le réseau de transactions. L'exploitation minière et les frais et récompenses perçus par les mineurs sont discutés plus en détail dans [extraction minière] .

Cette section examine comment les frais de transaction sont inclus dans une transaction typique. La plupart des portefeuilles calculent et incluent automatiquement les frais de transaction. Toutefois, si vous créez des transactions par programme ou à l'aide d'une interface de ligne de commande, vous devez manuellement comptabiliser et inclure ces frais.

Les frais de transaction servent d'incitation à inclure (mine) une transaction dans le bloc suivant et aussi de dissuasion contre les abus du système en imposant un petit coût à chaque transaction. Les frais de transaction sont collectés par le mineur qui extrait le bloc qui enregistre la transaction sur la blockchain.

Les frais de transaction sont calculés en fonction de la taille de la transaction en kilo-octets, et non de la valeur de la transaction en bitcoin. Dans l'ensemble, les frais de transaction sont fixés en fonction des forces du marché au sein du réseau Bitcoin. Les mineurs hiérarchisent les transactions en fonction de nombreux critères différents, y compris les frais, et peuvent même traiter des transactions gratuitement dans certaines circonstances. Les frais de transaction affectent la priorité de traitement, ce qui signifie qu'une transaction avec des frais suffisants est susceptible d'être incluse dans le prochain bloc extrait, alors qu'une transaction avec des frais insuffisants ou sans frais peut être retardée, traitée au mieux après quelques blocs, ou pas du tout traité. Les frais de transaction ne sont pas obligatoires et les transactions sans frais pourraient être traitées ultérieurement ; cependant, l'inclusion des frais de transaction encourage le traitement prioritaire.

Au fil du temps, la façon dont les frais de transaction sont calculés et leur effet sur la hiérarchisation des transactions a évolué. Au début, les frais de transaction étaient fixes et constants sur l'ensemble du réseau. Progressivement, la structure tarifaire s'est assouplie et peut être influencée par les forces du marché, en fonction de la capacité du réseau et du volume des transactions. Depuis au moins le début de 2016, les limites de capacité en bitcoin ont créé une concurrence

entre les transactions, entraînant des frais plus élevés et faisant effectivement des transactions gratuites une chose du passé. Les transactions sans frais ou avec des frais très bas sont rarement exploitées et parfois même pas propagées sur le réseau.

Dans Bitcoin Core, les politiques de relais de frais sont définies par l'option minrelaytxfee. Le minrelaytxfee par défaut actuel est de 0,00001 bitcoin ou un centième de millibitcoin par kilo-octet. Par conséquent, par défaut, les transactions avec des frais inférieurs à 0,00001 bitcoin sont traitées comme gratuites et ne sont relayées que s'il y a de la place dans le mempool ; sinon, ils sont abandonnés. Les nœuds Bitcoin peuvent remplacer la politique de relais de frais par défaut en ajustant la valeur de minrelaytxfee.

Tout service Bitcoin qui crée des transactions, y compris les portefeuilles, les échanges, les applications de vente au détail, etc., *doit* mettre en œuvre des frais dynamiques. Les frais dynamiques peuvent être mis en œuvre via un service tiers d'estimation des frais ou avec un algorithme d'estimation des frais intégré. En cas de doute, commencez par un service tiers et au fur et à mesure que vous acquérez de l'expérience, concevez et implémentez votre propre algorithme si vous souhaitez supprimer la dépendance tierce.

Les algorithmes d'estimation des frais calculent les frais appropriés, en fonction de la capacité et des frais offerts par les transactions « concurrentes ». Ces algorithmes vont du simpliste (frais moyen ou médian dans le dernier bloc) au sophistiqué (analyse statistique). Ils estiment les frais nécessaires (en satoshis par octet) qui donneront à une transaction une forte probabilité

d'être sélectionnée et incluse dans un certain nombre de blocs. La plupart des services offrent aux utilisateurs la possibilité de choisir des frais de priorité élevée, moyenne ou faible. Une priorité élevée signifie que les utilisateurs paient des frais plus élevés, mais la transaction est susceptible d'être incluse dans le bloc suivant. Une priorité moyenne et faible signifie que les utilisateurs paient des frais de transaction moins élevés, mais les transactions peuvent prendre beaucoup plus de temps à confirmer.

De nombreuses applications de portefeuille utilisent des services tiers pour le calcul des frais. Un service populaire est *https ://bitcoinfees.earn.com/* , qui fournit une API et un graphique visuel montrant les frais en satoshi / octet pour différentes priorités.

> *Conseil*
>
> *Les frais statiques ne sont plus viables sur le réseau Bitcoin. Les portefeuilles qui fixent des frais statiques produiront une expérience utilisateur médiocre car les transactions resteront souvent « bloquées » et resteront non confirmées. Les utilisateurs qui ne comprennent pas les transactions et les frais Bitcoin sont consternés par les transactions "bloquées" car ils pensent avoir perdu leur argent.*

Le graphique du service d'estimation des frais bitcoinfees.earn.com montre l'estimation en temps réel des frais par incréments de 10 satoshi / octet et le temps de confirmation attendu (en

minutes et en nombre de blocs) pour les transactions avec des frais dans chaque plage. Pour chaque fourchette de frais (par exemple, 61 à 70 satoshi / octet), deux barres horizontales indiquent le nombre de transactions non confirmées (1405) et le nombre total de transactions au cours des dernières 24 heures (102 975), avec des frais dans cette fourchette. Sur la base du graphique, les frais de priorité élevée recommandés à ce moment étaient de 80 satoshi / octet, des frais susceptibles d'entraîner l'extraction de la transaction dans le bloc suivant (délai de blocage nul). Pour la perspective, la taille médiane des transactions est de 226 octets, donc les frais recommandés pour cette taille de transaction seraient de 18 080 satoshis (0,00018080 BTC).

Les données d'estimation des frais peuvent être récupérées via une simple API HTTP REST, à l' adresse https ://bitcoinfees.earn.com/api/v1/fees/recommended . Par exemple, sur la ligne de commande à l'aide de la commande curl :

Utilisation de l'API d'estimation des frais

```
$ curl
https://bitcoinfees.earn.com/api/v1/fees/recommended

{"fastestFee":80,"halfHourFee":80,"hourFee":60}
```

L'API renvoie un objet JSON avec l'estimation des frais actuels pour la confirmation la plus rapide (fastFee), la confirmation dans les trois blocs (halfHourFee) et six blocs (hourFee), en satoshi par octet.

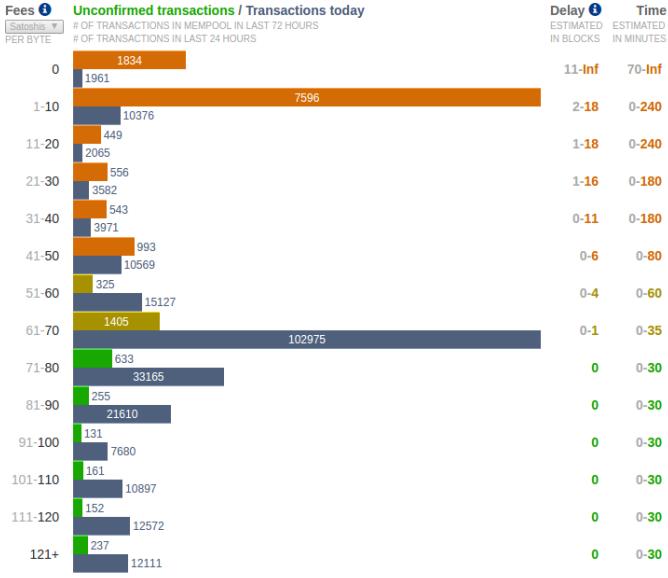


Figure 2. Service d'estimation des frais bitcoinfees.earn.com

Ajout de frais aux transactions

La structure de données des transactions n'a pas de champ pour les frais. Au lieu de cela, les frais sont sous-entendus comme la différence entre la somme des intrants et la somme des extrants. Tout montant excédentaire qui reste après que toutes les sorties ont été déduites de toutes les entrées est la redevance qui est perçue par les mineurs :

Les frais de transaction sont implicites, car l'excédent des intrants moins les extrants :

```
Fees = Sum(Inputs) – Sum(Outputs)
```

C'est un élément quelque peu déroutant des transactions et un point important à comprendre, car si vous construisez vos propres transactions, vous devez vous assurer de ne pas inclure par inadvertance des frais très élevés en sous-dépensant les intrants. Cela signifie que vous devez tenir compte de toutes les entrées, si nécessaire en créant du changement, ou vous finirez par donner aux mineurs un très gros pourboire !

Par exemple, si vous consommez un UTXO de 20 bitcoins pour effectuer un paiement de 1 bitcoin, vous devez inclure une sortie de modification de 19 bitcoins dans votre portefeuille. Sinon, les 19 bitcoins "restants" seront comptés comme des frais de transaction et seront collectés par le mineur qui extrait votre transaction dans un bloc. Bien que vous receviez un traitement prioritaire et que vous rendiez un mineur très heureux, ce n'est probablement pas ce que vous vouliez.

Avertissement

Si vous oubliez d'ajouter une sortie de modification dans une transaction construite manuellement, vous paierez la modification en tant que frais de transaction. Dire "Gardez le changement !" au mineur n'est peut-être pas ce que vous vouliez vraiment.

Voyons comment cela fonctionne dans la pratique, en examinant à nouveau l'achat de café d'Alice. Alice veut dépenser

0,015 bitcoin pour payer le café. Pour s'assurer que cette transaction est traitée rapidement, elle voudra inclure des frais de transaction, disons 0,0005. Cela signifiera que le coût total de la transaction sera de 0,0155. Son portefeuille doit donc se procurer un ensemble d'UTXO qui ajoute jusqu'à 0,0155 bitcoin ou plus et, si nécessaire, créer du changement. Disons que son portefeuille a un UTXO 0,1 bitcoin disponible. Il devra donc consommer cet UTXO, créer une sortie vers Bob's Cafe pour 0,015 et une deuxième sortie avec 0,0845 bitcoin en échange de son propre portefeuille, laissant 0,0005 bitcoin non alloué, en tant que frais implicite pour la transaction.

Regardons maintenant un scénario différent. Eugenia, la directrice de notre association caritative pour enfants aux Philippines, a organisé une collecte de fonds pour acheter des manuels scolaires pour les enfants. Elle a reçu plusieurs milliers de petits dons de personnes du monde entier, totalisant 50 bitcoins, son portefeuille est donc rempli de très petits paiements (UTXO). Maintenant, elle souhaite acheter des centaines de manuels scolaires à un éditeur local, en payant en bitcoin.

Comme l'application de portefeuille d'Eugenia essaie de construire une seule transaction de paiement plus grande, elle doit s'approvisionner à partir de l'ensemble UTXO disponible, qui est composé de nombreux montants plus petits. Cela signifie que la transaction résultante proviendra de plus d'une centaine d'UTXO de petite valeur en tant qu'entrées et d'une seule sortie, payant l'éditeur du livre. Une transaction avec autant d'entrées sera supérieure à un kilo-octet, peut-être plusieurs kilo-octets. En conséquence, il faudra des frais beaucoup plus élevés que la transaction de taille médiane.

L'application de portefeuille d'Eugenia calculera les frais appropriés en mesurant la taille de la transaction et en la multipliant par les frais par kilo-octet. De nombreux portefeuilles surpayeront les frais pour les transactions plus importantes afin de garantir que la transaction soit traitée rapidement. Les frais plus élevés ne sont pas dus au fait qu'Eugenia dépense plus d'argent, mais parce que sa transaction est plus complexe et de plus grande taille - les frais sont indépendants de la valeur en bitcoin de la transaction.

Scripts de transaction et langage de script

Le langage de script de transaction bitcoin, appelé *Script* , est un langage d'exécution basé sur une pile de notation inversée de type Forth. Si cela ressemble à du charabia, vous n'avez probablement pas étudié les langages de programmation des années 1960, mais ce n'est pas grave - nous allons tout expliquer dans ce chapitre. Le script de verrouillage placé sur un UTXO et le script de déverrouillage sont écrits dans ce langage de script. Lorsqu'une transaction est validée, le script de déverrouillage dans chaque entrée est exécuté à côté du script de verrouillage correspondant pour voir s'il satisfait la condition de dépense.

Le script est un langage très simple qui a été conçu pour être limité dans sa portée et exécutable sur une gamme de matériel, peut-être aussi simple qu'un périphérique embarqué. Il nécessite un traitement minimal et ne peut pas faire beaucoup des choses sophistiquées que les langages de programmation modernes peuvent faire. Pour son utilisation dans la validation

de l'argent programmable, il s'agit d'une fonction de sécurité délibérée.

Aujourd'hui, la plupart des transactions traitées via le réseau bitcoin ont le formulaire « Paiement à l'adresse bitcoin de Bob » et sont basées sur un script appelé script Pay-to-Public-Key-Hash. Cependant, les transactions bitcoin ne sont pas limitées au script "Paiement à l'adresse bitcoin de Bob". En fait, les scripts de verrouillage peuvent être écrits pour exprimer une grande variété de conditions complexes. Afin de comprendre ces scripts plus complexes, nous devons d'abord comprendre les bases des scripts de transaction et du langage de script.

Dans cette section, nous allons démontrer les composants de base du langage de script de transaction bitcoin et montrer comment il peut être utilisé pour exprimer des conditions simples de dépenses et comment ces conditions peuvent être satisfaites en déverrouillant des scripts.

Conseil

La validation des transactions Bitcoin n'est pas basée sur un modèle statique, mais est plutôt réalisée par l'exécution d'un langage de script. Ce langage permet d' exprimer une variété presque infinie de conditions. C'est ainsi que le bitcoin obtient la puissance de « l'argent programmable ».

Incomplétude de Turing

Le langage de script de transaction bitcoin contient de nombreux opérateurs, mais il est délibérément limité d'une manière importante : il n'y a pas de boucles ou de capacités de contrôle de flux complexes autres que le contrôle de flux conditionnel. Cela garantit que le langage n'est pas *Turing Complete* , ce qui signifie que les scripts ont une complexité limitée et des temps d'exécution prévisibles. Le script n'est pas un langage à usage général. Ces limitations garantissent que le langage ne peut pas être utilisé pour créer une boucle infinie ou une autre forme de « bombe logique » qui pourrait être intégrée dans une transaction d'une manière qui provoque une attaque par déni de service contre le réseau Bitcoin. N'oubliez pas que chaque transaction est validée par chaque nœud complet du réseau Bitcoin. Un langage limité empêche le mécanisme de validation des transactions d'être utilisé comme une vulnérabilité.

Vérification apatride

Le langage de script de transaction bitcoin est sans état, en ce sens qu'il n'y a pas d'état avant l'exécution du script, ni d'état enregistré après l'exécution du script. Par conséquent, toutes les informations nécessaires pour exécuter un script sont contenues dans le script. Un script s'exécutera de la même manière sur n'importe quel système. Si votre système vérifie un script, vous pouvez être sûr que tous les autres systèmes du réseau Bitcoin vérifieront également le script, ce qui signifie qu'une transaction valide est valide pour tout le monde et tout le monde le sait. Cette prévisibilité des résultats est un avantage

essentiel du système Bitcoin.

Construction de script (verrouillage + déverrouillage)

Le moteur de validation des transactions de Bitcoin repose sur deux types de scripts pour valider les transactions : un script de verrouillage et un script de déverrouillage.

Un script de verrouillage est une condition de dépense placée sur une sortie : il spécifie les conditions qui doivent être remplies pour dépenser la sortie à l'avenir. Historiquement, le script de verrouillage était appelé *scriptPubKey* , car il contenait généralement une clé publique ou une adresse bitcoin (hachage de clé publique). Dans ce livre, nous l'appelons un "script de verrouillage" pour reconnaître la gamme beaucoup plus large de possibilités de cette technologie de script. Dans la plupart des applications Bitcoin, ce que nous appelons un script de verrouillage apparaîtra dans le code source sous le nom de scriptPubKey. Vous verrez également le script de verrouillage appelé script *témoin* ou plus généralement en tant que *puzzle cryptographique* . Ces termes signifient tous la même chose, à différents niveaux d'abstraction.

Un script de déverrouillage est un script qui « résout » ou satisfait les conditions placées sur une sortie par un script de verrouillage et permet de dépenser la sortie. Les scripts de déverrouillage font partie de chaque entrée de transaction. La plupart du temps, ils contiennent une signature numérique produite par le portefeuille de l'utilisateur à partir de sa clé privée. Historiquement, le script de déverrouillage s'appelait *scriptSig* , car il contenait généralement une signature numérique. Dans

la plupart des applications Bitcoin, le code source fait référence au script de déverrouillage sous le nom de scriptSig. Vous verrez également le script de déverrouillage appelé *témoin* . Dans ce livre, nous l'appelons un « script de déverrouillage » pour reconnaître la gamme beaucoup plus large d'exigences de script de verrouillage, car tous les scripts de déverrouillage ne doivent pas contenir des signatures.

Chaque nœud de validation Bitcoin validera les transactions en exécutant les scripts de verrouillage et de déverrouillage ensemble. Chaque entrée contient un script de déverrouillage et fait référence à un UTXO existant précédemment. Le logiciel de validation copiera le script de déverrouillage, récupérera l'UTXO référencé par l'entrée et copiera le script de verrouillage à partir de cet UTXO. Le script de déverrouillage et de verrouillage est ensuite exécuté en séquence. L'entrée est valide si le script de déverrouillage satisfait aux conditions du script de verrouillage (voir Exécution séparée des scripts de déverrouillage et de verrouillage). Toutes les entrées sont validées indépendamment, dans le cadre de la validation globale de la transaction.

Notez que l'UTXO est enregistré en permanence dans la blockchain, et est donc invariable et n'est pas affecté par les tentatives infructueuses de le dépenser par référence dans une nouvelle transaction. Seule une transaction valide qui satisfait correctement les conditions de la sortie a pour résultat que la sortie est considérée comme "dépensée" et supprimée de l'ensemble des sorties de transaction non dépensées (ensemble UTXO).

La combinaison de scriptSig et scriptPubKey pour évaluer un script de transaction est un exemple des scripts de déverrouillage et de verrouillage pour le type le plus courant de transaction bitcoin (un paiement à un hachage de clé publique), montrant le script combiné résultant de la concaténation des scripts de déverrouillage et de verrouillage avant la validation du script.

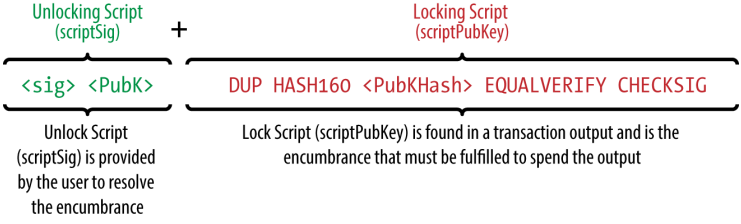


Figure 3. Combinaison de scriptSig et scriptPubKey pour évaluer un script de transaction

La pile d'exécution de script

Le langage de script de Bitcoin est appelé un langage basé sur la pile car il utilise une structure de données appelée *pile* . Une pile est une structure de données très simple qui peut être visualisée comme une pile de cartes. Une pile permet deux opérations : push et pop. Push ajoute un élément au-dessus de la pile. Pop supprime l'élément supérieur de la pile. Les opérations sur une pile ne peuvent agir que sur l'élément le plus haut de la pile. Une structure de données de pile est également appelée file d'attente Last-In-First-Out ou « LIFO ».

Le langage de script exécute le script en traitant chaque élément

de gauche à droite. Les nombres (constantes de données) sont poussés sur la pile. Les opérateurs poussent ou sautent un ou plusieurs paramètres de la pile, agissent sur eux et peuvent pousser un résultat sur la pile. Par exemple, OP_ADD fera apparaître deux éléments de la pile, les ajoutera et poussera la somme résultante sur la pile.

Les opérateurs conditionnels évaluent une condition, produisant un résultat booléen VRAI ou FAUX. Par exemple, OP_EQUAL saute deux éléments de la pile et pousse TRUE (TRUE est représenté par le nombre 1) s'ils sont égaux ou FALSE (représenté par zéro) s'ils ne sont pas égaux. Les scripts de transaction Bitcoin contiennent généralement un opérateur conditionnel, afin qu'ils puissent produire le résultat TRUE qui signifie une transaction valide.

Un script simple

Appliquons maintenant ce que nous avons appris sur les scripts et les piles à quelques exemples simples.

Dans la validation de script de Bitcoin faisant des calculs simples , le script 2 3 OP_ADD 5 OP_EQUAL montre l'opérateur d'addition arithmétique OP_ADD, en ajoutant deux nombres et en mettant le résultat sur la pile, suivi de l'opérateur conditionnel OP_EQUAL, qui vérifie que la somme résultante est égale à 5 Par souci de concision, le préfixe OP_ est omis dans l'exemple pas à pas. Pour plus de détails sur les opérateurs et les fonctions de script disponibles.

Bien que la plupart des scripts de verrouillage se réfèrent à un

hachage de clé publique (essentiellement, une adresse bitcoin), nécessitant ainsi une preuve de propriété pour dépenser les fonds, le script n'a pas besoin d'être aussi complexe. Toute combinaison de scripts de verrouillage et de déverrouillage qui aboutit à une valeur TRUE est valide. L'arithmétique simple que nous avons utilisée comme exemple de langage de script est également un script de verrouillage valide qui peut être utilisé pour verrouiller une sortie de transaction.

Utilisez une partie de l'exemple de script arithmétique comme script de verrouillage :

```
3 OP_ADD 5 OP_EQUAL
```

qui peut être satisfaite par une transaction contenant une entrée avec le script de déverrouillage :

```
2
```

Le logiciel de validation combine les scripts de verrouillage et de déverrouillage et le script résultant est :

```
2 3 OP_ADD 5 OP_EQUAL
```

Comme nous l'avons vu dans l'exemple étape par étape de la validation de script de Bitcoin en faisant des calculs simples , lorsque ce script est exécuté, le résultat est OP_TRUE, ce qui rend la transaction valide. Non seulement il s'agit d'un script

de verrouillage de sortie de transaction valide, mais l'UTXO résultant pourrait être dépensé par toute personne ayant les compétences en arithmétique pour savoir que le nombre 2 satisfait le script.

Conseil

Les transactions sont valides si le premier résultat de la pile est VRAI (noté {0x01}), toute autre valeur différente de zéro, et non OP_0, ou si la pile est vide après l'exécution du script. Les transactions ne sont pas valides si la valeur supérieure de la pile est FALSE (une valeur vide de longueur nulle, notée {}) ou si l'exécution du script est arrêtée explicitement par un opérateur, tel que OP_VERIFY, OP_RETURN ou un terminateur conditionnel tel que OP_ENDIF.

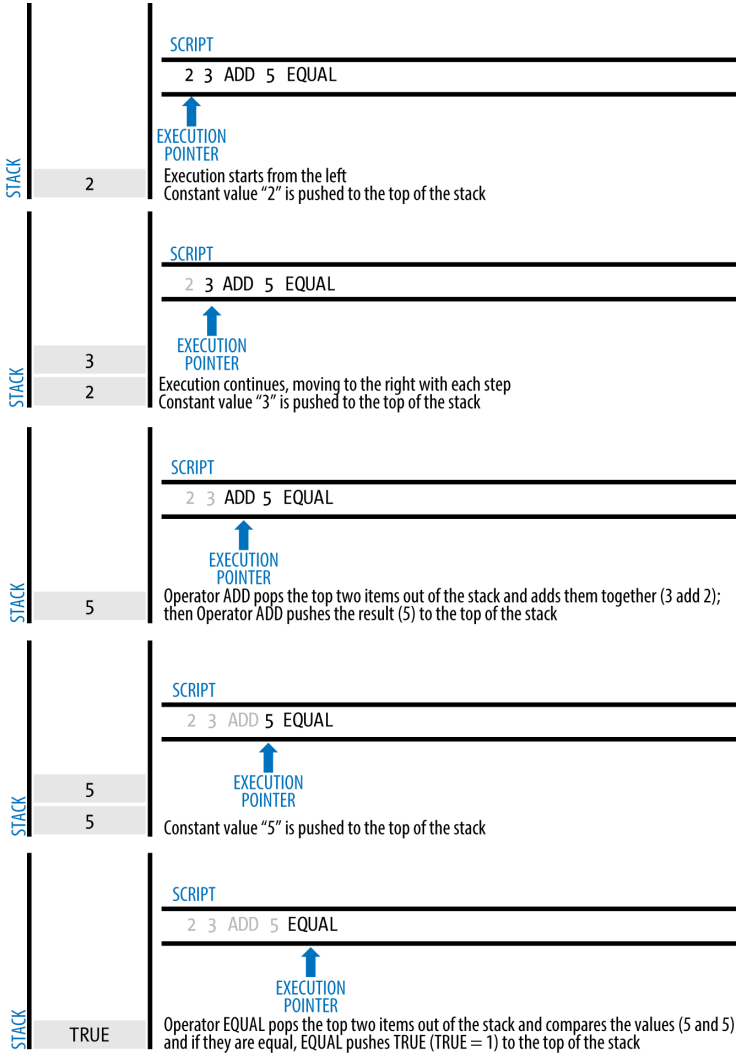


Figure 4. Validation du script de Bitcoin en maths simples

Ce qui suit est un script légèrement plus complexe, qui calcule

2 + 7 - 3 + 1. Notez que lorsque le script contient plusieurs opérateurs dans une ligne, la pile permet aux résultats d'un opérateur d'être agis par l'opérateur suivant :

```
2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL
```

Essayez de valider vous-même le script précédent à l'aide d'un crayon et de papier. Lorsque l'exécution du script se termine, vous devez vous retrouver avec la valeur TRUE sur la pile.

Exécution séparée des scripts de déverrouillage et de verrouillage

Dans le client bitcoin d'origine, les scripts de déverrouillage et de verrouillage étaient concaténés et exécutés en séquence. Pour des raisons de sécurité, cela a été modifié en 2010, en raison d'une vulnérabilité qui permettait à un script de déverrouillage mal formé de pousser des données sur la pile et de corrompre le script de verrouillage. Dans l'implémentation actuelle, les scripts sont exécutés séparément avec la pile transférée entre les deux exécutions, comme décrit ci-après.

Tout d'abord, le script de déverrouillage est exécuté à l'aide du moteur d'exécution de pile. Si le script de déverrouillage est exécuté sans erreur (par exemple, il ne reste plus de pointeurs "pendants"), la pile principale est copiée et le script de verrouillage est exécuté. Si le résultat de l'exécution du script de verrouillage avec les données de pile copiées à partir du script de déverrouillage est "TRUE", le script de déverrouillage a réussi

à résoudre les conditions imposées par le script de verrouillage et, par conséquent, l'entrée est une autorisation valide pour dépenser l'UTXO . Si un résultat autre que "TRUE" reste après l'exécution du script combiné, l'entrée est invalide car elle n'a pas satisfait aux conditions de dépense placées sur l'UTXO.

Payer à la clé publique-Hash (P2PKH)

La grande majorité des transactions traitées sur le réseau bitcoin dépensent des sorties verrouillées avec un script Pay-to-Public-Key-Hash ou "P2PKH". Ces sorties contiennent un script de verrouillage qui verrouille la sortie sur un hachage de clé publique, plus communément appelé adresse bitcoin. Une sortie verrouillée par un script P2PKH peut être déverrouillée (dépensée) en présentant une clé publique et une signature numérique créée par la clé privée correspondante (voir Signatures numériques (ECDSA)).

Par exemple, regardons à nouveau le paiement d'Alice à Bob's Cafe. Alice a effectué un paiement de 0,015 bitcoin à l'adresse bitcoin du café. Cette sortie de transaction aurait un script de verrouillage de la forme :

```
OP_DUP OP_HASH160 <Cafe Public Key Hash>
OP_EQUALVERIFY OP_CHECKSIG
```

Le hachage de la clé publique du café équivaut à l'adresse bitcoin du café, sans le codage Base58Check. La plupart des applications afficheraient le *hachage de la clé publique* en codage

hexadécimal et non le format habituel de l'adresse Bitcoin Base58Check qui commence par un "1".

Le script de verrouillage précédent peut se contenter d'un script de déverrouillage de la forme :

```
<Cafe Signature> <Cafe Public Key>
```

Les deux scripts formeraient ensemble le script de validation combiné suivant :

```
<Cafe Signature> <Cafe Public Key> OP_DUP OP_HASH160
<Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

Une fois exécuté, ce script combiné sera évalué à TRUE si, et seulement si, le script de déverrouillage correspond aux conditions définies par le script de verrouillage. En d'autres termes, le résultat sera VRAI si le script de déverrouillage a une signature valide de la clé privée du café qui correspond au hachage de clé publique défini comme un engagement.

Les figures # P2PubKHash1 et # P2PubKHash2 montrent (en deux parties) une exécution pas à pas du script combiné, ce qui prouvera qu'il s'agit d'une transaction valide.

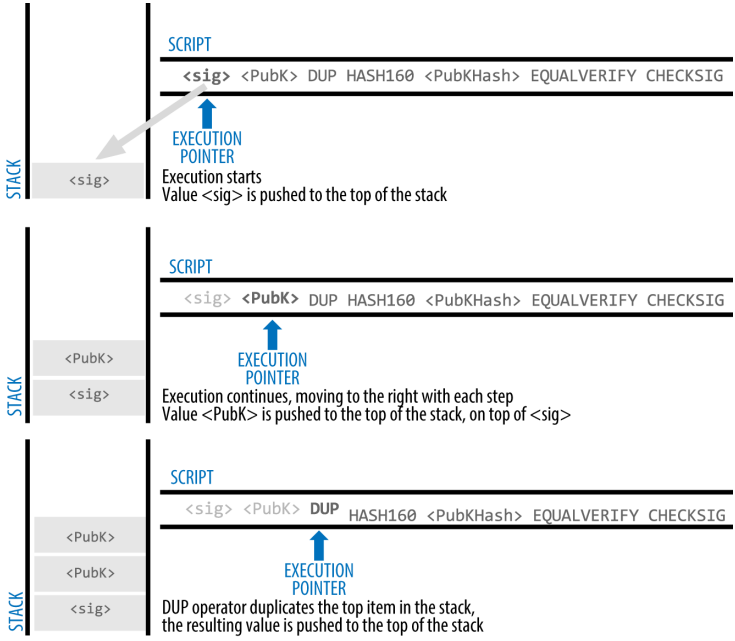


Figure 5. Évaluation d'un script pour une transaction P2PKH (partie 1 sur 2)

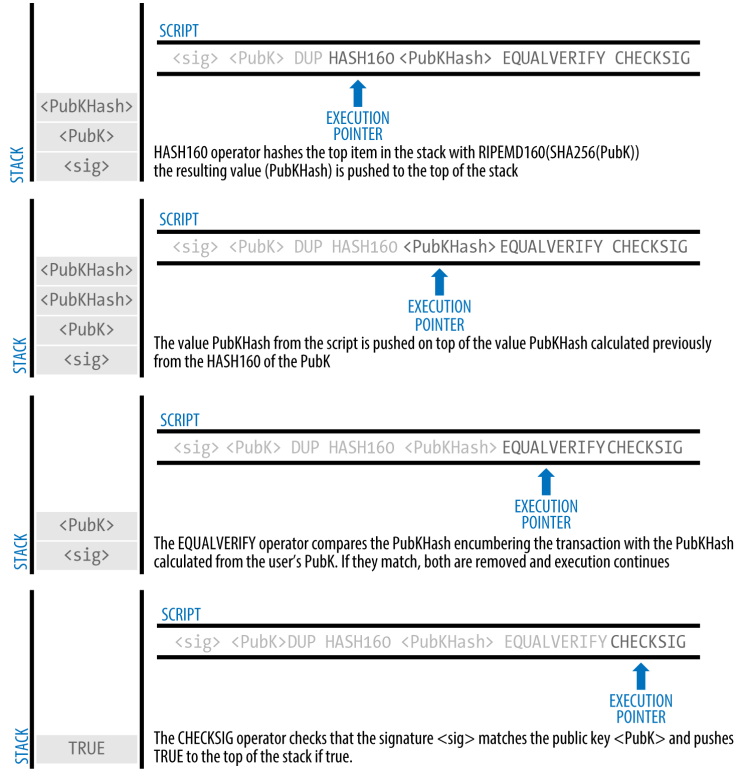


Figure 6. Évaluation d'un script pour une transaction P2PKH (partie 2 sur 2)

Signatures numériques (ECDSA)

Jusqu'à présent, nous n'avons pas approfondi de détails sur les « signatures numériques ». Dans cette section, nous examinons le fonctionnement des signatures numériques et comment elles

peuvent présenter une preuve de propriété d'une clé privée sans révéler cette clé privée.

L'algorithme de signature numérique utilisé dans Bitcoin est l' *algorithme de signature numérique à courbe elliptique* , ou *ECDSA* . ECDSA est l'algorithme utilisé pour les signatures numériques basées sur des paires de clés privées / publiques à courbe elliptique , comme décrit dans [la courbe elliptique du chapitre 4] . ECDSA est utilisé par les fonctions de script OP_CHECKSIG, OP_CHECKSIGVERIFY, OP_CHECKMULTISIG et OP_CHECKMULTISIGVERIFY. Chaque fois que vous les voyez dans un script de verrouillage, le script de déverrouillage doit contenir une signature ECDSA.

Une signature numérique a trois objectifs en bitcoin. Premièrement, la signature prouve que le propriétaire de la clé privée, qui est implicitement le propriétaire des fonds, a *autorisé* la dépense de ces fonds. Deuxièmement, la preuve de l'autorisation est *indéniable* (non répudiation). Troisièmement, la signature prouve que la transaction (ou des parties spécifiques de la transaction) n'a pas et *ne peut pas être modifiée* par quiconque après sa signature.

Notez que chaque entrée de transaction est signée indépendamment. Ceci est essentiel, car ni les signatures ni les entrées ne doivent appartenir ou être appliquées par les mêmes « propriétaires ». En fait, un schéma de transaction spécifique appelé "CoinJoin" utilise ce fait pour créer des transactions multi-parties pour la confidentialité.

Note

Chaque entrée de transaction et toute signature qu'elle peut contenir est totalement *indépendante de toute autre entrée ou signature. Plusieurs parties peuvent collaborer pour construire des transactions et signer une seule entrée chacune.*

Définition de Wikipedia d'une « signature numérique »

Une signature numérique est un schéma mathématique permettant de démontrer l'authenticité d'un message ou de documents numériques. Une signature numérique valide donne au destinataire des raisons de croire que le message a été créé par un expéditeur connu (authentification), que l'expéditeur ne peut pas nier avoir envoyé le message (non-répudiation) et que le message n'a pas été altéré en transit (intégrité).

Source : https ://en.wikipedia.org/wiki/Digital_signature

Comment fonctionnent les signatures numériques

Une signature numérique est un *schéma mathématique* qui se compose de deux parties. La première partie est un algorithme de création d'une signature, à l'aide d'une clé privée (la clé de signature), à partir d'un message (la transaction). La deuxième partie est un algorithme qui permet à quiconque de vérifier la signature, étant donné également le message et une clé publique.

Créer une signature numérique

Dans l'implémentation de l'algorithme ECDSA par bitcoin,

le « message » signé est la transaction, ou plus précisément un hachage d'un sous-ensemble spécifique des données de la transaction (voir Types de hachage de signature (SIGHASH)). La clé de signature est la clé privée de l'utilisateur. Le résultat est la signature :

\(\(Sig = F_{sig}(F_{hash}(m), dA)\)\)
où :

- *dA* est la clé privée de signature
- *m* est la transaction (ou une partie de celle-ci)
- F_{hash} est la fonction de hachage
- F_{sig} est l'algorithme de signature
- *Sig* est la signature résultante

Plus de détails sur les mathématiques de l'ECDSA peuvent être trouvés dans ECDSA Math plus bas.

La fonction F_{sig} produit une signature Sig composée de deux valeurs, communément appelées R et S :

```
Sig = (R, S)
```

Maintenant que les deux valeurs R et S ont été calculées, elles sont sérialisées dans un flux d'octets en utilisant un schéma de codage standard international appelé les *règles de codage distinctes* , ou *DER* .

Sérialisation des signatures (DER)

Regardons à nouveau la transaction qu'Alice a créée. Dans l'entrée de transaction, il y a un script de déverrouillage qui contient la signature encodée DER suivante du portefeuille d'Alice :

```
3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c7
```

Cette signature est un flux d'octets sérialisé des valeurs R et S produites par le portefeuille d'Alice pour prouver qu'elle possède la clé privée autorisée à dépenser cette sortie. Le format de sérialisation se compose de neuf éléments comme suit :

- 0x30 - indiquant le début d'une séquence DER
- 0x45 - la longueur de la séquence (69 octets)
- 0x02 : une valeur entière suit
- 0x21 - la longueur de l'entier (33 octets)
- R-00884d142d86652a3f47ba4746ec719bbfbd040a570b1d eccbb6498c75c4ae24cb
- 0x02 - un autre entier suit
- 0x20 - la longueur de l'entier (32 octets)
- S — 4b9f039ff08df09cbe9f6addac960298cad530a863ea8f 53982c09db8f6e3813
- Un suffixe (0x01) indiquant le type de hachage utilisé (SIGHASH_ALL)

Voyez si vous pouvez décoder la signature sérialisée (encodée en DER) d'Alice en utilisant cette liste. Les nombres importants sont R et S ; le reste des données fait partie du schéma de codage

DER.

Vérification de la signature

Pour vérifier la signature, il faut avoir la signature (R et S), la transaction sérialisée et la clé publique (qui correspond à la clé privée utilisée pour créer la signature). Essentiellement, la vérification d'une signature signifie « Seul le propriétaire de la clé privée qui a généré cette clé publique aurait pu produire cette signature sur cette transaction ».

L'algorithme de vérification de signature prend le message (un hachage de la transaction ou des parties de celui-ci), la clé publique du signataire et la signature (valeurs R et S), et retourne TRUE si la signature est valide pour ce message et la clé publique.

Types de hachage de signature (SIGHASH)

Les signatures numériques sont appliquées aux messages qui, dans le cas du bitcoin, sont les transactions elles-mêmes. La signature implique un *engagement* du signataire sur des données de transaction spécifiques. Dans sa forme la plus simple, la signature s'applique à l'ensemble de la transaction, validant ainsi toutes les entrées, sorties et autres champs de transaction. Cependant, une signature ne peut s'engager que sur un sous-ensemble des données d'une transaction, ce qui est utile pour un certain nombre de scénarios comme nous le verrons dans cette section.

Les signatures Bitcoin permettent d'indiquer quelle partie des

données d'une transaction est incluse dans le hachage signé par la clé privée à l'aide d'un indicateur SIGHASH. L'indicateur SIGHASH est un octet unique qui est ajouté à la signature. Chaque signature a un drapeau SIGHASH et le drapeau peut être différent d'une entrée à l'autre. Une transaction avec trois entrées signées peut avoir trois signatures avec différents indicateurs SIGHASH, chaque signature signant (validant) différentes parties de la transaction.

N'oubliez pas que chaque entrée peut contenir une signature dans son script de déverrouillage. En conséquence, une transaction qui contient plusieurs entrées peut avoir des signatures avec différents indicateurs SIGHASH qui valident différentes parties de la transaction dans chacune des entrées. Notez également que les transactions Bitcoin peuvent contenir des entrées de différents « propriétaires », qui peuvent signer une seule entrée dans une transaction partiellement construite (et invalide), collaborant avec d'autres pour rassembler toutes les signatures nécessaires pour effectuer une transaction valide. De nombreux types d'indicateurs SIGHASH n'ont de sens que si vous pensez à plusieurs participants collaborant en dehors du réseau Bitcoin et mettant à jour une transaction partiellement signée.

Il existe trois indicateurs SIGHASH : ALL, NONE et SINGLE, comme indiqué dans les types SIGHASH et leur signification .

SIGHASH flag	Value	Description
ALL	0x01	Signature applies to all inputs and outputs
NONE	0x02	Signature applies to all inputs, none of the outputs
SINGLE	0x03	Signature applies to all inputs but only the one output with the same index number as the signed input

Tableau 4. Types de SIGHASH et leurs significations

De plus, il existe un indicateur de modificateur SIGHASH_ANYONECANPAY, qui peut être combiné avec chacun des indicateurs précédents. Lorsque ANYONECANPAY est défini, une seule entrée est signée, laissant le reste (et leurs numéros de séquence) ouverts pour modification. ANYONECANPAY a la valeur 0x80 et est appliqué par OU au niveau du bit, ce qui donne les indicateurs combinés comme indiqué dans les types SIGHASH avec des modificateurs et leurs significations .

SIGHASH flag	Value	Description
ALL|ANYONECANPAY	0x81	Signature applies to one input and all outputs
NONE|ANYONECANPAY	0x82	Signature applies to one input, none of the outputs
SINGLE|ANYONECANPAY	0x83	Signature applies to one input and the output with the same index number

Tableau 5. Types de SIGHASH avec modificateurs et leurs significations

La signature s'applique à une entrée et la sortie avec le même numéro d'index

Ces combinaisons d'indicateurs sont résumées dans Résumé des différentes combinaisons de sighash

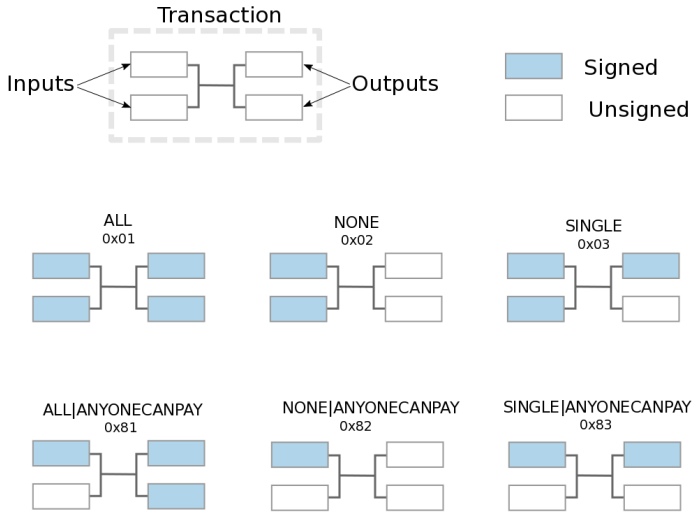


Figure 7. Résumé des différentes combinaisons de sighash

La façon dont les indicateurs SIGHASH sont appliqués lors de la signature et de la vérification est qu'une copie de la transaction est effectuée et que certains champs à l'intérieur sont tronqués (mis à zéro et vidés). La transaction résultante est sérialisée. L'indicateur SIGHASH est ajouté à la fin de la transaction sérialisée et le résultat est haché. Le hachage lui-même est le « message » qui est signé. En fonction de l'indicateur SIGHASH

utilisé, différentes parties de la transaction sont tronquées. Le hachage résultant dépend de différents sous-ensembles de données dans la transaction. En incluant le SIGHASH comme dernière étape avant le hachage, la signature valide également le type SIGHASH, donc il ne peut pas être modifié (par exemple, par un mineur).

Note

Tous les types SIGHASH signent le champ transaction nLocktime. De plus, le type SIGHASH lui-même est ajouté à la transaction avant sa signature, de sorte qu'il ne peut pas être modifié une fois signé.

Dans l'exemple de la transaction d'Alice (voir la liste dans Sérialisation des signatures (DER)), nous avons vu que la dernière partie de la signature encodée en DER était 01, qui est le drapeau SIGHASH_ALL . Cela verrouille les données de transaction, de sorte que la signature d'Alice valide l'état de toutes les entrées et sorties. Il s'agit du formulaire de signature le plus courant.

Examinons certains des autres types de SIGHASH et comment ils peuvent être utilisés dans la pratique :

ALL|ANYONECANPAY

Cette construction peut être utilisée pour effectuer une transaction de type "crowdfunding". Quelqu'un qui tente de lever des fonds peut construire une transaction avec un seul résultat. Le résultat unique verse le montant "objectif" à la collecte de fonds. Une telle transaction n'est évidemment pas valide, car il

n'a pas d'entrées. Cependant, d'autres peuvent maintenant le modifier en ajoutant leur propre entrée, en tant que don. Ils signent leur propre entrée avec ALL | ANYONECANPAY. À moins que suffisamment d'entrées ne soient rassemblées pour atteindre la valeur de la sortie, la transaction n'est pas valide. Chaque don est une « promesse de don » qui ne peut pas être collectée par la collecte de fonds tant que le montant total de l'objectif n'est pas atteint.

NONE

Cette construction peut être utilisée pour créer un « chèque au porteur » ou un « chèque en blanc » d'un montant spécifique. Il valide l'entrée, mais permet de modifier le script de verrouillage de sortie. Tout le monde peut écrire sa propre adresse bitcoin dans le script de verrouillage de sortie et valider la transaction. Cependant, la valeur de sortie elle-même est verrouillée par la signature.

NONE | ANYONECANPAY

Cette construction peut être utilisée pour construire un "dépoussiéreur". Les utilisateurs qui ont de minuscules UTXO dans leur portefeuille ne peuvent pas les dépenser car le coût des frais dépasse la valeur de la poussière. Avec ce type de signature, la poussière UTXO peut être donnée à quiconque pour l'agréger et la dépenser quand bon lui semble.

Il y a quelques propositions pour modifier ou étendre le système SIGHASH. Une de ces propositions est *Bitmask Sighash Modes* par Glenn Willen de Blockstream, dans le cadre du projet Elements. Cela vise à créer un remplacement flexible pour les types SIGHASH qui permet « des masques de bits

arbitraires, réinscriptibles par les mineurs » d'entrées et de sorties »qui peuvent exprimer« des schémas de pré-engagement contractuels plus complexes, tels que des offres signées avec changement dans un échange d'actifs distribué ».

Note

Vous ne verrez pas les indicateurs SIGHASH présentés en option dans l'application de portefeuille d'un utilisateur. À quelques exceptions près, les portefeuilles construisent des scripts P2PKH et signent avec des indicateurs SIGHASH_ALL. Pour utiliser un indicateur SIGHASH différent, vous devez écrire un logiciel pour construire et signer des transactions. Plus important encore, les indicateurs SIGHASH peuvent être utilisés par des applications Bitcoin à usage spécial qui permettent de nouvelles utilisations.

Mathématiques ECDSA

Comme mentionné précédemment, les signatures sont créées par une fonction mathématique *F sig* qui produit une signature constituée de deux valeurs *R* et *S* . Dans cette section, nous examinons plus en détail la fonction F_{sig} .

L'algorithme de signature génère d'abord une paire de clés publiques privées *éphémères* (temporaires). Cette paire de clés temporaire est utilisée dans le calcul des valeurs *R* et *S* , après une transformation impliquant la clé privée de signature et le hachage de la transaction.

La paire de clés temporaire est basée sur un nombre aléatoire k , qui est utilisé comme clé privée temporaire. À partir de k , nous générons la clé publique temporaire correspondante P (calculée comme $P = k*G$, de la même manière que les clés publiques bitcoin sont dérivées ;). Le R, valeur de la signature numérique est alors la coordonnée x de la clé publique éphémère P .

À partir de là, l'algorithme calcule la valeur S de la signature, telle que :

$S = k^{-1} (Hash(m) + dA * R) \ mod \ n$

où :

- k est la clé privée éphémère
- R est la coordonnée x de la clé publique éphémère
- dA est la clé privée de signature
- m est les données de transaction
- n est le premier ordre de la courbe elliptique

La vérification est l'inverse de la fonction de génération de signature, en utilisant les valeurs R , S et la clé publique pour calculer une valeur P , qui est un point sur la courbe elliptique (la clé publique éphémère utilisée dans la création de signature) :

$P = S^{-1} * Hash(m) * G + S^{-1} * R * Qa$

où :

- R et S sont les valeurs de signature
- Qa est la clé publique d'Alice
- m est les données de transaction qui ont été signées
- G est le point du générateur de courbe elliptique

Si la coordonnée x du point calculé *P* est égale à *R* , alors le vérificateur peut conclure que la signature est valide.

Notez que lors de la vérification de la signature, la clé privée n'est ni connue ni révélée.

Conseil

L'ECDSA est nécessairement un élément mathématique assez compliqué ; une explication complète dépasse le cadre de ce livre. Un certain nombre d'excellents guides en ligne vous guident pas à pas : recherchez "ECDSA expliqué" ou essayez celui-ci : http ://bit.ly/2r0HhGB .

L'importance de l'aléatoire dans les signatures

Comme nous l'avons vu dans ECDSA Math , l'algorithme de génération de signature utilise une clé aléatoire *k* , comme base d'une paire de clés privée / publique éphémère. La valeur de *k* n'est pas importante, *tant qu'elle est aléatoire* . Si la même valeur *k* est utilisée pour produire deux signatures sur différents messages (transactions), alors la *clé privée de* signature peut être calculée par n'importe qui. La réutilisation de la même valeur pour *k* dans un algorithme de signature conduit à une exposition de la clé privée !

Avertissement

Si la même valeur k *est utilisée dans l'algorithme de signature sur deux transactions différentes, la clé privée peut être calculée et exposée au monde !*

Ce n'est pas seulement une possibilité théorique. Nous avons vu ce problème conduire à l'exposition des clés privées dans quelques implémentations différentes d'algorithmes de signature de transaction dans Bitcoin. Des gens se sont fait voler des fonds en raison de la réutilisation par inadvertance d'une valeur k . La raison la plus courante de réutilisation d'une valeur k est un générateur de nombres aléatoires mal initialisé.

Pour éviter cette vulnérabilité, la meilleure pratique de l'industrie consiste à ne pas générer k avec un générateur de nombres aléatoires avec entropie, mais plutôt à utiliser un processus déterministe-aléatoire amorcé avec les données de transaction elles-mêmes. Cela garantit que chaque transaction produit un k différent . L'algorithme standard de l'industrie pour l'initialisation déterministe de k est défini dans la RFC 6979 , publiée par l'Internet Engineering Task Force.

Si vous implémentez un algorithme pour signer des transactions en bitcoin, vous *devez* utiliser la RFC 6979 ou un algorithme aléatoire déterministe similaire pour vous assurer de générer un k différent pour chaque transaction.

Adresses, soldes et autres abstractions Bitcoin

Nous avons commencé ce chapitre en découvrant que les transactions sont très différentes « dans les coulisses » de la façon dont elles sont présentées dans les portefeuilles, les explorateurs de chaînes de blocs et d'autres applications destinées aux utilisateurs. De nombreux concepts simplistes et familiers des chapitres précédents, tels que les adresses et les soldes Bitcoin,

semblent être absents de la structure de transaction. Nous avons vu que les transactions ne contiennent pas d'adresses bitcoin, en soi, mais fonctionnent à travers des scripts qui verrouillent et déverrouillent des valeurs discrètes de bitcoin. Les soldes ne sont présents nulle part dans ce système et pourtant chaque application de portefeuille affiche bien en évidence le solde du portefeuille de l'utilisateur.

Maintenant que nous avons exploré ce qui est réellement inclus dans une transaction bitcoin, nous pouvons examiner comment les abstractions de niveau supérieur sont dérivées des composants apparemment primitifs de la transaction.

Regardons à nouveau comment la transaction d'Alice a été présentée sur un explorateur de blocs populaire (la transaction d'Alice à Bob's Cafe).

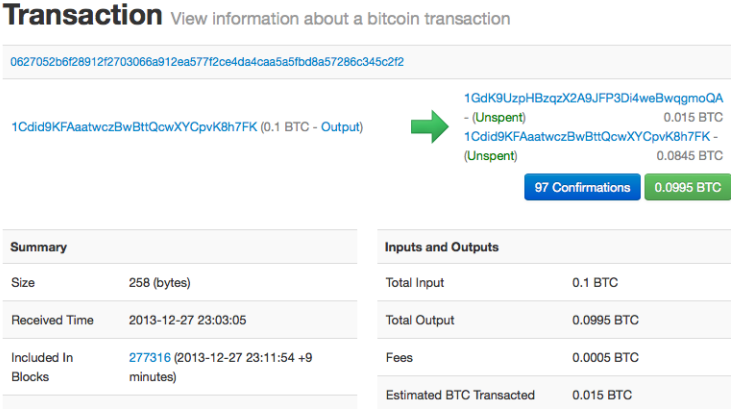


Figure 8. Transaction d'Alice à Bob's Cafe

Sur le côté gauche de la transaction, l'explorateur de chaînes de blocs montre l'adresse bitcoin d'Alice comme « expéditeur ». En fait, cette information ne se trouve pas dans la transaction elle-même. Lorsque l'explorateur de blockchain fait référence à la transaction, il fait également référence à la transaction précédente associée à l'entrée et extrait la première sortie de cette transaction plus ancienne. Dans cette sortie se trouve un script de verrouillage qui verrouille l'UTXO sur le hachage de clé publique d'Alice (un script P2PKH). L'explorateur de blockchain a extrait le hachage de la clé publique et l'a codé à l'aide du codage Base58Check pour produire et afficher l'adresse bitcoin qui représente cette clé publique.

De même, sur le côté droit, l'explorateur de blockchain montre les deux sorties ; le premier à l'adresse bitcoin de Bob et le second à l'adresse bitcoin d'Alice (en tant que changement). Une fois de plus, pour créer ces adresses bitcoin, l'explorateur de blockchain a extrait le script de verrouillage de chaque sortie, l'a reconnu comme un script P2PKH et extrait le hachage de clé publique de l'intérieur. Enfin, l'explorateur de blockchain a réencodé chaque hachage de clé publique avec Base58Check pour produire et afficher les adresses bitcoin.

Si vous deviez cliquer sur l'adresse bitcoin de Bob, l'explorateur de blockchain vous montrerait la vue dans Le solde de l'adresse bitcoin de Bob .

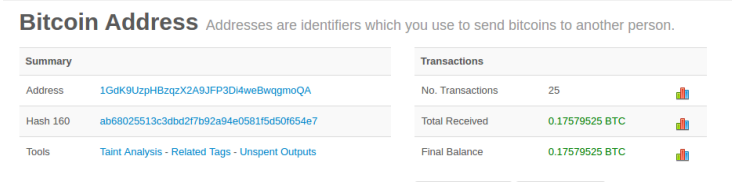


Figure 9. Le solde de l'adresse bitcoin de Bob

L'explorateur de blockchain affiche le solde de l'adresse bitcoin de Bob. Mais nulle part dans le système Bitcoin il n'y a de concept d '« équilibre ». Au contraire, les valeurs affichées ici sont construites par l'explorateur de blockchain comme suit.

Pour construire le montant "Total Reçu", l'explorateur de blockchain décode d'abord le codage Base58Check de l'adresse bitcoin pour récupérer le hachage 160 bits de la clé publique de Bob qui est codé dans l'adresse. Ensuite, l'explorateur de chaînes de blocs recherchera dans la base de données des transactions, à la recherche de sorties avec des scripts de verrouillage P2PKH contenant le hachage de la clé publique de Bob. En additionnant la valeur de toutes les sorties, l'explorateur de blockchain peut produire la valeur totale reçue.

Construire le solde actuel (affiché comme « solde final ») nécessite un peu plus de travail. L'explorateur de blockchain conserve une base de données séparée des sorties qui ne sont actuellement pas dépensées, l'ensemble UTXO. Pour maintenir cette base de données, l'explorateur de chaînes de blocs doit surveiller le réseau bitcoin, ajouter l'UTXO nouvellement créé et supprimer l'UTXO dépensé, en temps réel, tels qu'ils apparaissent dans les transactions non confirmées.

Il s'agit d'un processus compliqué qui dépend du suivi des transactions à mesure qu'elles se propagent, ainsi que du maintien d'un consensus avec le réseau Bitcoin pour garantir que la chaîne correcte est suivie. Parfois, l'explorateur de blockchain se désynchronise et sa perspective de l'ensemble UTXO est incomplète ou incorrecte.

À partir de l'ensemble UTXO, l'explorateur de chaînes de blocs résume la valeur de toutes les sorties non dépensées faisant référence au hachage de la clé publique de Bob et produit le numéro de « solde final » affiché à l'utilisateur.

Afin de produire cette image unique, avec ces deux « soldes », l'explorateur de la blockchain doit indexer et rechercher parmi des dizaines, des centaines, voire des centaines de milliers de transactions.

En résumé, les informations présentées aux utilisateurs via des applications de portefeuille, des explorateurs de chaînes de blocs et d'autres interfaces utilisateur Bitcoin sont souvent composées d'abstractions de plus haut niveau qui sont dérivées en recherchant de nombreuses transactions différentes, en inspectant leur contenu et en manipulant les données qu'elles contiennent. En présentant cette vue simpliste des transactions Bitcoin qui ressemblent à des chèques bancaires d'un expéditeur à un destinataire, ces applications doivent résumer de nombreux détails sous-jacents. Ils se concentrent principalement sur les types courants de transactions : P2PKH avec des signatures SIGHASH_ALL sur chaque entrée. Ainsi, alors que les applications Bitcoin peuvent présenter plus de 80% de toutes les transactions de manière facile à lire, elles sont parfois

déconcertées par des transactions qui s'écartent de la norme. Les transactions qui contiennent des scripts de verrouillage plus complexes, ou différents indicateurs SIGHASH, ou de nombreuses entrées et sorties, démontrent la simplicité et la faiblesse de ces abstractions.

Chaque jour, des centaines de transactions ne contenant pas de sorties P2PKH sont confirmées sur la blockchain. Les explorateurs de la blockchain les présentent souvent avec des messages d'avertissement rouges indiquant qu'ils ne peuvent pas décoder une adresse.

Comme nous le verrons dans le prochain chapitre, ce ne sont pas forcément des transactions étranges. Ce sont des transactions qui contiennent des scripts de verrouillage plus complexes que le P2PKH commun. Nous apprendrons ensuite à décoder et à comprendre des scripts plus complexes et les applications qu'ils prennent en charge.

8

Transactions et scripts avancés

Introduction

Dans le chapitre précédent, nous avons présenté les éléments de base des transactions Bitcoin et examiné le type de script de transaction le plus courant, le script P2PKH. Dans ce chapitre, nous examinerons des scripts plus avancés et comment nous pouvons l'utiliser pour créer des transactions avec des conditions complexes.

Tout d'abord, nous examinerons les scripts *multisignatures* . Ensuite, nous examinerons le deuxième script de transaction le plus courant, *Pay-to-Script-Hash* , qui ouvre tout un monde de scripts complexes. Ensuite, nous examinerons de nouveaux opérateurs de script qui ajoutent une dimension temporelle au bitcoin, via des *timelocks* . Enfin, nous examinerons *Segregated Witness* , une modification architecturale de la structure des transactions.

Multisignature

Les scripts multisignatures définissent une condition dans laquelle N clés publiques sont enregistrées dans le script et au moins M de celles-ci doivent fournir des signatures pour débloquer les fonds. Ceci est également connu sous le nom de schéma M-of-N, où N est le nombre total de clés et M est le seuil de signatures requis pour la validation. Par exemple, une multisignature 2 sur 3 est celle où trois clés publiques sont répertoriées comme signataires potentiels et au moins deux de celles-ci doivent être utilisées pour créer des signatures pour une transaction valide afin de dépenser les fonds.

À l'heure actuelle, *les* scripts multisignatures *standard* sont limités à au plus 3 clés publiques répertoriées, ce qui signifie que vous pouvez faire n'importe quoi, d'une multisignature 1 sur 1 à 3 sur 3 ou toute combinaison dans cette plage. La limitation à 3 clés répertoriées pourrait être levée au moment de la publication de ce livre, vérifiez donc la fonction IsStandard ( ) pour voir ce qui est actuellement accepté par le réseau. Notez que la limite de 3 clés s'applique uniquement aux scripts multisignatures standard (également appelés "nus"), et non aux scripts multisignatures enveloppés dans un script Pay-to-Script-Hash (P2SH). Les scripts multisignatures P2SH sont limités à 15 clés, permettant jusqu'à 15 sur 15 multisignatures. Cette limitation est également imposée par la fonction IsStandard ( ). Nous en apprendrons davantage sur le P2SH dans Pay-to-Script-Hash (P2SH) .

La forme générale d'un script de verrouillage définissant une

condition de multi-signature M-of-N est :

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N
CHECKMULTISIG
```

où N est le nombre total de clés publiques répertoriées et M est le seuil de signatures requises pour dépenser la sortie.

Un script de verrouillage définissant une condition de signature multiple 2 sur 3 ressemble à ceci :

```
2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

Le script de verrouillage précédent peut être satisfait d'un script de déverrouillage contenant n'importe quelle combinaison de deux signatures des clés privées correspondant aux trois clés publiques répertoriées :

```
<Signature B> <Signature C>
```

Les deux scripts formeraient ensemble le script de validation combiné :

```
<Signature B> <Signature C> 2 <Public Key A> <Public
Key B> <Public Key C> 3 CHECKMULTISIG
```

Une fois exécuté, ce script combiné sera évalué à TRUE si, et seulement si, le script de déverrouillage correspond aux conditions définies par le script de verrouillage. Dans ce cas, la condition est de savoir si le script de déverrouillage a une signature valide des deux clés privées qui correspondent à deux des trois clés publiques définies comme un encombrement.

Un bug dans l'exécution de CHECKMULTISIG

Il y a un bogue dans l'exécution de CHECKMULTISIG qui nécessite une légère solution de contournement. Lorsque CHECKMULTISIG s'exécute, il doit consommer M + N + 2 éléments sur la pile en tant que paramètres. Cependant, en raison du bogue, CHECKMULTISIG affichera une valeur supplémentaire ou une valeur de plus que prévu.

Examinons cela plus en détail à l'aide de l'exemple de validation précédent :

```
<Signature B> <Signature C> 2 <Public Key A> <Public
Key B> <Public Key C> 3 CHECKMULTISIG
```

Tout d'abord, CHECKMULTISIG fait apparaître l'élément supérieur, qui est N (dans cet exemple "3"). Ensuite, il apparaît N

éléments, qui sont les clés publiques qui peuvent signer. Dans cet exemple, les clés publiques A, B et C. Ensuite, il affiche un élément, qui est M, le quorum (combien de signatures sont nécessaires). Ici M = 2. À ce stade, CHECKMULTISIG devrait afficher les M éléments finaux, qui sont les signatures, et voir s'ils sont valides. Cependant, malheureusement, un bogue dans l'implémentation fait apparaître CHECKMULTISIG un élément de plus (M + 1 au total) qu'il ne le devrait. L'élément supplémentaire n'est pas pris en compte lors de la vérification des signatures , il n'a donc aucun effet direct sur CHECKMULTISIG lui-même. Cependant, une valeur supplémentaire doit être présente car si elle n'est pas présente, lorsque CHECKMULTISIG tente de faire apparaître une pile vide, cela provoquera une erreur de pile et un échec de script (marquant la transaction comme invalide). Étant donné que l'élément supplémentaire n'est pas pris en compte, il peut s'agir de n'importe quoi, mais généralement 0 est utilisé.

Parce que ce bogue est devenu une partie des règles de consensus, il doit maintenant être répliqué pour toujours. Par conséquent, la validation correcte du script ressemblerait à ceci :

```
0 <Signature B> <Signature C> 2 <Public Key A>
<Public Key B> <Public Key C> 3 CHECKMULTISIG
```

Ainsi, le script de déverrouillage réellement utilisé dans multisig n'est pas :

```
<Signature B> <Signature C>
```

mais à la place c'est :

```
0 <Signature B> <Signature C>
```

À partir de maintenant, si vous voyez un script de déverrouillage multisig, vous devriez vous attendre à voir un 0 supplémentaire au début, dont le seul but est de contourner un bogue qui est devenu accidentellement une règle de consensus.

Pay-to-Script-Hash (P2SH)

Pay-to-Script-Hash (P2SH) a été introduit en 2012 en tant que nouveau type de transaction puissant qui simplifie considérablement l'utilisation de scripts de transaction complexes. Pour expliquer la nécessité du P2SH, prenons un exemple pratique.

Dans le chapitre 2, nous avons présenté Mohammed, un importateur d'électronique basé à Dubaï. La société de Mohammed utilise largement la fonction multisignature de Bitcoin pour ses comptes d'entreprise. Les scripts multisignatures sont l'une des utilisations les plus courantes des capacités de script avancées de Bitcoin et sont une fonctionnalité très puissante. La société de Mohammed utilise un script multisignature pour tous les paiements des clients, connu en termes comptables sous le nom de « comptes clients » ou AR. Avec le système multisignature,

tous les paiements effectués par les clients sont verrouillés de telle manière qu'ils nécessitent au moins deux signatures pour être libérées, de la part de Mohammed et de l'un de ses partenaires ou de son avocat qui dispose d'une clé de sauvegarde. Un système multisignature comme celui-ci offre des contrôles de gouvernance d'entreprise et protège contre le vol, le détournement de fonds ou la perte.

Le script résultant est assez long et ressemble à ceci :

```
2 <Mohammed's Public Key> <Partner1 Public Key>
<Partner2 Public Key> <Partner3 Public Key> <Attorney
Public Key> 5 CHECKMULTISIG
```

Bien que les scripts multisignatures soient une fonctionnalité puissante, ils sont difficiles à utiliser. Compte tenu du script précédent, Mohammed devrait communiquer ce script à chaque client avant le paiement. Chaque client devrait utiliser un logiciel de portefeuille Bitcoin spécial avec la possibilité de créer des scripts de transaction personnalisés, et chaque client devrait comprendre comment créer une transaction à l'aide de scripts personnalisés. De plus, la transaction résultante serait environ cinq fois plus importante qu'une simple transaction de paiement, car ce script contient des clés publiques très longues. Le fardeau de cette transaction extra-large serait supporté par le client sous forme de frais. Enfin, un grand script de transaction comme celui-ci serait transporté dans l'ensemble UTXO dans la RAM de chaque nœud complet, jusqu'à ce qu'il soit dépensé. Tous ces problèmes rendent l'utilisation de scripts

de verrouillage complexes difficile dans la pratique.

P2SH a été développé pour résoudre ces difficultés pratiques et pour rendre l'utilisation de scripts complexes aussi simple qu'un paiement à une adresse bitcoin. Avec les paiements P2SH, le script de verrouillage complexe est remplacé par son empreinte numérique, un hachage cryptographique. Lorsqu'une transaction tentant de dépenser l'UTXO est présentée ultérieurement, elle doit contenir le script qui correspond au hachage, en plus du script de déverrouillage. En termes simples, P2SH signifie « payer à un script correspondant à ce hachage, un script qui sera présenté plus tard lorsque cette sortie sera dépensée ».

Dans les transactions P2SH, le script de verrouillage qui est remplacé par un hachage est appelé *script de rachat* car il est présenté au système au moment de la rédemption plutôt que comme un script de verrouillage. Le script complexe sans P2SH montre le script sans P2SH et le script complexe car P2SH montre le même script encodé avec P2SH.

Locking Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Unlocking Script	0 Sig1 Sig2

Tableau 1. Script complexe sans P2SH

Redeem Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Locking Script	HASH160 <20-byte hash of redeem script> EQUAL
Unlocking Script	0 Sig1 Sig2 <redeem script>

Tableau 2. Script complexe comme P2SH

Comme vous pouvez le voir dans les tableaux, avec P2SH, le script complexe qui détaille les conditions de dépense de la sortie (script de rachat) n'est pas présenté dans le script de verrouillage. Au lieu de cela, seul un hachage de celui-ci se trouve dans le script de verrouillage et le script de rachat lui-même est présenté plus tard, dans le cadre du script de déverrouillage lorsque la sortie est épuisée. Cela déplace le fardeau des frais et de la complexité de l'expéditeur (qui crée la transaction) au destinataire (qui déverrouille et passe la transaction).

Examinons la société de Mohammed, le script multisignature complexe et les scripts P2SH qui en résultent.

Tout d'abord, le script multisignature que la société de Mohammed utilise pour tous les paiements entrants des clients :

```
2 <Mohammed's Public Key> <Partner1 Public Key>
<Partner2 Public Key> <Partner3 Public Key> <Attorney
Public Key> 5 CHECKMULTISIG
```

Si les espaces réservés sont remplacés par des clés publiques réelles (affichées ici sous forme de nombres de 520 bits commençant par 04), vous pouvez voir que ce script devient très long :

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01
5 CHECKMULTISIG
```

Ce script entier peut à la place être représenté par un hachage cryptographique de 20 octets, en appliquant d'abord l'algorithme de hachage SHA256, puis en appliquant l'algorithme RIPEMD160 sur le résultat.

Nous utilisons libbitcoin-explorer (bx) sur la ligne de commande pour produire le hachage du script, comme suit :

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA0
\
[04A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2
\
[047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D573781
\
[0421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A
\
[043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A
\
```

```
5 CHECKMULTISIG \
| bx script-encode | bx sha256 | bx ripemd160
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

La série de commandes ci-dessus encode d'abord le script de rachat multisig de Mohammed en tant que script bitcoin codé hexadécimal sérialisé. La commande bx suivante calcule le hachage SHA256 de cela. La commande bx suivante effectue à nouveau un hachage avec RIPEMD160, produisant le hachage de script final :

Le hachage de 20 octets du script de rachat de Mohammed est :

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

Une transaction P2SH verrouille la sortie sur ce hachage au lieu du script de rachat plus long, en utilisant le script de verrouillage :

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

ce qui, comme vous pouvez le voir, est beaucoup plus court. Au lieu de « payer à ce script multisignature à 5 touches », la transaction équivalente P2SH est « payer à un script avec ce hachage ». Un client effectuant un paiement à l'entreprise de

Mohammed n'a qu'à inclure ce script de verrouillage beaucoup plus court dans son paiement. Lorsque Mohammed et ses partenaires veulent dépenser cet UTXO, ils doivent présenter le script de rachat original (celui dont le hachage a verrouillé l'UTXO) et les signatures nécessaires pour le déverrouiller, comme ceci :

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

Les deux scripts sont combinés en deux étapes. Tout d'abord, le script de rachat est vérifié par rapport au script de verrouillage pour s'assurer que le hachage correspond :

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160
<redeem scriptHash> EQUAL
```

Si le hachage du script de rachat correspond, le script de déverrouillage est exécuté seul, pour déverrouiller le script de rachat :

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```

Presque tous les scripts décrits dans ce chapitre ne peuvent être implémentés qu'en tant que scripts P2SH. Par exemple, un

script de verrouillage multisignature standard 2 sur 5 ne peut pas être utilisé directement dans le script de verrouillage d'un UTXO, car IsStandard ( ) invaliderait la transaction. Pour se conformer, un script de verrouillage P2SH peut être utilisé à la place, comme vu ci-dessus. Une transaction qui inclut alors un script de déverrouillage P2SH peut être utilisée pour racheter cet UTXO et sera valide tant qu'elle ne contient pas plus de 15 clés publiques.

Conseil

N'oubliez pas qu'en raison de la politique définie par la fonction IsStandard ( ) au moment de la rédaction de cet article, les scripts multisignatures standard sont limités à au plus 3 clés publiques répertoriées, tandis que les scripts P2SH sont limités à au plus 15 clés publiques répertoriées. Les scripts multisignatures standard peuvent invalider les transactions par le biais de leur script de verrouillage ou de *déverrouillage, tandis que les scripts P2SH peuvent invalider les transactions* uniquement *par le biais de leur script de déverrouillage . En effet, IsStandard ( ) n'a aucun moyen de dire si le hachage d'un script de rachat dans un script de verrouillage comprend plus de signatures que la limite de taille actuellement imposée, de sorte qu'il ne peut observer que les scripts de déverrouillage dans les entrées de transaction.*

Adresses P2SH

Un autre élément important de la fonctionnalité P2SH est la possibilité de coder un hachage de script en tant qu'adresse,

comme défini dans BIP-13. Les adresses P2SH sont des encodages Base58Check du hachage de 20 octets d'un script, tout comme les adresses bitcoin sont des encodages Base58Check du hachage de 20 octets d'une clé publique. Les adresses P2SH utilisent le préfixe de version « 5 », ce qui entraîne des adresses codées Base58Check commençant par un « 3 ».

Par exemple, le script complexe de Mohammed, haché et encodé en Base58Check en tant qu'adresse P2SH, devient 39RF6JqABi HdYHkfChV6USGMe6Nsr66Gzw. Nous pouvons le confirmer avec la commande bx :

```
echo \
'54c557e07dde5bb6cb791c7a540e0a4796f5e97e'\
 | bx address-encode -v 5
39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw
```

Désormais, Mohammed peut donner cette "adresse" à ses clients et ils peuvent utiliser presque n'importe quel portefeuille Bitcoin pour effectuer un simple paiement, comme s'il s'agissait d'une adresse Bitcoin. Le préfixe 3 leur donne un indice qu'il s'agit d'un type d'adresse spécial, correspondant à un script au lieu d'une clé publique, mais sinon, cela fonctionne exactement de la même manière qu'un paiement à une adresse bitcoin.

Les adresses P2SH cachent toute la complexité, de sorte que la personne effectuant un paiement ne voit pas le script.

Avantages de P2SH

La fonction P2SH offre les avantages suivants par rapport à l'utilisation directe de scripts complexes dans le verrouillage des sorties :

- Les scripts complexes sont remplacés par des empreintes digitales plus courtes dans la sortie de la transaction, ce qui rend la transaction plus petite.
- Les scripts peuvent être codés comme une adresse, de sorte que l'expéditeur et le portefeuille de l'expéditeur n'ont pas besoin d'une ingénierie complexe pour implémenter P2SH.
- P2SH transfère le fardeau de la construction du script au destinataire, pas à l'expéditeur.
- P2SH déplace la charge de stockage des données pour le long script de la sortie (qui en plus d'être stockée sur la blockchain est dans l'ensemble UTXO) vers l'entrée (stockée uniquement sur la blockchain).
- P2SH déplace le fardeau du stockage des données pour le long script du moment présent (paiement) à un moment futur (lorsqu'il est dépensé).
- P2SH déplace les frais de transaction plus élevés d'un long script de l'expéditeur vers le destinataire, qui doit inclure le long script de rachat pour le dépenser.

Utiliser le script et la validation

Avant la version 0.9.2 du client Bitcoin Core, Pay-to-Script-Hash était limité aux types standard de scripts de transaction Bitcoin, par la fonction IsStandard ( ). Cela signifie que le script de rachat présenté dans la transaction de dépense ne peut être que l'un des types standard : P2PK, P2PKH ou multisig.

À partir de la version 0.9.2 du client Bitcoin Core, les transactions P2SH peuvent contenir n'importe quel script valide, ce qui rend la norme P2SH beaucoup plus flexible et permet l'expérimentation de nombreux types de transactions nouveaux et complexes.

Vous ne pouvez pas mettre un P2SH dans un script de rachat P2SH, car la spécification P2SH n'est pas récursive. De plus, bien qu'il soit techniquement possible d'inclure RETURN (voir Sortie d'enregistrement de données (RETURN)) dans un script de rachat, car rien dans les règles ne vous empêche de le faire, cela n'est d'aucune utilité pratique car l'exécution de RETURN pendant la validation entraînera la transaction. pour être marqué comme invalide.

Notez que comme le script d'échange n'est pas présenté au réseau tant que vous n'avez pas tenté de dépenser une sortie P2SH, si vous verrouillez une sortie avec le hachage d'un script d'échange non valide, il sera traité malgré tout. L'UTXO sera verrouillé avec succès. Cependant, vous ne pourrez pas le dépenser car la transaction de dépense, qui comprend le script de remboursement , ne sera pas acceptée car il s'agit d'un script non valide. Cela crée un risque, car vous pouvez verrouiller Bitcoin dans un P2SH qui ne peut pas être dépensé plus tard. Le réseau acceptera le script de verrouillage P2SH même s'il correspond à un script de rachat non valide, car le hachage du script ne donne aucune indication sur le script qu'il représente.

Avertissement

Les scripts de verrouillage P2SH contiennent le hachage

d'un script de rachat, qui ne donne aucun indice sur le contenu du script de rachat lui-même. La transaction P2SH sera considérée comme valide et acceptée même si le script d'échange n'est pas valide. Vous pourriez accidentellement verrouiller le bitcoin de manière à ce qu'il ne puisse pas être dépensé plus tard.

Sortie d'enregistrement de données (RETURN)

Le grand livre distribué et horodaté de Bitcoin, la blockchain, a des utilisations potentielles bien au-delà des paiements. De nombreux développeurs ont essayé d'utiliser le langage de script de transaction pour tirer parti de la sécurité et de la résilience du système pour des applications telles que les services de notaire numérique, les certificats d'actions et les contrats intelligents. Les premières tentatives d'utilisation du langage de script de Bitcoin à ces fins impliquaient la création de sorties de transaction qui enregistraient des données sur la blockchain ; par exemple, pour enregistrer l'empreinte numérique d'un fichier de manière à ce que n'importe qui puisse établir la preuve de l'existence de ce fichier à une date précise en se référant à cette transaction.

L'utilisation de la blockchain de Bitcoin pour stocker des données non liées aux paiements Bitcoin est un sujet controversé. De nombreux développeurs considèrent cette utilisation abusive et veulent la décourager. D'autres y voient une démonstration des puissantes capacités de la technologie blockchain et souhaitent encourager une telle expérimentation. Ceux qui s'opposent à l'inclusion de données de non-paiement affirment

que cela provoque un « gonflement de la blockchain », accablant ceux qui exécutent des nœuds bitcoin complets de supporter le coût du stockage sur disque pour des données que la blockchain n'était pas censée transporter. De plus, de telles transactions créent des UTXO qui ne peuvent pas être dépensés, en utilisant l'adresse bitcoin de destination comme un champ de 20 octets de forme libre. Étant donné que l'adresse est utilisée pour les données, elle ne correspond pas à une clé privée et l'UTXO résultant ne peut *jamais* être dépensé ; c'est un faux paiement. Ces transactions qui ne peuvent jamais être dépensées ne sont donc jamais supprimées de l'ensemble UTXO et provoquent une augmentation permanente de la taille de la base de données UTXO, ou "gonflement".

Dans la version 0.9 du client Bitcoin Core, un compromis a été atteint avec l'introduction de l'opérateur RETURN. RETURN permet aux développeurs d'ajouter 80 octets de données de non-paiement à une sortie de transaction. Cependant, contrairement à l'utilisation de "faux" UTXO, l'opérateur RETURN crée une sortie explicitement *prouvée non dépensable* , qui n'a pas besoin d'être stockée dans l'ensemble UTXO. Les sorties RETURN sont enregistrées sur la blockchain, elles consomment donc de l'espace disque et contribuent à l'augmentation de la taille de la blockchain, mais elles ne sont pas stockées dans l'ensemble UTXO et ne gonflent donc pas le pool de mémoire UTXO et alourdissent les nœuds complets avec le coût de plus RAM chère.

Les scripts RETURN ressemblent à ceci :

```
RETURN <data>
```

La partie données est limitée à 80 octets et représente le plus souvent un hachage, tel que la sortie de l'algorithme SHA256 (32 octets). De nombreuses applications placent un préfixe devant les données pour aider à identifier l'application. Par exemple, le service de notarisation numérique Preuve d'existence utilise le préfixe de 8 octets DOCPROOF, qui est encodé en ASCII 44 4f 43 50 52 4f 4f 46 en hexadécimal.

Gardez à l'esprit qu'il n'y a pas de "script de déverrouillage" qui correspond à RETURN qui pourrait éventuellement être utilisé pour "dépenser" une sortie RETURN. L'intérêt de RETURN est que vous ne pouvez pas dépenser l'argent bloqué dans cette sortie, et qu'il n'est donc pas nécessaire de le conserver dans l'ensemble UTXO comme étant potentiellement dépensable - RETURN est sans *doute inutilisable* . RETURN est généralement une sortie avec un montant de zéro bitcoin , car tout bitcoin affecté à une telle sortie est effectivement perdu à jamais. Si un RETOUR est référencé en tant qu'entrée dans une transaction, le moteur de validation de script arrêtera l'exécution du script de validation et marquera la transaction comme invalide. L'exécution de RETURN provoque essentiellement le script "RETURN" avec un FALSE et s'arrête. Ainsi, si vous référencez accidentellement une sortie RETURN en tant qu'entrée dans une transaction, cette transaction n'est pas valide.

Une transaction standard (conforme aux vérifications IsStandard ( )) ne peut avoir qu'une seule sortie RETURN. Cependant,

une seule sortie RETURN peut être combinée dans une transaction avec des sorties de tout autre type.

Deux nouvelles options de ligne de commande ont été ajoutées dans Bitcoin Core à partir de la version 0.10. L'option datacarrier contrôle le relais et l'extraction des transactions RETURN, la valeur par défaut étant "1" pour les autoriser. L'option datacarriersize prend un argument numérique spécifiant la taille maximale en octets du script RETURN, 83 octets par défaut, ce qui permet un maximum de 80 octets de données RETURN plus un octet de l'opcode RETURN et deux octets de l'opcode PUSHDATA.

Note

RETURN a été initialement proposé avec une limite de 80 octets, mais la limite a été réduite à 40 octets lorsque la fonctionnalité a été libérée. En février 2015, dans la version 0.10 de Bitcoin Core, la limite a été relevée à 80 octets. Les nœuds peuvent choisir de ne pas relayer ou d'exploiter RETURN, ou uniquement de relayer et d'exploiter RETURN contenant moins de 80 octets de données.

Timelocks

Les délais sont des restrictions sur les transactions ou les extrants qui n'autorisent les dépenses qu'après un certain temps. Bitcoin a eu une fonction de blocage du temps au niveau des transactions depuis le début. Il est implémenté par le champ nLocktime dans une transaction. Deux nouvelles

fonctionnalités de timelock ont été introduites fin 2015 et mi-2016 qui offrent des timelocks de niveau UTXO. Ce sont CHECKLOCKTIMEVERIFY et CHECKSEQUENCEVERIFY.

Les timelocks sont utiles pour postdater les transactions et bloquer les fonds à une date ultérieure. Plus important encore, les délais étendent les scripts Bitcoin dans la dimension du temps, ouvrant la porte à des contrats intelligents complexes à plusieurs étapes.

Transaction Locktime (nLocktime)

Depuis le début, Bitcoin a eu une fonction de blocage du temps au niveau des transactions. Le délai de verrouillage de la transaction est un paramètre au niveau de la transaction (un champ dans la structure de données de transaction) qui définit la première heure à laquelle une transaction est valide et peut être relayée sur le réseau ou ajoutée à la blockchain. Locktime est également connu sous le nom de nLocktime à partir du nom de variable utilisé dans la base de code Bitcoin Core. Il est mis à zéro dans la plupart des transactions pour indiquer une propagation et une exécution immédiates. Si nLocktime est différent de zéro et inférieur à 500 millions, il est interprété comme une hauteur de bloc, ce qui signifie que la transaction n'est pas valide et n'est pas relayée ou incluse dans la blockchain avant la hauteur de bloc spécifiée. S'il est supérieur ou égal à 500 millions, il est interprété comme un horodatage d'époque Unix (secondes depuis le 1er janvier 1970) et la transaction n'est pas valide avant l'heure spécifiée. Les transactions avec nLocktime spécifiant un bloc ou une heure à venir doivent être conservées par le système d'origine et transmises au réseau

Bitcoin uniquement après qu'elles sont devenues valides. Si une transaction est transmise au réseau avant le nLocktime spécifié, la transaction sera rejetée par le premier nœud comme non valide et ne sera pas relayée vers d'autres nœuds. L'utilisation de nLocktime équivaut à postdater un chèque papier.

Limitations du temps de verrouillage des transactions

nLocktime a la limitation que s'il permet de dépenser certaines sorties dans le futur, il ne rend pas impossible de les dépenser jusqu'à ce moment-là. Nous allons expliquer que par l'exemple suivant.

Alice signe une transaction en dépensant l'une de ses sorties à l' adresse de Bob et définit la transaction nLocktime sur 3 mois dans le futur. Alice envoie cette transaction à Bob pour qu'elle soit suspendue. Avec cette transaction, Alice et Bob savent que :

- Bob ne peut pas transmettre la transaction pour racheter les fonds avant que 3 mois ne se soient écoulés.
- Bob peut transmettre la transaction après 3 mois.

Pourtant :

- Alice peut créer une autre transaction, en dépensant deux fois les mêmes entrées sans temps de verrouillage. Ainsi, Alice peut passer le même UTXO avant que les 3 mois ne se soient écoulés.
- Bob n'a aucune garantie qu'Alice ne le fera pas .

Il est important de comprendre les limites de la transaction nLocktime. La seule garantie est que Bob ne pourra pas l'échanger avant que 3 mois se soient écoulés. Il n'y a aucune garantie que Bob obtiendra les fonds. Pour obtenir une telle garantie, la restriction de verrouillage temporel doit être placée sur l'UTXO lui-même et faire partie du script de verrouillage, plutôt que sur la transaction. Ceci est réalisé par la prochaine forme de blocage du temps, appelée Vérifier la vérification du temps de verrouillage.

Vérifier le temps de verrouillage Vérifier (CLTV)

En décembre 2015, une nouvelle forme de timelock a été introduite dans le bitcoin en tant que mise à niveau soft fork. Basé sur une spécification dans BIP-65, un nouvel opérateur de script appelé *CHECKLOCKTIMEVERIFY* (*CLTV*) a été ajouté au langage de script. CLTV est un timelock par sortie, plutôt qu'un timelock par transaction comme c'est le cas avec nLocktime. Cela permet une plus grande flexibilité dans la manière dont les délais sont appliqués.

En termes simples, en ajoutant l'opcode CLTV dans le script de rachat d'une sortie, il restreint la sortie, de sorte qu'elle ne puisse être dépensée qu'après l'écoulement du temps spécifié.

Conseil

Alors que nLocktime est un timelock au niveau de la transaction, CLTV est un timelock basé sur la sortie.

CLTV ne remplace pas nLocktime, mais restreint plutôt UTXO

spécifique de sorte qu'ils ne puissent être dépensés que dans une transaction future avec nLocktime défini sur une valeur supérieure ou égale.

L'opcode CLTV prend un paramètre en entrée, exprimé sous forme de nombre dans le même format que nLocktime (soit une hauteur de bloc ou une heure d'époque Unix). Comme indiqué par le suffixe VERIFY, CLTV est le type d'opcode qui arrête l'exécution du script si le résultat est FALSE. S'il en résulte TRUE, l'exécution se poursuit.

Afin de verrouiller une sortie avec CLTV, vous l'insérez dans le script de rachat de la sortie dans la transaction qui crée la sortie. Par exemple, si Alice paie l'adresse de Bob, la sortie contiendrait normalement un script P2PKH comme celui-ci :

```
DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY
CHECKSIG
```

Pour la verrouiller à un moment, disons dans 3 mois, la transaction serait une transaction P2SH avec un script de rachat comme celui-ci :

```
<now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160
<Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

où <now {plus} 3 mois> est une hauteur de bloc ou une valeur de temps estimée à 3 mois à partir du moment où la transaction est minée : hauteur de bloc actuelle + 12 960 (blocs) ou heure actuelle de l'époque Unix + 7 760 000 (secondes). Pour l'instant, ne vous inquiétez pas de l'opcode DROP qui suit CHECKLOCKTIMEVERIFY ; cela sera expliqué sous peu.

Lorsque Bob essaie de dépenser cet UTXO, il construit une transaction qui fait référence à l'UTXO en tant qu'entrée. Il utilise sa signature et sa clé publique dans le script de déverrouillage de cette entrée et définit le nLocktime de la transaction pour qu'il soit égal ou supérieur au verrou temporel dans l'ensemble CHECKLOCKTIMEVERIFY Alice. Bob diffuse ensuite la transaction sur le réseau Bitcoin.

La transaction de Bob est évaluée comme suit. Si le paramètre CHECKLOCKTIMEVERIFY Alice set est inférieur ou égal au nLocktime de la transaction de dépense, l'exécution du script se poursuit (agit comme si une opération « no operation » ou NOP opcode était exécutée). Sinon, l'exécution du script s'arrête et la transaction est considérée comme invalide.

Plus précisément, CHECKLOCKTIMEVERIFY échoue et interrompt l'exécution, marquant la transaction invalide si (source : BIP-65) :

1. la pile est vide ; ou alors
2. l'élément supérieur de la pile est inférieur à 0 ; ou alors
3. le type de timelock (hauteur par rapport à l'horodatage) de l'élément supérieur de la pile et le champ nLocktime ne sont pas les mêmes ; ou alors

4. l'élément supérieur de la pile est supérieur au champ nLocktime de la transaction ; ou alors
5. le champ nSequence de l'entrée est 0xffffffff.

Note

CLTV et nLocktime utilisent le même format pour décrire les timelocks, soit une hauteur de bloc ou le temps écoulé en secondes depuis l'époque Unix. De manière critique, lorsqu'il est utilisé ensemble, le format de nLocktime doit correspondre à celui de CLTV dans les sorties - ils doivent tous deux référencer soit la hauteur du bloc, soit le temps en secondes.

Après l'exécution, si CLTV est satisfait, le paramètre de temps qui l'a précédé reste comme l'élément supérieur de la pile et peut devoir être supprimé, avec DROP, pour une exécution correcte des opcodes de script suivants. Vous verrez souvent CHECKLOCKTIMEVERIFY suivi de DROP dans les scripts pour cette raison.

En utilisant nLocktime en conjonction avec CLTV, le scénario décrit dans les limitations de temps de verrouillage de transaction change. Alice ne peut plus dépenser l'argent (car il est verrouillé avec la clé de Bob) et Bob ne peut plus le dépenser avant l'expiration du délai de verrouillage de 3 mois.

En introduisant la fonctionnalité de timelock directement dans le langage de script, CLTV nous permet de développer des scripts complexes très intéressants

La norme est définie dans le BIP-65 (CHECKLOCKTIMEVERIFY) .

Timelocks relatifs

nLocktime et CLTV sont tous deux *des horloges absolues* en ce sens qu'ils spécifient un point dans le temps absolu. Les deux prochaines fonctionnalités de timelock que nous examinerons sont *des timelocks relatifs* en ce sens qu'ils spécifient, comme condition de dépense d'une sortie, un temps écoulé à partir de la confirmation de la sortie dans la blockchain.

Les délais relatifs sont utiles car ils permettent à une chaîne de deux transactions interdépendantes ou plus d'être tenues hors chaîne, tout en imposant une contrainte de temps sur une transaction qui dépend du temps écoulé depuis la confirmation d'une transaction précédente. En d'autres termes, l'horloge ne commence à compter que lorsque l'UTXO est enregistré sur la blockchain. Cette fonctionnalité est particulièrement utile dans les canaux d'état bidirectionnels et Lightning Networks.

Les délais relatifs, comme les délais absolus, sont implémentés avec à la fois une fonctionnalité au niveau de la transaction et un opcode au niveau du script. Le blocage temporel relatif au niveau de la transaction est implémenté en tant que règle de consensus sur la valeur de nSequence, un champ de transaction défini dans chaque entrée de transaction. Les délais relatifs au niveau du script sont implémentés avec l'opcode CHECKSEQUENCEVERIFY (CSV).

Les horloges relatives sont implémentées selon les spécifications de BIP-68, Temps de verrouillage relatif utilisant

des numéros de séquence imposés par consensus et BIP-112, CHECKSEQUENCEVERIFY .

Le BIP-68 et le BIP-112 ont été activés en mai 2016 en tant que mise à niveau souple des règles de consensus.

Timelocks relatifs avec nSequence

Des délais relatifs peuvent être définis sur chaque entrée d'une transaction, en définissant le champ nSequence dans chaque entrée.

Signification originale de nSequence

Le champ nSequence était à l'origine destiné (mais jamais correctement implémenté) pour permettre la modification des transactions dans le mempool. Dans cette utilisation, une transaction contenant des entrées avec une valeur nSequence inférieure à 2 32 - 1 (0xFFFFFFFF) a indiqué une transaction qui n'était pas encore « finalisée ». Une telle transaction serait conservée dans le mempool jusqu'à ce qu'elle soit remplacée par une autre transaction dépensant les mêmes entrées avec une valeur nSequence plus élevée. Une fois qu'une transaction reçue dont les entrées avaient une valeur nSequence de 0xFFFFFFFF, elle serait considérée comme "finalisée" et exploitée.

La signification originale de nSequence n'a jamais été correctement implémentée et la valeur de nSequence est habituellement définie sur 0xFFFFFFFF dans les transactions qui n'utilisent pas de timelocks. Pour les transactions avec nLocktime ou CHECKLOCKTIMEVERIFY, la valeur nSequence doit être

définie sur moins de 2 31 pour que les gardes de timelock aient un effet, comme expliqué ci-dessous.

nSéquence en tant que délai relatif imposé par consensus

Depuis l'activation du BIP-68, de nouvelles règles de consensus s'appliquent pour toute transaction contenant une entrée dont la valeur nSequence est inférieure à 2 31 (le bit 1 « 31 n'est pas positionné). Par programme, cela signifie que si le bit le plus significatif (bit 1 « 31) n'est pas défini, c'est un drapeau qui signifie "temps de verrouillage relatif". Sinon (bit 1 « 31 défini), la valeur nSequence est réservée à d'autres utilisations telles que l'activation de CHECKLOCKTIMEVERIFY, nLocktime, Opt-In-Replace-By-Fee et d'autres développements futurs.

Les entrées de transaction avec des valeurs de nSequence inférieures à 2 31 sont interprétées comme ayant un verrou temporel relatif. Une telle transaction n'est valide que lorsque l'entrée a vieilli du montant relatif de la barre temporelle. Par exemple, une transaction avec une entrée avec un blocage temporel relatif nSequence de 30 blocs n'est valide que lorsqu'au moins 30 blocs se sont écoulés depuis le moment où l'UTXO référencé dans l'entrée a été extrait. Puisque nSequence est un champ par entrée, une transaction peut contenir n'importe quel nombre d'entrées verrouillées dans le temps, qui doivent toutes avoir suffisamment vieilli pour que la transaction soit valide. Une transaction peut inclure à la fois des entrées verrouillées dans le temps (nSequence <2 31) et des entrées sans blocage temporel relatif (nSequence> = 2 31).

La valeur nSequence est spécifiée en blocs ou en secondes,

mais dans un format légèrement différent de celui utilisé dans nLocktime. Un indicateur de type est utilisé pour différencier les valeurs comptant les blocs et les valeurs comptant le temps en secondes. L'indicateur de type est positionné dans le 23e bit le moins significatif (c'est-à-dire la valeur 1 « 22). Si l'indicateur de type est défini, la valeur nSequence est interprétée comme un multiple de 512 secondes. Si l'indicateur de type n'est pas défini, la valeur nSequence est interprétée comme un nombre de blocs.

Lors de l'interprétation de nSequence comme un verrou temporel relatif, seuls les 16 bits les moins significatifs sont pris en compte. Une fois que les indicateurs (bits 32 et 23) sont évalués, la valeur de nSequence est généralement "masquée" avec un masque de 16 bits (par exemple, nSequence & 0x0000FFFF).

La définition BIP-68 du codage nSequence (Source : BIP-68) montre la disposition binaire de la valeur nSequence, telle que définie par BIP-68.

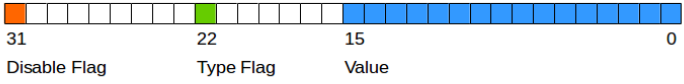


Figure 1. Définition BIP-68 du codage nSequence (Source : BIP-68)

Les délais relatifs basés sur l'application par consensus de la valeur nSequence sont définis dans le BIP-68.

La norme est définie dans BIP-68, Temps de verrouillage relatif utilisant des numéros de séquence imposés par consensus .

Timelocks relatifs avec CSV

Tout comme CLTV et nLocktime, il existe un opcode de script pour les timelocks relatifs qui exploite la valeur nSequence dans les scripts. Cet opcode est CHECKSEQUENCEVERIFY, communément appelé CSV en abrégé.

L'opcode CSV, lorsqu'il est évalué dans le script de remboursement d'un UTXO, permet de dépenser uniquement dans une transaction dont la valeur nSequence d'entrée est supérieure ou égale au paramètre CSV. Essentiellement, cela limite les dépenses de l'UTXO jusqu'à ce qu'un certain nombre de blocs ou de secondes se soient écoulés par rapport au temps où l'UTXO a été miné.

Comme avec CLTV, la valeur au format CSV doit correspondre au format de la valeur nSequence correspondante. Si CSV est spécifié en termes de blocs, nSequence doit l'être également. Si CSV est spécifié en termes de secondes, nSequence doit l'être également.

Les délais relatifs avec CSV sont particulièrement utiles lorsque plusieurs transactions (chaînées) sont créées et signées, mais pas propagées, lorsqu'elles sont maintenues « hors chaîne ». Une transaction enfant ne peut pas être utilisée tant que la transaction parent n'a pas été propagée, exploitée et vieillie à l'heure spécifiée dans le délai relatif.

CSV est défini en détail dans BIP-112, CHECKSEQUENCEVERIFY .

Median-Time-Past

Dans le cadre de l'activation des horloges relatives, il y a également eu un changement dans la façon dont le « temps » est calculé pour les horloges (absolues et relatives). En bitcoin, il existe une différence subtile, mais très significative, entre l'heure du mur et l'heure du consensus. Bitcoin est un réseau décentralisé, ce qui signifie que chaque participant a sa propre perspective du temps. Les événements sur le réseau ne se produisent pas instantanément partout. La latence du réseau doit être prise en compte dans la perspective de chaque nœud. Finalement, tout est synchronisé pour créer un grand livre commun. Bitcoin atteint un consensus toutes les 10 minutes sur l'état du grand livre tel qu'il existait dans le *passé* .

Les horodatages définis dans les en-têtes de bloc sont définis par les mineurs. Il existe un certain degré de latitude autorisé par les règles de consensus pour tenir compte des différences de précision d'horloge entre les nœuds décentralisés. Cependant, cela crée une incitation malheureuse pour les mineurs à mentir à propos de l'heure dans un bloc afin de gagner des frais supplémentaires en incluant des transactions bloquées dans le temps qui ne sont pas encore arrivées à maturité. Consultez la section suivante pour plus d'informations.

Pour supprimer l'incitation au mensonge et renforcer la sécurité des délais, un PIF a été proposé et activé en même temps que les PIF pour les délais relatifs. Il s'agit du BIP-113, qui définit

une nouvelle mesure consensuelle du temps appelée *Median-Time-Past* .

Median-Time-Past est calculé en prenant les horodatages des 11 derniers blocs et en trouvant la médiane. Ce temps médian devient alors le temps de consensus et est utilisé pour tous les calculs de timelock. En prenant le point médian d'environ deux heures dans le passé, l'influence de l'horodatage d'un bloc est réduite. En incorporant 11 blocs, aucun mineur ne peut influencer les horodatages afin de gagner des frais sur les transactions avec un délai qui n'a pas encore mûri.

Median-Time-Past modifie la mise en œuvre des calculs de temps pour nLocktime, CLTV, nSequence et CSV. L'heure de consensus calculée par Median-Time-Past est toujours d'environ une heure derrière l'heure de l'horloge murale. Si vous créez des transactions de verrouillage temporel, vous devez en tenir compte lors de l'estimation de la valeur souhaitée à encoder dans nLocktime, nSequence, CLTV et CSV.

Median-Time-Past est spécifié dans BIP-113 .

Timelock Defence contre les sniping payants

Fee-sniping est un scénario d'attaque théorique, où les mineurs qui tentent de réécrire les blocs passés « snip » les transactions plus coûteuses des blocs futurs pour maximiser leur rentabilité.

Par exemple, disons que le bloc le plus élevé existant est le bloc n ° 100 000. Si au lieu d'essayer d'exploiter le bloc # 100 001 pour étendre la chaîne, certains mineurs tentent de

récupérer le # 100 000. Ces mineurs peuvent choisir d'inclure toute transaction valide (qui n'a pas encore été minée) dans leur bloc candidat # 100000. Ils n'ont pas à reminer le bloc avec les mêmes transactions. En fait, ils sont incités à sélectionner les transactions les plus rentables (frais par ko les plus élevés) à inclure dans leur bloc. Ils peuvent inclure toutes les transactions qui se trouvaient dans l '« ancien » bloc # 100 000, ainsi que toutes les transactions du mempool actuel. Essentiellement, ils ont la possibilité d'extraire des transactions du « présent » dans le « passé » réécrit lorsqu'ils recréent le bloc # 100 000.

Aujourd'hui, cette attaque n'est pas très lucrative, car la récompense de bloc est beaucoup plus élevée que le total des frais par bloc. Mais à un moment donné dans le futur, les frais de transaction représenteront la majorité de la récompense minière (ou même l'intégralité de la récompense minière). À ce moment-là, ce scénario devient inévitable.

Pour éviter le « sniping de frais », lorsque Bitcoin Core crée des transactions, il utilise nLocktime pour les limiter au « bloc suivant », par défaut. Dans notre scénario, Bitcoin Core définirait nLocktime à 100 001 sur toute transaction créée. Dans des circonstances normales, ce nLocktime n'a aucun effet - les transactions ne peuvent être incluses que dans le bloc # 100 001 de toute façon ; c'est le prochain bloc.

Mais dans le cadre d'une attaque blockchain / double dépense, les mineurs ne seraient pas en mesure d'extraire des transactions à frais élevés du mempool, car toutes ces transactions seraient verrouillées dans le temps pour bloquer # 100,001. Ils ne peuvent récupérer que 100 000 $ avec toutes les transactions

valides à ce moment-là, ne gagnant essentiellement pas de nouveaux frais.

Pour ce faire, Bitcoin Core définit le nLocktime sur toutes les nouvelles transactions sur <bloc actuel # + 1> et définit le nSequence sur toutes les entrées sur 0xFFFFFFFE pour activer nLocktime.

Scripts avec contrôle de flux (clauses conditionnelles)

L'une des fonctionnalités les plus puissantes de Bitcoin Script est le contrôle de flux, également connu sous le nom de clauses conditionnelles. Vous connaissez probablement le contrôle de flux dans divers langages de programmation qui utilisent la construction IF … THEN … ELSE. Les clauses conditionnelles Bitcoin ont un aspect un peu différent, mais sont essentiellement la même construction.

À un niveau de base, les opcodes conditionnels bitcoin nous permettent de construire un script de rachat qui a deux façons d'être déverrouillé, en fonction d'un résultat TRUE / FALSE de l'évaluation d'une condition logique. Par exemple, si x est TRUE, le script de rachat est A et le script de rachat ELSE est B.

De plus, les expressions conditionnelles bitcoin peuvent être "imbriquées" indéfiniment, ce qui signifie qu'une clause conditionnelle peut en contenir une autre, qui en contient une autre, etc. Le contrôle de flux de script Bitcoin peut être utilisé pour construire des scripts très complexes avec des centaines voire

des milliers de chemins d'exécution possibles . Il n'y a pas de limite à l'imbrication, mais les règles de consensus imposent une limite à la taille maximale, en octets, d'un script.

Bitcoin implémente le contrôle de flux à l'aide des opcodes IF, ELSE, ENDIF et NOTIF. En outre, les expressions conditionnelles peuvent contenir des opérateurs booléens tels que BOOLAND, BOOLOR et NOT.

À première vue, vous pouvez trouver les scripts de contrôle de flux du bitcoin déroutants. En effet, Bitcoin Script est un langage de pile. De la même façon que 1 {plus} 1 regarde « en arrière » lorsqu'il est exprimé comme 1 1 ADD, les clauses de contrôle de flux dans Bitcoin regardent également « en arrière ».

Dans la plupart des langages de programmation traditionnels (procéduraux), le contrôle de flux ressemble à ceci :

Pseudocode de contrôle de flux dans la plupart des langages de programmation

```
if (condition):
  code to run when condition is true
else:
  code to run when condition is false
code to run in either case
```

Dans un langage basé sur la pile comme Bitcoin Script, la condition logique vient avant le IF, ce qui lui donne un aspect « en arrière », comme ceci :

Contrôle de flux de script Bitcoin

```
condition
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

Lors de la lecture du script Bitcoin, rappelez-vous que la condition en cours d'évaluation *précède* l'opcode IF.

Clauses conditionnelles avec opcodes VERIFY

Une autre forme de conditionnel dans Bitcoin Script est tout opcode qui se termine par VERIFY. Le suffixe VERIFY signifie que si la condition évaluée n'est pas TRUE, l'exécution du script se termine immédiatement et la transaction est considérée comme invalide.

Contrairement à une clause IF, qui offre des chemins d'exécution alternatifs, le suffixe VERIFY agit comme une *clause de garde* , ne continuant que si une condition préalable est remplie.

Par exemple, le script suivant nécessite la signature de Bob et une pré-image (secrète) qui produit un hachage spécifique. Les deux conditions doivent être remplies pour le déverrouiller :

Un script de rachat avec une clause de garde EQUALVERIFY.

```
HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey>
CHECKSIG
```

Pour utiliser ceci, Bob doit construire un script de déverrouillage qui présente une pré-image valide et une signature :

Un script de déverrouillage pour satisfaire le rachat au- dessus de script

```
<Bob's Sig> <hash pre-image>
```

Sans présenter la pré-image, Bob ne peut pas accéder à la partie du script qui vérifie sa signature.

Ce script peut être écrit avec un IF à la place :

Un script de rachat avec une clause de garde IF

```
HASH160 <expected hash> EQUAL
IF
   <Bob's Pubkey> CHECKSIG
ENDIF
```

Le script de déverrouillage de Bob est identique :

Un script de déverrouillage pour satisfaire le rachat au- dessus de script

```
<Bob's Sig> <hash pre-image>
```

Le script avec IF fait la même chose que l'utilisation d'un opcode avec un suffixe VERIFY ; ils fonctionnent tous deux comme des clauses de garde. Cependant, la construction VERIFY est plus efficace, en utilisant deux opcodes de moins.

Alors, quand utilisons-nous VERIFY et quand utilisons-nous IF ? Si tout ce que nous essayons de faire est de joindre une précondition (clause de garde), alors VERIFY est mieux. Si, cependant, nous voulons avoir plus d'un chemin d'exécution (contrôle de flux), alors nous avons besoin d'une clause de contrôle de flux IF … ELSE.

> *Conseil*
>
> *Un opcode tel que EQUAL poussera le résultat (TRUE / FALSE) sur la pile, le laissant là pour évaluation par les opcodes suivants. En revanche, le suffixe de l'opcode EQUALVERIFY ne laisse rien sur la pile. Les opcodes qui se terminent par VERIFY ne laissent pas le résultat sur la pile.*

Utilisation du contrôle de flux dans les scripts

Une utilisation très courante du contrôle de flux dans Bitcoin Script est de construire un script de rachat qui offre plusieurs chemins d'exécution, chacun étant une manière différente de racheter l'UTXO.

Prenons un exemple simple, où nous avons deux signataires, Alice et Bob, et l'un ou l'autre peut échanger. Avec multisig, cela serait exprimé sous la forme d'un script multisig 1 sur 2. Par souci de démonstration, nous ferons la même chose avec une clause IF :

```
IF
 <Alice's Pubkey> CHECKSIG
ELSE
 <Bob's Pubkey> CHECKSIG
ENDIF
```

En regardant ce script de rachat , vous vous demandez peut-être : "Où est la condition ? Il n'y a rien qui précède la clause IF !"

La condition ne fait pas partie du script de rachat . Au lieu de cela, la condition sera proposée dans le script de déverrouillage, permettant à Alice et Bob de "choisir" le chemin d'exécution de leur choix.

Alice rachète ceci avec le script de déverrouillage :

```
<Alice's Sig> 1
```

Le 1 à la fin sert de condition (TRUE) qui fera que la clause IF exécute le premier chemin de remboursement pour lequel Alice a une signature.

Pour que Bob utilise ceci, il devrait choisir le deuxième chemin d'exécution en donnant une valeur FALSE à la clause IF :

```
<Bob's Sig> 0
```

Le script de déverrouillage de Bob met un 0 sur la pile, ce qui oblige la clause IF à exécuter le second script (ELSE), qui nécessite la signature de Bob.

Puisque les clauses IF peuvent être imbriquées, nous pouvons créer un "labyrinthe" de chemins d'exécution. Le script de déverrouillage peut fournir une "carte" sélectionnant le chemin d'exécution réellement exécuté :

```
IF
  script A
ELSE
  IF
    script B
  ELSE
    script C
  ENDIF
ENDIF
```

Dans ce scénario, il existe trois chemins d'exécution (script A, script B et script C). Le script de déverrouillage fournit un chemin sous la forme d'une séquence de valeurs TRUE ou FALSE. Pour sélectionner le script de chemin B, par exemple, le script de déverrouillage doit se terminer par 1 0 (TRUE, FALSE). Ces valeurs seront poussées sur la pile, de sorte que la deuxième

valeur (FALSE) se retrouve en haut de la pile. La clause IF externe affiche la valeur FALSE et exécute la première clause ELSE. Ensuite, la valeur TRUE se déplace vers le haut de la pile et est évaluée par le IF interne (imbriqué), en sélectionnant le chemin d'exécution B.

En utilisant cette construction, nous pouvons créer des scripts de rachat avec des dizaines ou des centaines de chemins d'exécution, chacun offrant une manière différente de valider l'UTXO. Pour dépenser, nous construisons un script de déverrouillage qui navigue dans le chemin d'exécution en plaçant les valeurs TRUE et FALSE appropriées sur la pile à chaque point de contrôle de flux.

Exemple de script complexe

Dans cette section, nous combinons de nombreux concepts de ce chapitre en un seul exemple.

Notre exemple utilise l'histoire de Mohammed, le propriétaire de l'entreprise à Dubaï qui exploite une entreprise d'import / export.

Dans cet exemple, Mohammed souhaite construire un compte de capital d'entreprise avec des règles flexibles. Le schéma qu'il crée nécessite différents niveaux d'autorisation en fonction des délais. Les participants au programme multisig sont Mohammed, ses deux partenaires Saeed et Zaira et leur avocat d'entreprise Abdul. Les trois partenaires prennent des décisions

basées sur une règle de majorité, donc deux des trois doivent être d'accord. Cependant, en cas de problème avec leurs clés, ils souhaitent que leur avocat puisse récupérer les fonds avec l'une des trois signatures partenaires. Enfin, si tous les partenaires sont indisponibles ou incapables pendant un certain temps, ils souhaitent que l'avocat puisse gérer directement le compte.

Voici le script de rachat conçu par Mohammed pour y parvenir (préfixe du numéro de ligne XX) :

Multi-signature variable avec Timelock

```
01  IF
02    IF
03      2
04    ELSE
05      <30 days> CHECKSEQUENCEVERIFY DROP
06      <Abdul the Lawyer's Pubkey> CHECKSIGVERIFY
07      1
08    ENDIF
09    <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's
Pubkey> 3 CHECKMULTISIG
10  ELSE
11    <90 days> CHECKSEQUENCEVERIFY DROP
12    <Abdul the Lawyer's Pubkey> CHECKSIG
13  ENDIF
```

Le script de Mohammed implémente trois chemins d'exécution en utilisant des clauses de contrôle de flux IF … ELSE imbriquées.

Dans le premier chemin d'exécution, ce script fonctionne

comme un simple multisig 2 sur 3 avec les trois partenaires. Ce chemin d'exécution se compose des lignes 3 et 9. La ligne 3 définit le quorum du multisig sur 2 (2 sur 3). Ce chemin d'exécution peut être sélectionné en mettant TRUE TRUE à la fin du script de déverrouillage :

Script de déverrouillage pour le premier chemin d'exécution (multisig 2 sur 3)

```
0 <Mohammed's Sig> <Zaira's Sig> TRUE TRUE
```

> *Conseil*
>
> *Le 0 au début de ce script de déverrouillage est dû à un bogue dans CHECKMULTISIG qui fait apparaître une valeur supplémentaire de la pile. La valeur supplémentaire n'est pas prise en compte par CHECKMULTISIG, mais elle doit être présente ou le script échoue. Pousser 0 (habituellement) est une solution de contournement au bogue, comme décrit dans Un bogue dans l'exécution de CHECKMULTISIG .*

Le deuxième chemin d'exécution ne peut être utilisé qu'après 30 jours à compter de la création de l'UTXO. À ce moment-là, il faut la signature d'Abdul l'avocat et l'un des trois associés (un multisig 1 sur 3). Ceci est réalisé par la ligne 7, qui définit le quorum pour le multisig à 1. Pour sélectionner ce chemin d'exécution, le script de déverrouillage se terminerait par FALSE TRUE :

Script de déverrouillage pour le deuxième chemin d'exécution (Avocat + 1 sur 3)

```
0 <Abdul the Lawyer's Sig> <Saeed's Sig> FALSE TRUE
```

Conseil

Pourquoi FALSE TRUE ? N'est-ce pas en arrière ? Parce que les deux valeurs sont poussées vers la pile, avec FALSE poussé en premier, puis TRUE poussé en second. TRUE est donc d' abord *sauté par le premier opcode IF.*

Enfin, la troisième voie d'exécution permet à Abdul l'avocat de dépenser les fonds seul, mais seulement après 90 jours. Pour sélectionner ce chemin d'exécution, le script de déverrouillage doit se terminer par FALSE :

Script de déverrouillage pour le troisième chemin d'exécution (avocat uniquement)

```
<Abdul the Lawyer's Sig> FALSE
```

Essayez d'exécuter le script sur papier pour voir comment il se comporte sur la pile.

Quelques autres choses à considérer lors de la lecture de cet exemple. Voyez si vous pouvez trouver les réponses :

- Pourquoi l'avocat ne peut-il pas utiliser le troisième chemin

d'exécution à tout moment en le sélectionnant avec FALSE sur le script de déverrouillage ?
- Combien de chemins d'exécution peuvent être utilisés respectivement 5, 35 et 105 jours après l'extraction de l'UTXO ?
- Les fonds sont-ils perdus si l'avocat perd sa clé ? Votre réponse change-t-elle si 91 jours se sont écoulés ?
- Comment les partenaires « remettent-ils » l'horloge à zéro tous les 29 ou 89 jours pour empêcher l'avocat d'accéder aux fonds ?
- Pourquoi certains opcodes CHECKSIG dans ce script ont-ils le suffixe VERIFY alors que d'autres n'en ont pas ?

Témoin séparé

Segregated Witness (segwit) est une mise à niveau des règles de consensus Bitcoin et du protocole réseau, proposée et mise en œuvre en tant que soft-fork BIP-9 qui a été activé sur le réseau principal de Bitcoin le 1er août 2017.

En cryptographie, le terme « témoin » est utilisé pour décrire une solution à un puzzle cryptographique. En termes de bitcoin, le témoin satisfait une condition cryptographique placée sur une sortie de transaction non dépensée (UTXO).

Dans le contexte du bitcoin, une signature numérique est *un type de témoin* , mais un témoin est plus largement toute solution qui peut satisfaire les conditions imposées à un UTXO et débloquer cet UTXO pour les dépenses. Le terme « témoin » est un terme plus général désignant un « script de déverrouillage »

ou « scriptSig ».

Avant l'introduction de segwit, chaque entrée dans une transaction était suivie des données témoins qui la déverrouillaient. Les données du témoin ont été intégrées à la transaction dans le cadre de chaque entrée. Le terme *témoin séparé* , ou *segwit* pour faire court, signifie simplement séparer la signature ou le script de déverrouillage d'une sortie spécifique. Pensez à « scriptSig séparé » ou « signature séparée » dans la forme la plus simple.

Le témoin séparé est donc une modification architecturale du bitcoin qui vise à déplacer les données de témoin du champ scriptSig (script de déverrouillage) d'une transaction vers une structure de données de *témoin* distincte qui accompagne une transaction. Les clients peuvent demander des données de transaction avec ou sans les données de témoin qui l'accompagnent.

Dans cette section, nous examinerons certains des avantages de Segregated Witness, décrirons le mécanisme utilisé pour déployer et implémenter ce changement d'architecture et démontrerons l'utilisation de Segregated Witness dans les transactions et les adresses.

Le témoin séparé est défini par les PIF suivants :

BIP-141
La définition principale de témoin isolé.

BIP-143
Vérification de la signature de transaction pour le programme témoin de la version 0

BIP-144

Services homologues : nouveaux messages réseau et formats de sérialisation

BIP-145

Mises à jour getblocktemplate pour les témoins séparés (pour l'exploitation minière)

BIP-173

Format d'adresse Base32 pour les sorties témoins natives v0-16

Pourquoi Segregated Witness ?

Segregated Witness est un changement architectural qui a plusieurs effets sur l'évolutivité, la sécurité, les incitations économiques et les performances du bitcoin :

Malléabilité des transactions

En déplaçant le témoin en dehors des données de transaction, le hachage de transaction utilisé comme identifiant n'inclut plus les données de témoin. Étant donné que les données témoins sont la seule partie de la transaction qui peut être modifiée par un tiers (voir Identificateurs de transaction), leur suppression supprime également la possibilité d'attaques de malléabilité des transactions. Avec Segregated Witness, les hachages de transaction deviennent immuables par quiconque autre que le créateur de la transaction, ce qui améliore considérablement la mise en œuvre de nombreux autres protocoles qui reposent sur une construction de transactions Bitcoin avancée, tels que les canaux de paiement, les transactions chaînées et les réseaux

éclair.

Gestion des versions de script

Avec l'introduction des scripts de témoins séparés, chaque script de verrouillage est précédé d'un numéro de *version de script* , similaire à la façon dont les transactions et les blocs ont des numéros de version. L'ajout d'un numéro de version de script permet de mettre à niveau le langage de script d'une manière rétrocompatible (c'est-à-dire en utilisant des mises à niveau de soft fork) pour introduire de nouveaux opérandes, syntaxe ou sémantique de script. La possibilité de mettre à niveau le langage de script de manière non perturbatrice accélérera considérablement le taux d'innovation dans le bitcoin.

Évolutivité du réseau et du stockage

Les données des témoins contribuent souvent beaucoup à la taille totale d'une transaction. Les scripts plus complexes tels que ceux utilisés pour les canaux multisig ou de paiement sont très volumineux. Dans certains cas, ces scripts représentent la majorité (plus de 75%) des données d'une transaction. En déplaçant les données des témoins en dehors des données de transaction, Segregated Witness améliore l'évolutivité du bitcoin. Les nœuds peuvent élaguer les données des témoins après avoir validé les signatures, ou les ignorer complètement lors de la vérification simplifiée des paiements. Les données témoins n'ont pas besoin d'être transmises à tous les nœuds et n'ont pas besoin d'être stockées sur le disque par tous les nœuds.

Optimisation de la vérification des signatures

Segregated Witness met à niveau les fonctions de signature

(CHECKSIG, CHECKMULTISIG, etc.) pour réduire la complexité de calcul de l'algorithme. Avant segwit, l'algorithme utilisé pour produire une signature nécessitait un nombre d' opérations de hachage proportionnel à la taille de la transaction. Les calculs de hachage de données ont augmenté en O (n 2) par rapport au nombre d'opérations de signature, introduisant une charge de calcul substantielle sur tous les nœuds vérifiant la signature. Avec segwit, l'algorithme est modifié pour réduire la complexité à O (n).

Amélioration de la signature hors ligne

Les signatures de témoins séparés incorporent la valeur (montant) référencée par chaque entrée dans le hachage qui est signé. Auparavant, un périphérique de signature hors ligne, tel qu'un portefeuille matériel, devait vérifier le montant de chaque entrée avant de signer une transaction. Cela était généralement accompli en diffusant en continu une grande quantité de données sur les transactions précédentes référencées en tant qu'entrées. Étant donné que le montant fait désormais partie du hachage d'engagement qui est signé, un appareil hors ligne n'a pas besoin des transactions précédentes. Si les montants ne correspondent pas (sont faussés par un système en ligne compromis), la signature sera invalide.

Comment fonctionne Segregated Witness

À première vue, Segregated Witness semble être un changement dans la façon dont les transactions sont construites et donc une fonctionnalité au niveau des transactions, mais ce n'est pas le cas. Au contraire, le témoin séparé est un changement dans la façon dont chaque UTXO est dépensé et est donc une fonction

par sortie.

Une transaction peut dépenser des sorties de témoins séparés ou des sorties traditionnelles (témoins en ligne) ou les deux. Par conséquent, cela n'a pas beaucoup de sens de qualifier une transaction de « transaction de témoin séparé ». Nous devrions plutôt désigner les extrants de transaction spécifiques comme des « extrants de témoins séparés ».

Lorsqu'une transaction passe un UTXO, elle doit fournir un témoin. Dans un UTXO traditionnel, le script de verrouillage nécessite que les données témoins soient fournies en *ligne* dans la partie d'entrée de la transaction qui passe l'UTXO. Un UTXO de témoin séparé, cependant, spécifie un script de verrouillage qui peut être satisfait avec des données de témoin en dehors de l'entrée (séparé).

Fourche souple (compatibilité descendante)

Le témoin séparé est un changement important dans la façon dont les extrants et les transactions sont architecturés. Un tel changement nécessiterait normalement un changement simultané de chaque nœud bitcoin et de chaque portefeuille pour changer les règles de consensus - ce que l'on appelle un hard fork. Au lieu de cela, le témoin séparé est introduit avec un changement beaucoup moins perturbateur, qui est rétrocompatible, connu sous le nom de soft fork. Ce type de mise à niveau permet aux logiciels non mis à niveau d'ignorer les modifications et de continuer à fonctionner sans aucune interruption.

Les sorties de témoins séparés sont construites de sorte que les systèmes plus anciens qui ne sont pas compatibles avec segwit puissent toujours les valider. Pour un ancien portefeuille ou nœud, une sortie de témoin séparé ressemble à une sortie que *tout le monde peut dépenser* . De telles sorties peuvent être dépensées avec une signature vide, par conséquent le fait qu'il n'y ait pas de signature à l'intérieur de la transaction (elle est séparée) n'invalide pas la transaction. Les nouveaux portefeuilles et nœuds de minage, cependant, voient la sortie du témoin séparé et s'attendent à trouver un témoin valide pour cela dans les données de témoin de la transaction.

Résultats de témoins séparés et exemples de transactions

Examinons certains de nos exemples de transactions et voyons comment elles changeraient avec Segregated Witness. Nous allons d' abord voir comment un paiement Pay-to-Public-Key-Hash (P2PKH) est transformé avec le programme Segregated Witness. Ensuite, nous examinerons l'équivalent de Segregated Witness pour les scripts Pay-to-Script-Hash (P2SH). Enfin, nous verrons comment les deux programmes Segregated Witness précédents peuvent être intégrés dans un script P2SH.

Hachage de clé publique Pay-to-Witness (P2WPKH)

Dans le chapitre 2 , Alice a créé une transaction pour payer Bob pour une tasse de café. Cette transaction a créé une sortie P2PKH d'une valeur de 0,015 BTC qui pouvait être dépensée par Bob. Le script de la sortie ressemble à ceci :

Exemple de script de sortie P2PKH

```
DUP HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
EQUALVERIFY CHECKSIG
```

Avec Segregated Witness, Alice créerait un script Pay-to-Witness-Public-Key-Hash (P2WPKH), qui ressemble à ceci :

Exemple de script de sortie P2WPKH

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Comme vous pouvez le voir, le script de verrouillage d'une sortie de témoin séparé est beaucoup plus simple qu'une sortie traditionnelle. Il se compose de deux valeurs qui sont transmises à la pile d'évaluation de script. Pour un ancien client Bitcoin (non compatible avec Segwit), les deux poussées ressembleraient à une sortie que tout le monde peut dépenser et ne nécessite pas de signature (ou plutôt, peut être dépensée avec une signature vide). Pour un client plus récent, compatible avec segwit, le premier nombre (0) est interprété comme un numéro de version (la *version témoin*) et la deuxième partie (20 octets) est l'équivalent d'un script de verrouillage appelé *programme témoin* . Le programme témoin de 20 octets est simplement le hachage de la clé publique, comme dans un script P2PKH.

Maintenant, nous allons regarder à la transaction correspondante que Bob utilise pour passer cette sortie. Pour le script d'origine (nonsegwit), la transaction de Bob devrait inclure une signature dans l'entrée de transaction :

Transaction décodée montrant une sortie P2PKH dépensée avec une signature

```
[...]
"Vin" : [
"txid":
"0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2"
"vout": 0,
      "scriptSig": "<Bob's scriptSig>",
]
[...]
```

Cependant, pour dépenser la sortie de témoin séparé, la transaction n'a pas de signature dans la partie d'entrée. Au lieu de cela, la transaction de Bob a un scriptSig vide dans les données de transaction (la première partie d'une transaction, qui comprend la partie d'entrée) et inclut sa signature dans les données de témoin (la deuxième partie d'une transaction, qui est séparée des données de transaction) :

Transaction décodée montrant une sortie P2WPKH dépensée avec des données témoins distinctes

```
[...]
"Vin" : [
"txid":
"0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2"
"vout": 0,
      "scriptSig": "",
]
[...]
"witness": "<Bob's witness data>"
[...]
```

Construction de portefeuille de P2WPKH

Il est extrêmement important de noter que P2WPKH ne doit être créé que par le bénéficiaire (destinataire) et non converti par l'expéditeur à partir d'une clé publique connue, d'un script P2PKH ou d'une adresse. Le destinataire n'a aucun moyen de savoir si le portefeuille de l'expéditeur a la capacité de construire des transactions segwit et de dépenser des sorties P2WPKH.

En outre, les sorties P2WPKH doivent être construites à partir du hachage d'une clé publique *compressée* . Les clés publiques non compressées ne sont pas standard dans segwit et peuvent être explicitement désactivées par un futur soft fork. Si le hachage utilisé dans le P2WPKH provient d'une clé publique non compressée, il peut ne pas être dépensé et vous risquez de perdre des fonds. Les sorties P2WPKH doivent être créées par le portefeuille du bénéficiaire en dérivant une clé publique compressée à partir de sa clé privée.

Avertissement

P2WPKH doit être construit par le bénéficiaire (destinataire) en convertissant une clé publique compressée en un hachage P2WPKH. Vous ne devez jamais transformer un script P2PKH, une adresse bitcoin ou une clé publique non compressée en un script témoin P2WPKH.

Paiement au témoin-Script-Hash (P2WSH)

Le deuxième type de programme témoin correspond à un script Pay-to-Script-Hash (P2SH). Nous avons vu ce type de script

dans Pay-to-Script-Hash (P2SH) . Dans cet exemple, P2SH a été utilisé par la société de Mohammed pour exprimer un script multisignature. Les paiements à l'entreprise de Mohammed ont été encodés avec un script de verrouillage comme celui-ci :

Exemple de script de sortie P2SH

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

Ce script P2SH fait référence au hachage d'un *script de rachat* qui définit une exigence de signature multiple 2 sur 5 pour dépenser des fonds. Pour dépenser cette sortie, la société de Mohammed présenterait le script de rachat (dont le hachage correspond au hachage du script dans la sortie P2SH) et les signatures nécessaires pour satisfaire ce script de rachat, le tout à l'intérieur de l'entrée de transaction :

Transaction décodée montrant une sortie P2SH dépensée

```
[...]
"Vin" : [
"txid": "abcdef12345...",
"vout": 0,
      "scriptSig": "<SigA> <SigB> <2 PubA PubB PubC
      PubD PubE 5 CHECKMULTISIG>",
]
```

Maintenant, let regard sur la façon dont cet exemple entier serait mis à niveau vers segwit. Si les clients de Mohammed utilisaient un portefeuille compatible segwit, ils effectueraient

un paiement, créant une sortie Pay-to-Witness-Script-Hash (P2WSH) qui ressemblerait à ceci :

Exemple de script de sortie P2WSH

```
0
a9b7b38d972cabc7961dbfbcb841ad4508d133c47ba87457b4a0e8aae86dbb89
```

Encore une fois, comme avec l'exemple de P2WPKH, vous pouvez voir que le script équivalent Segregated Witness est beaucoup plus simple et omet les différents opérandes de script que vous voyez dans les scripts P2SH. Au lieu de cela, le programme Segregated Witness se compose de deux valeurs transmises à la pile : une version témoin (0) et le hachage SHA256 de 32 octets du script de rachat.

La société de Mohammed peut dépenser la sortie P2WSH en présentant le script de rachat correct et des signatures suffisantes pour le satisfaire. Le script de rachat et les signatures seraient séparés en *dehors* des données de transaction de dépenses dans le cadre des données de témoin. Dans l'entrée de transaction, le portefeuille de Mohammed mettrait un script vide

Transaction décodée montrant une sortie P2WSH dépensée avec des données témoins distinctes

```
[...]
"Vin" : [
```

```
"txid": "abcdef12345...",
"vout": 0,
      "scriptSig": "",
]
[...]
"witness": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE
5 CHECKMULTISIG>"
[...]
```

Conseil

Alors que P2SH utilise le hachage RIPEMD160 (SHA256 (script)) de 20 octets, le programme témoin P2WSH utilise un hachage SHA256 (script) de 32 octets. Cette différence dans la sélection de l'algorithme de hachage est délibérée et offre une sécurité renforcée à P2WSH (128 bits de sécurité en P2WSH contre 80 bits de sécurité en P2SH). Il est également utilisé pour différencier les deux types de programmes témoins (P2WPKH et P2WSH) en utilisant la longueur du hachage (voir ci-dessous).

Différenciation entre P2WPKH et P2WSH

Dans les deux sections précédentes, nous avons présenté deux types de programmes de témoins : Pay-to-Witness-Public-Key-Hash (P2WPKH) et Pay-to-Witness-Script-Hash (P2WSH) . Les deux types de programmes témoins consistent en un numéro de version à un octet suivi d'un hachage plus long. Ils ont un aspect

très similaire, mais sont interprétés très différemment : l'un est interprété comme un hachage de clé publique, qui est satisfait par une signature et l'autre comme un hachage de script, qui est satisfait par un script de rachat. La différence critique entre eux est la longueur du hachage :

- Le hachage de la clé publique dans P2WPKH est de 20 octets
- Le hachage du script dans P2WSH est de 32 octets

C'est la seule différence qui permet à un portefeuille de différencier les deux types de programmes de témoins. En regardant la longueur du hachage, un portefeuille peut déterminer de quel type de programme témoin il s'agit, P2WPKH ou P2WSH.

Mise à niveau du Segregated Witness

Comme nous pouvons le voir dans les exemples précédents, la mise à niveau vers Segregated Witness est un processus en deux étapes. Tout d'abord, les portefeuilles doivent créer des sorties de type segwit spéciales. Ensuite, ces sorties peuvent être dépensées par des portefeuilles qui savent comment construire des transactions de témoins séparés. Dans les exemples, le portefeuille d'Alice était compatible avec segwit et capable de créer des sorties spéciales avec des scripts de témoins séparés. Le portefeuille de Bob est également compatible avec Segwit et capable de dépenser ces sorties. Ce qui n'est peut-être pas évident d'après l'exemple, c'est qu'en pratique, le portefeuille d'Alice a besoin de *savoir* que Bob utilise un portefeuille compatible segwit et peut dépenser ces sorties. Sinon, si le portefeuille de Bob n'est pas mis à niveau et qu'Alice tente d'effectuer des paiements segwit à Bob, le portefeuille de Bob

ne pourra pas détecter ces paiements.

Conseil

Pour les types de paiement P2WPKH et P2WSH, les portefeuilles de l'expéditeur et du destinataire doivent être mis à niveau pour pouvoir utiliser segwit. En outre, le portefeuille de l'expéditeur doit savoir que le portefeuille du destinataire est compatible avec Segwit.

Le témoin séparé ne sera pas mis en œuvre simultanément sur l'ensemble du réseau. Au contraire, Segregated Witness est implémenté comme une mise à niveau rétrocompatible, dans laquelle les *anciens et les nouveaux clients peuvent coexister* . Les développeurs de portefeuille mettront à niveau indépendamment le logiciel de portefeuille pour ajouter des fonctionnalités de segwit. Les types de paiement P2WPKH et P2WSH sont utilisés lorsque l'expéditeur et le destinataire sont tous deux conscients de segwit. Le P2PKH et le P2SH traditionnels continueront de fonctionner pour les portefeuilles non mis à niveau. Cela laisse deux scénarios importants, qui sont abordés dans la section suivante :

- Capacité du portefeuille d'un expéditeur qui n'est pas compatible avec Segwit d'effectuer un paiement sur le portefeuille d'un destinataire qui peut traiter des transactions segwit
- Capacité du portefeuille d'un expéditeur qui est conscient de segwit à reconnaître et à distinguer les destinataires qui sont conscients de segwit et ceux qui ne le sont pas, par leurs *adresses* .

Intégration de témoins séparés dans P2SH

Nous allons supposer, par exemple, que le portefeuille d'Alice n'est pas mis à niveau vers segwit, mais le porte - monnaie de Bob est mis à jour et peut gérer les transactions segwit. Alice et Bob peuvent utiliser les « anciennes » transactions non segwit. Mais Bob voudrait probablement utiliser segwit pour réduire les frais de transaction, en profitant de la réduction qui s'applique aux données des témoins.

Dans ce cas, le portefeuille de Bob peut construire une adresse P2SH qui contient un script segwit à l'intérieur. Le portefeuille d'Alice voit cela comme une adresse P2SH "normale" et peut y effectuer des paiements sans aucune connaissance de segwit. Le portefeuille de Bob peut ensuite dépenser ce paiement avec une transaction segwit, en tirant pleinement parti de segwit et en réduisant les frais de transaction.

Les deux formes de scripts témoins, P2WPKH et P2WSH, peuvent être intégrées dans une adresse P2SH. Le premier est noté P2SH (P2WPKH) et le second est noté P2SH (P2WSH).

Pay-to-Witness-Public-Key-Hash dans Pay-to-Script-Hash

La première forme de script témoin que nous examinerons est P2SH (P2WPKH). Il s'agit d'un programme témoin Pay-to-Witness-Public-Key-Hash, intégré dans un script Pay-to-Script-Hash, afin qu'il puisse être utilisé par un portefeuille qui ne connaît pas segwit.

Le portefeuille de Bob construit un programme témoin

P2WPKH avec la clé publique de Bob. Ce programme témoin est ensuite haché et le hachage résultant est codé en tant que script P2SH. Le script P2SH est converti en une adresse bitcoin, qui commence par un "3", comme nous l'avons vu dans la section Pay-to-Script-Hash (P2SH) .

Le portefeuille de Bob commence avec le programme témoin P2WPKH que nous avons vu plus tôt :

Programme de témoins P2WPKH de Bob

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

Le programme témoin P2WPKH comprend la version témoin et le hachage de clé publique de 20 octets de Bob.

Le portefeuille de Bob hache ensuite le programme témoin précédent, d'abord avec SHA256, puis avec RIPEMD160, produisant un autre hachage de 20 octets.

Utilisons bx sur la ligne de commande pour répliquer cela :

HASH160 du programme témoin P2WPKH

```
echo \
'0 [ab68025513c3dbd2f7b92a94e0581f5d50f654e7]'\
 | bx script-encode | bx sha256 | bx ripemd160
3e0547268b3b19288b3adef9719ec8659f4b2b0b
```

Ensuite, le hachage du script de rachat est converti en une adresse bitcoin. Utilisons à nouveau bx sur la ligne de commande :

Adresse P2SH

```
echo \
'3e0547268b3b19288b3adef9719ec8659f4b2b0b' \
| bx address-encode -v 5
37Lx99uaGn5avKBxiW26HjedQE3LrDCZru
```

Maintenant, Bob peut afficher cette adresse pour que les clients paient leur café. Le portefeuille d'Alice peut effectuer un paiement à 37Lx99uaGn5avKBxiW26HjedQE3LrDCZru, comme il le ferait à toute autre adresse bitcoin.

Pour payer Bob, le portefeuille d'Alice verrouillerait la sortie avec un script P2SH :

```
HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL
```

Même si le portefeuille d'Alice ne prend pas en charge segwit, le paiement qu'il crée peut être dépensé par Bob avec une transaction segwit.

Pay-to-Witness-Script-Hash dans Pay-to-Script-Hash

De même, un programme témoin P2WSH pour un script multisig ou un autre script compliqué peut être intégré dans

un script et une adresse P2SH, ce qui permet à n'importe quel portefeuille d'effectuer des paiements compatibles avec segwit.

Comme nous l'avons vu dans Pay-to-Witness-Script-Hash (P2WSH) , la société de Mohammed utilise des paiements de témoins séparés avec des scripts multisignatures. Pour permettre à tout client de payer son entreprise, que son portefeuille soit ou non mis à niveau pour segwit, le portefeuille de Mohammed peut intégrer le programme témoin P2WSH dans un script P2SH.

Tout d'abord, le portefeuille de Mohammed hache le script de rachat avec SHA256 (une seule fois). Utilisons bx pour le faire sur la ligne de commande :

Le portefeuille de Mohammed crée un programme de témoins P2WSH

```
echo \
2 \
[04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA0
\
[04A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1FCBFC2
\
[047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D573781
\
[0421D65CBD7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A
\
[043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A
\
5 CHECKMULTISIG \
| bx script-encode | bx sha256
9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Ensuite, le script de rachat haché est transformé en un programme témoin P2WSH :

```
0
9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

Ensuite, le programme témoin lui-même est haché avec SHA256 et RIPEMD160, produisant un nouveau hachage de 20 octets, comme utilisé dans le P2SH traditionnel. Utilisons bx sur la ligne de commande pour ce faire :

Le HASH160 du programme témoin P2WSH

```
echo \
'0
[9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
 | bx script-encode | bx sha256 | bx ripemd160
86762607e8fe87c0c37740cddee880988b9455b2
```

Ensuite, le portefeuille construit une adresse bitcoin P2SH à partir de ce hachage. Encore une fois, nous utilisons bx pour calculer sur la ligne de commande :

Adresse bitcoin P2SH

```
echo \
'86762607e8fe87c0c37740cddee880988b9455b2'\
 | bx address-encode -v 5
3Dwz1MXhM6EfFoJChHCxh1jWHb8GQqRenG
```

Désormais, les clients de Mohammed peuvent effectuer des paiements à cette adresse sans avoir besoin de prendre en charge segwit. Pour envoyer un paiement à Mohammed, un portefeuille verrouillerait la sortie avec le script P2SH suivant :

Script P2SH utilisé pour verrouiller les paiements sur le multi-sig de Mohammed

```
HASH160 86762607e8fe87c0c37740cddee880988b9455b2 EQUAL
```

La société de Mohammed peut ensuite construire des transactions segwit pour dépenser ces paiements, en tirant parti des fonctionnalités de segwit, y compris des frais de transaction moins élevés.

Adresses des témoins séparés

Même après l'activation de Segwit, la mise à niveau de la plupart des portefeuilles prendra un certain temps. Dans un premier temps, segwit sera intégré dans P2SH, comme nous l'avons vu dans la section précédente, pour faciliter la compatibilité entre les portefeuilles segwit et inconscients.

Cependant, une fois que les portefeuilles prennent largement en charge segwit, il est logique d'encoder les scripts témoins directement dans un format d'adresse natif conçu pour segwit, plutôt que de les intégrer dans P2SH.

Le format d'adresse segwit natif est défini dans BIP-173 :

BIP-173

Format d'adresse Base32 pour les sorties témoins natives v0-16

BIP-173 encode uniquement les scripts témoins (P2WPKH et P2WSH). Il n'est pas compatible avec les scripts P2PKH ou P2SH non segwit. BIP-173 est un encodage Base32 avec somme de contrôle, par rapport à l'encodage Base58 d'une adresse bitcoin "traditionnelle". Les adresses BIP-173 sont également appelées adresses *bech32* , prononcées "beh-ch trente-deux ", faisant allusion à l'utilisation d'un algorithme de détection d'erreur "BCH" et d'un jeu de codage à 32 caractères.

Les adresses BIP-173 utilisent un jeu de 32 caractères alphanumériques en minuscules uniquement, soigneusement sélectionnés pour réduire les erreurs de lecture ou de frappe. En choisissant un jeu de caractères en minuscules uniquement, bech32 est plus facile à lire, à parler et 45% plus efficace à encoder dans les codes QR.

L'algorithme de détection d'erreur BCH est une grande amélioration par rapport à l'algorithme de somme de contrôle précédent (de Base58Check), permettant non seulement la détection mais également la *correction* des erreurs. Les interfaces de saisie d'adresse (telles que les champs de texte dans les formulaires) peuvent détecter et mettre en évidence le caractère le plus probablement mal saisi lorsqu'elles détectent une erreur.

À partir de la spécification BIP-173, voici quelques exemples d'adresses bech32 :

Mainnet P2WPKH

bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4

Testnet P2WPKH

tb1qw508d6qejxtdg4y5r3zarvary0c5xw7kxpjzsx

Mainnet P2WSH

bc1qrp33g0q5c5txsp9arysrx4k6zdkfs4nce4xj0gdcccefvpysxf3qccfmv3

Testnet P2WSH

tb1qrp33g0q5c5txsp9arysrx4k6zdkfs4nce4xj0gdcccefvpysxf3q0sl5k7

Comme vous pouvez le voir dans ces exemples, une chaîne segwit bech32 comprend jusqu'à 90 caractères et se compose de trois parties :

La partie lisible par l'homme

Ce préfixe "bc" ou "tb" identifie le réseau principal ou le réseau de test

Le séparateur

Le chiffre "1", qui ne fait pas partie du jeu de codage à 32 caractères et ne peut apparaître dans cette position que comme séparateur

La partie données

Un minimum de 6 caractères alphanumériques, le script témoin encodé par la somme de contrôle

À l'heure actuelle, seuls quelques portefeuilles acceptent ou produisent des adresses segwit bech32 natives, mais à mesure que l'adoption de segwit augmente, vous les verrez de plus en plus souvent.

Les adresses Bitcoin non-segwit (héritées) et segwit affichent les adresses bitcoin non-segwit (héritées) et segwit.

Type	Encoding	Prefix
Legacy P2PKH Address	Base58	1
Legacy Testnet P2PKH Address	Base58	m or n
Legacy P2SH Address	Base58	3
Legacy Testnet P2SH Address	Base58	2
Nested (embedded) Segwit P2SH(P2WPKH) Address	Base58	3
Nested (embedded) Segwit P2SH(P2WSH) Address	Base58	3
Native Segwit P2WPKH Address	Bech32	bc1
Native Segwit Testnet P2WPKH Address	Bech32	tb1
Native Segwit P2WSH Address	Bech32	bc1
Native Segwit Testnet P2WSH Address	Bech32	tb1

Tableau 3. Adresses Bitcoin non-segwit (héritées) et segwit

Identifiants de transaction

L'un des plus grands avantages de Segregated Witness est qu'il élimine la malléabilité des transactions tierces.

Avant segwit, les transactions pouvaient voir leurs signatures subtilement modifiées par des tiers, en changeant leur ID de transaction (hachage) sans changer les propriétés fondamentales (entrées, sorties, montants). Cela a créé des opportunités d'attaques par déni de service ainsi que d'attaques contre des logiciels de portefeuille mal écrits qui supposaient que les hachages de transaction non confirmés étaient immuables.

Avec l'introduction de Segregated Witness, les transactions ont deux identifiants, txid et wtxid. L'ID de transaction traditionnel txid est le double hachage SHA256 de la transaction sérialisée, sans les données de témoin. Une transaction wtxid est le double hachage SHA256 du nouveau format de sérialisation de la transaction avec les données témoins.

Le txid traditionnel est calculé exactement de la même manière qu'avec une transaction nonsegwit. Cependant, comme une transaction segwit pure (une transaction qui ne contient que des entrées segwit) a des scriptSigs vides dans chaque entrée, aucune partie de la transaction ne peut être modifiée par un tiers. Par conséquent, dans une transaction segwit pure, le txid est immuable par un tiers, même lorsque la transaction n'est pas confirmée.

Le wtxid est comme un ID "étendu", en ce que le hachage incorpore également les données témoins. Si une transaction

est transmise sans données de témoin, alors le wtxid et le txid sont identiques. Notez que puisque le wtxid inclut des données de témoin (signatures) et que les données de témoin peuvent être malléables, le wtxid doit être considéré comme malléable jusqu'à ce que la transaction soit confirmée. Seul le txid d'une transaction segwit pure peut être considéré comme immuable par des tiers.

> *Conseil*
>
> *Les transactions de témoins séparés ont deux ID : txid et wtxid. Le txid est le hachage de la transaction sans les données du témoin et le wtxid est le hachage incluant les données du témoin. Seules les transactions segwit pures (transactions qui ne contiennent que des entrées segwit) ont un txid qui n'est pas sensible à la malléabilité des transactions tierces.*

Nouvel algorithme de signature des témoins séparés

Segregated Witness modifie la sémantique des quatre fonctions de vérification de signature (CHECKSIG, CHECKSIGVERIFY, CHECKMULTISIG et CHECKMULTISIGVERIFY), en changeant la façon dont un hachage d'engagement de transaction est calculé.

Les signatures dans les transactions Bitcoin sont appliquées sur un *hachage d'engagement* , qui est calculé à partir des données de transaction, verrouillant des parties spécifiques des données indiquant l'engagement du signataire envers ces valeurs. Par exemple, dans une simple signature de type SIGHASH_ALL, le

hachage d'engagement inclut toutes les entrées et sorties.

Malheureusement, la façon dont le hachage d'engagement a été calculé a introduit la possibilité qu'un nœud vérifiant la signature puisse être contraint d'effectuer un nombre important de calculs de hachage. Plus précisément, les opérations de hachage augmentent en O (n 2) par rapport au nombre d'opérations de signature dans la transaction. Un attaquant pourrait donc créer une transaction avec un très grand nombre d'opérations de signature, obligeant l'ensemble du réseau bitcoin à effectuer des centaines ou des milliers d'opérations de hachage pour vérifier la transaction.

Segwit a représenté une opportunité de résoudre ce problème en modifiant la façon dont le hachage d'engagement est calculé. Pour les programmes témoins de la version 0 de segwit, la vérification de la signature se produit à l'aide d'un algorithme de hachage d'engagement amélioré comme spécifié dans BIP-143.

Le nouvel algorithme atteint deux objectifs importants. Premièrement, le nombre d'opérations de hachage augmente d'un O (n) beaucoup plus graduel au nombre d'opérations de signature, ce qui réduit la possibilité de créer des attaques par déni de service avec des transactions trop complexes. Deuxièmement, le hachage de l'engagement inclut désormais également la valeur (les montants) de chaque entrée dans le cadre de l'engagement. Cela signifie qu'un signataire peut s'engager sur une valeur d'entrée spécifique sans avoir besoin de "récupérer" et de vérifier la transaction précédente référencée par l'entrée. Dans le cas d'appareils hors ligne, tels que les portefeuilles matériels, cela simplifie considérablement la communication entre l'hôte

et le portefeuille matériel, supprimant la nécessité de diffuser les transactions précédentes pour validation. Un portefeuille matériel peut accepter la valeur d'entrée « comme indiqué » par un hôte non approuvé. Étant donné que la signature n'est pas valide si cette valeur d'entrée n'est pas correcte, le portefeuille matériel n'a pas besoin de valider la valeur avant de signer l'entrée.

Incitations économiques pour les témoins isolés

Les nœuds de minage Bitcoin et les nœuds complets entraînent des coûts pour les ressources utilisées pour prendre en charge le réseau Bitcoin et la blockchain. À mesure que le volume des transactions Bitcoin augmente, le coût des ressources (CPU, bande passante réseau, espace disque, mémoire) augmente également. Les mineurs sont compensés pour ces coûts par des frais proportionnels à la taille (en octets) de chaque transaction. Les nœuds complets non miniers ne sont pas compensés, ils supportent donc ces coûts car ils ont besoin d'exécuter un nœud d'index complet entièrement validé faisant autorité, peut-être parce qu'ils utilisent le nœud pour exploiter une entreprise Bitcoin.

Sans frais de transaction, la croissance des données Bitcoin augmenterait sans doute considérablement. Les frais visent à aligner les besoins des utilisateurs de Bitcoin sur le fardeau que leurs transactions imposent au réseau, grâce à un mécanisme de découverte des prix basé sur le marché.

Le calcul des frais en fonction de la taille de la transaction traite toutes les données de la transaction comme égales en coût. Mais

du point de vue des nœuds complets et des mineurs, certaines parties d'une transaction entraînent des coûts beaucoup plus élevés. Chaque transaction ajoutée au réseau bitcoin affecte la consommation de quatre ressources sur les nœuds :

Espace disque

Chaque transaction est stockée dans la blockchain, ajoutant à la taille totale de la blockchain. La blockchain est stockée sur disque, mais le stockage peut être optimisé en « élaguant » (en supprimant) les anciennes transactions.

CPU

Chaque transaction doit être validée, ce qui nécessite du temps CPU.

Bande passante

Chaque transaction est transmise (par propagation d'inondation) sur le réseau au moins une fois. Sans aucune optimisation du protocole de propagation de bloc, les transactions sont à nouveau transmises dans le cadre d'un bloc, doublant ainsi l'impact sur la capacité du réseau.

Mémoire

Les nœuds qui valident les transactions conservent l'index UTXO ou l'ensemble UTXO entier en mémoire pour accélérer la validation. La mémoire étant au moins un ordre de grandeur plus chère que le disque, la croissance de l'ensemble UTXO contribue de manière disproportionnée au coût de fonctionnement d'un nœud.

Comme vous pouvez le voir dans la liste, toutes les parties d'une

transaction n'ont pas un impact égal sur le coût d'exécution d'un nœud ou sur la capacité du bitcoin à évoluer pour prendre en charge plus de transactions. La partie la plus chère d'une transaction sont les sorties nouvellement créées, car elles sont ajoutées à l'ensemble UTXO en mémoire. En comparaison, les signatures (c'est-à-dire les données témoins) ajoutent le moins de charge au réseau et le coût d'exécution d'un nœud, car les données témoins ne sont validées qu'une seule fois et ne sont plus jamais utilisées. En outre, immédiatement après la réception d'une nouvelle transaction et la validation des données de témoin, les nœuds peuvent supprimer ces données de témoin. Si les frais sont calculés sur la taille de la transaction, sans faire de distinction entre ces deux types de données, alors les incitations du marché des frais ne sont pas alignées sur les coûts réels imposés par une transaction. En fait, la structure actuelle des frais encourage en fait le comportement inverse, car les données des témoins constituent la plus grande partie d'une transaction.

Les incitations créées par les frais sont importantes car elles affectent le comportement des portefeuilles. Tous les portefeuilles doivent mettre en œuvre une stratégie pour assembler des transactions qui prend en compte un certain nombre de facteurs, tels que la confidentialité (réduire la réutilisation des adresses), la fragmentation (effectuer beaucoup de modifications lâches) et les frais. Si les frais motivent massivement les portefeuilles à utiliser le moins d'entrées possible dans les transactions, cela peut conduire à des stratégies de sélection UTXO et de changement d'adresse qui gonflent par inadvertance l'ensemble UTXO.

Les transactions consomment UTXO dans leurs entrées et créent de nouveaux UTXO avec leurs sorties. Une transaction, par conséquent, qui a plus d'entrées que de sorties entraînera une diminution de l'ensemble UTXO, tandis qu'une transaction qui a plus de sorties que d'entrées entraînera une augmentation de l'ensemble UTXO. Examinons la *différence* entre les entrées et les sorties et appelons cela le "Net-new-UTXO". Il s'agit d' une mesure importante, car elle nous indique l'impact d'une transaction sur la ressource la plus chère du réseau, l'ensemble UTXO en mémoire. Une transaction avec Net-new-UTXO positif ajoute à ce fardeau. Une transaction avec un Net-new-UTXO négatif réduit la charge. Nous souhaitons donc encourager les transactions soit négatives Net-new-UTXO, soit neutres avec zéro Net-new-UTXO.

Prenons un exemple des incitations créées par le calcul des frais de transaction, avec et sans témoin distinct. Nous examinerons deux transactions différentes. La transaction A est une transaction à 3 entrées et 2 sorties, qui a une métrique Net-new-UTXO de –1, ce qui signifie qu'elle consomme un UTXO de plus qu'elle n'en crée, réduisant l'UTXO défini de un. La transaction B est une transaction à 2 entrées et 3 sorties, qui a une métrique Net-new-UTXO de 1, ce qui signifie qu'elle ajoute un UTXO à l'ensemble UTXO, imposant un coût supplémentaire sur l'ensemble du réseau Bitcoin. Les deux transactions utilisent des scripts multisignatures (2 sur 3) pour démontrer comment des scripts complexes augmentent l'impact des témoins séparés sur les honoraires. Nous allons supposer une feerate de transaction de 30 Satoshi par octet et une réduction de taxe de 75% sur les données du témoin :

Sans témoin séparé

Frais de transaction A : 28590 satoshi
Frais de transaction B : 20760 satoshi

Avec témoin séparé

Frais de transaction A : 12255 satoshi
Frais de transaction B : 10,425 satoshi

Les deux transactions sont moins coûteuses lorsqu'un témoin séparé est mis en œuvre. En comparant les coûts entre les deux transactions, nous voyons qu'avant Segregated Witness, la transaction avec le Net-new-UTXO positif a des économies de coûts significatives. Avec Segregated Witness, la différence de coût se réduit considérablement en termes absolus et relatifs. Alors qu'il faudrait que les intrants deviennent moins chers que les extrants pour inciter à la consolidation des ensembles UTXO, cette remise réduit l'incitation à créer de nouveaux UTXO afin d' éviter d'utiliser plus d'intrants.

Segregated Witness a donc deux effets principaux sur les frais payés par les utilisateurs de Bitcoin. Premièrement, segwit réduit le coût global des transactions en actualisant les données des témoins et en augmentant la capacité de la blockchain Bitcoin. Deuxièmement, la réduction de segwit sur les données des témoins atténue en partie un désalignement des incitations qui peut avoir créé par inadvertance plus de ballonnement dans l'ensemble UTXO.

9

Le réseau Bitcoin

Architecture de réseau peer-to-peer

Bitcoin est structuré comme une architecture de réseau peer-to-peer au-dessus d'Internet. Le terme peer-to-peer, ou P2P, signifie que les ordinateurs qui participent au réseau sont pairs les uns aux autres, qu'ils sont tous égaux, qu'il n'y a pas de nœuds "spéciaux" et que tous les nœuds partagent la charge de fournir le réseau. Les nœuds de réseau s'interconnectent dans un réseau maillé avec une topologie "plate". Il n'y a pas de serveur, pas de service centralisé et pas de hiérarchie au sein du réseau. Les nœuds d'un réseau P2P fournissent et consomment des services en même temps, la réciprocité agissant comme une incitation à la participation. Les réseaux P2P sont intrinsèquement résilients, décentralisés et ouverts. Un exemple prééminent d'architecture de réseau P2P était le premier Internet lui-même, où les nœuds sur le réseau IP étaient égaux. L'architecture Internet d'aujourd'hui est plus

hiérarchique, mais le protocole Internet conserve toujours son essence de topologie plate. Au-delà du bitcoin, l'application la plus importante et la plus réussie des technologies P2P est le partage de fichiers, avec Napster comme pionnier et BitTorrent comme l'évolution la plus récente de l'architecture.

L'architecture réseau P2P de Bitcoin est bien plus qu'un choix de topologie. Bitcoin est un système de paiement numérique P2P de par sa conception, et l'architecture du réseau est à la fois le reflet et le fondement de cette caractéristique fondamentale. La décentralisation du contrôle est un principe de conception de base qui ne peut être atteint et maintenu que par un réseau de consensus P2P plat et décentralisé.

Le terme « réseau bitcoin » fait référence à l'ensemble de nœuds exécutant le protocole Bitcoin P2P. En plus du protocole Bitcoin P2P, il existe d'autres protocoles tels que Stratum qui sont utilisés pour l'exploitation minière et les portefeuilles légers ou mobiles. Ces protocoles supplémentaires sont fournis par des serveurs de routage de passerelle qui accèdent au réseau bitcoin à l'aide du protocole bitcoin P2P, puis étendent ce réseau aux nœuds exécutant d'autres protocoles. Par exemple, les serveurs Stratum connectent les nœuds de minage Stratum via le protocole Stratum au réseau bitcoin principal et relient le protocole Stratum au protocole Bitcoin P2P. Nous utilisons le terme « réseau bitcoin étendu » pour désigner le réseau global qui comprend le protocole bitcoin P2P, les protocoles de pool-mining, le protocole Stratum et tout autre protocole associé connectant les composants du système bitcoin.

Types de nœuds et rôles

Bien que les nœuds du réseau Bitcoin P2P soient égaux, ils peuvent jouer des rôles différents en fonction de la fonctionnalité qu'ils prennent en charge. Un nœud bitcoin est un ensemble de fonctions : le routage, la base de données blockchain, l'exploitation minière et les services de portefeuille. Un nœud complet avec ces quatre fonctions est affiché dans Un nœud de réseau bitcoin avec les quatre fonctions : portefeuille, mineur, base de données blockchain complète et routage réseau (fig1)

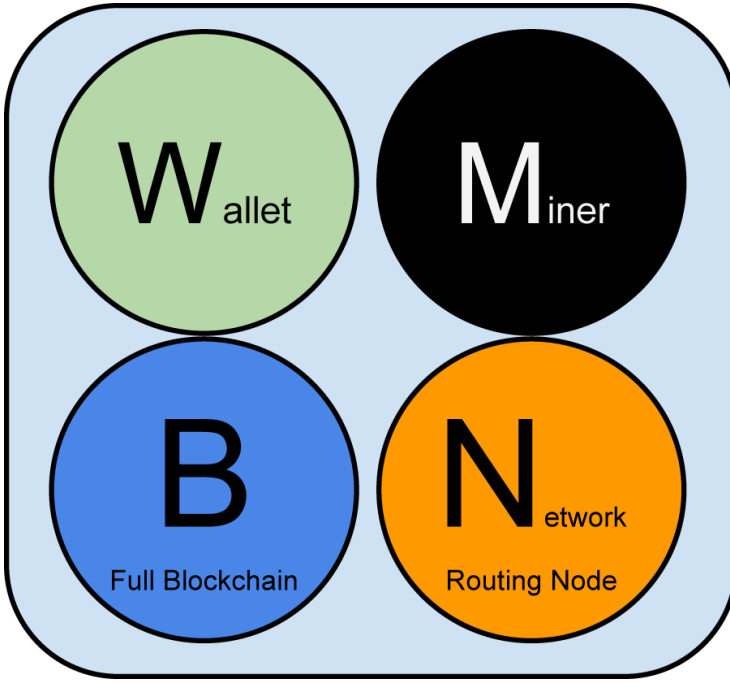


Figure 1. Un nœud de réseau bitcoin avec les quatre fonctions : portefeuille, mineur, base de données blockchain complète et routage réseau

Tous les nœuds incluent la fonction de routage pour participer au réseau et peuvent inclure d'autres fonctionnalités. Tous les nœuds valident et propagent les transactions et les blocs, et découvrent et maintiennent les connexions aux pairs. Dans l'exemple de nœud complet dans Un nœud de réseau bitcoin avec les quatre fonctions : portefeuille, mineur, base de données blockchain complète et routage réseau , la fonction de routage est indiquée par un cercle nommé "Nœud de routage

réseau" ou par la lettre "N".

Certains nœuds, appelés nœuds complets, conservent également une copie complète et à jour de la blockchain. Les nœuds complets peuvent vérifier de manière autonome et autoritaire toute transaction sans référence externe. Certains nœuds ne gèrent qu'un sous-ensemble de la blockchain et vérifient les transactions à l'aide d'une méthode appelée *vérification de paiement simplifiée* , ou SPV. Ces nœuds sont appelés nœuds SPV ou nœuds légers. Dans l'exemple de nœud complet de la figure, la fonction de base de données de blockchain à nœud complet est indiquée par un cercle appelé « Full Blockchain » ou la lettre « B ». Dans Le réseau bitcoin étendu montrant divers types de nœuds, passerelles et protocoles , les nœuds SPV sont dessinés sans le cercle « B », ce qui montre qu'ils ne disposent pas d'une copie complète de la blockchain.

Les nœuds de minage se font concurrence pour créer de nouveaux blocs en exécutant du matériel spécialisé pour résoudre l'algorithme de preuve de travail. Certains nœuds de minage sont également des nœuds complets, conservant une copie complète de la blockchain, tandis que d'autres sont des nœuds légers participant à l'extraction de pool et dépendant d'un serveur de pool pour maintenir un nœud complet. La fonction minière est affichée dans le nœud complet sous la forme d'un cercle appelé "Mineur" ou la lettre "M."

Les portefeuilles utilisateur peuvent faire partie d'un nœud complet, comme c'est généralement le cas avec les clients bitcoin de bureau. De plus en plus, de nombreux portefeuilles d'utilisateurs, en particulier ceux exécutés sur des appareils à

ressources limitées tels que les smartphones, sont des nœuds SPV. La fonction de portefeuille est affichée dans Un nœud de réseau bitcoin avec les quatre fonctions : portefeuille, mineur, base de données blockchain complète et routage réseau sous la forme d'un cercle appelé "Wallet" ou la lettre "W."

En plus des principaux types de nœuds sur le protocole Bitcoin P2P, il existe des serveurs et des nœuds exécutant d'autres protocoles, tels que des protocoles de pool de minage spécialisés et des protocoles d'accès client légers.

Différents types de nœuds sur le réseau bitcoin étendu montrent les types de nœuds les plus courants sur le réseau bitcoin étendu.

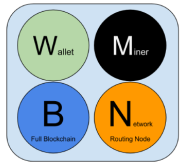
Le réseau Bitcoin étendu

Le réseau bitcoin principal, exécutant le protocole Bitcoin P2P, se compose de 5000 à 8000 nœuds d'écoute exécutant différentes versions du client de référence Bitcoin (Bitcoin Core) et de quelques centaines de nœuds exécutant diverses autres implémentations du protocole Bitcoin P2P, telles que Bitcoin Classic , Bitcoin Unlimited, BitcoinJ , Libbitcoin , btcd et bcoin . Un petit pourcentage des nœuds sur le réseau Bitcoin P2P sont également des nœuds miniers, en concurrence dans le processus de minage, en validant les transactions et en créant de nouveaux blocs. Diverses grandes entreprises s'interfacent avec le réseau Bitcoin en exécutant des clients à nœud complet basés sur le client Bitcoin Core, avec des copies complètes de la blockchain et d'un nœud de réseau, mais sans fonctions

d'extraction ou de portefeuille. Ces nœuds agissent comme des routeurs de périphérie du réseau, permettant à divers autres services (échanges, portefeuilles, explorateurs de blocs, traitement des paiements marchands) d'être construits par-dessus.

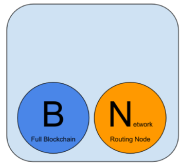
Le réseau bitcoin étendu comprend le réseau exécutant le protocole Bitcoin P2P, décrit précédemment, ainsi que des nœuds exécutant des protocoles spécialisés. Un certain nombre de serveurs de pool et de passerelles de protocole sont attachés au réseau principal Bitcoin P2P qui connectent des nœuds exécutant d'autres protocoles. Ces autres nœuds de protocole sont principalement des nœuds d'extraction de pool (voir [extraction]) et des clients de portefeuille légers, qui ne portent pas une copie complète de la blockchain.

Le réseau Bitcoin étendu montrant divers types de nœuds, passerelles et protocoles montre le réseau Bitcoin étendu avec les différents types de nœuds, serveurs de passerelle, routeurs de périphérie et clients de portefeuille et les divers protocoles qu'ils utilisent pour se connecter les uns aux autres.



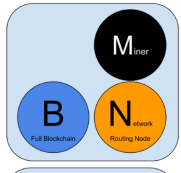
Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



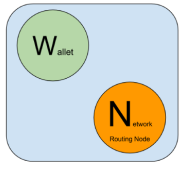
Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.



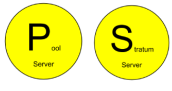
Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.



Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.

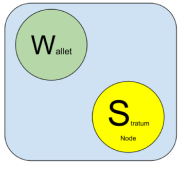


Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.

Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.



Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

Figure 2. Différents types de nœuds sur le réseau bitcoin étendu

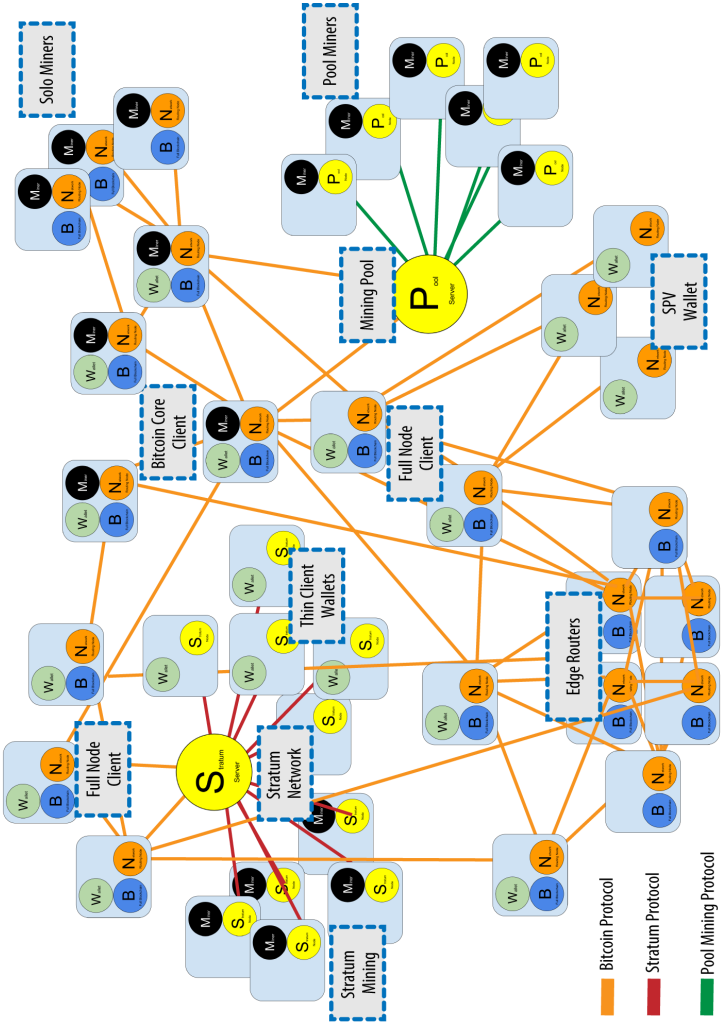


Figure 3. Le réseau Bitcoin étendu montrant divers types de nœuds, passerelles et protocoles

Réseaux de relais Bitcoin

Alors que le réseau Bitcoin P2P répond aux besoins généraux d'une grande variété de types de nœuds, il présente une latence de réseau trop élevée pour les besoins spécialisés des nœuds de minage Bitcoin.

Les mineurs de Bitcoin sont engagés dans une compétition urgente pour résoudre le problème de la preuve de travail et étendre la blockchain (voir [exploitation minière]). En participant à cette compétition, les mineurs de bitcoin doivent minimiser le temps entre la propagation d'un bloc gagnant et le début du prochain tour de compétition. Dans le secteur minier, la latence du réseau est directement liée aux marges bénéficiaires.

Un *réseau de relais Bitcoin* est un réseau qui tente de minimiser la latence dans la transmission des blocs entre les mineurs. Le réseau de relais Bitcoin original a été créé par le développeur principal Matt Corallo en 2015 pour permettre une synchronisation rapide des blocs entre les mineurs avec une latence très faible. Le réseau se composait de plusieurs nœuds spécialisés hébergés sur l'infrastructure Amazon Web Services à travers le monde et servait à connecter la majorité des mineurs et des pools de minage.

Le réseau de relais Bitcoin d'origine a été remplacé en 2016 avec l'introduction du *moteur de relais Bitcoin Fast Internet* ou *FIBRE* , également créé par le développeur principal Matt Corallo . FIBRE est un réseau de relais basé sur UDP qui relaie les

blocs au sein d'un réseau de nœuds. FIBER met en œuvre *une* optimisation de *bloc compact* pour réduire davantage la quantité de données transmises et la latence du réseau.

Un autre réseau de relais (toujours en phase de proposition) est *Falcon* , basé sur des recherches à l'Université Cornell. Falcon utilise le "cut-through-routing" au lieu de "store-and-forward" pour réduire la latence en propageant des parties de blocs au fur et à mesure de leur réception plutôt que d'attendre la réception d'un bloc complet.

Les réseaux de relais ne remplacent pas le réseau P2P de Bitcoin. Au lieu de cela, ce sont des réseaux superposés qui fournissent une connectivité supplémentaire entre les nœuds ayant des besoins spécialisés. Comme les autoroutes ne remplacent pas les routes rurales, mais plutôt des raccourcis entre deux points à fort trafic, vous avez toujours besoin de petites routes pour vous connecter aux autoroutes.

Découverte du réseau

Lorsqu'un nouveau nœud démarre, il doit découvrir d'autres nœuds bitcoin sur le réseau pour participer. Pour démarrer ce processus, un nouveau nœud doit découvrir au moins un nœud existant sur le réseau et s'y connecter. L'emplacement géographique des autres nœuds n'est pas pertinent ; la topologie du réseau bitcoin n'est pas définie géographiquement. Par conséquent, tous les nœuds bitcoin existants peuvent être sélectionnés au hasard.

Pour se connecter à un homologue connu, les nœuds établissent une connexion TCP, généralement au port 8333 (le port généralement connu comme celui utilisé par bitcoin), ou un autre port si celui-ci est fourni. Lors de l'établissement d'une connexion, le nœud commencera une "prise de contact" (voir La prise de contact initiale entre pairs) en transmettant un message de version, qui contient des informations d'identification de base, notamment :

nVersion
La version du protocole Bitcoin P2P que le client « parle » (par exemple, 70002)

nLocalServices
Une liste des services locaux pris en charge par le nœud, actuellement uniquement NODE_ NETWORK

nTime
L'heure actuelle

addrYou
L'adresse IP du nœud distant vue de ce nœud

addrMe
L'adresse IP du nœud local, telle que découverte par le nœud local

subver
Une sous-version montrant le type de logiciel fonctionnant sur ce nœud (par exemple, /Satoshi :0.9.2.1/)

BestHeight
La hauteur de bloc de la blockchain de ce nœud

(Voir GitHub pour un exemple du message réseau de version.)

Le message de version est toujours le premier message envoyé par un pair à un autre pair. L'homologue local recevant un message de version examinera la nVersion signalée par l'homologue distant et décidera si l'homologue distant est compatible. Si l'homologue distant est compatible, l'homologue local accusera réception du message de version et établira une connexion en envoyant un message verack .

Comment un nouveau nœud trouve-t-il des pairs ? La première méthode consiste à interroger le DNS à l'aide d' un certain nombre de « graines DNS », qui sont des serveurs DNS qui fournissent une liste d'adresses IP de nœuds bitcoin. Certaines de ces graines DNS fournissent une liste statique d'adresses IP de nœuds d'écoute Bitcoin stables. Certaines des graines DNS sont des implémentations personnalisées de BIND (Berkeley Internet Name Daemon) qui renvoient un sous-ensemble aléatoire à partir d'une liste d'adresses de nœuds bitcoin collectées par un robot d'exploration ou un nœud bitcoin de longue durée. Le client Bitcoin Core contient les noms de neuf graines DNS différentes. La diversité de propriété et la diversité de mise en œuvre des différentes semences DNS offrent un haut niveau de fiabilité pour le processus d'amorçage initial. Dans le client Bitcoin Core, l'option d'utiliser les graines DNS est contrôlée par le commutateur d'option - dnsseed (défini sur 1 par défaut, pour utiliser la graine DNS).

Alternativement, un nœud d'amorçage qui ne sait rien du réseau doit recevoir l'adresse IP d'au moins un nœud bitcoin, après quoi il peut établir des connexions grâce à d'autres introductions. L'argument de ligne de commande - seednode peut être utilisé pour se connecter à un nœud juste pour les introductions en l'utilisant comme graine. Une fois que le nœud d'amorçage initial a été utilisé pour former des introductions, le client se déconnectera et utilisera les pairs nouvellement découverts.

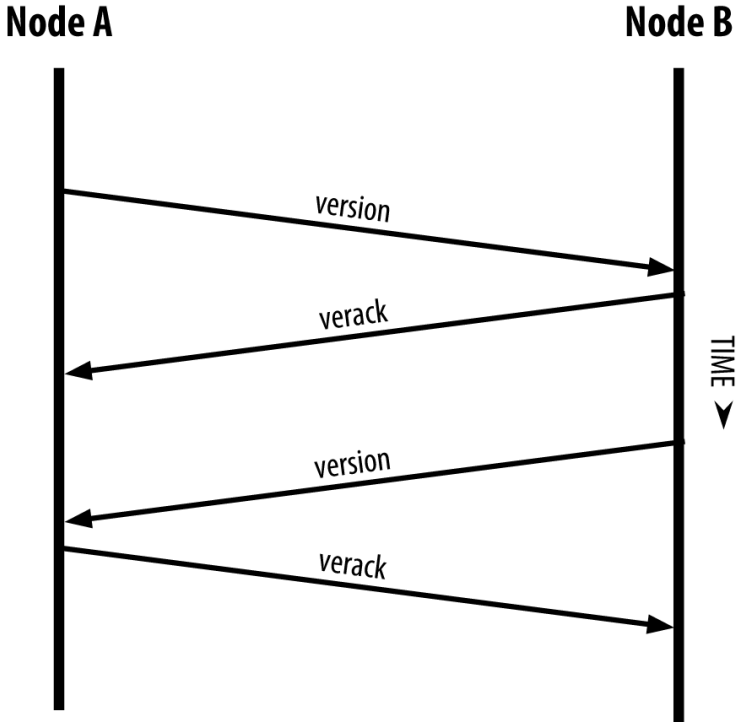


Figure 4. The initial handshake between peers

Une fois qu'une ou plusieurs connexions sont établies, le nouveau nœud enverra un message addr contenant sa propre adresse IP à ses voisins. Les voisins, à leur tour, transmettront le message addr à leurs voisins, s'assurant que le nœud nouvellement connecté devienne bien connu et mieux connecté. De plus, le nœud nouvellement connecté peut envoyer getaddr aux voisins, leur demandant de renvoyer une liste d'adresses IP d'autres pairs. De cette façon, un nœud peut trouver des pairs auxquels se connecter et annoncer son existence sur le réseau pour que d'autres nœuds le trouvent. La propagation et la découverte d'adresses affichent le protocole de découverte d'adresses.

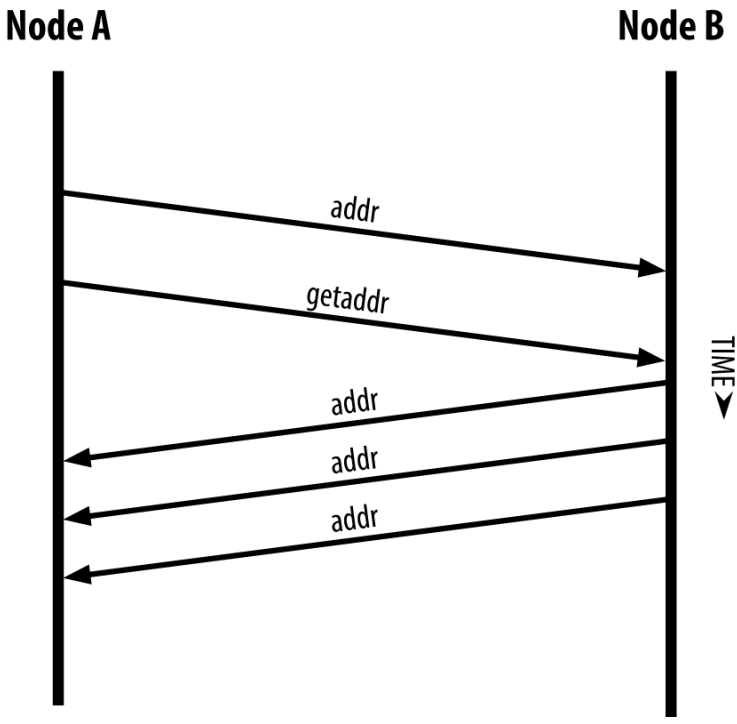


Figure 5. Propagation et découverte d'adresses

Un nœud doit se connecter à quelques pairs différents afin d'établir divers chemins dans le réseau Bitcoin. Les chemins ne sont pas persistants - les nœuds vont et viennent - et le nœud doit donc continuer à découvrir de nouveaux nœuds car il perd les anciennes connexions et assister les autres nœuds lors de leur démarrage. Une seule connexion est nécessaire pour amorcer, car le premier nœud peut offrir des introductions à ses nœuds homologues et ces pairs peuvent proposer d'autres introductions. Il est également inutile et gaspilleur de ressources

réseau de se connecter à plus d'une poignée de nœuds. Après l'amorçage, un nœud se souviendra de ses dernières connexions homologues réussies, de sorte qu'en cas de redémarrage, il puisse rapidement rétablir les connexions avec son ancien réseau homologue. Si aucun des anciens homologues ne répond à sa demande de connexion, le nœud peut utiliser les nœuds d'amorçage pour redémarrer.

Sur un nœud exécutant le client Bitcoin Core, vous pouvez lister les connexions homologues avec la commande getpeerinfo :

```
$ bitcoin-cli getpeerinfo
```

```
[
    {
        "addr" : "85.213.199.39:8333",
        "services" : "00000001",
        "lastsend" : 1405634126,
        "lastrecv" : 1405634127,
        "bytessent" : 23487651,
        "bytesrecv" : 138679099,
        "conntime" : 1405021768,
        "pingtime" : 0.00000000,
        "version" : 70002,
        "subver" : "/Satoshi:0.9.2.1/",
        "inbound" : false,
        "startingheight" : 310131,
        "banscore" : 0,
        "syncnode" : true
```

```
    },
    {
        "addr" : "58.23.244.20:8333",
        "services" : "00000001",
        "lastsend" : 1405634127,
        "lastrecv" : 1405634124,
        "bytessent" : 4460918,
        "bytesrecv" : 8903575,
        "conntime" : 1405559628,
        "pingtime" : 0.00000000,
        "version" : 70001,
        "subver" : "/Satoshi:0.8.6/",
        "inbound" : false,
        "startingheight" : 311074,
        "banscore" : 0,
        "syncnode" : false
    }
]
```

Pour remplacer la gestion automatique des pairs et pour spécifier une liste d'adresses IP, les utilisateurs peuvent fournir l'option -connect = < IPAddress > et spécifier une ou plusieurs adresses IP. Si cette option est utilisée, le nœud se connectera uniquement aux adresses IP sélectionnées, au lieu de découvrir et de maintenir automatiquement les connexions homologues.

S'il n'y a pas de trafic sur une connexion, les nœuds enverront périodiquement un message pour maintenir la connexion. Si un nœud n'a pas communiqué sur une connexion pendant plus de 90 minutes, il est supposé être déconnecté et un nouveau pair sera recherché. Ainsi, le réseau s'adapte dynamiquement aux nœuds transitoires et aux problèmes de réseau , et peut

croître et se réduire de manière organique selon les besoins sans aucun contrôle central.

Nœuds complets

Les nœuds complets sont des nœuds qui maintiennent une blockchain complète avec toutes les transactions. Plus précisément, ils devraient probablement être appelés « nœuds de blockchain complets ». Dans les premières années de Bitcoin, tous les nœuds étaient des nœuds complets et actuellement, le client Bitcoin Core est un nœud de blockchain complet. Au cours des deux dernières années, cependant, de nouvelles formes de clients Bitcoin ont été introduites qui ne maintiennent pas une blockchain complète mais fonctionnent comme des clients légers. Nous les examinerons plus en détail dans la section suivante.

Les nœuds de blockchain complets conservent une copie complète et à jour de la blockchain Bitcoin avec toutes les transactions, qu'ils construisent et vérifient indépendamment, en commençant par le tout premier bloc (bloc de genèse) et en construisant jusqu'au dernier bloc connu du réseau . Un nœud de blockchain complet peut vérifier de manière indépendante et faisant autorité toute transaction sans recours ni dépendance à tout autre nœud ou source d'informations. Le nœud blockchain complet s'appuie sur le réseau pour recevoir des mises à jour sur les nouveaux blocs de transactions, qu'il vérifie ensuite et incorpore dans sa copie locale de la blockchain.

L'exécution d'un nœud blockchain complet vous offre une expérience Bitcoin pure : une vérification indépendante de toutes les transactions sans avoir besoin de s'appuyer sur d'autres systèmes ou de leur faire confiance. Il est facile de savoir si vous exécutez un nœud complet, car il nécessite plus de cent gigaoctets de stockage persistant (espace disque) pour stocker la blockchain complète. Si vous avez besoin de beaucoup de disque et que la synchronisation avec le réseau prend deux à trois jours, vous exécutez un nœud complet. C'est le prix de l'indépendance totale et de la libération de l'autorité centrale.

Il existe quelques implémentations alternatives de clients bitcoin blockchain complets, construits à l'aide de différents langages de programmation et architectures logicielles. Cependant, l'implémentation la plus courante est le client de référence Bitcoin Core, également connu sous le nom de client Satoshi. Plus de 75% des nœuds du réseau Bitcoin exécutent différentes versions de Bitcoin Core. Il est identifié comme "Satoshi" dans la chaîne de sous-version envoyée dans le message de version et affiché par la commande getpeerinfo comme nous l'avons vu précédemment ; par exemple, /Satoshi :0.8.6/.

Échanger "Inventaire"

La première chose qu'un nœud complet fera une fois qu'il se connectera à ses pairs est d' essayer de construire une blockchain complète. S'il s'agit d'un tout nouveau nœud et qu'il n'a pas du tout de blockchain, il ne connaît qu'un seul bloc, le bloc genesis, qui est intégré de manière statique dans

le logiciel client. En commençant par le bloc n ° 0 (le bloc de genèse), le nouveau nœud devra télécharger des centaines de milliers de blocs pour se synchroniser avec le réseau et rétablir la blockchain complète.

Le processus de synchronisation de la blockchain commence par le message de version, car il contient BestHeight , la hauteur actuelle de la blockchain d'un nœud (nombre de blocs). Un nœud verra les messages de version de ses pairs, saura combien de blocs ils ont chacun et pourra comparer au nombre de blocs dont il dispose dans sa propre blockchain. Les nœuds appairés échangeront un message getblocks contenant le hachage (empreinte digitale) du bloc supérieur sur leur blockchain locale. L'un des pairs pourra identifier le hachage reçu comme appartenant à un bloc qui n'est pas au sommet, mais appartient plutôt à un bloc plus ancien, en déduisant ainsi que sa propre blockchain locale est plus longue que celle de son homologue.

Le pair qui a la plus longue blockchain a plus de blocs que l'autre nœud et peut identifier les blocs dont l'autre nœud a besoin pour « rattraper ». Il identifiera les 500 premiers blocs à partager et transmettra leurs hachages à l'aide d'un message inv (inventaire). Le nœud manquant ces blocs les récupérera ensuite, en émettant une série de messages getdata demandant les données de bloc complètes et en identifiant les blocs demandés à l'aide des hachages du message inv.

Allons supposer, par exemple, qu'un noeud a seul le bloc de genèse. Il recevra alors un message d'inv de ses pairs contenant les hachages des 500 blocs suivants de la chaîne. Il commencera

à demander des blocs à tous ses pairs connectés, en répartissant la charge et en s'assurant qu'il ne submerge aucun pair de requêtes. Le nœud garde la trace du nombre de blocs "en transit" par connexion homologue, c'est-à-dire des blocs qu'il a demandés mais non reçus, en vérifiant qu'il ne dépasse pas une limite (MAX_BLOCKS_IN_TRANSIT_PER_PEER). De cette façon, s'il a besoin de beaucoup de blocs, il n'en demandera que de nouveaux au fur et à mesure que les demandes précédentes seront satisfaites, permettant aux pairs de contrôler le rythme des mises à jour et de ne pas submerger le réseau. Au fur et à mesure que chaque bloc est reçu, il est ajouté à la blockchain, comme nous le verrons dans [blockchain] . Au fur et à mesure que la blockchain locale se construit, davantage de blocs sont demandés et reçus, et le processus se poursuit jusqu'à ce que le nœud rattrape le reste du réseau.

Ce processus de comparaison de la blockchain locale avec les pairs et de récupération des blocs manquants se produit chaque fois qu'un nœud se déconnecte pendant une période de temps . Qu'un nœud soit hors ligne depuis quelques minutes et qu'il manque quelques blocs, ou un mois et qu'il manque quelques milliers de blocs, il commence par envoyer des getblocks , obtient une réponse inv et commence à télécharger les blocs manquants. Le nœud synchronisant la blockchain en récupérant des blocs d'un pair montre l'inventaire et le protocole de propagation de bloc.

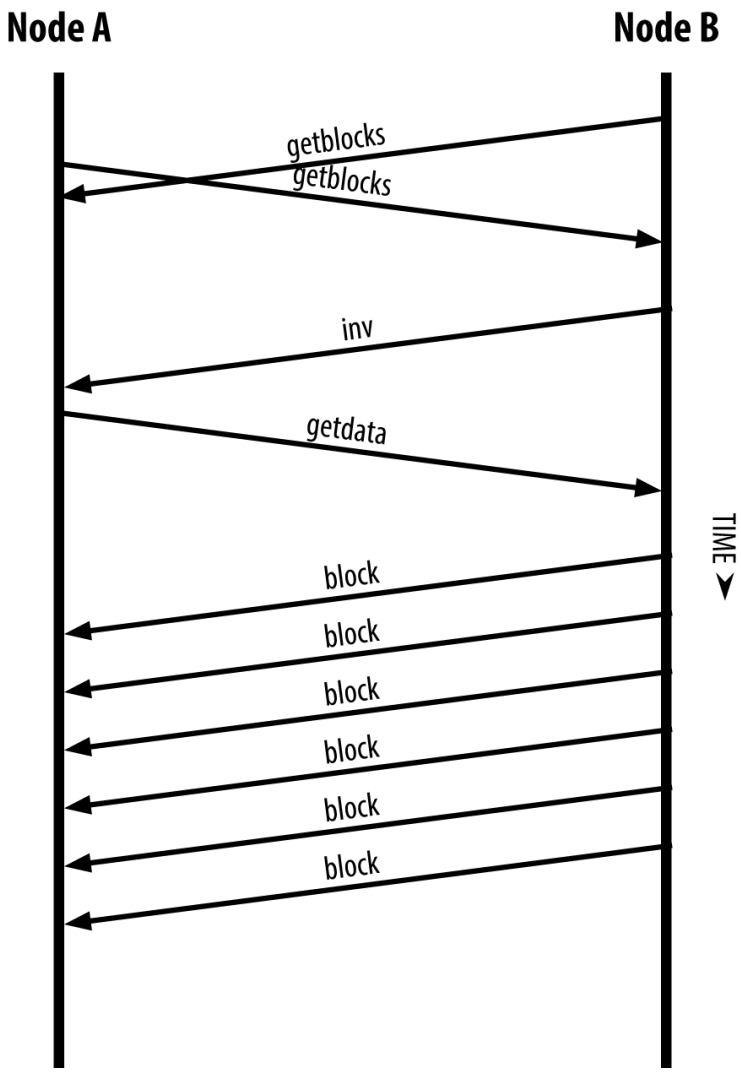


Figure 6. Nœud synchronisant la blockchain en récupérant des blocs d'un pair

Nœuds de vérification simplifiée des paiements (SPV)

Tous les nœuds n'ont pas la capacité de stocker la blockchain complète. De nombreux clients Bitcoin sont conçus pour fonctionner sur des appareils à contraintes d'espace et de puissance, tels que les smartphones, les tablettes ou les systèmes embarqués. Pour ces appareils, une méthode de *vérification de paiement simplifiée* (SPV) est utilisée pour leur permettre de fonctionner sans stocker la blockchain complète. Ces types de clients sont appelés clients SPV ou clients légers. Alors que l'adoption de Bitcoin augmente, le nœud SPV devient la forme la plus courante de nœud Bitcoin, en particulier pour les portefeuilles Bitcoin.

Les nœuds SPV téléchargent uniquement les en-têtes de bloc et ne téléchargent pas les transactions incluses dans chaque bloc. La chaîne de blocs résultante, sans transactions, est 1000 fois plus petite que la blockchain complète. Les nœuds SPV ne peuvent pas construire une image complète de tous les UTXO disponibles pour les dépenses car ils ne connaissent pas toutes les transactions sur le réseau. Les nœuds SPV vérifient les transactions à l'aide d'une méthode légèrement différente qui s'appuie sur des pairs pour fournir des vues partielles des parties pertinentes de la blockchain à la demande.

Par analogie, un nœud complet est comme un touriste dans une ville étrange, équipé d'une carte détaillée de chaque rue et de chaque adresse. En comparaison, un nœud SPV est comme un

touriste dans une ville étrange demandant à des inconnus au hasard des directions détaillées tout en ne connaissant qu'une seule avenue principale. Bien que les deux touristes puissent vérifier l'existence d'une rue en la visitant, le touriste sans carte ne sait pas ce qui se trouve dans l'une des rues secondaires et ne sait pas quelles autres rues existent. Positionné en face du 23 Church Street, le touriste sans carte ne peut pas savoir s'il y a une douzaine d'autres adresses "23 Church Street" dans la ville et si c'est la bonne. La meilleure chance pour le touriste mapless est de demander à suffisamment de gens et d'espérer que certains d'entre eux n'essaient pas de l'attaquer.

SPV vérifie les transactions en fonction de leur *profondeur* dans la blockchain au lieu de leur *hauteur* . Alors qu'un nœud de blockchain complet construira une chaîne entièrement vérifiée de milliers de blocs et de transactions descendant la blockchain (dans le temps) jusqu'au bloc de genèse, un nœud SPV vérifiera la chaîne de tous les blocs (mais pas toutes les transactions) et reliez cette chaîne à la transaction d'intérêt.

Par exemple, lors de l'examen d'une transaction dans le bloc 300000, un nœud complet relie tous les 300000 blocs au bloc genesis et construit une base de données complète d'UTXO, établissant la validité de la transaction en confirmant que l'UTXO reste non dépensé. Un nœud SPV ne peut pas valider si l'UTXO n'est pas dépensé. Au lieu de cela, le nœud SPV établira un lien entre la transaction et le bloc qui la contient, en utilisant un *chemin merkle* (voir [merkle_trees]). Ensuite, le nœud SPV attend de voir les six blocs 300 001 à 300 006 empilés au-dessus du bloc contenant la transaction et le vérifie en établissant sa profondeur sous les blocs 300 006 à 300 001.

Le fait que d'autres nœuds du réseau aient accepté le bloc 300 000 et ensuite effectué le travail nécessaire pour produire six blocs supplémentaires en plus de celui-ci est la preuve, par procuration, que la transaction n'était pas une double dépense.

Un nœud SPV ne peut pas être persuadé qu'une transaction existe dans un bloc alors que la transaction n'existe pas en fait. Le nœud SPV établit l'existence d'une transaction dans un bloc en demandant une preuve de chemin de merkle et en validant la preuve de travail dans la chaîne de blocs. Cependant, l'existence d'une transaction peut être "cachée" d'un nœud SPV. Un nœud SPV peut définitivement prouver qu'une transaction existe mais ne peut pas vérifier qu'une transaction, telle qu'une double dépense du même UTXO, n'existe pas car il n'a pas d'enregistrement de toutes les transactions. Cette vulnérabilité peut être utilisée dans une attaque par déni de service ou pour une attaque à double dépense contre les nœuds SPV. Pour se défendre contre cela, un nœud SPV doit se connecter de manière aléatoire à plusieurs nœuds, pour augmenter la probabilité qu'il soit en contact avec au moins un nœud honnête. Ce besoin de se connecter de manière aléatoire signifie que les nœuds SPV sont également vulnérables aux attaques de partitionnement de réseau ou aux attaques Sybil, où ils sont connectés à de faux nœuds ou de faux réseaux et n'ont pas accès à des nœuds honnêtes ou au vrai réseau Bitcoin.

Dans la plupart des cas, les nœuds SPV bien connectés sont suffisamment sécurisés, établissant un équilibre entre les besoins en ressources, l'aspect pratique et la sécurité. Pour une sécurité infaillible, cependant, rien ne vaut l'exécution d'un nœud blockchain complet.

Conseil

Un nœud blockchain complet vérifie une transaction en vérifiant toute la chaîne de milliers de blocs en dessous afin de garantir que l'UTXO n'est pas dépensé, tandis qu'un nœud SPV vérifie à quelle profondeur le bloc est enterré par une poignée de blocs au-dessus.

Pour obtenir les en-têtes de bloc, les nœuds SPV utilisent un message getheaders au lieu de getblocks . Le pair répondant enverra jusqu'à 2 000 en-têtes de bloc en utilisant un seul message d'en-têtes. Le processus est par ailleurs le même que celui utilisé par un nœud complet pour récupérer des blocs complets. Les nœuds SPV définissent également un filtre sur la connexion aux pairs, pour filtrer le flux des futurs blocs et transactions envoyés par les pairs. Toutes les transactions d'intérêt sont récupérées à l'aide d'une requête getdata . Le pair génère un message tx contenant les transactions, en réponse. Le nœud SPV synchronisant les en-têtes de bloc montre la synchronisation des en-têtes de bloc.

Étant donné que les nœuds SPV doivent récupérer des transactions spécifiques afin de les vérifier de manière sélective, ils créent également un risque de confidentialité. Contrairement aux nœuds blockchain complets, qui collectent toutes les transactions au sein de chaque bloc, les demandes du nœud SPV pour des données spécifiques peuvent révéler par inadvertance les adresses de leur portefeuille. Par exemple, un tiers surveillant un réseau pourrait suivre toutes les transactions demandées par un portefeuille sur un nœud SPV et les utiliser pour associer des adresses bitcoin à l'utilisateur de ce portefeuille, détruisant ainsi la confidentialité de l'utilisateur.

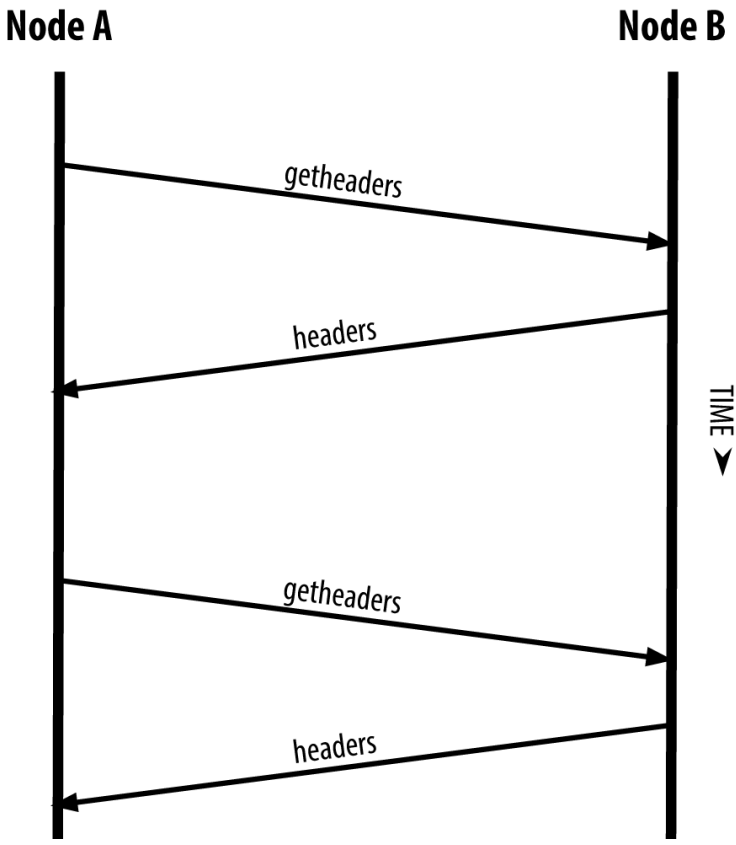


Figure 7. Nœud SPV synchronisant les en- têtes de bloc

Peu de temps après l'introduction de SPV / nœuds légers, les développeurs de bitcoin ont ajouté une fonctionnalité appelée *filtres de floraison* pour traiter les risques de confidentialité des nœuds SPV. Les filtres Bloom permettent aux nœuds SPV de recevoir un sous-ensemble des transactions sans révéler préci-

sément les adresses qui les intéressent, grâce à un mécanisme de filtrage qui utilise des probabilités plutôt que des modèles fixes.

Filtres Bloom

Un filtre de floraison est un filtre de recherche probabiliste qui offre un moyen efficace d'exprimer un modèle de recherche tout en protégeant la confidentialité. Ils sont utilisés par les nœuds SPV pour demander à leurs pairs des transactions correspondant à un modèle spécifique, sans révéler exactement quelles adresses, clés ou transactions ils recherchent.

Dans notre analogie précédente, un touriste sans carte demande son chemin vers une adresse précise, "23 Church St." Si elle demande à des inconnus leur chemin vers cette rue, elle révèle par inadvertance sa destination. Un filtre de floraison, c'est comme demander : "Y a-t-il des rues dans ce quartier dont le nom se termine par RCH?" Une question comme celle-là en révèle un peu moins sur la destination souhaitée que de demander "23 Church St." En utilisant cette technique, un touriste pourrait spécifier l'adresse souhaitée plus en détail, par exemple "se terminant par URCH" ou moins en détail par "se terminant par H." En faisant varier la précision de la recherche, le touriste révèle plus ou moins d'informations , au détriment d'obtenir des résultats plus ou moins précis. Si elle demande un modèle moins spécifique, elle obtient beaucoup plus d'adresses possibles et une meilleure confidentialité, mais la plupart des résultats ne sont pas pertinents. Si elle demande un modèle

très spécifique, elle obtient moins de résultats mais perd sa confidentialité.

Les filtres Bloom remplissent cette fonction en permettant à un nœud SPV de spécifier un modèle de recherche pour les transactions qui peuvent être réglées vers la précision ou la confidentialité. Un filtre de floraison plus spécifique produira des résultats précis, mais au détriment de la révélation des modèles qui intéressent le nœud SPV, révélant ainsi les adresses appartenant au portefeuille de l'utilisateur. Un filtre de floraison moins spécifique produira plus de données sur plus de transactions, dont beaucoup ne sont pas pertinentes pour le nœud, mais permettra au nœud de maintenir une meilleure confidentialité.

Comment fonctionnent les filtres Bloom

Les filtres Bloom sont implémentés sous la forme d'un tableau de taille variable de N chiffres binaires (un champ de bits) et d'un nombre variable de M fonctions de hachage. Les fonctions de hachage sont conçues pour toujours produire une sortie comprise entre 1 et N, correspondant au tableau de chiffres binaires. Les fonctions de hachage sont générées de manière déterministe, de sorte que tout nœud implémentant un filtre de floraison utilisera toujours les mêmes fonctions de hachage et obtiendra les mêmes résultats pour une entrée spécifique. En choisissant des filtres de floraison de différentes longueurs (N) et un nombre différent (M) de fonctions de hachage, le filtre de floraison peut être réglé, en faisant varier le niveau de précision et donc de confidentialité.

Dans un exemple de filtre de floraison simpliste, avec un

champ de 16 bits et trois fonctions de hachage , nous utilisons un très petit tableau de 16 bits et un ensemble de trois fonctions de hachage pour démontrer le fonctionnement des filtres de floraison.

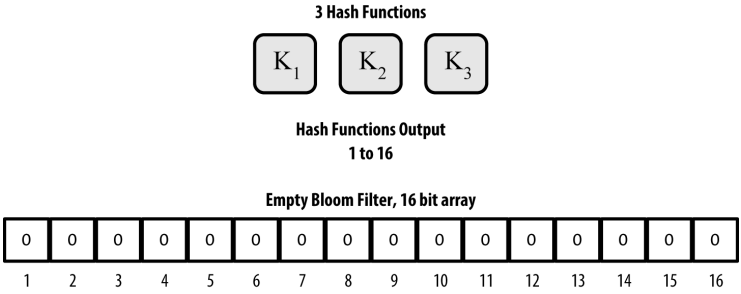


Figure 8. Exemple de filtre de floraison simpliste, avec un champ de 16 bits et trois fonctions de hachage

Le filtre de floraison est initialisé de sorte que le tableau de bits soit entièrement zéros. Pour ajouter un motif au filtre de floraison, le motif est haché tour à tour par chaque fonction de hachage. L'application de la première fonction de hachage à l'entrée donne un nombre compris entre 1 et N. Le bit correspondant dans le tableau (indexé de 1 à N) est trouvé et mis à 1, enregistrant ainsi la sortie de la fonction de hachage. Ensuite, la fonction de hachage suivante est utilisée pour définir un autre bit et ainsi de suite. Une fois que toutes les fonctions de hachage M ont été appliquées, le motif de recherche sera "enregistré" dans le filtre de floraison sous la forme de M bits qui ont été modifiés de 0 à 1.

L'ajout d'un motif "A" à notre simple filtre de floraison est un exemple d'ajout d'un motif "A" au simple filtre de floraison illustré dans Un exemple de filtre de floraison simpliste, avec un champ de 16 bits et trois fonctions de hachage .

Ajouter un deuxième motif est aussi simple que de répéter ce processus. Le motif est haché par chaque fonction de hachage à tour de rôle et le résultat est enregistré en définissant les bits sur 1. Notez que lorsqu'un filtre de floraison est rempli de plus de motifs, le résultat d'une fonction de hachage peut coïncider avec un bit déjà défini sur 1, auquel cas le bit n'est pas modifié. Essentiellement, au fur et à mesure que plus de motifs enregistrent sur des bits qui se chevauchent, le filtre de floraison commence à devenir saturé avec plus de bits mis à 1 et la précision du filtre diminue. C'est pourquoi le filtre est une structure de données probabiliste - il devient moins précis à mesure que davantage de modèles sont ajoutés. La précision dépend du nombre de motifs ajoutés par rapport à la taille du tableau de bits (N) et du nombre de fonctions de hachage (M). Un plus grand tableau de bits et plus de fonctions de hachage peuvent enregistrer plus de modèles avec une plus grande précision. Un tableau de bits plus petit ou moins de fonctions de hachage enregistrera moins de modèles et produira moins de précision.

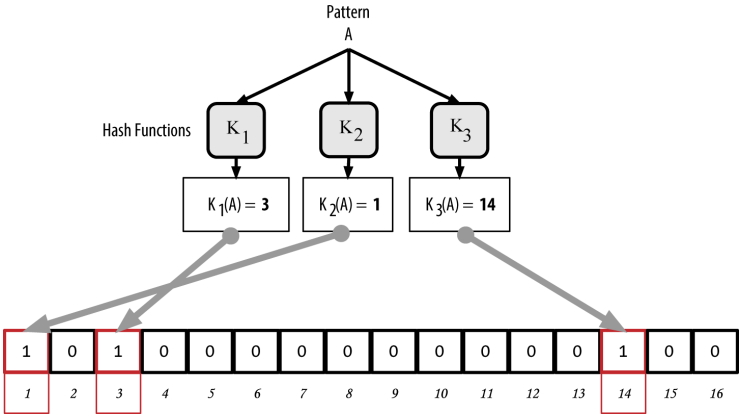


Figure 9. Ajout d'un motif "A" à notre filtre de floraison simple

L'ajout d'un deuxième motif "B" à notre filtre de floraison simple est un exemple d'ajout d'un deuxième motif "B" au filtre de floraison simple.

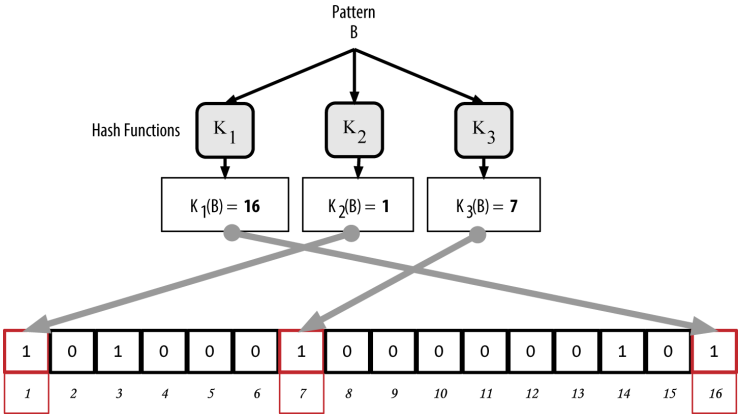


Figure 10. Ajout d'un deuxième motif "B" à notre filtre de floraison simple

Pour tester si un motif fait partie d'un filtre de bloom, le motif est haché par chaque fonction de hachage et le motif de bits résultant est testé par rapport au tableau de bits. Si tous les bits indexés par les fonctions de hachage sont mis à 1, alors le motif est *probablement* enregistré dans le filtre de floraison. Étant donné que les bits peuvent être définis en raison du chevauchement de plusieurs modèles, la réponse n'est pas certaine, mais est plutôt probabiliste. En termes simples, une correspondance positive de filtre de floraison est un "Peut-être, Oui".

Test de l'existence du motif "X" dans le filtre de floraison. Le résultat est une correspondance positive probabiliste, signifiant «Peut-être». est un exemple de test de l'existence du motif "X" dans le filtre de floraison simple. Les bits correspondants sont mis à 1, donc le motif est probablement

une correspondance.

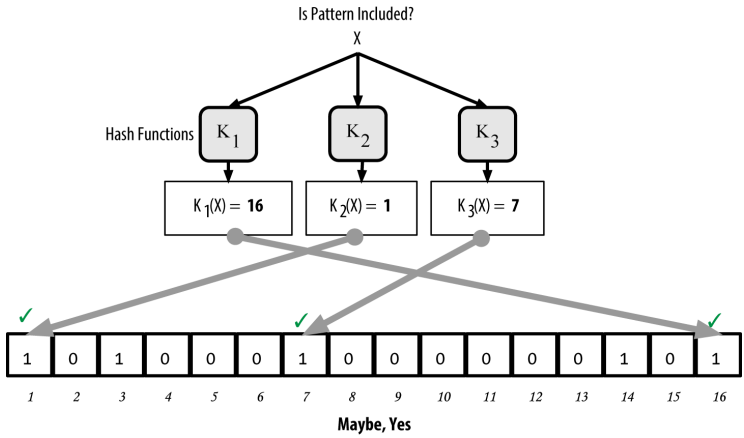


Figure 11. Test de l'existence du motif "X" dans le filtre de floraison. Le résultat est une correspondance positive probabiliste, signifiant « Peut-être ».

Au contraire, si un motif est testé par rapport au filtre de bloom et que l'un des bits est mis à 0, cela prouve que le motif n'a pas été enregistré dans le filtre de bloom. Un résultat négatif n'est pas une probabilité, c'est une certitude. En termes simples, une correspondance négative sur un filtre de floraison est un "Certainement pas !"

Test de l'existence du motif "Y" dans le filtre de floraison. Le résultat est une correspondance négative définitive, ce qui signifie "Certainement pas !" est un exemple de test de l'existence du motif "Y" dans le filtre de floraison simple. L'un des

bits correspondants est mis à 0, donc le motif n'est certainement pas une correspondance.

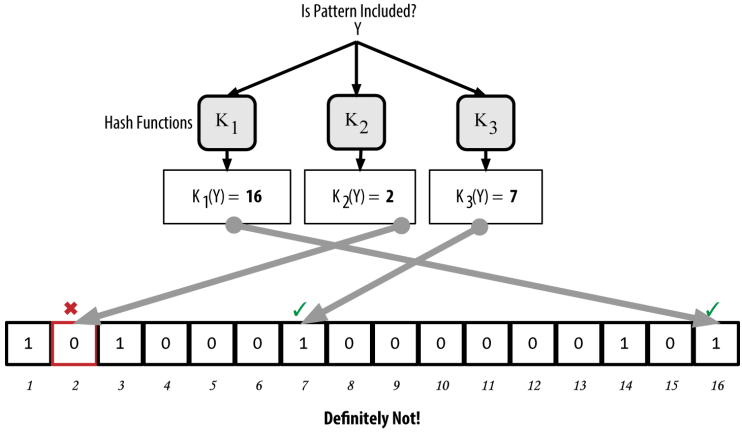


Figure 12. Test de l'existence du motif "Y" dans le filtre de bloom. Le résultat est une correspondance négative définitive, ce qui signifie "Certainement pas!"

Comment les nœuds SPV utilisent les filtres Bloom

Les filtres Bloom sont utilisés pour filtrer les transactions (et les blocs les contenant) qu'un nœud SPV reçoit de ses pairs, en sélectionnant uniquement les transactions d'intérêt pour le nœud SPV sans révéler les adresses ou les clés qui l'intéressent.

Un nœud SPV initialisera un filtre de floraison comme "vide";

dans cet état, le filtre de floraison ne correspondra à aucun modèle. Le nœud SPV dressera ensuite une liste de toutes les adresses, clés et hachages qui l'intéressent. Il le fera en extrayant le hachage de clé publique, le hachage de script et les ID de transaction de tout UTXO contrôlé par son portefeuille. Le nœud SPV ajoute ensuite chacun de ceux-ci au filtre de floraison, de sorte que le filtre de floraison « corresponde » si ces modèles sont présents dans une transaction, sans révéler les modèles eux-mêmes.

Le nœud SPV enverra alors un message de chargement de filtre à l'homologue, contenant le filtre de floraison à utiliser sur la connexion. Sur l'homologue, les filtres de floraison sont vérifiés par rapport à chaque transaction entrante. Le nœud complet vérifie plusieurs parties de la transaction par rapport au filtre de floraison, à la recherche d'une correspondance comprenant :

- L' ID de transaction
- Les composants de données des scripts de verrouillage de chacune des sorties de transaction (chaque clé et hachage dans le script)
- Chacune des entrées de transaction
- Chacun des composants de données de signature d'entrée (ou scripts témoins)

En comparant tous ces composants, les filtres Bloom peuvent être utilisés pour faire correspondre les hachages de clé publique, les scripts, les valeurs OP_RETURN, les clés publiques dans les signatures ou tout composant futur d'un contrat intelligent ou d'un script complexe.

Une fois le filtre établi, l'homologue testera alors la sortie de chaque transaction par rapport au filtre de floraison. Seules les transactions qui correspondent au filtre sont envoyées au nœud.

En réponse à un message getdata du nœud, les pairs enverront un message merkleblock qui ne contient que des en-têtes de bloc pour les blocs correspondant au filtre et un chemin merkle (voir [merkle_trees]) pour chaque transaction correspondante. Le pair enverra alors également des messages tx contenant les transactions correspondant au filtre.

Lorsque le nœud complet envoie des transactions au nœud SPV, le nœud SPV rejette tous les faux positifs et utilise les transactions correctement appariées pour mettre à jour son ensemble UTXO et le solde de son portefeuille. Au fur et à mesure qu'il met à jour sa propre vue de l'ensemble UTXO, il modifie également le filtre de floraison pour qu'il corresponde à toutes les futures transactions faisant référence à l'UTXO qu'il vient de trouver. Le nœud complet utilise ensuite le nouveau filtre de floraison pour faire correspondre les nouvelles transactions et l'ensemble du processus se répète.

Le nœud définissant le filtre de floraison peut ajouter de manière interactive des modèles au filtre en envoyant un message filteradd . Pour effacer le filtre de floraison, le nœud peut envoyer un message filterclear . Comme il n'est pas possible de supprimer un modèle d'un filtre de floraison, un nœud doit effacer et renvoyer un nouveau filtre de floraison si un modèle n'est plus souhaité.

Le protocole de réseau et le mécanisme de filtre de floraison pour les nœuds SPV sont définis dans BIP-37 (Peer Services) .

Nœuds SPV et confidentialité

Les nœuds qui implémentent SPV ont une confidentialité plus faible qu'un nœud complet. Un nœud complet reçoit toutes les transactions et ne révèle donc aucune information sur l'utilisation ou non d'une adresse dans son portefeuille. Un nœud SPV reçoit une liste filtrée de transactions liées aux adresses qui se trouvent dans son portefeuille. En conséquence, cela réduit la vie privée du propriétaire.

Les filtres Bloom sont un moyen de réduire la perte de confidentialité. Sans eux, un nœud SPV devrait répertorier explicitement les adresses qui l'intéressent, créant ainsi une grave violation de la vie privée. Cependant, même avec des filtres de floraison, un adversaire surveillant le trafic d'un client SPV ou qui y est connecté directement en tant que nœud dans le réseau P2P peut collecter suffisamment d'informations au fil du temps pour apprendre les adresses dans le portefeuille du client SPV.

Connexions cryptées et authentifiées

La plupart des nouveaux utilisateurs de bitcoin supposent que les communications réseau d'un nœud bitcoin sont cryptées. En fait, l'implémentation originale de Bitcoin communique entièrement en clair. Bien que ce ne soit pas un problème majeur de confidentialité pour les nœuds complets, c'est un

gros problème pour les nœuds SPV.

Afin d' augmenter la confidentialité et la sécurité du réseau Bitcoin P2P, il existe deux solutions qui assurent le cryptage des communications : *Tor Transport* et *P2P Authentication and Encryption* avec BIP-150/151.

Transport de Tor

Tor, qui signifie *The Onion Routing network* , est un projet logiciel et un réseau qui offre le cryptage et l'encapsulation des données via des chemins réseau aléatoires qui offrent anonymat, non traçabilité et confidentialité.

Bitcoin Core propose plusieurs options de configuration qui vous permettent d'exécuter un nœud bitcoin avec son trafic transporté sur le réseau Tor. De plus, Bitcoin Core peut également offrir un service caché Tor permettant à d'autres nœuds Tor de se connecter à votre nœud directement via Tor.

À partir de la version 0.12 de Bitcoin Core, un nœud offrira automatiquement un service Tor caché s'il est capable de se connecter à un service Tor local. Si Tor est installé et que le processus Bitcoin Core s'exécute en tant qu'utilisateur disposant des autorisations adéquates pour accéder au cookie d'authentification Tor, il devrait fonctionner automatiquement. Utilisez l'indicateur de débogage pour activer le débogage de Bitcoin Core pour le service Tor comme ceci :

```
$ bitcoind --daemon --debug=tor
```

Vous devriez voir "tor : ADD_ONION réussi" dans les journaux, indiquant que Bitcoin Core a ajouté un service caché au réseau Tor.

Vous pouvez trouver plus d'instructions sur l'exécution de Bitcoin Core en tant que service caché Tor dans la documentation Bitcoin Core (*docs / tor.md*) et divers didacticiels en ligne.

Authentification et chiffrement d'égal à égal

Deux propositions d'amélioration de Bitcoin, BIP-150 et BIP-151, ajoutent la prise en charge de l'authentification et du cryptage P2P dans le réseau Bitcoin P2P. Ces deux BIP définissent des services optionnels qui peuvent être proposés par des nœuds bitcoin compatibles. BIP-151 permet le cryptage négocié pour toutes les communications entre deux nœuds qui prennent en charge BIP-151. Le BIP-150 offre une authentification par les pairs en option qui permet aux nœuds de s'authentifier mutuellement à l'aide d'ECDSA et de clés privées. Le BIP-150 exige qu'avant l'authentification, les deux nœuds aient établi des communications cryptées conformément au BIP-151.

Depuis février 2021, BIP-150 et BIP-151 ne sont pas implémentés dans Bitcoin Core. Cependant, les deux propositions ont été mises en œuvre par au moins un client bitcoin alternatif nommé bcoin .

BIP-150 et BIP-151 permettent aux utilisateurs d'exécuter des clients SPV qui se connectent à un nœud complet de confiance, en utilisant le cryptage et l'authentification pour protéger la confidentialité du client SPV.

De plus, l'authentification peut être utilisée pour créer des réseaux de nœuds Bitcoin de confiance et empêcher les attaques Man-in-the-Middle. Enfin, le cryptage P2P, s'il était déployé à grande échelle, renforcerait la résistance du bitcoin à l'analyse du trafic et à la surveillance érodant la vie privée, en particulier dans les pays totalitaires où l'utilisation d'Internet est fortement contrôlée et surveillée.

La norme est définie dans BIP-150 (Peer Authentication) et BIP-151 (Peer-to-Peer Communication Encryption) .

Pools de transactions

Presque tous les nœuds du réseau bitcoin maintiennent une liste temporaire de transactions non confirmées appelée *pool de mémoire* , *mempool* ou *pool de transactions* . Les nœuds utilisent ce pool pour suivre les transactions connues du réseau mais qui ne sont pas encore incluses dans la blockchain. Par exemple, un nœud de portefeuille utilisera le pool de transactions pour suivre les paiements entrants dans le portefeuille de l'utilisateur qui ont été reçus sur le réseau mais qui ne sont pas encore confirmés.

Au fur et à mesure que les transactions sont reçues et vérifiées, elles sont ajoutées au pool de transactions et relayées vers les

nœuds voisins pour se propager sur le réseau.

Certaines implémentations de nœuds maintiennent également un pool séparé de transactions orphelines. Si les entrées d'une transaction font référence à une transaction qui n'est pas encore connue, telle qu'un parent manquant, la transaction orpheline sera stockée temporairement dans le pool orphelin jusqu'à ce que la transaction parent arrive.

Lorsqu'une transaction est ajoutée au pool de transactions, le pool d'orphelins est vérifié pour tous les orphelins qui référencent les sorties de cette transaction (ses enfants). Tous les orphelins correspondants sont ensuite validés. S'ils sont valides, ils sont supprimés du pool orphelin et ajoutés au pool de transactions, complétant ainsi la chaîne qui a commencé avec la transaction parent. À la lumière de la transaction nouvellement ajoutée, qui n'est plus orpheline, le processus est répété de manière récursive à la recherche d'autres descendants, jusqu'à ce qu'il n'y ait plus de descendants. Grâce à ce processus, l'arrivée d'une transaction mère déclenche une reconstruction en cascade de toute une chaîne de transactions interdépendantes en réunissant les orphelins avec leurs parents tout au long de la chaîne.

Le pool de transactions et le pool orphelin (le cas échéant) sont stockés dans la mémoire locale et ne sont pas enregistrés sur le stockage persistant ; ils sont plutôt alimentés dynamiquement à partir des messages réseau entrants. Lorsqu'un nœud démarre, les deux pools sont vides et sont progressivement remplis avec les nouvelles transactions reçues sur le réseau.

Certaines implémentations du client bitcoin maintiennent également une base de données ou un pool UTXO, qui est l'ensemble de toutes les sorties non dépensées sur la blockchain. Les utilisateurs de Bitcoin Core le trouveront dans le dossier chainstate / du répertoire de données de leur client. Bien que le nom « pool UTXO » ressemble au pool de transactions, il représente un ensemble de données différent. Contrairement à la transaction et aux pools orphelins, le pool UTXO n'est pas initialisé vide mais contient à la place des millions d'entrées de sorties de transaction non dépensées, tout ce qui n'est pas dépensé depuis le début jusqu'au bloc de genèse. Le pool UTXO peut être hébergé dans la mémoire locale ou en tant que table de base de données indexée sur un stockage persistant.

Alors que la transaction et les pools orphelins représentent la perspective locale d'un seul nœud et peuvent varier considérablement d'un nœud à l'autre en fonction du moment où le nœud a été démarré ou redémarré, le pool UTXO représente le consensus émergent du réseau et variera donc peu entre les nœuds. En outre, la transaction et les pools orphelins ne contiennent que des transactions non confirmées, tandis que le pool UTXO ne contient que des sorties confirmées.

10

La Blockchain

Introduction

La structure de données blockchain est une liste ordonnée et liée de blocs de transactions. La blockchain peut être stockée sous forme de fichier plat ou dans une simple base de données. Le client Bitcoin Core stocke les métadonnées de la blockchain à l'aide de la base de données LevelDB de Google . Les blocs sont liés « en arrière », chacun faisant référence au bloc précédent de la chaîne. La blockchain est souvent visualisée comme une pile verticale, avec des blocs superposés et le premier bloc servant de base à la pile. La visualisation des blocs empilés les uns sur les autres aboutit à l'utilisation de termes tels que « hauteur » pour désigner la distance par rapport au premier bloc et « haut » ou « pointe » pour désigner le bloc le plus récemment ajouté.

Chaque bloc dans la blockchain est identifié par un hachage,

généré à l'aide de l'algorithme de hachage cryptographique SHA256 sur l'en-tête du bloc. Chaque bloc fait également référence à un bloc précédent, connu sous le nom de bloc *parent* , par le biais du champ "hachage de bloc précédent" dans l'en-tête du bloc. En d'autres termes, chaque bloc contient le hachage de son parent dans son propre en-tête. La séquence de hachages reliant chaque bloc à son parent crée une chaîne remontant jusqu'au premier bloc jamais créé, connu sous le nom de *bloc de genèse* .

Bien qu'un bloc n'ait qu'un seul parent, il peut temporairement avoir plusieurs enfants. Chacun des enfants fait référence au même bloc que son parent et contient le même hachage (parent) dans le champ "hachage du bloc précédent". Plusieurs enfants surviennent lors d'un « fork » de la blockchain, une situation temporaire qui se produit lorsque différents blocs sont découverts presque simultanément par différents mineurs (voir [fourches]). Finalement, un seul bloc enfant devient une partie de la blockchain et le « fork » est résolu. Même si un bloc peut avoir plus d'un enfant, chaque bloc ne peut avoir qu'un seul parent. Cela est dû au fait qu'un bloc a un seul champ "hachage de bloc précédent" référençant son parent unique.

Le champ "hachage du bloc précédent" se trouve à l'intérieur de l'en-tête du bloc et affecte ainsi le hachage du bloc *actuel* . La propre identité de l'enfant change si l'identité du parent change. Lorsque le parent est modifié de quelque manière que ce soit, le hachage du parent change. Le hachage modifié du parent nécessite un changement du pointeur "hachage de bloc précédent" de l'enfant. Cela entraîne à son tour le changement de hachage de l'enfant, ce qui nécessite une modification du

pointeur du petit-enfant, qui à son tour change le petit-enfant, etc. Cet effet de cascade garantit qu'une fois qu'un bloc a de nombreuses générations le suivant, il ne peut pas être modifié sans forcer un recalcul de tous les blocs suivants. Parce qu'un tel recalcul nécessiterait d'énormes calculs (et donc une consommation d'énergie), l'existence d'une longue chaîne de blocs rend la profonde histoire de la blockchain immuable, ce qui est une caractéristique clé de la sécurité de Bitcoin.

Une façon de penser à la blockchain est comme des couches dans une formation géologique ou un échantillon de carotte de glacier. Les couches superficielles peuvent changer avec les saisons, voire être emportées par le vent avant d'avoir le temps de se déposer. Mais une fois que vous atteignez quelques centimètres de profondeur, les couches géologiques deviennent de plus en plus stables. Au moment où vous regardez quelques centaines de mètres plus bas, vous regardez un instantané du passé qui n'a pas été perturbé pendant des millions d'années. Dans la blockchain, les quelques blocs les plus récents peuvent être révisés s'il y a un recalcul de la chaîne en raison d'un fork. Les six premiers blocs sont comme quelques centimètres de terre végétale. Mais une fois que vous entrez plus profondément dans la blockchain, au-delà de six blocs, les blocs sont de moins en moins susceptibles de changer. Après 100 blocs en arrière, il y a tellement de stabilité que la transaction coinbase - la transaction contenant du bitcoin nouvellement extrait - peut être dépensée. Il y a quelques milliers de blocs (un mois) et la blockchain est une histoire établie, à toutes fins pratiques. Alors que le protocole permet toujours à une chaîne d'être défait par une chaîne plus longue et que la possibilité qu'un bloc soit inversé existe toujours, la probabilité d'un tel événement

diminue avec le temps jusqu'à ce qu'il devienne infinitésimal.

Structure d'un bloc

Un bloc est une structure de données de conteneur qui agrège les transactions pour les inclure dans le grand livre public, la blockchain. Le bloc est constitué d'un en-tête contenant des métadonnées, suivi d'une longue liste de transactions qui constituent l'essentiel de sa taille. L'en-tête de bloc est de 80 octets, tandis que la transaction moyenne est d'au moins 400 octets et le bloc moyen contient plus de 1900 transactions. Un bloc complet, avec toutes les transactions, est donc 10 000 fois plus grand que l'en-tête du bloc. La structure d'un bloc décrit la structure d'un bloc.

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1–9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Tableau 1. La structure d'un bloc

En-tête de bloc

L'en-tête de bloc se compose de trois ensembles de métadonnées de bloc. Tout d'abord, il y a une référence à un hachage de bloc précédent, qui connecte ce bloc au bloc précédent dans la blockchain. Le deuxième ensemble de métadonnées, à savoir la *difficulté* , l' *horodatage* et le *nonce* , se rapporte à la concurrence minière, comme détaillé dans [extraction] . Le troisième élément de métadonnées est la racine de l'arborescence merkle , une structure de données utilisée pour résumer efficacement toutes les transactions du bloc. La structure de l'en-tête de bloc décrit la structure d'un en-tête de bloc.

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (in seconds elapsed since Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

Tableau 2. La structure de l'en-tête de bloc

Un compteur utilisé pour l'algorithme de preuve de travail

Le nonce, la cible de difficulté et l'horodatage sont utilisés dans le processus d'extraction et seront discutés plus en détail dans [extraction] .

Identificateurs de bloc : hachage d'en-tête de bloc et hauteur de bloc

L'identifiant principal d'un bloc est son hachage cryptographique, une empreinte numérique, faite en hachant l'en-tête de bloc deux fois via l'algorithme SHA256. Le hachage de 32 octets qui en résulte est appelé *hachage de bloc,* mais est plus précisément le *hachage d'en-tête de bloc* , car seul l'en-tête de bloc est utilisé pour le calculer. Par exemple, 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f est le hachage de bloc du premier bloc bitcoin jamais créé. Le hachage de bloc identifie un bloc de manière unique et sans ambiguïté et peut être dérivé indépendamment par n'importe quel nœud en double simplement l'en-tête de bloc avec l'algorithme SHA256.

Notez que le hachage de bloc n'est pas réellement inclus dans la structure de données du bloc, ni lorsque le bloc est transmis sur le réseau, ni lorsqu'il est stocké sur le stockage de persistance d'un nœud dans le cadre de la blockchain. Au lieu de cela, le hachage du bloc est calculé par chaque nœud lorsque le bloc est reçu du réseau. Le hachage de bloc peut être stocké dans une table de base de données distincte dans le cadre des métadonnées du bloc, pour faciliter l'indexation et une récupération plus rapide des blocs à partir du disque.

Une deuxième façon d'identifier un bloc est sa position dans la blockchain, appelée *hauteur de bloc* . Le premier bloc jamais créé est à la hauteur de bloc 0 (zéro) et est le même bloc qui était précédemment référencé par le hachage de bloc suivant 000000000019d6689c085ae165831e934ff763ae46a2a6c17 2b3f1b60a8ce26f. Un bloc peut ainsi être identifié de deux manières : en référençant le hachage du bloc ou en référençant la hauteur du bloc. Chaque bloc suivant ajouté "au-dessus" de ce premier bloc est une position "plus haute" dans la blockchain, comme des boîtes empilées les unes sur les autres.

De plus, le terme *hauteur de bloc actuelle* indique la taille de la blockchain en blocs à un moment donné. Par exemple, la hauteur actuelle des blocs au 1er mars 2021 était d'environ 672 722, ce qui signifie qu'il y avait 672 722 blocs empilés au-dessus du premier bloc créé en janvier 2009.

Contrairement au hachage de bloc, la hauteur de bloc n'est pas un identifiant unique. Bien qu'un seul bloc ait toujours une hauteur de bloc spécifique et invariante, l'inverse n'est pas vrai : la hauteur de bloc n'identifie pas toujours un seul bloc. Deux blocs ou plus peuvent avoir la même hauteur de bloc, en compétition pour la même position dans la blockchain. Ce scénario est discuté en détail dans la section [fourches] . La hauteur du bloc ne fait pas non plus partie de la structure de données du bloc ; il n'est pas stocké dans le bloc. Chaque nœud identifie dynamiquement la position (hauteur) d'un bloc dans la blockchain lorsqu'il est reçu du réseau Bitcoin. La hauteur du bloc peut également être stockée sous forme de métadonnées dans une table de base de données indexée pour une récupération plus rapide.

Conseil

Le hachage de *bloc d'un* bloc *identifie toujours un seul bloc de manière unique. Un bloc a également toujours une* hauteur de bloc *spécifique . Cependant, il n'est pas toujours possible qu'une hauteur de bloc spécifique puisse identifier un seul bloc. Au contraire, deux blocs ou plus pourraient se disputer une seule position dans la blockchain.*

Le bloc Genesis

Le premier bloc de la blockchain s'appelle le bloc de genèse et a été créé en 2009. C'est l'ancêtre commun de tous les blocs de la blockchain, ce qui signifie que si vous commencez à n'importe quel bloc et suivez la chaîne en arrière dans le temps, vous finirez par arriver. au bloc de genèse.

Chaque nœud commence toujours par une blockchain d'au moins un bloc car le bloc de genèse est codé statiquement dans le logiciel client Bitcoin, de sorte qu'il ne peut pas être modifié. Chaque nœud « connaît » toujours le hachage et la structure du bloc de genèse, l'heure fixe à laquelle il a été créé et même la transaction unique à l'intérieur. Ainsi, chaque nœud a le point de départ de la blockchain, une « racine » sécurisée à partir de laquelle construire une blockchain de confiance.

Voir le bloc de genèse codé statiquement dans le client Bitcoin Core, dans *chainparams.cpp* .

Le hachage d'identificateur suivant appartient au bloc de

genèse :

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Vous pouvez rechercher ce hachage de bloc dans n'importe quel site Web d'explorateur de blocs, tel que *blockchain.info* , et vous trouverez une page décrivant le contenu de ce bloc, avec une URL contenant ce hachage :

https ://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Utilisation du client de référence Bitcoin Core sur la ligne de commande :

```
$ bitcoin-cli getblock
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

```
{
    "hash" :
    "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce
    "confirmations" : 308321,
    "size" : 285,
    "height" : 0,
    "version" : 1,
    "merkleroot" :
    "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda
    "tx" : [
```

```
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7a
    ],
    "time" : 1231006505,
    "nonce" : 2083236893,
    "bits" : "1d00ffff",
    "difficulty" : 1.00000000,
    "nextblockhash" :
    "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb
}
```

Le bloc de genèse contient un message caché. L' entrée de la transaction coinbase contient le texte "The Times 03 / Jan / 2009 Chancellor on the brink of second bailout for banks". Ce message visait à prouver la date la plus ancienne de création de ce bloc, en référençant le titre du journal britannique *The Times* . Il sert également de rappel ironique de l'importance d'un système monétaire indépendant, le lancement de Bitcoin se produisant en même temps qu'une crise monétaire mondiale sans précédent. Le message a été intégré dans le premier bloc par Satoshi Nakamoto, le créateur de Bitcoin.

Lier des blocs dans la blockchain

Les nœuds complets Bitcoin conservent une copie locale de la blockchain, en commençant par le bloc de genèse. La copie locale de la blockchain est constamment mise à jour au fur et à mesure que de nouveaux blocs sont trouvés et utilisés pour étendre la chaîne. Lorsqu'un nœud reçoit des blocs entrants

du réseau, il validera ces blocs, puis les liera à la blockchain existante. Pour établir un lien, un nœud examinera l'en-tête de bloc entrant et recherchera le « hachage de bloc précédent ».

Allons supposer, par exemple, qu'un noeud a 277,314 blocs dans la copie locale du blockchain. Le dernier bloc que le nœud connaît est le bloc 277,314, avec un hachage d'en-tête de bloc de :

```
00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249
```

Le nœud bitcoin reçoit ensuite un nouveau bloc du réseau, qu'il analyse comme suit :

```
{
    "size" : 43560,
    "version" : 2,
    "previousblockhash" :
        "00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a
    "merkleroot" :
        "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1
    "time" : 1388185038,
    "difficulty" : 1180923195.25802612,
    "nonce" : 4215469401,
    "tx" : [
        "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7c

 #[... many more transactions omitted ...]

        "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a0
```

```
    ]
}
```

En regardant ce nouveau bloc, le nœud trouve le champ previousblockhash , qui contient le hachage de son bloc parent. Il s'agit d'un hachage connu du nœud, celui du dernier bloc de la chaîne à hauteur 277,314. Par conséquent, ce nouveau bloc est un enfant du dernier bloc de la chaîne et étend la blockchain existante. Le nœud ajoute ce nouveau bloc à la fin de la chaîne, allongeant la blockchain avec une nouvelle hauteur de 277315. Les blocs liés dans une chaîne par référence au hachage d'en-tête de bloc précédent montre la chaîne de trois blocs, liés par des références dans le champ previousblockhash .

Arbres Merkle

Chaque bloc de la blockchain Bitcoin contient un résumé de toutes les transactions du bloc à l'aide d'un *arbre de merkle* .

Un *arbre merkle* , également connu sous le nom d' *arbre de hachage binaire* , est une structure de données utilisée pour résumer et vérifier efficacement l'intégrité de grands ensembles de données. Les arbres Merkle sont des arbres binaires contenant des hachages cryptographiques. Le terme « arbre » est utilisé en informatique pour décrire une structure de données ramifiée, mais ces arbres sont généralement affichés à l'envers

avec la « racine » en haut et les « feuilles » en bas d'un diagramme, comme vous le verrez dans les exemples qui suivent.

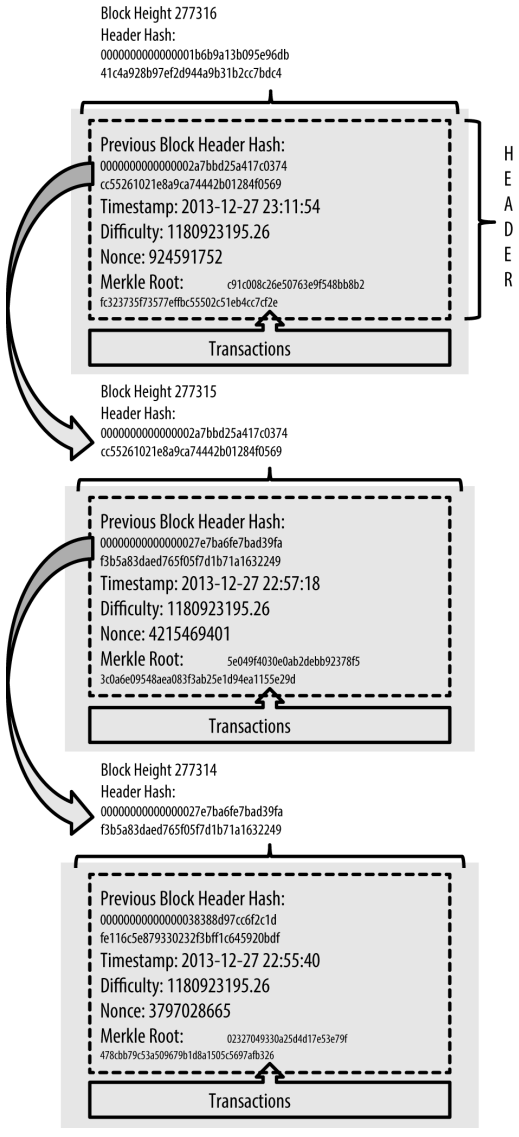


Figure 1. Blocs liés dans une chaîne par référence au hachage d'en-tête de bloc précédent

Les arbres Merkle sont utilisés dans Bitcoin pour résumer toutes les transactions d'un bloc, produisant une empreinte numérique globale de l'ensemble des transactions, fournissant un processus très efficace pour vérifier si une transaction est incluse dans un bloc. Un arbre merkle est construit en hachant récursivement des paires de nœuds jusqu'à ce qu'il n'y ait qu'un seul hachage, appelé *racine* ou *racine merkle* . L'algorithme de hachage cryptographique utilisé dans les arbres de merkle de Bitcoin est SHA256 appliqué deux fois, également connu sous le nom de double-SHA256.

Lorsque N éléments de données sont hachés et résumés dans un arbre merkle , vous pouvez vérifier si un élément de données est inclus dans l'arbre avec au plus 2 * log ~ 2 ~ (N) calculs, ce qui en fait une structure de données très efficace.

L' arbre de merkle est construit de bas en haut. Dans l'exemple suivant, nous commençons par quatre transactions, A, B, C et D, qui forment les *feuilles* de l' arborescence merkle , comme indiqué dans Calcul des nœuds dans un arbre merkle . Les transactions ne sont pas stockées dans l' arborescence merkle ; au lieu de cela, leurs données sont hachées et le hachage résultant est stocké dans chaque nœud feuille comme H_A , H_B, H_C et H_D :

```
HA = SHA256(SHA256(Transaction A))
```

Les paires consécutives de nœuds feuilles sont ensuite résumées dans un nœud parent, en concaténant les deux hachages et en

les hachant ensemble. Par exemple, pour construire le nœud parent H_{AB} , les deux hachages de 32 octets des enfants sont concaténés pour créer une chaîne de 64 octets. Cette chaîne est ensuite double-hachée pour produire le hachage du nœud parent :

```
HAB = SHA256(SHA256(HA + HB))
```

Le processus se poursuit jusqu'à ce qu'il n'y ait qu'un seul nœud en haut, le nœud connu sous le nom de racine de merkle . Ce hachage de 32 octets est stocké dans l'en-tête de bloc et résume toutes les données des quatre transactions. Le calcul des nœuds dans une arborescence de merkle montre comment la racine est calculée par des hachages par paires des nœuds.

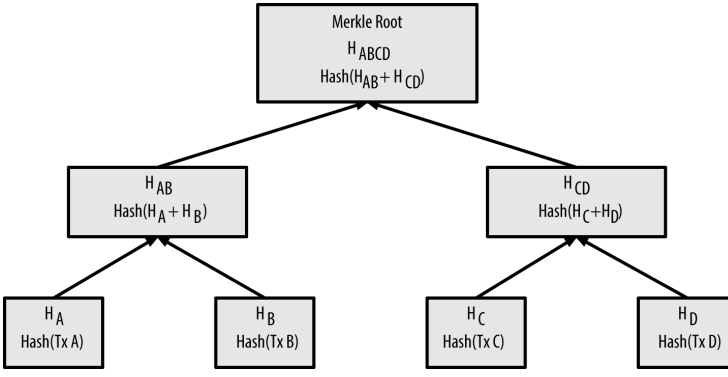


Figure 2. Calcul des nœuds dans une arborescence de merkle

Parce que le Merkle arbre est un arbre binaire, il a besoin d' un nombre pair de nœuds feuilles. S'il y a un nombre impair de transactions à résumer, le dernier hachage de transaction sera dupliqué pour créer un nombre pair de nœuds feuilles, également appelé *arbre équilibré* . Ceci est illustré dans La duplication d'un élément de données permet d'obtenir un nombre pair d'éléments de données , où la transaction C est dupliquée.

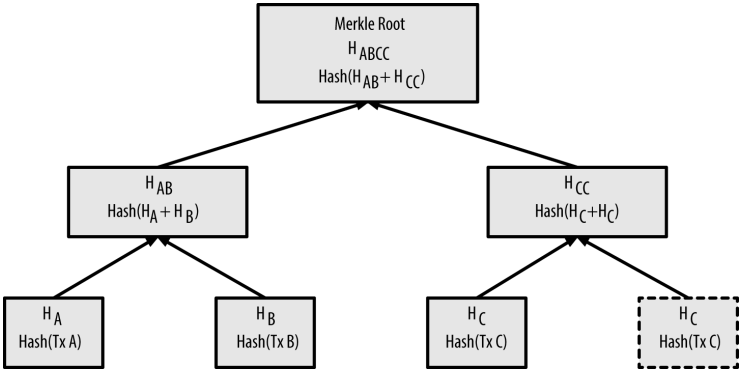


Figure 3. La duplication d'un élément de données permet d'obtenir un nombre pair d' éléments de données

La même méthode pour construire un arbre à partir de quatre transactions peut être généralisée pour construire des arbres de toute taille. Dans Bitcoin, il est courant d'avoir plusieurs centaines à plus d'un millier de transactions dans un seul bloc, qui sont résumées exactement de la même manière, ne produisant que 32 octets de données en tant que racine de

merkle unique . Dans Un arbre merkle résumant de nombreux éléments de données , vous verrez un arbre construit à partir de 16 transactions. Notez que bien que la racine semble plus grande que les nœuds feuilles dans le diagramme, elle a exactement la même taille, seulement 32 octets. Qu'il y ait une transaction ou cent mille transactions dans le bloc, la racine merkle les résume toujours en 32 octets.

Pour prouver qu'une transaction spécifique est incluse dans un bloc, un nœud n'a besoin que de produire des hachages log ~ 2 ~ (N) 32 octets, constituant un *chemin d'authentification* ou *un chemin de merkle* reliant la transaction spécifique à la racine de l'arbre. Ceci est d'autant plus important que le nombre de transactions augmente, car le logarithme en base 2 du nombre de transactions augmente beaucoup plus lentement. Cela permet aux nœuds bitcoin de produire efficacement des chemins de 10 ou 12 hachages (320 à 384 octets), qui peuvent fournir la preuve d'une seule transaction sur plus de mille transactions dans un bloc de la taille d'un mégaoctet.

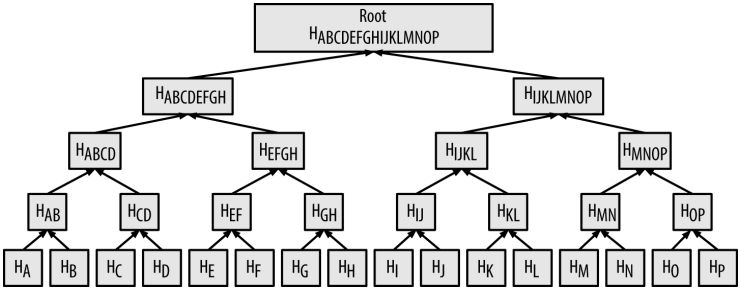


Figure 4. Un arbre de merkle résumant de nombreux éléments de données

Dans un chemin de merkle utilisé pour prouver l'inclusion d'un élément de données , un nœud peut prouver qu'une transaction K est incluse dans le bloc en produisant un chemin de merkle qui ne fait que quatre hachages de 32 octets (128 octets au total). Le chemin se compose des quatre hachages (représentés avec un arrière-plan ombré dans un chemin de merkle utilisé pour prouver l'inclusion d'un élément de données)H_L, H_{IJ}, H_{MNOP}, and $H_{ABCDEFGH}$. Avec ces quatre hachages fournis comme chemin d'authentification, n'importe quel nœud peut prouver que H_K (avec un fond noir en bas du diagramme) est inclus dans la racine de merkle en calculant quatre hachages par paire supplémentaires H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, et la racine de l'arborescence merkle (soulignée en pointillés dans le diagramme).

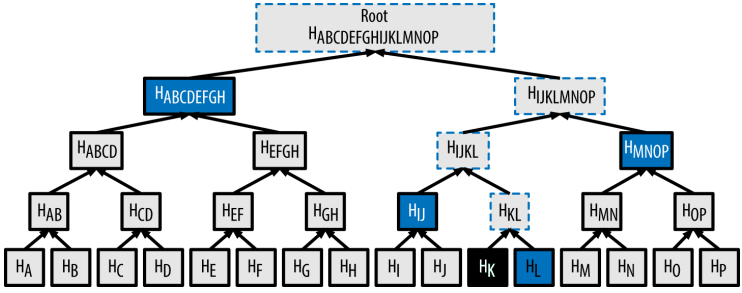


Figure 5. Un chemin de merkle utilisé pour prouver l'inclusion d'un élément de données

Le code de Construction d'un arbre merkle montre le processus de création d'un arbre merkle à partir des hachages des

nœuds feuilles jusqu'à la racine, en utilisant la bibliothèque libbitcoin pour certaines fonctions d'assistance.

Exemple 1. Construction d'un arbre de merkle

```
link:code/merkle.cpp[]
```

La compilation et l'exécution de l' exemple de code merkle montre le résultat de la compilation et de l'exécution du code merkle .

Exemple 2. Compilation et exécution de l' exemple de code merkle

```
# Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags
--libs libbitcoin)# Run the merkle executable
$ ./merkle
Current merkle hash list:
  32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d270
  30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540

Current merkle hash list:
  d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba

Result:
d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

L'efficacité des arbres merkle devient évidente à mesure que l'échelle augmente. L'efficacité de l'arborescence de Merkle

indique la quantité de données qui doit être échangée en tant que chemin de merkle pour prouver qu'une transaction fait partie d'un bloc.

Number of transactions	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

Tableau 3. Efficacité de l'arbre de Merkle

Comme vous pouvez le voir dans le tableau, alors que la taille du bloc augmente rapidement, de 4 Ko avec 16 transactions à une taille de bloc de 16 Mo pour s'adapter à 65535 transactions, le chemin de merkle requis pour prouver l'inclusion d'une transaction augmente beaucoup plus lentement, de 128 octets à seulement 512 octets. Avec les arbres merkle , un nœud peut télécharger uniquement les en-têtes de bloc (80 octets par bloc) et être toujours capable d'identifier l'inclusion d'une transaction dans un bloc en récupérant un petit chemin merkle à partir d'un nœud complet, sans stocker ni transmettre la grande majorité des blockchain, qui peut avoir une taille de plusieurs gigaoctets. Les nœuds qui ne maintiennent pas une blockchain complète, appelés nœuds de vérification simplifiée des paiements (SPV), utilisent des chemins de merkle pour vérifier les transactions

sans télécharger des blocs complets.

Arbres Merkle et vérification simplifiée des paiements (SPV)

Les arbres Merkle sont largement utilisés par les nœuds SPV. Les nœuds SPV n'ont pas toutes les transactions et ne téléchargent pas de blocs complets, bloquent simplement les entêtes. Afin de vérifier qu'une transaction est incluse dans un bloc, sans avoir à télécharger toutes les transactions dans le bloc, ils utilisent un chemin d'authentification, ou chemin de merkle .

Prenons, par exemple, un nœud SPV qui s'intéresse aux paiements entrants à une adresse contenue dans son portefeuille. Le nœud SPV établira un filtre de floraison (voir [bloom_filters]) sur ses connexions aux pairs pour limiter les transactions reçues à seulement celles contenant des adresses d'intérêt. Lorsqu'un pair voit une transaction qui correspond au filtre de floraison, il enverra ce bloc en utilisant un message merkleblock . Le message merkleblock contient l'en-tête du bloc ainsi qu'un chemin merkle qui relie la transaction d'intérêt à la racine merkle dans le bloc. Le nœud SPV peut utiliser ce chemin de merkle pour connecter la transaction au bloc et vérifier que la transaction est incluse dans le bloc. Le nœud SPV utilise également l'en-tête de bloc pour lier le bloc au reste de la blockchain. La combinaison de ces deux liens, entre la transaction et le bloc, et entre le bloc et la blockchain, prouve que la transaction est enregistrée dans la blockchain. Dans

l'ensemble, le nœud SPV aura reçu moins d'un kilo-octet de données pour l'en-tête de bloc et le chemin de merkle , une quantité de données plus de mille fois inférieure à un bloc complet (environ 1 mégaoctet actuellement).

Blockchains de test de Bitcoin

Vous serez peut-être surpris d'apprendre qu'il existe plus d'une blockchain Bitcoin. La blockchain bitcoin « principale », celle créée par Satoshi Nakamoto le 3 janvier 2009, celle avec le bloc de genèse que nous avons étudié dans ce chapitre, s'appelle *mainnet* . Il existe d'autres blockchains Bitcoin qui sont utilisées à des fins de test : actuellement *testnet* , *segnet* et *regtest* . Examinons chacun à son tour.

Testnet - Le terrain de jeu de test de Bitcoin

Testnet est le nom de la blockchain, du réseau et de la devise de test utilisés à des fins de test. Le testnet est un réseau P2P en direct complet, avec des portefeuilles, des bitcoins de test (pièces de testnet), l'exploitation minière et toutes les autres fonctionnalités du réseau principal . Il n'y a en réalité que deux différences : les pièces testnet sont censées être sans valeur et la difficulté d'extraction doit être suffisamment faible pour que tout le monde puisse extraire des pièces testnet relativement facilement (en les gardant sans valeur).

Tout développement logiciel destiné à une utilisation en production sur le réseau principal de Bitcoin doit d'abord être

testé sur testnet avec des pièces de test. Cela protège à la fois les développeurs des pertes monétaires dues aux bogues et le réseau des comportements involontaires dus aux bogues.

Garder les pièces sans valeur et l'extraction facile, cependant, n'est pas facile. Malgré les appels des développeurs, certaines personnes utilisent des équipements miniers avancés (GPU et ASIC) pour miner sur testnet . Cela augmente la difficulté, rend impossible l'extraction avec un processeur et rend finalement assez difficile l'obtention de pièces de test que les gens commencent à les valoriser, donc elles ne sont pas sans valeur. En conséquence, de temps en temps, le testnet doit être mis au rebut et redémarré à partir d'un nouveau bloc de genèse, ce qui réinitialise la difficulté.

Le testnet actuel est appelé *testnet3* , la troisième itération de testnet , redémarré en février 2011 pour réinitialiser la difficulté du testnet précédent .

Gardez à l'esprit que testnet3 est une grande blockchain, dépassant 25 Go en 2021. Il faudra environ un jour pour se synchroniser complètement et utiliser les ressources de votre ordinateur. Pas autant que le réseau principal , mais pas vraiment "léger" non plus. Un bon moyen d'exécuter un nœud testnet est de créer une image de machine virtuelle (par exemple, VirtualBox, Docker, Cloud Server, etc.) dédiée à cet effet.

Utilisation de testnet

Bitcoin Core, comme presque tous les autres logiciels Bitcoin, prend entièrement en charge le fonctionnement sur testnet au lieu de mainnet . Toutes les fonctions de Bitcoin Core

fonctionnent sur testnet , y compris le portefeuille, l'extraction de pièces de testnet et la synchronisation d'un nœud testnet complet .

Pour démarrer Bitcoin Core sur testnet au lieu de mainnet, vous utilisez le commutateur testnet :

```
$ bitcoind -testnet
```

Dans les journaux, vous devriez voir que bitcoind construit une nouvelle blockchain dans le sous-répertoire testnet3 du répertoire bitcoind par défaut :

```
bitcoind: Using data directory
/home/username/.bitcoin/testnet3
```

Pour vous connecter à bitcoind , vous utilisez l'outil de ligne de commande bitcoin-cli, mais vous devez également le basculer en mode testnet :

```
$ bitcoin-cli -testnet getblockchaininfo
{
  "chain": "test",
  "blocks": 1088,
  "headers": 139999,
  "bestblockhash":
  "0000000063d29909d475a1c4ba26da64b368e56cce5d925097bf3a208437012
  "difficulty": 1,
  "mediantime": 1337966158,
  "verificationprogress": 0.001644065914099759,
```

```
"chainwork":
"0000000000000000000000000000000000000000000000000000044104410
"pruned": false,
"softforks": [

[...]
```

Vous pouvez également exécuter sur testnet3 avec d'autres implémentations de nœuds complets, telles que btcd (écrit en Go) et bcoin (écrit en JavaScript), pour expérimenter et apprendre dans d'autres langages et frameworks de programmation.

En 2021, testnet3 prend en charge toutes les fonctionnalités du réseau principal , y compris le témoin séparé. Par conséquent, testnet3 peut également être utilisé pour tester les fonctionnalités des témoins séparés.

Segnet : le réseau de test des témoins séparés

En 2016, un réseau de test spécialisé a été lancé pour aider au développement et au test de Segregated Witness (alias segwit ; voir [segwit]). Cette blockchain de test s'appelle segnet et peut être rejointe en exécutant une version spéciale (branche) de Bitcoin Core.

Depuis que segwit a été ajouté à testnet3, il n'est plus nécessaire d'utiliser segnet pour tester les fonctionnalités de segwit .

À l'avenir, il est probable que nous verrons d'autres blockchains de testnet spécialement conçues pour tester une seule fonction-

nalité ou un changement architectural majeur, comme le segnet .

Regtest : la blockchain locale

Regtest , qui signifie « Regression Testing », est une fonctionnalité Bitcoin Core qui vous permet de créer une blockchain locale à des fins de test. Contrairement à testnet3, qui est une blockchain de test publique et partagée, les blockchains regtest sont destinées à être exécutées en tant que systèmes fermés pour des tests locaux. Vous lancez une blockchain regtest à partir de zéro, créant un bloc de genèse local. Vous pouvez ajouter d'autres nœuds au réseau ou l' exécuter avec un seul nœud uniquement pour tester le logiciel Bitcoin Core.

Pour démarrer Bitcoin Core en mode regtest , vous utilisez l' indicateur regtest :

```
$ bitcoind -regtest
```

Tout comme avec testnet , Bitcoin Core initialisera une nouvelle blockchain dans le sous- répertoire *regtest* de votre répertoire par défaut bitcoind :

```
bitcoind: Using data directory
/home/username/.bitcoin/regtest
```

Pour utiliser l'outil de ligne de commande, vous devez également spécifier l' indicateur de regtest . Essayons la commande

getblockchaininfo pour inspecter la blockchain regtest :

```
$ bitcoin-cli -regtest getblockchaininfo
{
  "chain": "regtest",
  "blocks": 0,
  "headers": 0,
  "bestblockhash":
  "0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e22
  "difficulty": 4.656542373906925e-10,
  "mediantime": 1296688602,
  "verificationprogress": 1,
  "chainwork":
  "0000000000000000000000000000000000000000000000000000000000000
  "pruned": false,
  [...]
```

Comme vous pouvez le voir, il n'y a pas encore de blocages. Explorons-en quelques-uns (500 blocs) et gagnez la récompense :

```
$ bitcoin-cli -regtest generate 500
[
  "7afed70259f22c2bf11e406cb12ed5c0657b6e16a6477a9f8b28e2046b5ba1
  "1aca2f154a80a9863a9aac4c72047a6d3f385c4eec5441a4aafa6acaa1dada
  "4334ecf6fb022f30fbd764c3ee778fabbd53b4a4d1950eae8a91f1f5158ed2
  "5f951d34065efeaf64e54e91d00b260294fcdfc7f05dbb5599aec84b957a77
  "43744b5e77c1dfece9d05ab5f0e6796ebe627303163547e69e27f55d0f2b93
   [...]
  "6c31585a48d4fc2b3fd25521f4515b18aefb59d0def82bd9c2185c4ecb7543
]
```

Il ne faudra que quelques secondes pour extraire tous ces blocs,

ce qui facilite certainement les tests. Si vous vérifiez le solde de votre portefeuille, vous verrez que vous avez gagné une récompense pour les 400 premiers blocs (les récompenses de coinbase doivent avoir une profondeur de 100 blocs avant de pouvoir les dépenser) :

```
$ bitcoin-cli -regtest getbalance
12462.50000000
```

Utilisation des chaînes de blocs de test pour le développement

Les différentes chaînes de blocs de Bitcoin (regtest , segnet , testnet3, mainnet) offrent une gamme d'environnements de test pour le développement de Bitcoin. Utilisez les blockchains de test que vous développiez pour Bitcoin Core ou pour un autre client de consensus à nœud complet ; une application telle qu'un portefeuille, une bourse, un site de commerce électronique ; ou même le développement de nouveaux contrats intelligents et de scripts complexes.

Vous pouvez utiliser les blockchains de test pour établir un pipeline de développement. Testez votre code localement sur un regtest au fur et à mesure que vous le développez. Une fois que vous êtes prêt à l'essayer sur un réseau public, passez à testnet pour exposer votre code à un environnement plus dynamique avec plus de diversité de code et d'applications. Enfin, une fois

que vous êtes sûr que votre code fonctionne comme prévu, passez au réseau principal pour le déployer en production. Au fur et à mesure que vous apportez des modifications, des améliorations, des corrections de bogues, etc., redémarrez le pipeline, en déployant chaque changement d'abord sur regtest , puis sur testnet et enfin en production.

11

Exploitation minière et consensus

Introduction

Le mot « exploitation minière » est quelque peu trompeur. En évoquant l'extraction des métaux précieux, il concentre notre attention sur la récompense de l'exploitation minière, le nouveau bitcoin créé dans chaque bloc. Bien que l'exploitation minière soit incitée par cette récompense, le but principal de l'exploitation minière n'est pas la récompense ou la génération de nouvelles pièces. Si vous considérez le minage uniquement comme le processus par lequel les pièces sont créées, vous confondez les moyens (incitations) comme l'objectif du processus. L'exploitation minière est le mécanisme qui sous-tend la chambre de compensation décentralisée, par lequel les transactions sont validées et compensées. L'exploitation minière est l'invention qui rend le bitcoin spécial, un mécanisme de sécurité décentralisé qui est à la base de l'argent numérique P2P.

Le minage *sécurise le système Bitcoin* et permet l'émergence d'un *consensus à l'* échelle du réseau *sans autorité centrale* . La récompense des pièces de monnaie nouvellement frappées et des frais de transaction est un système d'incitation qui aligne les actions des mineurs avec la sécurité du réseau, tout en mettant en œuvre simultanément l'offre monétaire.

Conseil

Le but du minage n'est pas la création de nouveau bitcoin. C'est le système d'incitation. L'exploitation minière est le mécanisme par lequel la sécurité *de Bitcoin est* décentralisée

Les mineurs valident les nouvelles transactions et les enregistrent dans le grand livre global. Un nouveau bloc, contenant les transactions qui ont eu lieu depuis le dernier bloc, est « miné » toutes les 10 minutes en moyenne, ajoutant ainsi ces transactions à la blockchain. Les transactions qui font partie d'un bloc et ajoutées à la blockchain sont considérées comme « confirmées », ce qui permet aux nouveaux propriétaires de bitcoin de dépenser le bitcoin qu'ils ont reçu dans ces transactions.

Les mineurs reçoivent deux types de récompenses en échange de la sécurité fournie par l'exploitation minière : de nouvelles pièces créées avec chaque nouveau bloc, également appelées *récompense de bloc* ou *récompense de coinbase* , et les frais de transaction de toutes les transactions incluses dans le bloc. Pour gagner cette récompense, les mineurs s'affrontent pour résoudre un problème mathématique difficile basé sur un algo-

rithme de hachage cryptographique. La solution au problème, appelée Proof-of-Work, est incluse dans le nouveau bloc et sert de preuve que le mineur a dépensé un effort informatique important. La concurrence pour résoudre l'algorithme de preuve de travail pour gagner la récompense et le droit d'enregistrer les transactions sur la blockchain est la base du modèle de sécurité de Bitcoin.

Le processus est appelé minage parce que la récompense (nouvelle génération de pièces) est conçue pour simuler des rendements décroissants, tout comme l'extraction de métaux précieux. La masse monétaire de Bitcoin est créée par l'exploitation minière, de la même manière qu'une banque centrale émet de la nouvelle monnaie en imprimant des billets de banque. La quantité maximale de bitcoin nouvellement créé qu'un mineur peut ajouter à un bloc diminue environ tous les quatre ans (ou précisément tous les 210000 blocs). Il a commencé à 50 bitcoins par bloc en janvier 2009 et réduit de moitié à 25 bitcoins par bloc en novembre 2012. Il a diminué de moitié à 12,5 bitcoins en juillet 2016 et à nouveau à 6,25 bitcoins en mai 2020. Sur la base de cette formule, les récompenses minières en bitcoins diminuent de façon exponentielle jusqu'en 2140 environ, date à laquelle tous les bitcoins (20,99999998 millions) auront été émis. Après 2140, aucun nouveau bitcoin ne sera émis.

Les mineurs de Bitcoin perçoivent également des frais sur les transactions. Chaque transaction comprend généralement des frais de transaction, sous la forme d'un excédent de bitcoin entre les entrées et les sorties de la transaction. Le mineur bitcoin gagnant peut "conserver la monnaie" sur les transactions incluses dans le bloc gagnant. Aujourd'hui, les frais représentent

0,5% ou moins du revenu d'un mineur de bitcoin, la grande majorité provenant du bitcoin nouvellement créé. Cependant, à mesure que la récompense de bloc diminue au fil du temps et que le nombre de transactions par bloc augmente, une plus grande proportion des revenus miniers de Bitcoin proviendra des frais. Peu à peu, la récompense minière sera dominée par les frais de transaction, qui constitueront la principale incitation pour les mineurs. Après 2140, la quantité de nouveaux bitcoins dans chaque bloc tombe à zéro et l'extraction de bitcoins ne sera encouragée que par les frais de transaction.

Dans ce chapitre, nous examinerons d'abord l'exploitation minière en tant que mécanisme d'offre monétaire, puis la fonction la plus importante de l'exploitation minière : le mécanisme de consensus décentralisé qui sous-tend la sécurité du bitcoin.

Pour comprendre l'exploitation minière et le consensus, nous suivrons la transaction d'Alice au fur et à mesure qu'elle est reçue et ajoutée à un bloc par l'équipement minier de Jing. Ensuite, nous suivrons le bloc au fur et à mesure qu'il est extrait, ajouté à la blockchain et accepté par le réseau Bitcoin à travers le processus de consensus émergent.

Économie Bitcoin et création de devises

Les Bitcoins sont « frappés » lors de la création de chaque bloc à un taux fixe et décroissant. Chaque bloc, généré en moyenne toutes les 10 minutes, contient du bitcoin entièrement nouveau, créé à partir de rien. Tous les 210000 blocs, ou environ tous les quatre ans, le taux d'émission de devises est diminué de 50%. Pendant les quatre premières années de fonctionnement du

réseau, chaque bloc contenait 50 nouveaux bitcoins.

En novembre 2012, le nouveau taux d'émission de bitcoins a été ramené à 25 bitcoins par bloc. En juillet 2016, il a été ramené à 12,5 bitcoins par bloc, et en mai 2020, il a de nouveau été ramené à 6,25 bitcoins par bloc. Le taux des nouvelles pièces diminue de cette manière exponentiellement sur 32 "moitiés" jusqu'au bloc 6 720 000 (extrait approximativement en 2137), lorsqu'il atteint l'unité monétaire minimale de 1 satoshi. Enfin, après 6,93 millions de blocs, en environ 2140, près de 2 099 999 997 690 000 satoshis, soit près de 21 millions de bitcoins, seront émis. Par la suite, les blocs ne contiendront aucun nouveau bitcoin et les mineurs seront récompensés uniquement par les frais de transaction. L'offre de monnaie bitcoin au fil du temps sur la base d'un taux d'émission géométriquement décroissant montre le bitcoin total en circulation au fil du temps, à mesure que l'émission de monnaie diminue.

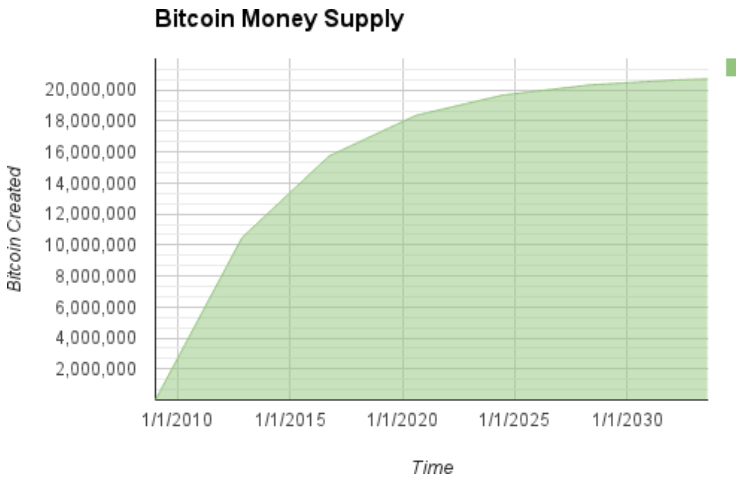


Figure 1. Offre de monnaie bitcoin au fil du temps sur la base d'un taux d'émission géométriquement décroissant

Note

Le nombre maximum de pièces extraites est la limite supérieure *des récompenses minières possibles pour Bitcoin. En pratique, un mineur peut intentionnellement miner un bloc en prenant moins que la pleine récompense. Ces blocs ont déjà été minés et d'autres pourraient l'être à l'avenir, ce qui entraînera une émission totale moindre de la monnaie.*

Dans l'exemple de code ci-dessus nommé "Un script pour calculer le montant total de bitcoin qui sera émis" , nous calculons le montant total de bitcoin qui sera émis.

Exemple 1. Un script pour calculer la quantité totale de bitcoins qui sera émise

```
link:code/max_money.py[]
```

L'exécution du script max_money.py affiche la sortie produite par l'exécution de ce script.

Exemple 2. Exécution du script max_money.py

```
$ python max_money.py
Total BTC to ever be created: 2100000000000000.0
Satoshis
```

L'émission finie et décroissante crée une offre monétaire fixe qui résiste à l'inflation. Contrairement à une monnaie fiduciaire, qui peut être imprimée en nombre infini par une banque centrale, le bitcoin ne peut jamais être gonflé par impression.

Argent déflationniste

La conséquence la plus importante et débattue des émissions monétaires fixes et décroissantes est que la monnaie a tendance à être intrinsèquement *déflationniste* . La déflation est le phénomène d'appréciation de la valeur due à une inadéquation de l'offre et de la demande qui fait monter la valeur (et le taux de change) d'une monnaie. Le contraire de l'inflation, la déflation des prix, signifie que la monnaie a plus de pouvoir d'achat au fil du temps.

De nombreux économistes affirment qu'une économie défla-

tionniste est un désastre qui doit être évité à tout prix. En effet, dans une période de déflation rapide, les gens ont tendance à accumuler de l'argent au lieu de le dépenser, dans l'espoir que les prix chuteront. Un tel phénomène s'est déroulé pendant la « décennie perdue » du Japon, lorsqu'un effondrement complet de la demande a poussé la monnaie dans une spirale déflationniste.

Les experts Bitcoin affirment que la déflation n'est pas mauvaise en soi. La déflation est plutôt associée à un effondrement de la demande car c'est le seul exemple de déflation que nous devons étudier. Dans une monnaie fiduciaire avec possibilité d'impression illimitée, il est très difficile d'entrer dans une spirale déflationniste à moins qu'il n'y ait un effondrement complet de la demande et une réticence à imprimer de la monnaie. La déflation du bitcoin n'est pas causée par un effondrement de la demande, mais par une offre limitée de manière prévisible.

L'aspect positif de la déflation, bien sûr, est qu'elle est le contraire de l'inflation. L'inflation provoque une dépréciation lente mais inévitable de la monnaie, entraînant une forme d'imposition cachée qui punit les épargnants afin de renflouer les débiteurs (y compris les plus gros débiteurs, les gouvernements eux-mêmes). Les monnaies sous contrôle gouvernemental souffrent de l'aléa moral d'une émission de dette facile qui peut plus tard être effacée par une dépréciation au détriment des épargnants.

Il reste à voir si l'aspect déflationniste de la monnaie est un problème quand il n'est pas entraîné par une rétraction

économique rapide, ou un avantage parce que la protection contre l'inflation et l'avilissement l'emporte largement sur les risques de déflation.

Consensus décentralisé

Dans le chapitre précédent, nous avons examiné la blockchain, le grand livre public mondial (liste) de toutes les transactions, que tout le monde dans le réseau Bitcoin accepte comme l'enregistrement de propriété faisant autorité.

Mais comment tout le monde dans le réseau peut-il s'entendre sur une seule « vérité » universelle sur qui possède quoi, sans avoir à faire confiance à personne ? Tous les systèmes de paiement traditionnels dépendent d'un modèle de confiance qui a une autorité centrale fournissant un service de chambre de compensation, vérifiant et compensant essentiellement toutes les transactions. Bitcoin n'a pas d'autorité centrale, mais d'une manière ou d'une autre, chaque nœud complet a une copie complète d'un grand livre public auquel il peut faire confiance comme enregistrement faisant autorité. La blockchain n'est pas créée par une autorité centrale, mais est assemblée indépendamment par chaque nœud du réseau. D'une manière ou d'une autre, chaque nœud du réseau, agissant sur les informations transmises via des connexions réseau non sécurisées, peut arriver à la même conclusion et assembler une copie du même registre public que tout le monde. Ce chapitre examine le processus par lequel le réseau Bitcoin parvient à un consensus mondial sans autorité centrale.

La principale invention de Satoshi Nakamoto est le mécanisme décentralisé d' *un consensus émergent* . Emergent, car le consensus n'est pas atteint explicitement - il n'y a pas d'élections ou de moment fixe où le consensus se produit. Au lieu de cela, le consensus est un artefact émergent de l'interaction asynchrone de milliers de nœuds indépendants, tous suivant des règles simples. Toutes les propriétés du bitcoin, y compris la monnaie, les transactions, les paiements et le modèle de sécurité qui ne dépend pas de l'autorité centrale ou de la confiance, découlent de cette invention.

Le consensus décentralisé de Bitcoin émerge de l'interaction de quatre processus qui se produisent indépendamment sur les nœuds du réseau :

- Vérification indépendante de chaque transaction, par chaque nœud complet, sur la base d'une liste complète de critères
- Agrégation indépendante de ces transactions dans de nouveaux blocs par des nœuds miniers, associée à un calcul démontré via un algorithme de preuve de travail
- Vérification indépendante des nouveaux blocs par chaque nœud et assemblage dans une chaîne
- Sélection indépendante, par chaque nœud, de la chaîne avec le calcul le plus cumulatif démontré par Proof-of-Work

Dans les prochaines sections, nous examinerons ces processus et la manière dont ils interagissent pour créer la propriété émergente du consensus à l'échelle du réseau qui permet à tout nœud Bitcoin d'assembler sa propre copie du grand livre mondial faisant autorité, de confiance et public.

Vérification indépendante des transactions

Dans [le chapitre "transactions"] , nous avons vu comment le logiciel de portefeuille crée des transactions en collectant UTXO, en fournissant les scripts de déverrouillage appropriés, puis en construisant de nouvelles sorties attribuées à un nouveau propriétaire. La transaction résultante est ensuite envoyée aux nœuds voisins du réseau bitcoin afin qu'elle puisse être propagée sur l'ensemble du réseau bitcoin.

Cependant, avant de transmettre les transactions à ses voisins, chaque nœud bitcoin qui reçoit une transaction vérifiera d'abord la transaction. Cela garantit que seules les transactions valides sont propagées sur le réseau, tandis que les transactions non valides sont supprimées au premier nœud qui les rencontre.

Chaque nœud vérifie chaque transaction par rapport à une longue liste de contrôle de critères :

- La syntaxe et la structure des données de la transaction doivent être correctes.
- Aucune des listes d'entrées ou de sorties n'est vide.
- La taille de la transaction en octets est inférieure à MAX_BLOCK_SIZE.
- Chaque valeur de sortie, ainsi que le total, doit être dans la plage de valeurs autorisée (moins de 21 millions de pièces, plus que le seuil de *poussière*).
- Aucune des entrées n'a hash = 0, N = –1 (les transactions coinbase ne doivent pas être relayées).
- nLocktime est égal à INT_MAX, ou les valeurs nLocktime

et nSequence sont satisfaites selon MedianTimePast.
- La taille de la transaction en octets est supérieure ou égale à 82.
- Le nombre d'opérations de signature (SIGOPS) contenues dans la transaction est inférieur à la limite d'opération de signature.
- Le script de déverrouillage (scriptSig) ne peut pousser que des nombres sur la pile, et le script de verrouillage (scriptPubkey) doit correspondre aux formulaires IsStandard (cela rejette les transactions "non standard").
- Une transaction correspondante dans le pool ou dans un bloc de la branche principale doit exister.
- Pour chaque entrée, si la sortie référencée existe dans une autre transaction du pool, la transaction doit être rejetée.
- Pour chaque entrée, regardez dans la branche principale et le pool de transactions pour trouver sa transaction parente. Si la transaction parente est manquante pour une entrée, ce sera une transaction orpheline. Ajouter au pool de transactions orphelines, si une transaction correspondante n'est pas déjà dans le pool.
- Pour chaque entrée, si sa transaction parente est une transaction coinbase, elle doit avoir au moins COINBASE_MATURITY (100) confirmations.
- Pour chaque entrée, la sortie référencée doit exister et ne peut pas déjà être dépensée.
- En utilisant les transactions parentes pour obtenir les valeurs d'entrée, vérifiez que chaque valeur d'entrée, ainsi que la somme, se trouvent dans la plage de valeurs autorisée (moins de 21 millions de pièces, plus de 0).
- Rejeter si la somme des valeurs d'entrée est inférieure à la somme des valeurs de sortie.

- Refuser si les frais de transaction sont trop bas (minRelayTxFee) pour entrer dans un bloc vide.
- Les scripts de déverrouillage pour chaque entrée doivent être validés par rapport aux scripts de verrouillage de sortie correspondants.

Ces conditions peuvent être vues en détail dans les fonctions AcceptToMemoryPool, CheckTransaction et CheckInputs dans Bitcoin Core. Notez que les conditions changent avec le temps, pour faire face à de nouveaux types d'attaques par déni de service ou parfois pour assouplir les règles afin d'inclure davantage de types de transactions.

En vérifiant indépendamment à chaque transaction telle qu'elle est reçue et avant que la propager, chaque noeud construit un pool de valides (mais non confirmées) les opérations connues sous le *pool de transaction* , *pool de mémoire* , ou *MemPool* .

Nœuds de minage

Certains des nœuds du réseau bitcoin sont des nœuds spécialisés appelés *mineurs* . Dans [ch01_intro_what_is_bitcoin], nous avons présenté Jing, un étudiant en génie informatique à Shanghai, en Chine, qui est un mineur de bitcoin. Jing gagne des bitcoins en exécutant une « plate-forme minière », qui est un système informatique spécialisé conçu pour extraire des bitcoins. Le matériel minier spécialisé de Jing est connecté à un serveur exécutant un nœud bitcoin complet. Contrairement à Jing, certains mineurs minent sans nœud complet, comme nous

le verrons dans Mining Pools . Comme tous les autres nœuds complets, le nœud de Jing reçoit et propage des transactions non confirmées sur le réseau bitcoin. Le nœud de Jing, cependant, agrège également ces transactions dans de nouveaux blocs.

Le nœud de Jing est à l'écoute de nouveaux blocs, propagés sur le réseau bitcoin, comme le font tous les nœuds. Cependant, l'arrivée d'un nouveau bloc a une signification particulière pour un nœud minier. La compétition entre mineurs se termine effectivement par la propagation d'un nouveau bloc qui fait office d'annonce d'un gagnant. Pour les mineurs, recevoir un nouveau bloc valide signifie que quelqu'un d'autre a remporté la compétition et qu'il a perdu. Cependant, la fin d'un tour d'une compétition est également le début du tour suivant. Le nouveau bloc n'est pas seulement un drapeau à damier, marquant la fin de la course ; c'est aussi le pistolet de départ dans la course pour le prochain bloc.

Agrégation des transactions en blocs

Après avoir validé les transactions, un nœud bitcoin les ajoutera au *pool de mémoire* , ou *pool de transactions* , où les transactions attendent jusqu'à ce qu'elles puissent être incluses (minées) dans un bloc. Le nœud de Jing collecte, valide et relaie les nouvelles transactions comme n'importe quel autre nœud. Contrairement aux autres nœuds, cependant, le nœud de Jing agrégera ensuite ces transactions dans un *bloc candidat* .

Suivons les blocs créés à l'époque où Alice achetait une tasse de café au Bob's Cafe ; La transaction d'Alice a été incluse dans

le bloc 277,316. Dans le but de démontrer les concepts de ce chapitre, supposons que le bloc a été miné par le système de minage de Jing et suivez la transaction d'Alice au fur et à mesure qu'il fait partie de ce nouveau bloc.

Le nœud minier de Jing conserve une copie locale de la blockchain. Au moment où Alice achète la tasse de café, le nœud de Jing a assemblé une chaîne jusqu'au bloc 277314. Le nœud de Jing écoute les transactions, tente de miner un nouveau bloc et écoute également les blocs découverts par d'autres nœuds. Comme le nœud de Jing est minier, il reçoit le bloc 277315 via le réseau Bitcoin. L'arrivée de ce bloc signifie la fin du concours pour le bloc 277,315 et le début du concours pour créer le bloc 277,316.

Au cours des 10 minutes précédentes, alors que le nœud de Jing cherchait une solution au bloc 277 315, il collectait également des transactions en préparation du bloc suivant. À ce jour, il a collecté quelques centaines de transactions dans le pool de mémoire. Lors de la réception du bloc 277,315 et de sa validation, le nœud de Jing le comparera également à toutes les transactions dans le pool de mémoire et supprimera celles qui étaient incluses dans le bloc 277,315. Les transactions qui restent dans le pool de mémoire ne sont pas confirmées et attendent d'être enregistrées dans un nouveau bloc.

Le nœud de Jing construit immédiatement un nouveau bloc vide, un candidat pour le bloc 277,316. Ce bloc est appelé un *bloc candidat* car il n'est pas encore un bloc valide, car il ne contient pas de preuve de travail valide. Le bloc ne devient valide que si le mineur parvient à trouver une solution à l'algorithme de

preuve de travail.

Lorsque le nœud de Jing agrège toutes les transactions du pool de mémoire, le nouveau bloc candidat a 418 transactions avec des frais de transaction totaux de 0,09094928 bitcoin. Vous pouvez voir ce bloc dans la blockchain à l'aide de l'interface de ligne de commande du client Bitcoin Core, comme indiqué dans l'Utilisation de la ligne de commande pour récupérer le bloc 277,316 .

Exemple 3. Utilisation de la ligne de commande pour récupérer le bloc 277,316

```
$ bitcoin-cli getblockhash 277316

000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock
000000000000001b6b9a13b095e96db41c4a928b97ef2d9\
44a9b31b2cc7bdc4
```

```
{
    "hash" :
    "000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7
    "confirmations" : 35561,
    "size" : 218629,
    "height" : 277316,
    "version" : 2,
    "merkleroot" :
    "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7
    "tx" : [
```

```
        "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e742
        "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877

        ... 417 more transactions ...

      ],
    "time" : 1388185914,
    "nonce" : 924591752,
    "bits" : "1903a30c",
    "difficulty" : 1180923195.25802612,
    "chainwork" :
    "0000000000000000000000000000000000000000000000934695e92aaf53af
    "previousblockhash" :
    "0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0
}
```

La transaction Coinbase

La première transaction d'un bloc est une transaction spéciale, appelée *transaction coinbase* . Cette transaction est construite par le nœud de Jing et contient sa *récompense* pour l'effort minier.

> *Note*
>
> *Lorsque le bloc 277316 a été extrait, la récompense était de 25 bitcoins par bloc. Depuis lors, deux périodes de « réduction de moitié » se sont écoulées. La récompense de bloc est passée à 12,5 bitcoin en juillet 2016 et à 6,25 bitcoin en mai 2020.*

Le nœud de Jing crée la transaction coinbase comme un paiement à son propre portefeuille : “Payez l’adresse de Jing 25.09094928 bitcoin.” Le montant total de la récompense que Jing collecte pour l’extraction d’un bloc est la somme de la récompense coinbase (25 nouveaux bitcoins) et des frais de transaction (0,09094928) de toutes les transactions incluses dans le bloc, comme indiqué dans la transaction Coinbase .

Exemple 4. Transaction Coinbase

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f
1
```

```
{
    "hex" :
    "010000000100000000000000000000000000000000000000000000000000000000
    "txid" :
    "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424af
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "coinbase" :
            "03443b0403858402062f503253482f",
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 25.09094928,
```

```
            "n" : 0,
            "scriptPubKey" : {
                "asm" :
                "02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5c
                "hex" :
                "2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba
                "reqSigs" : 1,
                "type" : "pubkey",
                "addresses" : [
                    "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                ]
            }
        }
    ]
}
```


Contrairement aux transactions régulières, la transaction coinbase ne consomme pas (dépense) UTXO en tant qu'entrées. Au lieu de cela, il n'a qu'une seule entrée, appelée *coinbase* , qui crée du bitcoin à partir de rien. La transaction coinbase a une sortie, payable à la propre adresse bitcoin du mineur. La sortie de la transaction coinbase envoie la valeur de 25,09094928 bitcoin à l'adresse bitcoin du mineur ; dans ce cas, il s'agit de 1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N.

Récompense et frais Coinbase

Pour construire la transaction coinbase, le nœud de Jing calcule d'abord le montant total des frais de transaction en ajoutant toutes les entrées et sorties des 418 transactions qui ont été ajoutées au bloc. Les frais sont calculés comme suit :

```
Total Fees = Sum(Inputs) – Sum(Outputs)
```

Dans le bloc 277,316, le total des frais de transaction est de 0,09094928 bitcoin.

Ensuite, le nœud de Jing calcule la récompense correcte pour le nouveau bloc. La récompense est calculée en fonction de la hauteur du bloc, à partir de 50 bitcoins par bloc et réduite de moitié tous les 210000 blocs. Étant donné que ce bloc a une hauteur de 277316, la récompense correcte est de 25 bitcoins.

Le calcul peut être vu dans la fonction GetBlockSubsidy dans le client Bitcoin Core, comme indiqué dans Calcul de la récompense de bloc - Fonction GetBlockSubsidy, Bitcoin Core Client, main.cpp .

Exemple 5. Calcul de la récompense de bloc - Fonction GetBlockSubsidy, Bitcoin Core Client, main.cpp

```
CAmount GetBlockSubsidy(int nHeight, const
Consensus::Params& consensusParams)
{
    int halvings = nHeight /
    consensusParams.nSubsidyHalvingInterval;
    // Force block reward to zero when right shift is
    undefined.if (halvings >= 64)
        return 0;

    CAmount nSubsidy = 50 * COIN;
    // Subsidy is cut in half every 210,000 blocks
    which will occur approximately every 4 years.
```

```
    nSubsidy >>= halvings;
    return nSubsidy;
}
```

La subvention initiale est calculée en satoshis en multipliant 50 par la constante COIN (100 000 000 satoshis). Cela fixe la récompense initiale (nSubsidy) à 5 milliards de satoshis.

Ensuite, la fonction calcule le nombre de moitiés qui se sont produites en divisant la hauteur de bloc actuelle par l'intervalle de réduction de moitié (SubsidyHalvingInterval). Dans le cas du bloc 277 316, avec un intervalle de réduction de moitié tous les 210 000 blocs, le résultat est une réduction de moitié.

Le nombre maximum de moitiés autorisé est de 64, donc le code impose une récompense nulle (ne renvoie que les frais) si les 64 moitiés sont dépassées.

Ensuite, la fonction utilise l'opérateur de décalage binaire vers la droite pour diviser la récompense (nSubsidy) par deux pour chaque cycle de réduction de moitié. Dans le cas du bloc 277316, cela décalerait en binaire vers la droite la récompense de 5 milliards de satoshis une fois (une réduction de moitié) et résulterait en 2,5 milliards de satoshis, soit 25 bitcoin. L'opérateur de décalage binaire vers la droite est utilisé car il est plus efficace que plusieurs divisions répétées. Pour éviter un bug potentiel, l'opération de décalage est ignorée après 63 moitiés et la subvention est définie sur 0.

Enfin, la récompense coinbase (nSubsidy) est ajoutée aux frais de transaction (nFees) et la somme est retournée.

Conseil

Si le nœud minier de Jing écrit la transaction coinbase, qu'est-ce qui empêche Jing de se « récompenser » 100 ou 1000 bitcoins ? La réponse est qu'une récompense incorrecte ferait en sorte que le bloc soit considéré comme invalide par tout le monde, gaspillant l'électricité de Jing utilisée pour la preuve de travail. Jing ne peut dépenser la récompense que si le blocage est accepté par tout le monde.

Structure de la transaction Coinbase

Avec ces calculs, le nœud de Jing construit ensuite la transaction coinbase pour se payer 25,09094928 bitcoin.

Comme vous pouvez le voir dans la transaction Coinbase , la transaction Coinbase a un format spécial. Au lieu d'une entrée de transaction spécifiant un UTXO précédent à dépenser, il a une entrée « coinbase » . Comparons une entrée de transaction régulière avec une entrée de transaction coinbase. La structure d'une entrée de transaction "normale" montre la structure d'une entrée de transaction régulière, tandis que La structure d'une entrée de transaction coinbase montre la structure de l'entrée de transaction coinbase.

Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent, first one is 0
1–9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking script
4 bytes	Sequence Number	Usually set to 0xFFFFFFFF to opt out of BIP 125 and BIP 68

Tableau 1. La structure d'une entrée de transaction "normale"

Size	Field	Description
32 bytes	Transaction Hash	All bits are zero: Not a transaction hash reference
4 bytes	Output Index	All bits are ones: 0xFFFFFFFF
1–9 bytes (VarInt)	Coinbase Data Size	Length of the coinbase data, from 2 to 100 bytes
Variable	Coinbase Data	Arbitrary data used for extra nonce and mining tags. In v2 blocks; must begin with block height
4 bytes	Sequence Number	Set to 0xFFFFFFFF

Table 2. The structure of a coinbase transaction input

Dans une transaction coinbase, les deux premiers champs sont définis sur des valeurs qui ne représentent pas une référence UTXO. Au lieu d'un « hachage de transaction », le premier champ est rempli de 32 octets, tous mis à zéro. L '« index de sortie » est rempli de 4 octets, tous mis à 0xFF (255 décimal). Le "Unlocking Script" (scriptSig) est remplacé par des données coinbase, un champ de données utilisé par les mineurs, comme nous le verrons plus loin.

Données Coinbase

Les transactions Coinbase n'ont pas de champ de script de déverrouillage (aka, scriptSig). Au lieu de cela, ce champ est remplacé par les données coinbase, qui doivent être comprises entre 2 et 100 octets. À l'exception des premiers octets, le reste des données de la coinbase peut être utilisé par les mineurs comme ils le souhaitent ; ce sont des données arbitraires.

Dans le bloc de genèse, par exemple, Satoshi Nakamoto a ajouté le texte "The Times 03 / Jan / 2009 Chancellor on the brink of second bailout for banks" dans les données coinbase, en l'utilisant comme preuve de la date et pour transmettre un message. Actuellement, les mineurs utilisent les données coinbase pour inclure des valeurs nonce supplémentaires et des chaînes identifiant le pool de minage.

Les premiers octets de la coinbase étaient arbitraires, mais ce n'est plus le cas. Conformément à BIP-34, les blocs de version 2 (blocs avec le champ de version réglé sur 2) doivent contenir l'index de hauteur de bloc sous la forme d'une opération "push" de script au début du champ coinbase.

Dans le bloc 277,316, nous voyons que la coinbase (voir transaction Coinbase), qui se trouve dans le script de déverrouillage ou le champ scriptSig de l'entrée de transaction, contient la valeur hexadécimale 03443b0403858402062f5032 53482f. Décodons cette valeur.

Le premier octet, 03, indique au moteur d'exécution de script de pousser les trois octets suivants sur la pile de scripts. Les trois octets suivants, 0x443b04, sont la hauteur de bloc codée au format little-endian (octet le moins significatif en arrière en premier). Inversez l'ordre des octets et le résultat est 0x043b44, soit 277316 en décimal.

Les quelques chiffres hexadécimaux suivants (0385840206) sont utilisés pour coder un *nonce* supplémentaire (voir la solution Extra Nonce), ou une valeur aléatoire, utilisée pour trouver une solution de preuve de travail appropriée.

La dernière partie des données coinbase (2f503253482f) est la chaîne encodée en ASCII / P2SH / , qui indique que le nœud d'exploration de données qui a extrait ce bloc prend en charge l'amélioration P2SH définie dans BIP-16. L'introduction de la capacité P2SH exigeait une signalisation par les mineurs pour approuver le BIP-16 ou le BIP-17. Ceux qui approuvaient la mise en œuvre du BIP-16 devaient inclure la chaîne / P2SH / dans leurs données coinbase. Ceux qui approuvaient la mise en œuvre BIP-17 de P2SH devaient inclure la chaîne p2sh / CHV dans leurs données coinbase. Enfin, le BIP-16 a été élu vainqueur et de nombreux mineurs ont continué à inclure la chaîne / P2SH / dans leur coinbase pour indiquer qu'ils fournissent un support pour cette fonctionnalité.

Extraire les données coinbase du bloc genesis utilise la bibliothèque libbitcoin introduite dans [alt_libraries] pour extraire les données coinbase du bloc genesis, affichant le message de Satoshi. Notez que la bibliothèque libbitcoin contient une copie statique du bloc genesis, donc l'exemple de code peut récupérer le bloc genesis directement à partir de la bibliothèque.

Exemple 6. Extraire les données coinbase du bloc genesis

```
link:code/satoshi-words.cpp[]
```

Nous compilons le code avec le compilateur GNU C ++ et exécutons l'exécutable résultant, comme indiqué dans Compilation et exécution de l'exemple de code satoshi-words .

Exemple 7. Compilation et exécution de l'exemple de code satoshi-words

```
# Compile the code
$  g++ -o satoshi-words satoshi-words.cpp
$(pkg-config --cflags --libs libbitcoin)# Run the
executable
$ ./satoshi-words
^D<GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink
of second bailout for banks
```

Construire l'en-tête de bloc

Pour construire l'en-tête de bloc, le nœud d'exploration de données doit remplir six champs, comme indiqué dans La structure de l'en-tête de bloc .

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Target	The Proof-of-Work algorithm target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

Tableau 3. La structure de l'en-tête de bloc

Au moment où ce bloc 277,316 a été extrait, le numéro de version décrivant la structure du bloc est la version 2, qui est codée au format little-endian sur 4 octets sous la forme 0x02000000.

Ensuite, le nœud d'exploration de données doit ajouter le "Hash de bloc précédent" (également connu sous le nom de prevhash). C'est le hachage de l'en-tête de bloc du bloc 277,315, le bloc précédent reçu du réseau, que le nœud de Jing a accepté et sélectionné comme *parent* du bloc candidat 277,316. Le hachage d'en-tête de bloc pour le bloc 277,315 est :

0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569

Conseil

En sélectionnant le bloc parent *spécifique , indiqué par le champ Hash du bloc précédent dans l'en-tête du bloc candidat, Jing engage sa puissance de minage pour étendre la chaîne qui se termine dans ce bloc spécifique. En substance, c'est ainsi que Jing « vote » avec sa puissance minière pour la chaîne valide de difficulté la plus longue.*

L'étape suivante consiste à résumer toutes les transactions avec un arbre merkle, afin d'ajouter la racine merkle à l'en-tête du bloc. La transaction coinbase est répertoriée comme la première transaction du bloc. Ensuite, 418 transactions supplémentaires sont ajoutées après cela, pour un total de 419 transactions dans le bloc. Comme nous l'avons vu dans [merkle_trees] , il doit y avoir un nombre pair de nœuds "feuilles" dans l'arbre, donc la dernière transaction est dupliquée, créant 420 nœuds, chacun contenant le hachage d'une transaction. Les hachages de transaction sont ensuite combinés, par paires, créant chaque niveau de l'arbre, jusqu'à ce que toutes les transactions soient résumées en un nœud à la « racine » de l'arbre. La racine de l'arborescence merkle résume toutes les

transactions en une seule valeur de 32 octets, que vous pouvez voir répertoriée comme « racine merkle » dans Utilisation de la ligne de commande pour récupérer le bloc 277,316 , et ici :

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

Le nœud de minage de Jing ajoutera ensuite un horodatage de 4 octets, codé comme un horodatage « d'époque » Unix, qui est basé sur le nombre de secondes écoulées depuis minuit UTC, le jeudi 1er janvier 1970. L'heure 1388185914 est égale à vendredi décembre. 27, 2013, 23 :11 :54 UTC.

Le nœud de Jing remplit ensuite la cible, qui définit la preuve de travail requise pour en faire un bloc valide. La cible est stockée dans le bloc sous la forme d'une métrique de "bits cibles", qui est un codage d'exposant de mantisse de la cible. Le codage a un exposant de 1 octet, suivi d'une mantisse de 3 octets (coefficient). Dans le bloc 277,316, par exemple, la valeur des bits cibles est 0x1903a30c. La première partie 0x19 est un exposant hexadécimal, tandis que la partie suivante, 0x03a30c, est le coefficient. Le concept de cible est expliqué dans Retargeting to Adjust Difficulty et la représentation des « bits cibles » est expliquée dans Target Representation .

Le dernier champ est le nonce, qui est initialisé à zéro.

Avec tous les autres champs remplis, l'en-tête du bloc est maintenant terminé et le processus d'extraction peut commencer.

Le but est maintenant de trouver une valeur pour le nonce qui aboutit à un hachage d'en-tête de bloc égal ou inférieur à la cible. Le nœud de minage devra tester des milliards ou des billions de valeurs nonce avant de trouver un nonce qui satisfait l'exigence.

Extraire le bloc

Maintenant qu'un bloc candidat a été construit par le nœud de Jing, il est temps pour la plate-forme minière matérielle de Jing de « miner » le bloc, de trouver une solution à l'algorithme de preuve de travail qui rend le bloc valide. Tout au long de ce livre, nous avons étudié les fonctions de hachage cryptographiques utilisées dans divers aspects du système Bitcoin. La fonction de hachage SHA256 est la fonction utilisée dans le processus d'extraction de Bitcoin.

Dans les termes les plus simples, l'extraction est le processus de hachage de l'en-tête de bloc à plusieurs reprises, en modifiant un paramètre, jusqu'à ce que le hachage résultant corresponde à une cible spécifique. Le résultat de la fonction de hachage ne peut pas être déterminé à l'avance, ni ne peut être créé un modèle qui produira une valeur de hachage spécifique. Cette fonctionnalité des fonctions de hachage signifie que la seule façon de produire un résultat de hachage correspondant à une cible spécifique est d'essayer encore et encore, en modifiant de manière aléatoire l'entrée jusqu'à ce que le résultat de hachage souhaité apparaisse par hasard.

Algorithme de preuve de travail

Un algorithme de hachage prend une entrée de données de longueur arbitraire et produit un résultat déterministe de longueur fixe, une empreinte numérique de l'entrée. Pour toute entrée spécifique, le hachage résultant sera toujours le même et peut être facilement calculé et vérifié par toute personne implémentant le même algorithme de hachage. La caractéristique clé d'un algorithme de hachage cryptographique est qu'il est impossible, d'un point de vue informatique, de trouver deux entrées différentes qui produisent la même empreinte digitale (connue sous le nom de *collision*). En corollaire, il est également pratiquement impossible de sélectionner une entrée de manière à produire une empreinte digitale souhaitée, autre que d'essayer des entrées aléatoires.

Avec SHA256, la sortie a toujours une longueur de 256 bits, quelle que soit la taille de l'entrée. Dans l' exemple SHA256 , nous utiliserons l'interpréteur Python pour calculer le hachage SHA256 de la phrase, "Je suis Satoshi Nakamoto".

Exemple 8. Exemple SHA256

```
$ python
```

```
Python 3.7.3
>>> import hashlib
>>> hashlib.sha256(b"I am Satoshi
Nakamoto").hexdigest()
'5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e'
```

L'exemple SHA256 montre le résultat du calcul du hachage de "Je suis Satoshi Nakamoto" : 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Ce nombre de 256 bits est le *hachage* ou le *condensé* de la phrase et dépend de chaque partie de la phrase. L'ajout d'une seule lettre, d'un signe de ponctuation ou de tout autre caractère produira un hachage différent.

Maintenant, si nous changeons la phrase, nous devrions nous attendre à voir des hachages complètement différents. Essayons cela en ajoutant un nombre à la fin de notre phrase, en utilisant le simple script Python dans le script SHA256 pour générer de nombreux hachages en itérant sur un nonce .

Exemple 9. Script SHA256 pour générer de nombreux hachages en itérant sur un nonce

```
link:code/hash_example.py[]
```

L'exécution de ceci produira les hachages de plusieurs phrases, rendues différentes en ajoutant un nombre à la fin du texte. En incrémentant le nombre, nous pouvons obtenir différents hachages, comme indiqué dans la sortie SHA256 d'un script pour générer de nombreux hachages en itérant sur un nonce .

Exemple 10. Sortie SHA256 d'un script pour générer de nombreux hachages en itérant sur un nonce

```
$ python hash_example.py
```

```
I am Satoshi Nakamoto0 =>
a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 =>
f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 =>
ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 =>
bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 =>
bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 =>
eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 =>
4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 =>
790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 =>
702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 =>
7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 =>
c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 =>
7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 =>
60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 =>
0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 =>
27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 =>
394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 =>
8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 =>
dca9b8b4f8d8e1521fa4eaa46f4f0cd...
```

```
I am Satoshi Nakamoto18 =>
9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 =>
cda56022ecb5b67b2bc93a2d764e75f...
```

Chaque phrase produit un résultat de hachage complètement différent. Ils semblent complètement aléatoires, mais vous pouvez reproduire les résultats exacts de cet exemple sur n'importe quel ordinateur avec Python et voir les mêmes hachages exacts.

Le nombre utilisé comme variable dans un tel scénario est appelé un *nonce* . Le nonce est utilisé pour faire varier la sortie d'une fonction cryptographique, dans ce cas pour faire varier l'empreinte SHA256 de la phrase.

Pour relever le défi de cet algorithme, définissons une cible : trouvez une phrase qui produit un hachage hexadécimal commençant par un zéro. Heureusement, ce n'est pas difficile ! La sortie SHA256 d'un script pour générer de nombreux hachages en itérant sur un nonce montre que l'expression "Je suis Satoshi Nakamoto13" produit le hachage 0ebc56d59a34f5 082aaef3d66b37a661696c2b618e62432727216ba9531041a5, qui correspond à nos critères. Il a fallu 13 tentatives pour le trouver. En termes de probabilités, si la sortie de la fonction de hachage est uniformément répartie, nous nous attendons à trouver un résultat avec un 0 comme préfixe hexadécimal une fois tous les 16 hachages (un sur 16 chiffres hexadécimaux de 0 à F). En termes numériques, cela signifie trouver une valeur de

hachage inférieure à 0x100000000000000000000000000000000000000000000000000000000000000. Nous appelons ce seuil la *cible* et le but est de trouver un hachage numériquement égal ou inférieur à la cible. Si nous diminuons la cible, la tâche de trouver un hachage inférieur à la cible devient de plus en plus difficile.

Pour donner une analogie simple, imaginez un jeu où les joueurs lancent une paire de dés à plusieurs reprises, en essayant de lancer égal ou inférieur à une cible spécifiée. Au premier tour, l'objectif est de 11. À moins que vous ne lanciez un double-six, vous gagnez. Au tour suivant, l'objectif est de 10. Les joueurs doivent en lancer 10 ou moins pour gagner, encore une fois une tâche facile. Disons que quelques tours plus tard, la cible est ramenée à 5. Désormais, plus de la moitié des lancers de dés dépasseront la cible et seront donc invalides. Il faut exponentiellement plus de lancers de dés pour gagner, plus la cible baisse. Finalement, lorsque la cible est de 2 (le minimum possible), un seul lancer sur 36, soit 2% d'entre eux, produira un résultat gagnant.

Du point de vue d'un observateur qui sait que la cible du jeu de dés est 2, si quelqu'un a réussi à lancer un lancer gagnant, on peut supposer qu'il a tenté, en moyenne, 36 lancers. En d'autres termes, on peut estimer la quantité de travail qu'il faut pour réussir à partir de la difficulté imposée par la cible. Lorsque l'algorithme est basé sur une fonction déterministe telle que SHA256, l'entrée elle-même constitue la *preuve* qu'un certain *travail a* été effectué pour produire un résultat égal ou inférieur à la cible. Par conséquent, *preuve de travail* .

Conseil

Même si chaque tentative produit un résultat aléatoire, la probabilité de tout résultat possible peut être calculée à l'avance. Par conséquent, le résultat d'une difficulté spécifiée constitue la preuve d'une quantité de travail spécifique.

Dans la sortie SHA256 d'un script pour générer de nombreux hachages en itérant sur un nonce , le "nonce" gagnant est 13 et ce résultat peut être confirmé par n'importe qui indépendamment. N'importe qui peut ajouter le nombre 13 comme suffixe à la phrase "Je suis Satoshi Nakamoto" et calculer le hachage, en vérifiant qu'il est inférieur à la cible. Le résultat réussi est également la preuve de travail, car cela prouve que nous avons fait le travail pour trouver ce nonce. Bien qu'il ne faille qu'un seul calcul de hachage pour vérifier, il nous a fallu 13 calculs de hachage pour trouver un nonce qui a fonctionné. Si nous avions une cible inférieure (difficulté plus élevée), il faudrait beaucoup plus de calculs de hachage pour trouver un nonce approprié, mais un seul calcul de hachage que quiconque puisse vérifier. De plus, en connaissant la cible, n'importe qui peut estimer la difficulté à l'aide de statistiques et donc savoir combien de travail a été nécessaire pour trouver un tel nonce.

Conseil

La preuve de travail doit produire un hachage égal ou inférieur à *la cible. Une cible plus élevée signifie qu'il est moins difficile de trouver un hachage égal ou inférieur à la cible. Une cible inférieure signifie qu'il est plus difficile de trouver un hachage égal ou inférieur à la cible. La cible*

et la difficulté sont inversement liées.

La preuve de travail de Bitcoin est très similaire au défi présenté dans la sortie SHA256 d'un script pour générer de nombreux hachages en itérant sur un nonce . Le mineur construit un bloc candidat rempli de transactions. Ensuite, le mineur calcule le hachage de l'en-tête de ce bloc et voit s'il est égal ou inférieur à la *cible* actuelle . Si le hachage est supérieur à la cible, le mineur modifiera le nonce (généralement en l'incrémentant simplement de un) et réessayera. À la difficulté actuelle du réseau bitcoin, les mineurs doivent essayer des quadrillions de fois avant de trouver un nonce qui se traduit par un hachage d'en-tête de bloc suffisamment bas.

Un algorithme de preuve de travail très simplifié est implémenté en Python dans l'implémentation de preuve de travail simplifiée .

Exemple 11. Mise en œuvre simplifiée de la preuve de travail

```
link:code/proof-of-work-example.py[]
```

En exécutant ce code, vous pouvez définir la difficulté souhaitée (en bits, combien de bits de tête doivent être zéro) et voir combien de temps il faut à votre ordinateur pour trouver une solution. Dans Exécution de l'exemple de preuve de travail pour diverses difficultés , vous pouvez voir comment cela fonctionne sur un ordinateur portable moyen.

Exemple 12. Exécution de l'exemple de preuve de travail pour diverses difficultés

```
$ python proof-of-work-example.py*
```

```
Difficulty: 1 (0 bits)

[...]

Difficulty: 8 (3 bits)
Starting search...
Success with nonce 9
Hash is
1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Elapsed Time: 0.0004 seconds
Hashing Power: 25065 hashes per second
Difficulty: 16 (4 bits)
Starting search...
Success with nonce 25
Hash is
0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Elapsed Time: 0.0005 seconds
Hashing Power: 52507 hashes per second
Difficulty: 32 (5 bits)
Starting search...
Success with nonce 36
Hash is
029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hashes per second

[...]

Difficulty: 4194304 (22 bits)
```

```
Starting search...
Success with nonce 1759164
Hash is
0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Elapsed Time: 13.3201 seconds
Hashing Power: 132068 hashes per second
Difficulty: 8388608 (23 bits)
Starting search...
Success with nonce 14214729
Hash is
000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Elapsed Time: 110.1507 seconds
Hashing Power: 129048 hashes per second
Difficulty: 16777216 (24 bits)
Starting search...
Success with nonce 24586379
Hash is
0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Elapsed Time: 195.2991 seconds
Hashing Power: 125890 hashes per second

[...]

Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291
Hash is
0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per second
```

Comme vous pouvez le voir, augmenter la difficulté de 1 bit entraîne un doublement du temps nécessaire pour trouver une solution. Si vous pensez à tout l'espace numérique de 256 bits, chaque fois que vous contraignez un bit supplémentaire à zéro,

vous diminuez de moitié l'espace de recherche. Dans l'exemple Exécution de la preuve de travail pour diverses difficultés , il faut 84 millions de tentatives de hachage pour trouver un nonce qui produit un hachage avec 26 bits de tête comme zéro. Même à une vitesse de plus de 120 000 hachages par seconde, il faut encore 10 minutes sur un ordinateur portable pour trouver cette solution.

Au moment de l'écriture, le réseau tente de trouver un bloc dont le hachage d'en-tête est égal ou inférieur à :

```
0000000000000000029AB9000000000000000000000000000000000000000000
```

Comme vous pouvez le voir, il y a beaucoup de zéros au début de cette cible, ce qui signifie que la plage acceptable de hachages est beaucoup plus petite, il est donc plus difficile de trouver un hachage valide. Il faudra en moyenne plus de 1,8 hachages zêta (mille milliards de milliards de hachages) pour que le réseau découvre le bloc suivant. Cela semble être une tâche impossible, mais heureusement, le réseau apporte 3 exa-hachages par seconde (EH / sec) de puissance de traitement, ce qui sera en mesure de trouver un bloc en environ 10 minutes en moyenne.

Représentation de la cible

En utilisant la ligne de commande pour récupérer le bloc 277,316 , nous avons vu que le bloc contient la cible, dans une notation appelée « bits cibles » ou simplement « bits », qui

dans le bloc 277,316 a la valeur 0x1903a30c. Cette notation exprime la cible de preuve de travail sous la forme d'un format de coefficient / exposant, avec les deux premiers chiffres hexadécimaux pour l'exposant et les six chiffres hexadécimaux suivants comme coefficient. Dans ce bloc, par conséquent, l'exposant est 0x19 et le coefficient est 0x03a30c.

La formule pour calculer l'objectif de difficulté à partir de cette représentation est :

- target = coefficient * $2^{(8*(\text{exponent}-3))}$

En utilisant cette formule et la valeur des bits de difficulté 0x1903a30c, nous obtenons :

- target = 0x03a30c * $2^{0x08*(0x19-0x03)}$
- => target = 0x03a30c * $2^{(0x08*0x16)}$
- => target = 0x03a30c * 2^{0xB0}

qui en décimal est :

- => target = 238,348 * 2^{176}
- => target =
- 22,829,202,948,393,929,850,749,706,076,701,368,331,072, 452,018,388,575,715,328

retour à l'hexadécimal :

- => target =

- 0x0000000000000003A30C00000000000000000000000000000000000000000000

Cela signifie qu'un bloc valide pour la hauteur 277,316 est celui dont le hachage d'en-tête de bloc est inférieur à cette cible. En binaire, ce nombre doit avoir plus de 60 bits de tête mis à zéro. Avec ce niveau de difficulté, un seul mineur traitant 1 billion de hachages par seconde (1 térahash par seconde ou 1 TH / s) ne trouverait une solution qu'une fois tous les 8496 blocs ou une fois tous les 59 jours, en moyenne.

Reciblage pour ajuster la difficulté

Comme nous l'avons vu, la cible détermine la difficulté et affecte donc le temps nécessaire pour trouver une solution à l'algorithme de preuve de travail. Cela conduit à des questions évidentes : pourquoi la difficulté est-elle ajustable, qui l'ajuste et comment ?

Les blocs de Bitcoin sont générés toutes les 10 minutes, en moyenne. C'est le rythme cardiaque de Bitcoin et sous-tend la fréquence d'émission de devises et la vitesse de règlement des transactions. Il doit rester constant non seulement à court terme, mais sur une période de plusieurs décennies. Au cours de cette période, on s'attend à ce que la puissance informatique continue d'augmenter à un rythme rapide. En outre, le nombre de participants à l'exploitation minière et les ordinateurs qu'ils utilisent changeront également constamment. Pour maintenir le temps de génération de bloc à 10 minutes, la difficulté de l'extraction doit être ajustée pour tenir compte de ces

changements. En fait, la cible de preuve de travail est un paramètre dynamique qui est périodiquement ajusté pour atteindre un objectif d'intervalle de bloc de 10 minutes. En termes simples, la cible est définie de sorte que la puissance minière actuelle se traduise par un intervalle de bloc de 10 minutes.

Comment, alors, un tel ajustement se fait-il dans un réseau complètement décentralisé ? Le reciblage se produit automatiquement et sur chaque nœud indépendamment. Tous les 2,016 blocs, tous les nœuds reciblent la preuve de travail. L'équation de reciblage mesure le temps qu'il a fallu pour trouver les 2 016 derniers blocs et le compare au temps prévu de 20 160 minutes (2 016 blocs fois l'intervalle de bloc de 10 minutes souhaité). Le rapport entre la durée réelle et la durée souhaitée est calculé et un ajustement proportionné (vers le haut ou vers le bas) est effectué sur la cible. En termes simples : si le réseau trouve des blocs plus rapidement que toutes les 10 minutes, la difficulté augmente (la cible diminue). Si la découverte de bloc est plus lente que prévu, la difficulté diminue (la cible augmente).

L'équation peut être résumée comme suit :

```
New Target = Old Target * (Actual Time of Last 2016
Blocks / 20160 minutes)
```

Reciblage de la preuve de travail - CalculateNextWorkRequired () dans pow.cpp montre le code utilisé dans le client Bitcoin Core.

Exemple 13. Reciblage de la preuve de travail - CalculateNext WorkRequired () dans pow.cpp

```
// Limit adjustment stepint64_t nActualTimespan =
pindexLast->GetBlockTime() - nFirstBlockTime;
    LogPrintf("  nActualTimespan = %d  before
    bounds\n", nActualTimespan);
    if (nActualTimespan < params.nPowTargetTimespan/4)
        nActualTimespan = params.nPowTargetTimespan/4;
    if (nActualTimespan > params.nPowTargetTimespan*4)
        nActualTimespan = params.nPowTargetTimespan*4;

    // Retargetconst arith_uint256 bnPowLimit =
    UintToArith256(params.powLimit);
    arith_uint256 bnNew;
    arith_uint256 bnOld;
    bnNew.SetCompact(pindexLast->nBits);
    bnOld = bnNew;
    bnNew *= nActualTimespan;
    bnNew /= params.nPowTargetTimespan;

    if (bnNew > bnPowLimit)
        bnNew = bnPowLimit;
```

Note

Alors que l'étalonnage de la cible se produit tous les 2,016 blocs, en raison d'une erreur de décalage dans le client Bitcoin Core d'origine, il est basé sur le temps total des 2,015 blocs précédents (et non de 2,016 comme il se doit), ce qui entraîne un biais de reciblage. vers une difficulté plus élevée de 0,05%.

Les paramètres Interval (2,016 blocs) et TargetTimespan (deux semaines comme 1,209,600 secondes) sont définis dans *chainparams.cpp* .

Pour éviter une volatilité extrême de la difficulté, l'ajustement de reciblage doit être inférieur à un facteur de quatre (4) par cycle. Si l'ajustement cible requis est supérieur à un facteur de quatre, il sera ajusté d'un facteur de 4 et pas plus. Tout ajustement supplémentaire sera effectué au cours de la prochaine période de reciblage, car le déséquilibre persistera pendant les 2 016 blocs suivants. Par conséquent, les écarts importants entre la puissance de hachage et la difficulté peuvent prendre plusieurs 2 016 cycles de bloc pour s'équilibrer.

Conseil

La difficulté de l'extraction d'un bloc Bitcoin est d'environ " 10 minutes de traitement '' pour l'ensemble du réseau, en fonction du temps nécessaire pour extraire les 2,016 blocs précédents, ajusté tous les 2,016 blocs. Ceci est réalisé en abaissant ou en augmentant la cible.

Notez que la cible est indépendante du nombre de transactions ou de la valeur des transactions. Cela signifie que la quantité d'énergie de hachage et donc d'électricité dépensée pour sécuriser le bitcoin est également entièrement indépendante du nombre de transactions. Bitcoin peut évoluer, atteindre une adoption plus large et rester sécurisé sans aucune augmentation de la puissance de hachage par rapport au niveau actuel. L'augmentation de la puissance de hachage représente les forces du marché à mesure que de nouveaux mineurs entrent sur le

marché pour se disputer la récompense. Tant qu'une puissance de hachage suffisante est sous le contrôle des mineurs agissant honnêtement à la poursuite de la récompense, cela suffit pour empêcher les attaques de « prise de contrôle » et, par conséquent, il suffit de sécuriser le bitcoin.

La difficulté de l'exploitation minière est étroitement liée au coût de l'électricité et au taux de change du bitcoin par rapport à la monnaie utilisée pour payer l'électricité. Les systèmes d'extraction haute performance sont à peu près aussi efficaces que possible avec la génération actuelle de fabrication de silicium, convertissant l'électricité en calcul de hachage au taux le plus élevé possible. L'influence principale sur le marché minier est le prix d'un kilowattheure d'électricité en bitcoin, car cela détermine la rentabilité de l'exploitation minière et donc les incitations à entrer ou sortir du marché minier.

Extraction réussie du bloc

Comme nous l'avons vu précédemment, le nœud de Jing a construit un bloc candidat et l'a préparé pour l'exploitation minière. Jing dispose de plusieurs plates-formes minières matérielles avec des circuits intégrés spécifiques à l'application , où des centaines de milliers de circuits intégrés exécutent l'algorithme SHA256 en parallèle à des vitesses incroyables. Beaucoup de ces machines spécialisées sont connectées à son nœud de minage via USB ou un réseau local. Ensuite, le nœud de minage s'exécutant sur le bureau de Jing transmet l'en-tête de bloc à son matériel de minage, qui commence à tester des milliards de nonces par seconde. Parce que le nonce n'est que

de 32 bits, après avoir épuisé toutes les possibilités de nonce (environ 4 milliards), le matériel de minage change l'en-tête de bloc (ajustant l'espace nonce supplémentaire de la coinbase ou l'horodatage) et réinitialise le compteur de nonce, testant de nouvelles combinaisons.

Près de 11 minutes après avoir commencé à miner le bloc 277 316, l'une des machines d'extraction matérielle trouve une solution et la renvoie au nœud d'extraction.

Lorsqu'il est inséré dans l'en-tête de bloc, le nonce 924 591 752 produit un hachage de bloc de :

```
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

qui est inférieur à la cible :

```
0000000000000003A30C0000000000000000000000000000000000000000000
```

Immédiatement, le nœud minier de Jing transmet le bloc à tous ses pairs. Ils reçoivent, valident, puis propagent le nouveau bloc. Au fur et à mesure que le bloc se propage sur le réseau, chaque nœud l'ajoute à sa propre copie de la blockchain, l'étendant à une nouvelle hauteur de 277316 blocs. Lorsque les nœuds de minage reçoivent et valident le bloc, ils abandonnent leurs efforts pour trouver un bloc à la même hauteur et commencent immédiatement à calculer le bloc suivant dans la chaîne, en utilisant le bloc de Jing comme «parent». En construisant

au-dessus du bloc nouvellement découvert de Jing, les autres mineurs « votent » essentiellement avec leur pouvoir minier et approuvent le bloc de Jing et la chaîne qu'il étend.

Dans la section suivante, nous examinerons le processus utilisé par chaque nœud pour valider un bloc et sélectionner la chaîne la plus longue, créant ainsi le consensus qui forme la blockchain décentralisée.

Valider un nouveau bloc

La troisième étape du mécanisme de consensus de Bitcoin est la validation indépendante de chaque nouveau bloc par chaque nœud du réseau. Au fur et à mesure que le bloc nouvellement résolu se déplace sur le réseau, chaque nœud effectue une série de tests pour le valider avant de le propager à ses pairs. Cela garantit que seuls les blocs valides sont propagés sur le réseau. La validation indépendante garantit également que les mineurs qui agissent honnêtement obtiennent leurs blocs incorporés dans la blockchain, gagnant ainsi la récompense. Les mineurs qui agissent de manière malhonnête voient leurs blocages rejetés et non seulement perdent la récompense, mais gaspillent également les efforts déployés pour trouver une solution de preuve de travail, encourant ainsi le coût de l'électricité sans compensation.

Lorsqu'un nœud reçoit un nouveau bloc, il validera le bloc en le comparant à une longue liste de critères qui doivent tous être satisfaits ; sinon, le bloc est rejeté. Ces critères peuvent être vus dans le client Bitcoin Core dans les fonctions CheckBlock et

CheckBlockHeader et comprennent :

- La structure des données de bloc est syntaxiquement valide
- Le hachage de l'en-tête de bloc est égal ou inférieur à la cible (applique la preuve de travail)
- L'horodatage du bloc est inférieur à deux heures dans le futur (en tenant compte des erreurs de temps)
- La taille du bloc est dans les limites acceptables
- La première transaction (et seulement la première) est une transaction coinbase
- Toutes les transactions dans le bloc sont valides en utilisant la liste de contrôle des transactions décrite dans Vérification indépendante des transactions

La validation indépendante de chaque nouveau bloc par chaque nœud du réseau garantit que les mineurs ne peuvent pas tricher. Dans les sections précédentes, nous avons vu comment les mineurs peuvent écrire une transaction qui leur attribue le nouveau bitcoin créé dans le bloc et réclamer les frais de transaction. Pourquoi les mineurs n'écrivent-ils pas eux-mêmes une transaction pour mille bitcoins au lieu de la récompense correcte ? Parce que chaque nœud valide les blocs selon les mêmes règles. Une transaction coinbase invalide rendrait le bloc entier invalide, ce qui entraînerait le rejet du bloc et, par conséquent, cette transaction ne ferait jamais partie du grand livre. Les mineurs doivent construire un bloc parfait, basé sur les règles partagées que tous les nœuds suivent, et le miner avec une solution correcte à la preuve de travail. Pour ce faire, ils dépensent beaucoup d'électricité dans les mines, et s'ils trichent, toute l'électricité et les efforts sont gaspillés. C'est pourquoi

la validation indépendante est un élément clé du consensus décentralisé.

Assemblage et sélection de chaînes de blocs

La dernière étape du mécanisme de consensus décentralisé de Bitcoin est l'assemblage de blocs en chaînes et la sélection de la chaîne avec le plus de preuves de travail. Une fois qu'un nœud a validé un nouveau bloc, il tentera alors d'assembler une chaîne en connectant le bloc à la blockchain existante.

Les nœuds maintiennent trois ensembles de blocs : ceux connectés à la blockchain principale, ceux qui forment des branches de la blockchain principale (chaînes secondaires), et enfin, les blocs qui n'ont pas de parent connu dans les chaînes connues (orphelins). Les blocs non valides sont rejetés dès que l'un des critères de validation échoue et ne sont donc inclus dans aucune chaîne.

La "chaîne principale" à tout moment est la chaîne de blocs *valide* à laquelle est associée la preuve de travail la plus cumulative. Dans la plupart des cas, il s'agit également de la chaîne avec le plus de blocs, à moins qu'il n'y ait deux chaînes de longueur égale et une avec plus de preuve de travail. La chaîne principale aura également des branches avec des blocs qui sont des « frères et sœurs » des blocs de la chaîne principale. Ces blocs sont valides mais ne font pas partie de la chaîne principale. Ils sont conservés pour référence future, au cas où l'une de ces chaînes serait étendue pour dépasser la chaîne principale en

cours de travail. Dans la section suivante (Blockchain Forks), nous verrons comment les chaînes secondaires se produisent à la suite d'une extraction presque simultanée de blocs à la même hauteur.

Lorsqu'un nouveau bloc est reçu, un nœud essaiera de l'insérer dans la blockchain existante. Le nœud examinera le champ "hachage de bloc précédent" du bloc, qui est la référence au parent du bloc. Ensuite, le nœud tentera de trouver ce parent dans la blockchain existante. La plupart du temps, le parent sera la "pointe" de la chaîne principale, ce qui signifie que ce nouveau bloc étend la chaîne principale. Par exemple, le nouveau bloc 277,316 a une référence au hachage de son bloc parent 277,315. La plupart des nœuds qui reçoivent 277 316 auront déjà le bloc 277 315 comme bout de leur chaîne principale et relieront donc le nouveau bloc et étendront cette chaîne.

Parfois, comme nous le verrons dans Blockchain Forks , le nouveau bloc étend une chaîne qui n'est pas la chaîne principale. Dans ce cas, le nœud attachera le nouveau bloc à la chaîne secondaire qu'il étend, puis comparera le travail de la chaîne secondaire à la chaîne principale. Si la chaîne secondaire a plus de travail cumulatif que la chaîne principale, le nœud se *reconverra* sur la chaîne secondaire, ce qui signifie qu'il sélectionnera la chaîne secondaire comme nouvelle chaîne principale, faisant de l'ancienne chaîne principale une chaîne secondaire. Si le nœud est un mineur, il construira maintenant un bloc étendant cette nouvelle chaîne plus longue.

Si un bloc valide est reçu et qu'aucun parent n'est trouvé dans les chaînes existantes, ce bloc est considéré comme "orphelin". Les

blocs orphelins sont enregistrés dans le pool de blocs orphelins où ils resteront jusqu'à ce que leur parent soit reçu. Une fois que le parent est reçu et lié aux chaînes existantes, l'orphelin peut être retiré du pool d'orphelin et lié au parent, ce qui en fait une partie d'une chaîne. Les blocs orphelins se produisent généralement lorsque deux blocs qui ont été minés dans un court laps de temps l'un de l'autre sont reçus dans l'ordre inverse (enfant avant parent).

En sélectionnant la chaîne valide de travail cumulatif le plus élevé, tous les nœuds parviennent finalement à un consensus à l'échelle du réseau. Les écarts temporaires entre les chaînes sont finalement résolus au fur et à mesure que du travail est ajouté, étendant l'une des chaînes possibles. Les nœuds miniers « votent » avec leur puissance minière en choisissant la chaîne à étendre en minant le bloc suivant. Lorsqu'ils extraient un nouveau bloc et étendent la chaîne, le nouveau bloc lui-même représente leur vote.

Dans la section suivante, nous verrons comment les écarts entre les chaînes concurrentes (fourches) sont résolus par la sélection indépendante de la plus grande chaîne de travail cumulatif.

Fourches Blockchain

Parce que la blockchain est une structure de données décentralisée, différentes copies de celle-ci ne sont pas toujours cohérentes. Les blocs peuvent arriver à différents nœuds à des moments différents, ce qui fait que les nœuds ont des perspectives différentes de la blockchain. Pour résoudre ce problème, chaque nœud sélectionne et tente toujours d'étendre

la chaîne de blocs qui représente la plus grande preuve de travail, également appelée chaîne la plus longue ou chaîne de travail cumulative la plus grande. En additionnant le travail enregistré dans chaque bloc d'une chaîne, un nœud peut calculer la quantité totale de travail qui a été dépensée pour créer cette chaîne. Tant que tous les nœuds sélectionnent la plus grande chaîne de travail cumulatif, le réseau bitcoin mondial finit par converger vers un état cohérent. Les fourchettes se produisent comme des incohérences temporaires entre les versions de la blockchain, qui sont résolues par une éventuelle reconvergence lorsque davantage de blocs sont ajoutés à l'une des fourches.

Conseil

Les fourchettes de blockchain décrites dans cette section se produisent naturellement (accidentellement) à la suite de retards de transmission dans le réseau mondial. Plus loin dans ce chapitre, nous examinerons également les fourches délibérément induites (hard forks et soft forks), qui sont utilisées pour modifier les règles de consensus.

Dans les prochains schémas, nous suivons la progression d'un événement « fork » sur le réseau. Le diagramme est une représentation simplifiée du réseau bitcoin. À des fins d'illustration, différents blocs sont représentés sous différentes formes (étoile, triangle, triangle inversé, losange), répartis sur le réseau. Chaque nœud du réseau est représenté par un cercle.

Chaque nœud a sa propre perspective de la blockchain mondiale. Au fur et à mesure que chaque nœud reçoit des blocs de ses voisins, il met à jour sa propre copie de la blockchain,

en sélectionnant la plus grande chaîne de travail cumulatif. À des fins d'illustration, chaque nœud contient une forme qui représente le bloc qui, selon lui, est actuellement la pointe de la chaîne principale. Donc, si vous voyez une forme d'étoile dans le nœud, cela signifie que le bloc d'étoile est la pointe de la chaîne principale, en ce qui concerne ce nœud.

Dans le premier diagramme (avant la fourche - tous les nœuds ont la même perspective), le réseau a une perspective unifiée de la blockchain, avec le bloc en étoile comme pointe de la chaîne principale.

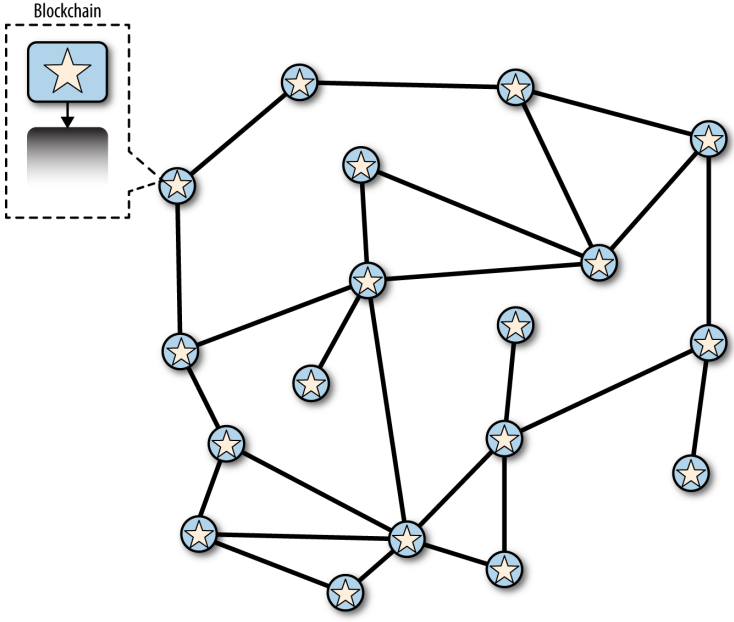


Figure 2. Avant la fourche : tous les nœuds ont la même perspective

Une "fourchette" se produit chaque fois qu'il y a deux blocs valides différents à la même hauteur de bloc en compétition pour former la plus longue blockchain. Cela se produit dans des conditions normales chaque fois que deux mineurs résolvent l'algorithme de preuve de travail dans un court laps de temps l'un de l'autre. Lorsque les deux mineurs découvrent une solution pour leurs blocs candidats respectifs, ils diffusent immédiatement leur propre bloc « gagnant » à leurs voisins immédiats qui commencent à propager le bloc sur le réseau. Chaque nœud qui reçoit un bloc valide l'intègrera dans sa blockchain, étendant la blockchain d'un bloc. Si ce nœud voit plus tard un autre bloc valide étendant le même parent (à la même hauteur de bloc), il connecte le deuxième bloc sur une chaîne secondaire, forçant sa chaîne principale. En conséquence, certains nœuds "verront" un bloc gagnant en premier, tandis que d'autres nœuds verront l'autre bloc gagnant en premier, et deux versions concurrentes de la blockchain émergeront.

Dans Visualisation d'un événement de fork de blockchain : deux blocs trouvés simultanément , nous voyons deux mineurs (Node X et Node Y) qui minent deux blocs différents presque simultanément. Ces deux blocs sont des enfants du bloc d'étoiles et prolongent la chaîne en construisant au-dessus du bloc d'étoiles. Pour nous aider à le suivre, l'un est visualisé comme un bloc triangulaire provenant du nœud X, et l'autre est représenté comme un bloc triangulaire à l'envers provenant du nœud Y.

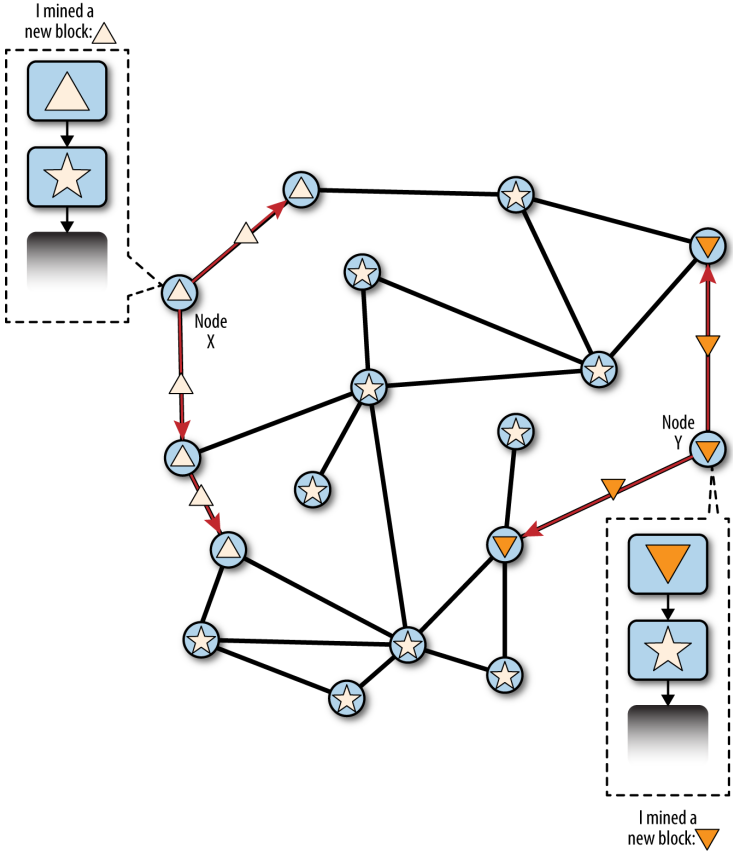


Figure 3. Visualisation d'un événement de fork de blockchain : deux blocs trouvés simultanément

Supposons, par exemple, que le mineur Node X trouve une solution Proof-of-Work pour un bloc "triangle" qui étend la blockchain, en s'appuyant sur le bloc parent "star". Presque simultanément, le mineur Node Y qui étendait également

la chaîne du bloc “étoile” trouve une solution pour le bloc “triangle inversé”, son bloc candidat. Maintenant, il y a deux blocs possibles ; celui que nous appelons « triangle », originaire du nœud X ; et celui que nous appelons « triangle à l’envers », originaire du nœud Y. Les deux blocs ont été extraits avec succès, les deux blocs sont valides (contiennent une solution valide à la preuve de travail) et les deux blocs étendent le même parent (bloc ” Star”). Les deux blocs contiennent probablement la plupart des mêmes transactions, avec seulement peut-être quelques différences dans l’ordre des transactions.

Au fur et à mesure que les deux blocs se propagent, certains nœuds reçoivent le bloc “triangle” en premier et certains reçoivent le bloc “triangle à l’envers” en premier. Comme le montre la visualisation d’un événement de fork de blockchain : deux blocs se propagent, divisant le réseau , le réseau se divise en deux perspectives différentes de la blockchain ; un côté surmonté du bloc triangle, l’autre du bloc triangle inversé.

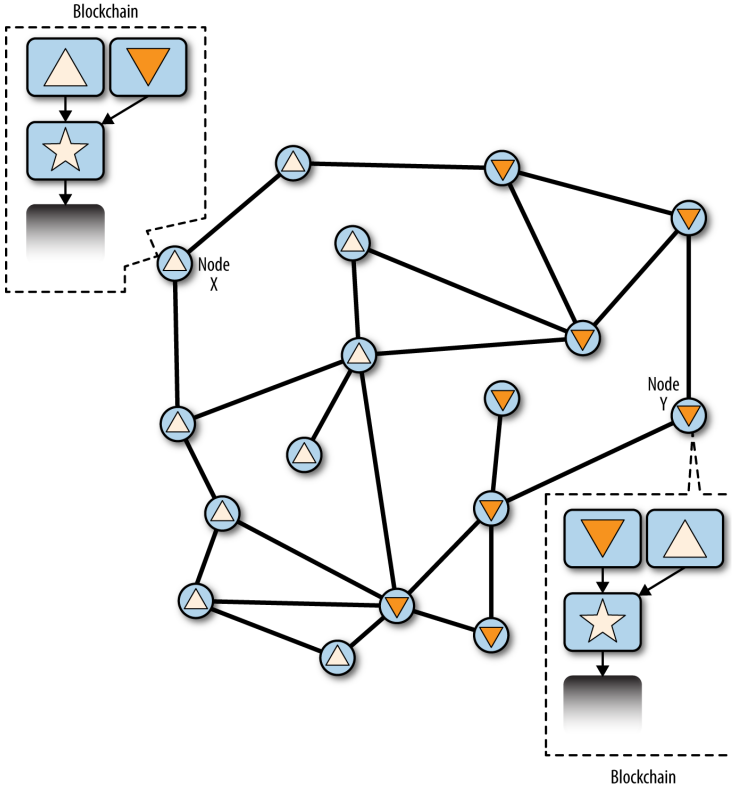


Figure 4. Visualisation d'un événement de fork de blockchain : deux blocs se propagent, divisant le réseau

Dans Visualisation d'un événement de fourchette de blockchain : deux blocs se propagent, divisant le réseau , le mineur Node X a extrait (créé) le bloc de triangle et a étendu la chaîne d'étoiles avec lui. Par conséquent, Node X considère initialement la chaîne avec le bloc "triangle" comme chaîne principale. Plus tard, Node X a également reçu le bloc "triangle à l'envers" qui

a été extrait par Node Y. Puisqu'il a été reçu en second, il est supposé avoir "perdu" la course. Pourtant, le bloc « triangle à l'envers » n'est pas écarté. Il est lié au bloc parent "étoile" et forme une chaîne secondaire. Alors que Node X suppose que sa chaîne principale est la chaîne gagnante, il conserve la chaîne « perdante » afin de disposer des informations nécessaires pour se reconvertir si la chaîne « perdante » finit par « gagner ».

De l'autre côté du réseau, le mineur Node Y construit une blockchain basée sur sa propre perspective de la séquence des événements. Le mineur Node Y a extrait (créé) le "triangle à l'envers" et considère initialement cette chaîne comme la chaîne principale (la chaîne "gagnante"). Lorsqu'il a reçu plus tard le bloc « triangle » qui a été extrait par le nœud X, il l'a connecté au bloc parent « étoile » en tant que chaîne secondaire.

Aucun des deux camps n'est « correct » ou « incorrect ». Les deux sont des perspectives valables de la blockchain. Ce n'est qu'avec le recul que l'on prévaudra, en fonction de la façon dont ces deux chaînes concurrentes sont prolongées par des travaux supplémentaires.

Chaque nœud d'exploration dont la perspective ressemble à Node X commencera immédiatement à extraire un bloc candidat qui étend la chaîne avec un « triangle » comme pointe. En liant le « triangle » en tant que parent de leurs blocs candidats, ils votent avec leur pouvoir de hachage. Leur vote soutient la chaîne qu'ils ont élue comme chaîne principale.

Tout nœud minier dont la perspective ressemble à Node Y commencera à construire un bloc candidat avec un "triangle

inversé" comme parent, étendant la chaîne qu'ils croient être la chaîne principale. Et ainsi, la course recommence.

Les fourchettes sont presque toujours résolues en un temps de bloc (10 minutes en moyenne). Alors qu'une partie de la puissance de hachage du réseau est dédiée à la construction au-dessus du « triangle » en tant que parent, une autre partie de la puissance de hachage se concentre sur la construction au-dessus du « triangle à l'envers ». Même si la puissance de hachage est répartie presque uniformément, il est probable qu'un groupe de mineurs trouve une solution et la propage avant que l'autre groupe de mineurs ait trouvé une solution. Disons, par exemple, que les mineurs construisant au-dessus du "triangle" trouvent un nouveau bloc "losange" qui étend la chaîne (par exemple, étoile-triangle-losange). Ils propagent immédiatement ce nouveau bloc et l'ensemble du réseau le voit comme une solution valide comme le montre la visualisation d'un événement de fork de la blockchain : un nouveau bloc étend un fork, reconvergeant le réseau. Le nœud X et le nœud Y considèrent désormais le bloc « triangle inversé » comme un bloc périmé. .

Tous les nœuds qui avaient choisi « triangle » comme vainqueur au tour précédent allongeront simplement la chaîne d'un bloc de plus. Cependant, les nœuds qui ont choisi « triangle inversé » comme vainqueur verront désormais deux chaînes : étoile-triangle-losange et étoile-triangle-tête-en-bas. La chaîne étoile-triangle-losange est désormais plus longue (travail plus cumulatif) que l'autre chaîne. En conséquence, ces nœuds définiront la chaîne étoile-triangle-losange comme chaîne principale et changeront la chaîne triangle étoile-tête-en-bas en chaîne

secondaire, comme indiqué dans Visualisation d'un événement de fourchette de blockchain : le réseau se reconvertit sur un nouvelle chaîne la plus longue . Il s'agit d'une reconvergence de chaîne, car ces nœuds sont obligés de revoir leur vision de la blockchain pour incorporer la nouvelle preuve d'une chaîne plus longue. Tous les mineurs travaillant sur l'extension de la chaîne triangle étoile-tête-en-bas arrêteront désormais ce travail car leur bloc candidat est désormais considéré comme un enfant d'un bloc périmé, car son "triangle inversé" parent ne se trouve plus sur la chaîne la plus longue. . Étant donné que le bloc triangle inversé est maintenant obsolète, le mineur Node Y (qui a extrait ce bloc) ne pourra pas dépenser la récompense minière pour ce bloc, même si ce bloc était valide et a été exploité avec succès. Les transactions dans le "triangle inversé" qui ne sont pas dans le "triangle" sont réinsérées dans le mempool pour être incluses dans le bloc suivant pour devenir une partie de la chaîne principale. L'ensemble du réseau se reconvertit sur une seule blockchain étoile-triangle-losange, avec "losange" comme dernier bloc de la chaîne. Tous les mineurs commencent immédiatement à travailler sur des blocs candidats qui font référence à "losange" comme leur parent pour étendre la chaîne étoile-triangle-losange.

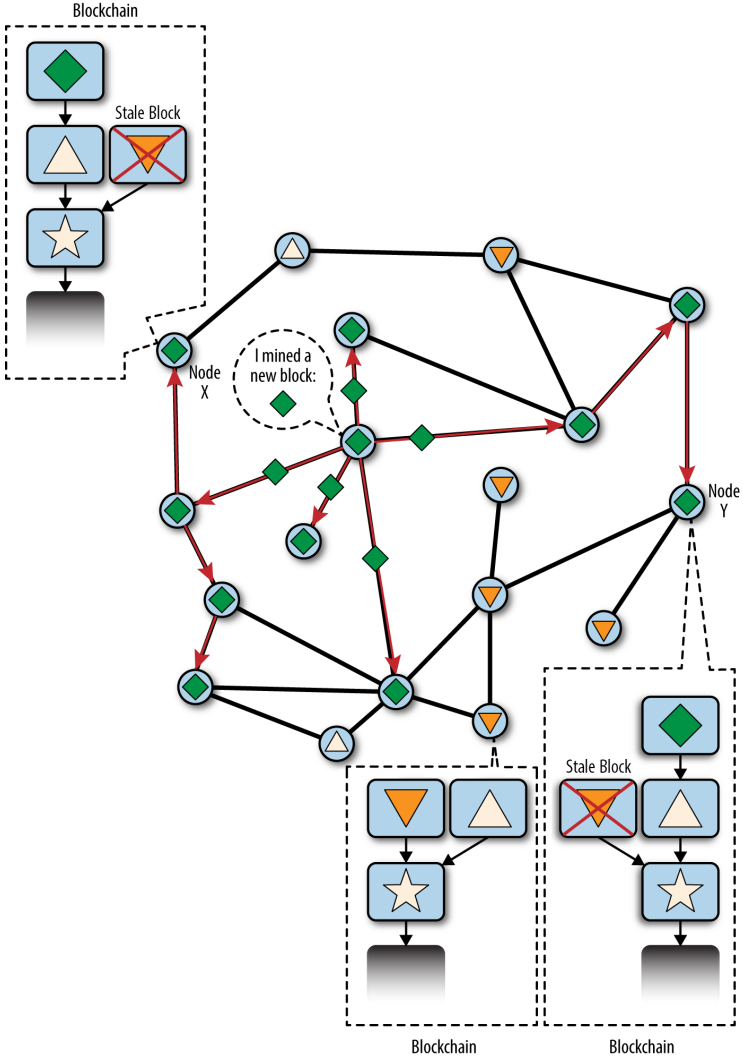


Figure 5. Visualisation d'un événement de fork de blockchain : un nouveau bloc étend un fork, reconvergeant le réseau. Le nœud X et le nœud Y considèrent désormais le bloc « triangle inversé » comme un bloc périmé.

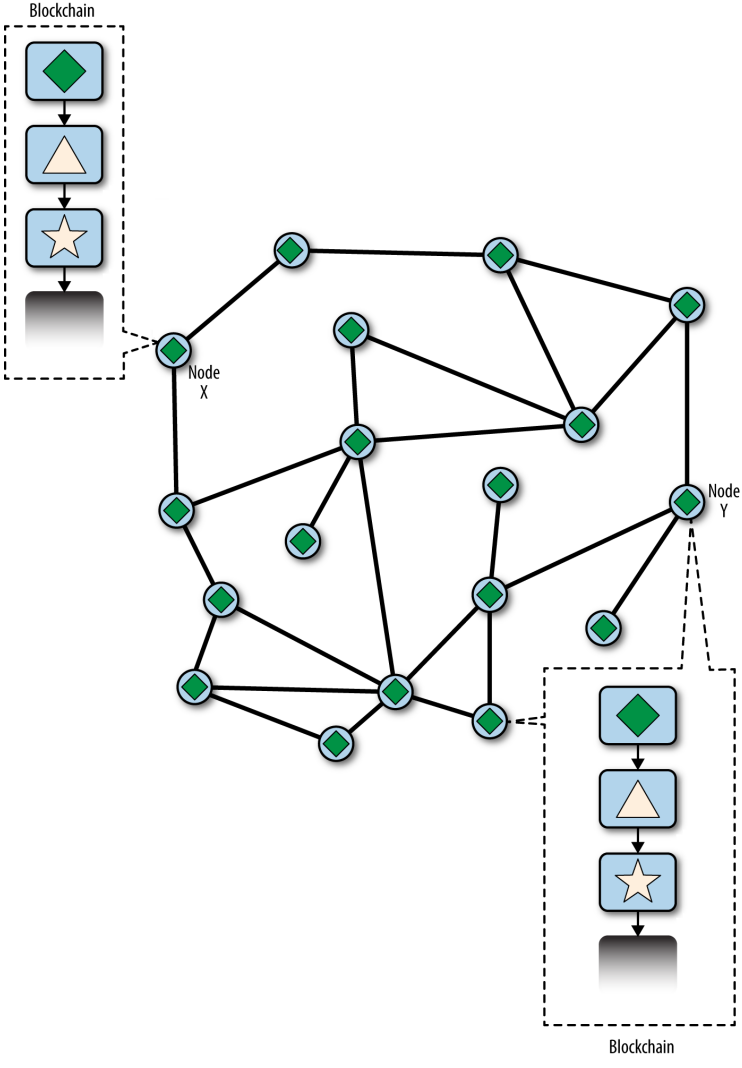


Figure 6. Visualisation d'un événement de fork de blockchain : le réseau se reconvertit sur une nouvelle chaîne la plus longue

Il est théoriquement possible qu'une fourche s'étende sur deux blocs, si deux blocs sont trouvés presque simultanément par des mineurs sur les « côtés » opposés d'une fourche précédente. Cependant, la probabilité que cela se produise est très faible. Alors qu'une fourche à un bloc peut se produire tous les jours, une fourche à deux blocs se produit au plus une fois toutes les quelques semaines.

L'intervalle de bloc de 10 minutes de Bitcoin est un compromis de conception entre les temps de confirmation rapides (règlement des transactions) et la probabilité d'un fork. Un temps de blocage plus rapide rendrait les transactions plus claires mais conduirait à des fourchettes de blockchain plus fréquentes, tandis qu'un temps de blocage plus lent réduirait le nombre de fourchettes mais ralentirait le règlement.

Exploitation minière et course au hachage

L'exploitation minière de Bitcoin est une industrie extrêmement compétitive. La puissance de hachage a augmenté de façon exponentielle chaque année d'existence de Bitcoin. Certaines années, la croissance a reflété un changement complet de technologie, comme en 2010 et 2011, lorsque de nombreux mineurs sont passés de l'exploitation minière CPU à l'extraction GPU et à l'exploitation minière FPGA (Field Programmable Gate Array). En 2013, l'introduction de l'exploitation minière ASIC a conduit à un autre bond de géant dans la puissance minière, en plaçant la fonction SHA256 directement sur des puces de silicium spécialisées à des fins d'exploitation minière.

Les premières puces de ce type pourraient fournir plus de puissance minière dans un seul boîtier que l'ensemble du réseau Bitcoin en 2010.

La liste suivante montre la puissance de hachage totale du réseau bitcoin en térahashes / sec (TH / sec), depuis sa création en 2009 (source : Blockchain.com) :

2009
0,000004 - 0,00001 TH / s (2,40 × croissance)

2010
0,00001 - 0,14 TH / s (14247 × croissance)

2011
0,14 - 9,49 TH / s (63,92 × croissance)

2012
9,49 - 22 TH / s (2,32 × croissance)

2013
22,04 - 15 942 TH / s (723,32 × croissance)

2014
15 942 - 306 333 TH / s (19,21 × croissance)

2015
306,333 - 881,232 TH / s (2,87 × croissance)

2016
881232 - 2807540 TH / s (3,18 × croissance)

2017

2807540 - 18206558 TH / s (6,48 × croissance)

2018

18,206,558 - 41,801,528 TH / s (2,29 × croissance)

2019

41,801,528 - 109,757,127 TH / s (2,62 × croissance)

2020

109,757,127 - 149,064,869 TH / s (1,35 × croissance)

Dans le graphique Puissance de hachage totale, térahashes par seconde (TH / s) (graphique sur une échelle linéaire) , nous pouvons voir que la puissance de hachage du réseau bitcoin a augmenté au cours des deux dernières années. Comme vous pouvez le voir, la concurrence entre les mineurs et la croissance du bitcoin a entraîné une augmentation exponentielle de la puissance de hachage (total des hachages par seconde sur le réseau).

Figure 7. Puissance de hachage totale, térahashes par seconde (TH / s) (graphique sur une échelle linéaire)

Au fur et à mesure que la quantité de puissance de hachage appliquée à l'extraction de bitcoin a explosé, la difficulté a augmenté pour l'égaler. La métrique de difficulté dans le graphique montré dans la métrique de difficulté minière de Bitcoin (graphique sur une échelle logarithmique) est mesurée comme un rapport de la difficulté actuelle sur la difficulté minimale (la difficulté du premier bloc).

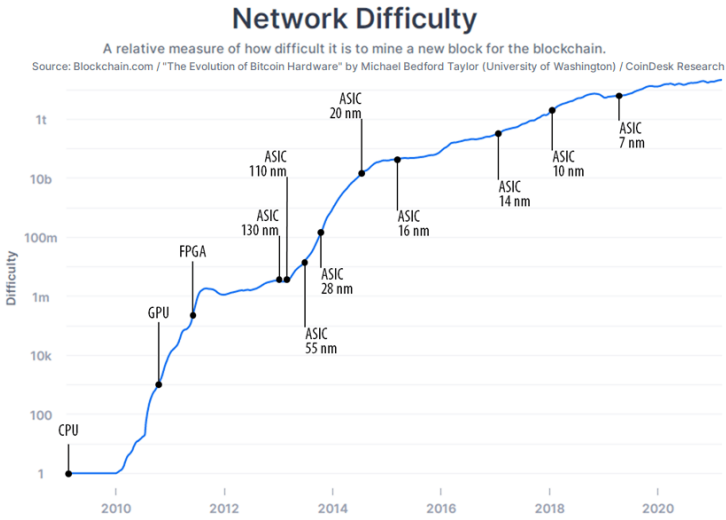


Figure 8. Métrique de difficulté d'extraction de Bitcoin (graphique sur une échelle logarithmique)

Au cours des deux dernières années, les puces minières ASIC sont devenues de plus en plus denses, se rapprochant de la pointe de la fabrication du silicium avec une taille de fonction (résolution) de 7 nanomètres (nm). Actuellement, les fabricants d'ASIC visent à dépasser les fabricants de puces de processeur à usage général, en concevant des puces avec une taille de caractéristique de 5 nm, car la rentabilité de l'exploitation minière conduit cette industrie encore plus rapidement que l'informatique générale. Il n'y a plus de sauts de géant dans l'extraction de bitcoins, car l'industrie a atteint le premier rang de la loi de Moore, qui stipule que la densité de calcul doublera environ tous les 18 mois. Pourtant, la puissance minière du réseau continue de progresser à un rythme exponentiel alors que la course aux puces à densité plus élevée s'accompagne

d'une course aux centres de données à densité plus élevée où des milliers de ces puces peuvent être déployées. Il ne s'agit plus de la quantité d'extraction qui peut être effectuée avec une seule puce, mais du nombre de puces pouvant être insérées dans un bâtiment, tout en dissipant la chaleur et en fournissant une puissance adéquate.

La solution Extra Nonce

Depuis 2012, l'extraction de bitcoins a évolué pour résoudre une limitation fondamentale dans la structure de l'en-tête de bloc. Au début du bitcoin, un mineur pouvait trouver un bloc en itérant dans le nonce jusqu'à ce que le hachage résultant soit égal ou inférieur à la cible. À mesure que la difficulté augmentait, les mineurs parcouraient souvent les 4 milliards de valeurs du nonce sans trouver un bloc. Cependant, cela a été facilement résolu en mettant à jour l'horodatage du bloc pour tenir compte du temps écoulé. Étant donné que l'horodatage fait partie de l'en-tête, la modification permettrait aux mineurs d'itérer à nouveau les valeurs du nonce avec des résultats différents. Cependant, une fois que le matériel minier dépassait 4 GH / s, cette approche devenait de plus en plus difficile car les valeurs nonce étaient épuisées en moins d'une seconde. Alors que l'équipement minier ASIC commençait à pousser puis à dépasser le taux de hachage TH / s, le logiciel minier avait besoin de plus d'espace pour les valeurs nonce afin de trouver des blocs valides. L'horodatage pourrait être légèrement allongé, mais le déplacer trop loin dans le futur rendrait le bloc invalide. Une nouvelle source de « changement » était nécessaire dans l'en-tête du bloc. La solution consistait à utiliser la transaction coinbase comme source de valeurs

nonce supplémentaires. Étant donné que le script coinbase peut stocker entre 2 et 100 octets de données, les mineurs ont commencé à utiliser cet espace comme espace nonce supplémentaire, ce qui leur a permis d'explorer une gamme beaucoup plus large de valeurs d'en-tête de bloc pour trouver des blocs valides. La transaction coinbase est incluse dans l'arborescence merkle, ce qui signifie que toute modification du script coinbase entraîne la modification de la racine merkle. Huit octets de nonce supplémentaire, plus les 4 octets de nonce "standard" permettent aux mineurs d'explorer un total de 2 96 (8 suivis de 28 zéros) possibilités *par seconde* sans avoir à modifier l'horodatage. Si, à l'avenir, les mineurs pouvaient utiliser toutes ces possibilités, ils pourraient alors modifier l'horodatage. Il y a également plus d'espace dans le script coinbase pour l'expansion future de l'espace nonce supplémentaire.

Piscines minières

Dans cet environnement hautement compétitif, les mineurs individuels travaillant seuls (également connus sous le nom de mineurs en solo) n'ont aucune chance. La probabilité qu'ils trouvent un bloc pour compenser leurs coûts d'électricité et de matériel est si faible que cela représente un pari, comme jouer à la loterie. Même le système minier ASIC grand public le plus rapide ne peut pas suivre le rythme des systèmes commerciaux qui empilent des dizaines de milliers de ces puces dans des entrepôts géants à proximité des centrales hydroélectriques. Les mineurs collaborent désormais pour former des pools de minage, mettant en commun leur pouvoir de hachage et partageant la récompense entre des milliers de participants. En participant à un pool, les mineurs obtiennent une plus

petite part de la récompense globale, mais sont généralement récompensés chaque jour, ce qui réduit l'incertitude.

Regardons un exemple spécifique. Supposons qu'un mineur a acheté du matériel minier avec un taux de hachage combiné de 14 000 gigahashes par seconde (GH / s), soit 14 TH / s. En 2017, cet équipement coûte environ 2500 USD. Le matériel consomme 1 375 watts (1,3 kW) d'électricité lorsqu'il fonctionne, 33 kW-heures par jour, à un coût de 1 $ à 2 $ par jour à des tarifs d'électricité très bas. À la difficulté actuelle du bitcoin, le mineur pourra extraire en solo un bloc environ une fois tous les 4 ans. Comment calculons-nous cette probabilité ? Il est basé sur un taux de hachage à l'échelle du réseau de 3 EH / s (en 2017), et le taux du mineur de 14 TH / s :

- $P = (14 * 10^{12} / 3 * 10^{18}) * 210240 = 0.98$

… Où 210240 est le nombre de blocs en quatre ans. Le mineur a une probabilité de 98% de trouver un bloc sur quatre ans, sur la base du taux de hachage global au début de la période.

Si le mineur trouve un seul bloc dans ce laps de temps, le paiement de 6,25 bitcoin, à environ 1000 USD par bitcoin, se traduira par un paiement unique de 6250 USD, ce qui produira un bénéfice net d'environ 750 USD. Cependant, la chance de trouver un bloc dans une période de 4 ans dépend de la chance du mineur. Il pourrait trouver deux blocs en 4 ans et réaliser un profit plus important. Ou il pourrait ne pas trouver de bloc pendant 5 ans et subir une grosse perte financière. Pire encore, la difficulté de l'algorithme Bitcoin Proof-of-Work est

susceptible d'augmenter considérablement au cours de cette période, au rythme actuel de croissance de la puissance de hachage, ce qui signifie que le mineur a, au plus, un an pour casser avant même le matériel. est effectivement obsolète et doit être remplacé par du matériel de minage plus puissant. Sur le plan financier, cela n'a de sens qu'à des coûts d'électricité très bas (moins de 1 cent par kW-heure) et uniquement à très grande échelle.

Les pools de minage coordonnent plusieurs centaines ou milliers de mineurs, via des protocoles de pool-mining spécialisés. Les mineurs individuels configurent leur équipement minier pour se connecter à un serveur de pool et spécifient une adresse bitcoin, qui recevra leur part des récompenses. Leur matériel de minage reste connecté au serveur de pool pendant le minage, synchronisant leurs efforts avec les autres mineurs. Ainsi, les mineurs du pool partagent l'effort pour extraire un bloc et ensuite partager les récompenses.

Les blocs réussis paient la récompense à une adresse bitcoin de pool, plutôt qu'à des mineurs individuels. Le serveur de pool effectuera périodiquement des paiements aux adresses bitcoin des mineurs, une fois que leur part des récompenses aura atteint un certain seuil. En règle générale, le serveur de pool facture un pourcentage des récompenses pour la fourniture du service d'extraction de pool.

Les mineurs participant à un pool se sont partagés le travail de recherche d'une solution pour un bloc candidat, gagnant des « parts » pour leur contribution minière. Le pool de minage fixe un objectif plus élevé (difficulté inférieure) pour gagner une

part, généralement plus de 1000 fois plus facile que l'objectif du réseau Bitcoin. Lorsqu'un membre du pool réussit à miner un bloc, la récompense est gagnée par le pool, puis partagée avec tous les mineurs proportionnellement au nombre d'actions qu'ils ont contribué à l'effort.

Les piscines sont ouvertes à tout mineur, grand ou petit, professionnel ou amateur. Un pool aura donc certains participants avec une seule petite machine minière, et d'autres avec un garage rempli de matériel minier haut de gamme. Certains exploiteront quelques dizaines de kilowatt d'électricité, d'autres exploiteront un centre de données consommant un mégawatt d'électricité. Comment un pool minier mesure-t-il les contributions individuelles, afin de répartir équitablement les récompenses, sans possibilité de tricherie ? La réponse est d'utiliser l'algorithme Proof-of-Work de Bitcoin pour mesurer la contribution de chaque mineur de pool, mais avec une difficulté inférieure afin que même les plus petits mineurs de pool gagnent une part assez souvent pour qu'il vaille la peine de contribuer au pool. En définissant une difficulté inférieure pour gagner des parts, le pool mesure la quantité de travail effectuée par chaque mineur. Chaque fois qu'un mineur de pool trouve un hachage d'en-tête de bloc égal ou inférieur à la cible du pool, il prouve qu'elle a effectué le travail de hachage pour trouver ce résultat. Plus important encore, le travail de recherche de partages contribue, de manière statistiquement mesurable, à l'effort global pour trouver un hachage égal ou inférieur à la cible du réseau bitcoin. Des milliers de mineurs qui tentent de trouver des hachages de faible valeur finiront par en trouver un suffisamment bas pour satisfaire la cible du réseau Bitcoin.

Revenons à l'analogie d'un jeu de dés. Si les joueurs de dés lancent des dés avec un objectif de lancer égal ou inférieur à quatre (la difficulté globale du réseau), un pool fixera un objectif plus facile, en comptant le nombre de fois que les joueurs du pool ont réussi à lancer égal ou inférieur à huit. Lorsque les joueurs du pool lancent égal ou inférieur à huit (l'objectif de partage du pool) mais supérieur à quatre (supérieur à la difficulté globale du réseau), ils gagnent des parts, mais ni eux ni le pool ne gagnent la partie parce qu'ils n'atteignent pas la partie. cible (égale ou inférieure à quatre). Les joueurs de pool atteindront beaucoup plus souvent l'objectif de pool le plus facile, ce qui leur permettra de gagner des parts très régulièrement, même s'ils n'atteignent pas l'objectif le plus difficile de gagner la partie. De temps en temps, l'un des joueurs du pool lancera un lancer de dés combiné égal ou inférieur à quatre, le joueur du pool gagne une part et l'ensemble du pool remporte la partie. Ensuite, les gains peuvent être distribués aux joueurs du pool en fonction du montant d'actions que chacun a gagné. Même si l'objectif de huit ou moins ne gagnait pas, c'était un moyen équitable de mesurer les lancers de dés pour les joueurs, et cela produit parfois un lancer de quatre ou moins.

De même, un pool de minage définira un objectif de pool (plus élevé et plus facile) qui garantira qu'un mineur de pool individuel puisse trouver souvent des hachages d'en-tête de bloc égaux ou inférieurs à la cible du pool, gagnant des parts. De temps en temps, l'une de ces tentatives produira un hachage d'en-tête de bloc égal ou inférieur à la cible du réseau bitcoin, ce qui en fera un bloc valide et l'ensemble du pool l'emportera.

Pools gérés

La plupart des pools de minage sont « gérés », ce qui signifie qu'une société ou un individu exécute un serveur de pool. Le propriétaire du serveur de pool est appelé l' *opérateur* du *pool* et il facture aux mineurs du pool un pourcentage des revenus.

Le serveur de pool exécute un logiciel spécialisé et un protocole d'extraction de pool qui coordonnent les activités des mineurs de pool. Le serveur de pool est également connecté à un ou plusieurs nœuds bitcoin complets et a un accès direct à une copie complète de la base de données blockchain. Cela permet au serveur de pool de valider les blocs et les transactions pour le compte des mineurs de pool, les soulageant ainsi de la charge d'exécuter un nœud complet. Pour les mineurs de pool, c'est une considération importante, car un nœud complet nécessite un ordinateur dédié avec au moins 300 à 350 Go de stockage persistant (disque) et au moins 2 à 4 Go de mémoire (RAM). En outre, le logiciel Bitcoin exécuté sur le nœud complet doit être surveillé, entretenu et mis à niveau fréquemment. Tout temps d'arrêt causé par un manque d'entretien ou un manque de ressources nuira à la rentabilité du mineur. Pour de nombreux mineurs, la possibilité de miner sans exécuter un nœud complet est un autre grand avantage de rejoindre un pool géré.

Les mineurs de pool se connectent au serveur de pool à l'aide d'un protocole d'exploration de données tel que Stratum (STM) ou GetBlockTemplate (GBT). Un ancien standard appelé GetWork (GWK) est pour la plupart obsolète depuis fin 2012, car il ne prend pas facilement en charge l'exploitation minière

à des taux de hachage supérieurs à 4 GH / s. Les protocoles STM et GBT créent des *modèles de* bloc qui contiennent un modèle d'en-tête de bloc candidat. Le serveur de pool construit un bloc candidat en agrégeant les transactions, en ajoutant une transaction coinbase (avec un espace nonce supplémentaire), en calculant la racine merkle et en établissant une liaison avec le hachage de bloc précédent. L'en-tête du bloc candidat est ensuite envoyé à chacun des mineurs du pool en tant que modèle. Chaque mineur de pool exploite ensuite le modèle de bloc, à une cible plus élevée (plus facile) que la cible du réseau bitcoin, et renvoie tous les résultats positifs au serveur de pool pour gagner des parts.

Pool de minage peer-to-peer (P2Pool)

Les pools gérés créent la possibilité de tricherie par l'opérateur du pool, qui pourrait diriger l'effort du pool pour doubler les transactions ou invalider les blocs (voir Attaques de consensus). En outre, les serveurs de pool centralisés représentent un point de défaillance unique. Si le serveur de pool est arrêté ou est ralenti par une attaque par déni de service, les mineurs de pool ne peuvent pas miner. En 2011, pour résoudre ces problèmes de centralisation, une nouvelle méthode de pool mining a été proposée et mise en œuvre : P2Pool, un pool de minage peer-to-peer sans opérateur central.

P2Pool fonctionne en décentralisant les fonctions du serveur de pool, en implémentant un système parallèle de type blockchain appelé *chaîne de partage* . Une chaîne de partage est une

blockchain fonctionnant à un niveau de difficulté inférieur à la blockchain Bitcoin. La chaîne de partage permet aux mineurs de pool de collaborer dans un pool décentralisé en exploitant des actions sur la chaîne de partage à un rythme d'un bloc de partage toutes les 30 secondes. Chacun des blocs de la chaîne de partage enregistre une part de récompense proportionnelle pour les mineurs du pool qui contribuent au travail, reportant les actions du bloc de partage précédent. Lorsque l'un des blocs de partage atteint également l'objectif du réseau bitcoin, il est propagé et inclus dans la blockchain Bitcoin, récompensant tous les mineurs de pool qui ont contribué à toutes les actions qui ont précédé le bloc de partage gagnant. Essentiellement, au lieu d'un serveur de pool gardant une trace des partages et des récompenses des mineurs de pool, la chaîne de partage permet à tous les mineurs de pool de garder une trace de toutes les actions en utilisant un mécanisme de consensus décentralisé comme le mécanisme de consensus blockchain de Bitcoin.

Le minage P2Pool est plus complexe que le minage de pool car il nécessite que les mineurs du pool exécutent un ordinateur dédié avec suffisamment d'espace disque, de mémoire et de bande passante Internet pour prendre en charge un nœud bitcoin complet et le logiciel du nœud P2Pool. Les mineurs de P2Pool connectent leur matériel de minage à leur nœud P2Pool local, qui simule les fonctions d'un serveur de pool en envoyant des modèles de blocs au matériel de minage. Sur P2Pool, les mineurs de pool individuels construisent leurs propres blocs candidats, agrégeant les transactions un peu comme les mineurs en solo, mais exploitent ensuite en collaboration sur la chaîne de partage. P2Pool est une approche hybride qui présente l'avantage de recevoir des paiements beaucoup plus granulaires que le minage

en solo, mais sans donner trop de contrôle à un opérateur de pool comme les pools gérés.

Même si P2Pool réduit la concentration d'énergie des exploitants de pools miniers, il est vraisemblablement vulnérable à 51% d'attaques contre la chaîne de partage elle-même. Une adoption beaucoup plus large de P2Pool ne résout pas le problème d'attaque de 51% pour le bitcoin lui-même. Au contraire, P2Pool rend le bitcoin plus robuste dans l'ensemble, dans le cadre d'un écosystème minier diversifié.

Attaques de consensus

Le mécanisme de consensus de Bitcoin est, au moins théoriquement, vulnérable aux attaques de mineurs (ou de pools) qui tentent d'utiliser leur pouvoir de hachage à des fins malhonnêtes ou destructrices. Comme nous l'avons vu, le mécanisme de consensus dépend du fait qu'une majorité de mineurs agissent honnêtement par intérêt personnel. Cependant, si un mineur ou un groupe de mineurs peut atteindre une part significative de la puissance minière, il peut attaquer le mécanisme de consensus afin de perturber la sécurité et la disponibilité du réseau bitcoin.

Il est important de noter que les attaques de consensus ne peuvent affecter que le consensus futur, ou au mieux, le passé le plus récent (des dizaines de blocs). Le grand livre de Bitcoin devient de plus en plus immuable au fil du temps. Alors qu'en théorie, un fork peut être réalisé à n'importe quelle profondeur, en pratique, la puissance de calcul nécessaire pour forcer un fork très profond est immense, ce qui rend les vieux

blocs pratiquement immuables. Les attaques par consensus n'affectent pas non plus la sécurité des clés privées et de l'algorithme de signature (ECDSA). Une attaque par consensus ne peut pas voler des bitcoins, dépenser des bitcoins sans signatures, rediriger des bitcoins ou autrement modifier les transactions passées ou les enregistrements de propriété. Les attaques par consensus ne peuvent affecter que les blocs les plus récents et provoquer des perturbations par déni de service lors de la création de futurs blocs.

Un scénario d'attaque contre le mécanisme de consensus est appelé « l'attaque à 51% ». Dans ce scénario, un groupe de mineurs, contrôlant une majorité (51%) de la puissance de hachage totale du réseau, s'entend pour attaquer le bitcoin. Avec la possibilité d'exploiter la majorité des blocs, les mineurs attaquants peuvent provoquer des « fourchettes » délibérées dans la blockchain et doubler les transactions ou exécuter des attaques par déni de service contre des transactions ou des adresses spécifiques. Une attaque fork / double-depense est l'endroit où l'attaquant provoque l'invalidation de blocs précédemment confirmés en passant sous eux et en reconverrant sur une chaîne alternative. Avec une puissance suffisante, un attaquant peut invalider six blocs ou plus d'affilée, entraînant l'invalidation des transactions considérées comme immuables (six confirmations). Notez qu'une double dépense ne peut être effectuée que sur les propres transactions de l'attaquant, pour lesquelles l'attaquant peut produire une signature valide. Double-dépenser ses propres transactions est rentable si, en invalidant une transaction, l'attaquant peut obtenir un paiement ou un produit d'échange irréversible sans le payer.

Examinons un exemple pratique d'attaque à 51%. Dans le premier chapitre, nous avons examiné une transaction entre Alice et Bob pour une tasse de café. Bob, le propriétaire du café, est prêt à accepter le paiement pour des tasses de café sans attendre la confirmation (extraction dans un bloc), car le risque de dépenser deux fois pour une tasse de café est faible par rapport à la commodité d'un service client rapide. . Ceci est similaire à la pratique des cafés qui acceptent les paiements par carte de crédit sans signature pour des montants inférieurs à 25 $, car le risque de rétrofacturation par carte de crédit est faible tandis que le coût de retarder la transaction pour obtenir une signature est comparativement plus élevé. En revanche, vendre un article plus cher pour Bitcoin court le risque d'une attaque de double dépense, où l'acheteur diffuse une transaction concurrente qui dépense les mêmes entrées (UTXO) et annule le paiement au commerçant. Une attaque à double dépense peut se produire de deux manières : soit avant qu'une transaction ne soit confirmée, soit si l'attaquant profite d'un fork de la blockchain pour annuler plusieurs blocs. Une attaque à 51% permet aux attaquants de doubler leurs propres transactions dans la nouvelle chaîne, annulant ainsi la transaction correspondante dans l'ancienne chaîne.

Dans notre exemple, l'attaquant malveillant Mallory se rend à la galerie de Carol et achète un magnifique triptyque représentant Satoshi Nakamoto dans le rôle de Prométhée. Carol vend des peintures "The Great Fire" pour 250 000 $ en bitcoins à Mallory. Au lieu d'attendre six confirmations ou plus sur la transaction, Carol enveloppe et remet les peintures à Mallory après une seule confirmation. Mallory travaille avec un complice, Paul, qui exploite un grand pool minier, et le complice lance une

attaque à 51% dès que la transaction de Mallory est incluse dans un bloc. Paul ordonne au pool de minage de redéfinir la même hauteur de bloc que le bloc contenant la transaction de Mallory, remplaçant le paiement de Mallory à Carol par une transaction qui double la même entrée que le paiement de Mallory. La transaction à double dépense consomme le même UTXO et le rembourse au portefeuille de Mallory, au lieu de le payer à Carol, ce qui permet essentiellement à Mallory de conserver le bitcoin. Paul dirige ensuite le pool de minage pour exploiter un bloc supplémentaire, afin de rendre la chaîne contenant la transaction de double dépense plus longue que la chaîne d'origine (provoquant une fourchette sous le bloc contenant la transaction de Mallory). Lorsque la fourchette de la blockchain se résout en faveur de la nouvelle chaîne (plus longue), la transaction à double dépensé remplace le paiement initial à Carol. Carol manque maintenant les trois peintures et n'a pas non plus de paiement Bitcoin. Tout au long de cette activité, les participants au pool de minage de Paul peuvent rester totalement inconscients de la tentative de double dépense, car ils minent avec des mineurs automatisés et ne peuvent pas surveiller chaque transaction ou blocage.

Pour se prémunir contre ce type d'attaque, un commerçant vendant des articles de grande valeur doit attendre au moins six confirmations avant de remettre le produit à l'acheteur. Alternativement, le commerçant doit utiliser un compte séquestre multisignature, en attendant à nouveau plusieurs confirmations après le financement du compte séquestre. Plus les confirmations s'écoulent, plus il devient difficile d'invalider une transaction avec une attaque à 51%. Pour les articles de grande valeur, le paiement par bitcoin sera toujours pratique

et efficace même si l'acheteur doit attendre 24 heures pour la livraison, ce qui correspondrait à environ 144 confirmations.

En plus d'une attaque à double dépense, l'autre scénario pour une attaque par consensus est de refuser le service à des participants Bitcoin spécifiques (adresses Bitcoin spécifiques). Un attaquant disposant d'une majorité de la puissance minière peut simplement ignorer des transactions spécifiques. S'ils sont inclus dans un bloc miné par un autre mineur, l'attaquant peut délibérément bifurquer et reminer ce bloc, en excluant à nouveau les transactions spécifiques. Ce type d'attaque peut entraîner un déni de service soutenu contre une adresse ou un ensemble d'adresses spécifique tant que l'attaquant contrôle la majorité de la puissance minière.

Malgré son nom, le scénario d'attaque à 51% ne nécessite pas en réalité 51% de la puissance de hachage. En fait, une telle attaque peut être tentée avec un pourcentage plus faible de la puissance de hachage. Le seuil de 51% est simplement le niveau auquel une telle attaque est presque assurée de réussir. Une attaque consensuelle est essentiellement un bras de fer pour le prochain bloc et le groupe « plus fort » a plus de chances de gagner. Avec moins de puissance de hachage, la probabilité de succès est réduite, car d'autres mineurs contrôlent la génération de certains blocs avec leur puissance minière « honnête ». Une façon de voir les choses est que plus un attaquant a de puissance de hachage, plus le fork qu'il peut créer délibérément longtemps, plus il peut invalider de blocs dans un passé récent ou plus il peut contrôler de blocs dans le futur. Des groupes de recherche sur la sécurité ont utilisé la modélisation statistique pour affirmer que divers types d'attaques par consensus sont possibles avec

aussi peu que 30% de la puissance de hachage.

L'augmentation massive de la puissance de hachage totale a sans doute rendu le bitcoin imperméable aux attaques d'un seul mineur. Il n'y a aucun moyen possible pour un mineur solo de contrôler plus d'un petit pourcentage de la puissance minière totale. Cependant, la centralisation du contrôle causée par les pools miniers a introduit le risque d'attaques à but lucratif par un opérateur de pool minier. L'opérateur de pool dans un pool géré contrôle la construction des blocs candidats et contrôle également quelles transactions sont incluses. Cela donne à l'opérateur du pool le pouvoir d'exclure des transactions ou d'introduire des transactions à double dépense. Si un tel abus de pouvoir est fait de manière limitée et subtile, un opérateur de pool pourrait en théorie tirer profit d'une attaque consensuelle sans se faire remarquer.

Cependant, tous les attaquants ne seront pas motivés par le profit. Un scénario d'attaque potentiel est celui où un attaquant a l'intention de perturber le réseau Bitcoin sans avoir la possibilité de profiter d'une telle perturbation. Une attaque malveillante visant à paralyser le bitcoin nécessiterait d'énormes investissements et une planification secrète, mais pourrait éventuellement être lancée par un attaquant bien financé, très probablement parrainé par l'État. Alternativement, un attaquant bien financé pourrait attaquer le consensus de Bitcoin en accumulant simultanément du matériel de minage, en compromettant les opérateurs de pool et en attaquant d'autres pools avec un déni de service. Tous ces scénarios sont théoriquement possibles, mais de plus en plus impraticables car la puissance de hachage globale du réseau Bitcoin continue de

croître de manière exponentielle.

Sans aucun doute, une attaque consensuelle sérieuse éroderait la confiance dans le bitcoin à court terme, provoquant peut-être une baisse significative des prix. Cependant, le réseau et le logiciel Bitcoin sont en constante évolution, de sorte que les attaques par consensus se heurteraient à des contre-mesures immédiates de la part de la communauté Bitcoin, rendant le Bitcoin plus robuste.

Changer les règles de consensus

Les règles du consensus déterminent la validité des transactions et des blocages. Ces règles sont la base de la collaboration entre tous les nœuds Bitcoin et sont responsables de la convergence de toutes les perspectives locales en une seule blockchain cohérente sur l'ensemble du réseau.

Si les règles de consensus sont invariables à court terme et doivent être cohérentes sur tous les nœuds, elles ne sont pas invariables à long terme. Afin d'évoluer et de développer le système Bitcoin, les règles doivent changer de temps en temps pour s'adapter à de nouvelles fonctionnalités, améliorations ou corrections de bogues. Contrairement au développement logiciel traditionnel, cependant, les mises à niveau vers un système de consensus sont beaucoup plus difficiles et nécessitent une coordination entre tous les participants.

Fourches dures

Dans Blockchain Forks, nous avons examiné comment le réseau bitcoin pouvait brièvement diverger, deux parties du réseau suivant deux branches différentes de la blockchain pendant une courte période. Nous avons vu comment ce processus se déroule naturellement, dans le cadre du fonctionnement normal du réseau et comment le réseau se reconvertit sur une blockchain commune après l'extraction d'un ou plusieurs blocs.

Il existe un autre scénario dans lequel le réseau peut diverger selon les deux chaînes suivantes : une modification des règles de consensus. Ce type de fork est appelé *hard fork* , car après le fork, le réseau ne se reconvertit pas sur une seule chaîne. Au lieu de cela, les deux chaînes évoluent indépendamment. Les hard forks se produisent lorsqu'une partie du réseau fonctionne selon un ensemble de règles de consensus différent de celui du reste du réseau. Cela peut se produire en raison d'un bogue ou d'un changement délibéré dans l'implémentation des règles de consensus.

Les hard forks peuvent être utilisés pour changer les règles du consensus, mais ils nécessitent une coordination entre tous les participants au système. Tous les nœuds qui ne sont pas mis à niveau vers les nouvelles règles de consensus sont incapables de participer au mécanisme de consensus et sont forcés de passer par une chaîne distincte au moment du hard fork. Ainsi, un changement introduit par un hard fork peut être considéré comme non "forward compatible", en ce sens que les systèmes non mis à niveau ne peuvent pas traiter les nouvelles règles de consensus après l'événement hard fork.

Examinons la mécanique d'une hard fork avec un exemple

spécifique.

Une blockchain avec des fourches montre une blockchain avec deux fourchettes. À la hauteur de bloc 4, une fourche à un bloc se produit. C'est le type de fork spontané que nous avons vu dans Blockchain Forks . Avec l'extraction du bloc 5, le réseau se reconvertit sur une chaîne et la fourche est résolue.

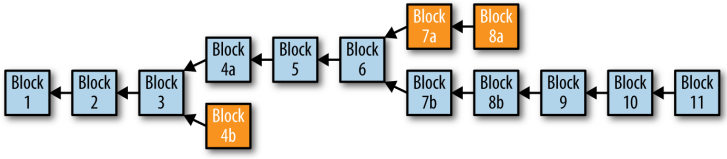


Figure 9. Une blockchain avec des fourches

Plus tard, cependant, à la hauteur de bloc 6, une fourche dure se produit. Supposons qu'une nouvelle implémentation du client soit publiée avec un changement des règles de consensus. À partir de la hauteur de bloc 7, les mineurs exécutant cette nouvelle implémentation accepteront un nouveau type de signature numérique, appelons-le une signature "Smores", qui n'est pas basée sur ECDSA. Immédiatement après, un nœud exécutant la nouvelle implémentation crée une transaction qui contient une signature Smores et un mineur avec le bloc 7b de mines logiciel mis à jour contenant cette transaction.

Tout nœud ou mineur qui n'a pas mis à niveau le logiciel pour valider les signatures Smores est désormais incapable de traiter

le bloc 7b. De leur point de vue, à la fois la transaction qui contenait une signature Smores et le bloc 7b qui contenait cette transaction sont invalides, car ils les évaluent sur la base des anciennes règles de consensus. Ces nœuds rejetteront la transaction et le bloc et ne les propageront pas. Les mineurs qui utilisent les anciennes règles n'accepteront pas le bloc 7b et continueront à miner un bloc candidat dont le parent est le bloc 6. En fait, les mineurs utilisant les anciennes règles peuvent même ne pas recevoir le bloc 7b si tous les nœuds auxquels ils sont connectés sont obéir également aux anciennes règles et donc ne pas propager le bloc. Finalement, ils pourront miner le bloc 7a, qui est valide selon les anciennes règles et ne contient aucune transaction avec les signatures Smores.

Les deux chaînes continuent de diverger à partir de ce point. Les mineurs de la chaîne « b » continueront d'accepter et d'exploiter les transactions contenant des signatures Smores, tandis que les mineurs de la chaîne « a » continueront à ignorer ces transactions. Même si le bloc 8b ne contient aucune transaction signée par Smores, les mineurs de la chaîne "a" ne peuvent pas la traiter. Pour eux, il semble être un bloc orphelin, car son parent "7b" n'est pas reconnu comme un bloc valide.

Hard Forks : logiciel, réseau, exploitation minière et chaîne

Pour les développeurs de logiciels, le terme « fork » a une autre signification, ajoutant de la confusion au terme « hard fork ». Dans les logiciels open source, un fork se produit lorsqu'un groupe de développeurs choisit de suivre une feuille de route logicielle différente et de démarrer une implémentation concur-

rente d'un projet open source. Nous avons déjà discuté de deux circonstances qui conduiront à un hard fork du bitcoin : un bug dans les règles de consensus et une modification délibérée des règles de consensus. Dans le cas d'un changement délibéré des règles de consensus, un fork du logiciel précède le hard fork. Cependant, pour que ce type de hard fork se produise, une nouvelle implémentation logicielle des règles de consensus doit être développée, adoptée et lancée.

Les exemples de fourches logicielles qui ont tenté de modifier les règles de consensus incluent Bitcoin XT, Bitcoin Classic et, plus récemment, Bitcoin Unlimited. Cependant, aucune de ces fourchettes logicielles n'a abouti à une fourche dure. Alors qu'un fork logiciel est une condition préalable nécessaire, il n'est pas en soi suffisant pour qu'un hard fork se produise. Pour qu'une hard fork se produise, l'implémentation concurrente doit être adoptée et les nouvelles règles activées, par les mineurs, les portefeuilles et les nœuds intermédiaires. À l'inverse, il existe de nombreuses implémentations alternatives de Bitcoin Core, et même des fourchettes logicielles, qui ne changent pas les règles de consensus et sauf bogue, peuvent coexister sur le réseau et interopérer sans provoquer de hard fork.

Les règles de consensus peuvent différer de manière évidente et explicite, dans la validation des transactions ou des blocages. Les règles peuvent également différer de manière plus subtile, dans la mise en œuvre des règles de consensus telles qu'elles s'appliquent aux scripts Bitcoin ou aux primitives cryptographiques telles que les signatures numériques. Enfin, les règles de consensus peuvent différer de manière imprévue en raison de contraintes de consensus implicites imposées par les limitations

du système ou les détails de mise en œuvre. Un exemple de ce dernier a été vu dans le hard fork imprévu lors de la mise à niveau de Bitcoin Core 0.7 vers 0.8, qui a été causée par une limitation dans l'implémentation de Berkeley DB utilisée pour stocker des blocs.

Conceptuellement, nous pouvons penser qu'un hard fork se développe en quatre étapes : un fork de logiciel, un fork de réseau, un fork de minage et un chain fork.

Le processus commence lorsqu'une implémentation alternative du client, avec des règles de consensus modifiées, est créée par les développeurs.

Lorsque cette implémentation fourchue est déployée dans le réseau, un certain pourcentage de mineurs, d'utilisateurs de portefeuille et de nœuds intermédiaires peuvent adopter et exécuter cette implémentation. Un fork qui en résultera dépendra du fait que les nouvelles règles de consensus s'appliquent aux blocs, aux transactions ou à un autre aspect du système. Si les nouvelles règles de consensus concernent les transactions, alors un portefeuille créant une transaction selon les nouvelles règles peut précipiter une fourchette de réseau, suivie d'une fourchette dure lorsque la transaction est minée dans un bloc. Si les nouvelles règles concernent des blocs, alors le processus de hard fork commencera lorsqu'un bloc est exploité selon les nouvelles règles.

Tout d'abord, le réseau va bifurquer. Les nœuds basés sur l'implémentation d'origine des règles de consensus rejetteront toutes les transactions et les blocs créés selon les nouvelles

règles. En outre, les nœuds suivant les règles de consensus d'origine interdiront temporairement et se déconnecteront de tous les nœuds qui leur envoient ces transactions et blocs invalides. En conséquence, le réseau se partitionnera en deux : les anciens nœuds ne resteront connectés qu'aux anciens nœuds et les nouveaux nœuds ne seront connectés qu'aux nouveaux nœuds. Une seule transaction ou un seul bloc basé sur les nouvelles règles se répercutera sur le réseau et entraînera la partition en deux réseaux.

Une fois qu'un mineur utilisant les nouvelles règles mine un bloc, la puissance et la chaîne minières se bifurquent également. Les nouveaux mineurs exploiteront le nouveau bloc, tandis que les anciens mineurs exploiteront une chaîne distincte basée sur les anciennes règles. Le réseau partitionné fera en sorte que les mineurs fonctionnant selon des règles de consensus distinctes ne recevront probablement pas les blocs de l'autre, car ils sont connectés à deux réseaux distincts.

Mineurs divergents et difficulté

À mesure que les mineurs divergent pour exploiter deux chaînes différentes, la puissance de hachage est répartie entre les chaînes. La puissance minière peut être répartie dans n'importe quelle proportion entre les deux chaînes. Les nouvelles règles ne peuvent être suivies que par une minorité ou par la grande majorité de la puissance minière.

Supposons, par exemple, une répartition de 80% à 20%, la majorité de la puissance minière utilisant les nouvelles règles de consensus. Supposons également que le fork se produit

immédiatement après une période de reciblage.

Les deux chaînes hériteraient chacune de la difficulté de la période de reciblage. Les nouvelles règles de consensus auraient 80% de la puissance minière disponible auparavant. Du point de vue de cette chaîne, la puissance minière a brutalement baissé de 20% par rapport à la période précédente. Des blocs seront trouvés en moyenne toutes les 12,5 minutes, ce qui représente la baisse de 20% de la puissance minière disponible pour étendre cette chaîne. Ce taux d'émission de blocs se poursuivra (sauf modification de la puissance de hachage) jusqu'à ce que les blocs de 2016 soient extraits, ce qui prendra environ 25 200 minutes (à 12,5 minutes par bloc) ou 17,5 jours. Après 17,5 jours, un reciblage se produira et la difficulté s'ajustera (réduite de 20%) pour produire à nouveau des blocs de 10 minutes, en fonction de la quantité réduite de puissance de hachage dans cette chaîne.

La chaîne minoritaire, qui exploite sous les anciennes règles avec seulement 20% de la puissance de hachage, devra faire face à une tâche beaucoup plus difficile. Sur cette chaîne, les blocs seront désormais minés toutes les 50 minutes en moyenne. La difficulté ne sera pas ajustée pour les blocs de 2016, qui prendront 100 800 minutes, soit environ 10 semaines à miner. En supposant une capacité fixe par bloc, cela entraînera également une réduction de la capacité de transaction d'un facteur 5, car il y a moins de blocs par heure disponibles pour enregistrer les transactions.

Fourches rigides litigieuses

C'est l'aube du développement logiciel consensuel. Tout comme

le développement open source a changé à la fois les méthodes et les produits des logiciels et a créé de nouvelles méthodologies, de nouveaux outils et de nouvelles communautés dans son sillage, le développement de logiciels par consensus représente également une nouvelle frontière en informatique. À partir des débats, des expériences et des tribulations de la feuille de route de développement de Bitcoin, nous verrons émerger de nouveaux outils de développement, pratiques, méthodologies et communautés.

Les hard forks sont considérés comme risqués car ils obligent une minorité à se moderniser ou à rester dans une chaîne minoritaire. Le risque de scinder l'ensemble du système en deux systèmes concurrents est considéré par beaucoup comme un risque inacceptable. En conséquence, de nombreux développeurs hésitent à utiliser le mécanisme de hard fork pour implémenter des mises à niveau des règles de consensus, à moins qu'il n'y ait un soutien quasi unanime de l'ensemble du réseau. Toutes les propositions de hard fork qui ne bénéficient pas d'un soutien quasi unanime sont considérées comme trop « litigieuses » pour être tentées sans risquer une partition du système.

La question des hard forks est très controversée dans la communauté de développement de Bitcoin, en particulier en ce qui concerne les modifications proposées aux règles de consensus contrôlant la limite de taille maximale des blocs. Certains développeurs s'opposent à toute forme de hard fork, la jugeant trop risquée. D'autres voient le mécanisme de hard fork comme un outil essentiel pour améliorer les règles de consensus d'une manière qui évite la « dette technique » et offre une rupture

nette avec le passé. Enfin, certains développeurs considèrent les hard forks comme un mécanisme qui devrait être utilisé rarement, avec beaucoup de planification préalable et seulement sous un consensus quasi unanime.

Nous avons déjà vu l'émergence de nouvelles méthodologies pour faire face aux risques des hard fork. Dans la section suivante, nous examinerons les soft forks et les méthodes BIP-34 et BIP-9 pour la signalisation et l'activation des modifications de consensus.

Fourches souples

Tous les changements de règle de consensus ne provoquent pas une fourchette difficile. Seuls les changements de consensus qui sont incompatibles vers l'avant provoquent un fork. Si le changement est implémenté de telle manière qu'un client non mis à niveau voit toujours la transaction ou le bloc comme valide selon les règles précédentes, le changement peut se produire sans un fork.

Le terme *soft fork a* été introduit pour distinguer cette méthode de mise à niveau d'un « hard fork ». En pratique, une fourche souple n'est pas du tout une fourchette. Un soft fork est un changement compatible avec les règles de consensus qui permet aux clients non mis à niveau de continuer à fonctionner en consensus avec les nouvelles règles.

Un aspect des soft fork qui n'est pas immédiatement évident est que les mises à niveau de soft fork ne peuvent être utilisées que pour contraindre les règles de consensus, pas pour les

étendre. Pour être compatibles avec le transfert, les transactions et les blocs créés sous les nouvelles règles doivent également être valides sous les anciennes règles, mais pas l'inverse. Les nouvelles règles ne peuvent limiter que ce qui est valide ; sinon, ils déclencheront un hard fork lorsqu'ils seront rejetés selon les anciennes règles.

Les fourchettes logicielles peuvent être implémentées de plusieurs manières : le terme ne spécifie pas de méthode particulière, mais plutôt un ensemble de méthodes qui ont toutes une chose en commun : elles ne nécessitent pas que tous les nœuds soient mis à niveau ou forcent les nœuds non mis à niveau à quitter consensus.

Fourches souples redéfinissant les opcodes NOP

Un certain nombre de soft forks ont été implémentés dans Bitcoin, sur la base de la réinterprétation des opcodes NOP. Bitcoin Script avait dix opcodes réservés pour une utilisation future, NOP1 à NOP10. Selon les règles de consensus, la présence de ces opcodes dans un script est interprétée comme un opérateur nul-puissant, ce qui signifie qu'ils n'ont aucun effet. L'exécution continue après l'opcode NOP comme s'il n'y était pas.

Un soft fork peut donc modifier la sémantique d'un code NOP pour lui donner un nouveau sens. Par exemple, BIP-65 (CHECKLOCKTIMEVERIFY) a réinterprété l'opcode NOP2. Les clients implémentant BIP-65 interprètent NOP2 comme OP_CHECKLOCKTIMEVERIFY et imposent une règle de consensus de verrouillage absolu sur UTXO qui contient cet

opcode dans leurs scripts de verrouillage. Ce changement est un soft fork car une transaction valide sous BIP-65 l'est également sur tout client qui n'implémente pas (ignorant) le BIP-65. Pour les anciens clients, le script contient un code NOP, qui est ignoré.

Autres moyens de mise à niveau de la fourche souple

La réinterprétation des opcodes NOP était à la fois prévue et un mécanisme évident pour les mises à niveau consensuelles. Récemment, cependant, un autre mécanisme de soft fork a été introduit qui ne repose pas sur les opcodes NOP pour un type très spécifique de changement de consensus. Ceci est examiné plus en détail dans [segwit] . Segwit est une modification architecturale de la structure d'une transaction, qui déplace le script de déverrouillage (témoin) de l'intérieur de la transaction vers une structure de données externe (en la séparant). Segwit a été initialement envisagé comme une mise à niveau hard fork, car il modifiait une structure fondamentale (transaction). En novembre 2015, un développeur travaillant sur Bitcoin Core a proposé un mécanisme par lequel segwit pourrait être introduit en tant que soft fork. Le mécanisme utilisé pour cela est une modification du script de verrouillage de UTXO créé sous les règles de segwit, de sorte que les clients non mis à niveau voient le script de verrouillage comme échangeable avec n'importe quel script de déverrouillage. En conséquence, segwit peut être introduit sans que chaque nœud soit mis à niveau ou séparé de la chaîne : un soft fork.

Il est probable qu'il existe d'autres mécanismes, encore à découvrir, par lesquels les mises à niveau peuvent être effectuées

de manière compatible avec les avancées en tant que soft fork.

Critiques des fourches souples

Les fourches logicielles basées sur les opcodes NOP sont relativement peu controversées. Les opcodes NOP ont été placés dans Bitcoin Script dans le but explicite de permettre des mises à niveau non perturbatrices.

Cependant, de nombreux développeurs s'inquiètent du fait que d'autres méthodes de mise à niveau de soft fork font des compromis inacceptables. Les critiques courantes des changements de soft fork incluent :

Dette technique

Parce que les fourches souples sont techniquement plus complexes qu'une mise à niveau de hard fork, elles introduisent *une dette technique* , un terme qui fait référence à l'augmentation du coût futur de la maintenance du code en raison des compromis de conception effectués dans le passé. La complexité du code augmente à son tour la probabilité de bogues et de vulnérabilités de sécurité.

Validation relaxation

Les clients non mis à niveau considèrent les transactions comme valides, sans évaluer les règles de consensus modifiées. En effet, les clients non mis à niveau ne valident pas en utilisant la gamme complète des règles de consensus, car ils sont aveugles aux nouvelles règles. Cela s'applique aux mises à niveau basées sur NOP, ainsi qu'à d'autres mises à niveau de soft fork.

Améliorations irréversibles

Parce que les soft forks créent des transactions avec des contraintes de consensus supplémentaires, elles deviennent des mises à niveau irréversibles dans la pratique. Si une mise à niveau soft fork devait être annulée après avoir été activée, toute transaction créée selon les nouvelles règles pourrait entraîner une perte de fonds selon les anciennes règles. Par exemple, si une transaction CLTV est évaluée selon les anciennes règles, il n'y a pas de contrainte de délai et elle peut être dépensée à tout moment. Par conséquent, les critiques soutiennent qu'une fourchette souple échouée qui a dû être inversée en raison d'un bogue entraînerait presque certainement une perte de fonds.

Signalisation de fourche souple avec version bloc

Étant donné que les fourchettes souples permettent aux clients non mis à niveau de continuer à fonctionner dans le cadre d'un consensus, le mécanisme pour « activer » une fourchette souple passe par les mineurs signalant leur disponibilité : une majorité de mineurs doivent convenir qu'ils sont prêts et disposés à appliquer les nouvelles règles de consensus. Pour coordonner leurs actions, il existe un mécanisme de signalisation qui leur permet de montrer leur soutien à un changement de règle de consensus. Ce mécanisme a été introduit avec l'activation du BIP-34 en mars 2013 et remplacé par l'activation du BIP-9 en juillet 2016.

Signalisation et activation du BIP-34

La première implémentation, dans BIP-34, utilisait le champ

de version de bloc pour permettre aux mineurs de signaler qu'ils étaient prêts pour un changement de règle de consensus spécifique. Avant BIP-34, la version bloc était définie sur "1" par *convention* non appliquée par *consensus* .

Le BIP-34 a défini un changement de règle de consensus qui exigeait que le champ de données coinbase d'une entrée de transaction coinbase contienne la hauteur de bloc. Avant BIP-34, les données de la coinbase pouvaient contenir toutes les données arbitraires que les mineurs choisissaient d'inclure. Après l'activation de BIP-34, les blocs valides devaient contenir une hauteur de bloc spécifique au début des données de la coinbase et être identifiés avec un numéro de version supérieur ou égal à "2".

Pour signaler le changement et l'activation du BIP-34, les mineurs définissent la version de bloc sur « 2 » au lieu de « 1 ». Cela n'a pas immédiatement rendu invalides les blocs de la version "1". Une fois activés, les blocs de la version "1" deviendraient invalides et tous les blocs de la version "2" devraient contenir la hauteur de bloc dans la coinbase pour être valides.

Le BIP-34 a défini un mécanisme d'activation en deux étapes, basé sur une fenêtre glissante de 1000 blocs. Un mineur signalerait sa disponibilité individuelle pour BIP-34 en construisant des blocs avec "2" comme numéro de version. À proprement parler, ces blocs n'avaient pas encore à se conformer à la nouvelle règle de consensus d'inclure la hauteur de bloc dans la transaction coinbase car la règle de consensus n'avait pas encore été activée. Les règles de consensus activées en deux étapes :

- Si 75% (750 des 1000 blocs les plus récents) sont marqués avec la version "2", alors les blocs de la version "2" doivent contenir la hauteur de bloc dans la transaction coinbase ou ils sont rejetés comme invalides. Les blocs de la version "1" sont toujours acceptés par le réseau et n'ont pas besoin de contenir une hauteur de bloc. Les anciennes et nouvelles règles de consensus coexistent pendant cette période.
- Lorsque 95% (950 des 1000 blocs les plus récents) sont de la version "2", les blocs de la version "1" ne sont plus considérés comme valides. Les blocs de la version "2" ne sont valides que s'ils contiennent la hauteur de bloc dans la coinbase (selon le seuil précédent). Par la suite, tous les blocs doivent se conformer aux nouvelles règles de consensus, et tous les blocs valides doivent contenir une hauteur de bloc dans la transaction coinbase.
-

Après une signalisation et une activation réussies selon les règles BIP-34, ce mécanisme a été utilisé deux fois de plus pour activer les fourches souples :

- Le codage DER strict des signatures BIP-66 a été activé par une signalisation de style BIP-34 avec une version de bloc "3" et des blocs de version "2" invalides.
- BIP-65 CHECKLOCKTIMEVERIFY a été activé par une signalisation de style BIP-34 avec une version de bloc "4" et des blocs de version "3" invalides.

Après l'activation du BIP-65, le mécanisme de signalisation et d'activation du BIP-34 a été retiré et remplacé par le mécanisme

de signalisation BIP-9 décrit ci-après.

Le standard est défini dans BIP-34 (Block v2, Height in Coinbase) .

Signalisation et activation BIP-9

Le mécanisme utilisé par BIP-34, BIP-66 et BIP-65 a réussi à activer trois fourches souples. Cependant, il a été remplacé car il présentait plusieurs limitations :

- En utilisant la valeur entière de la version de bloc, un seul soft fork pouvait être activé à la fois, il fallait donc une coordination entre les propositions de soft fork et un accord sur leur hiérarchisation et leur séquençage.
- De plus, comme la version en bloc était incrémentée, le mécanisme n'offrait pas un moyen simple de rejeter un changement et d'en proposer un autre. Si d'anciens clients étaient toujours en cours d'exécution, ils pourraient confondre la signalisation d'un nouveau changement avec la signalisation de la modification précédemment rejetée.
- Chaque nouvelle modification réduisait irrévocablement les versions de bloc disponibles pour les modifications futures.

Le BIP-9 a été proposé pour surmonter ces défis et améliorer le rythme et la facilité de mise en œuvre des changements futurs.

BIP-9 interprète la version de bloc comme un champ de bits au lieu d'un entier. Étant donné que la version de bloc était à l'origine utilisée comme un entier, les versions 1 à 4, seuls 29 bits restent disponibles pour être utilisés comme champ de

bits. Cela laisse 29 bits qui peuvent être utilisés pour signaler indépendamment et simultanément la disponibilité sur 29 propositions différentes.

Le BIP-9 définit également un temps maximum pour la signalisation et l'activation. De cette façon, les mineurs n'ont pas besoin de signaler pour toujours. Si une proposition n'est pas activée dans le délai TIMEOUT (défini dans la proposition), la proposition est considérée comme rejetée. La proposition peut être soumise à nouveau pour la signalisation avec un bit différent, renouvelant la période d'activation.

De plus, une fois que le TEMPS D'EXTINCTION s'est écoulé et qu'une fonction a été activée ou rejetée, le bit de signalisation peut être réutilisé pour une autre caractéristique sans confusion. Par conséquent, jusqu'à 29 changements peuvent être signalés en parallèle et après TIMEOUT, les bits peuvent être "recyclés" pour proposer de nouveaux changements.

Note

Alors que les bits de signalisation peuvent être réutilisés ou recyclés, tant que la période de vote ne se chevauche pas, les auteurs du BIP-9 recommandent que les bits ne soient réutilisés que lorsque cela est nécessaire ; un comportement inattendu peut se produire en raison de bogues dans des logiciels plus anciens. En bref, nous ne devrions pas nous attendre à voir une réutilisation tant que les 29 bits n'ont pas été utilisés une fois.

Les modifications proposées sont identifiées par une structure

de données qui contient les champs suivants :

Nom

Une brève description utilisée pour distinguer les propositions. Le plus souvent, le BIP décrivant la proposition, comme "bipN", où N est le numéro BIP.

bit

0 à 28, le bit de la version bloc que les mineurs utilisent pour signaler l'approbation de cette proposition.

Heure de début

Heure (basée sur le temps passé médian ou MTP) à laquelle la signalisation commence après laquelle la valeur du bit est interprétée comme indiquant l'état de préparation de la proposition.

heure de fin

Le temps (basé sur MTP) après lequel la modification est considérée comme rejetée si elle n'a pas atteint le seuil d'activation.

Contrairement au BIP-34, le BIP-9 compte la signalisation d'activation dans des intervalles entiers en fonction de la période de reciblage de difficulté des blocs de 2016. Pour chaque période de reciblage, si la somme des blocs de signalisation pour une proposition dépasse 95% (1916 sur 2016), la proposition sera activée une période de reciblage plus tard.

BIP-9 propose un diagramme d'état de proposition pour illustrer les différentes étapes et transitions d'une proposition,

comme indiqué dans le diagramme de transition d'état BIP-9 .

Les propositions commencent dans l'état DEFINED, une fois que leurs paramètres sont connus (définis) dans le logiciel Bitcoin. Pour les blocs avec MTP après l'heure de début, l'état de la proposition passe à STARTED. Si le seuil de vote est dépassé au cours d'une période de reciblage et que le délai n'a pas été dépassé, l'état de la proposition passe à LOCKED_IN. Une période de reciblage plus tard, la proposition devient ACTIVE. Les propositions restent à l'état ACTIF perpétuellement une fois qu'elles atteignent cet état. Si le délai d'attente s'écoule avant que le seuil de vote n'ait été atteint, l'état de la proposition passe à FAILED, indiquant une proposition rejetée. Les propositions ÉCHOUÉES restent perpétuellement dans cet état.

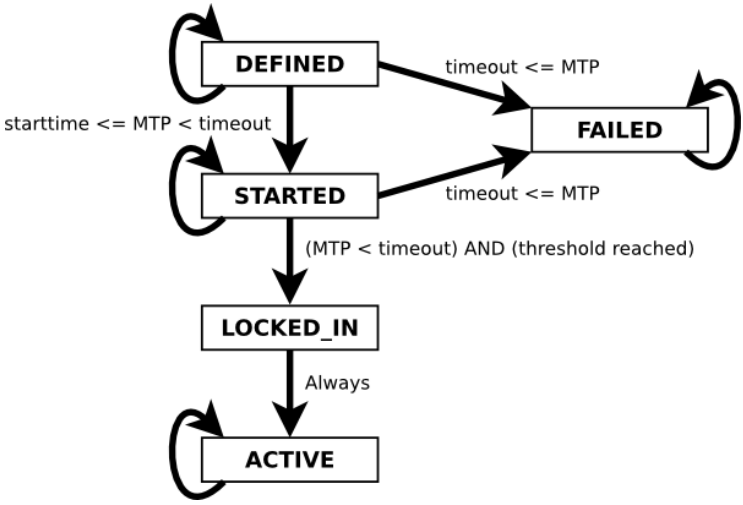


Figure 10. Diagramme de transition d'état BIP-9

Le BIP-9 a été mis en œuvre pour la première fois pour l'activation de CHECKSEQUENCEVERIFY et des BIP associés (68, 112, 113). La proposition nommée « csv » a été activée avec succès en juillet 2016.

La norme est définie dans BIP-9 (bits de version avec timeout et delay) .

Développement de logiciels de consensus

Le logiciel de consensus continue d'évoluer et il y a beaucoup de discussions sur les différents mécanismes pour changer les règles de consensus. De par sa nature même, le bitcoin place la barre très haut en matière de coordination et de consensus pour les changements. En tant que système décentralisé, il n'a aucune « autorité » qui puisse imposer sa volonté aux participants du réseau. Le pouvoir est diffusé entre plusieurs groupes tels que les mineurs, les développeurs principaux, les développeurs de portefeuilles, les bourses, les commerçants et les utilisateurs finaux. Les décisions ne peuvent être prises unilatéralement par aucune de ces circonscriptions. Par exemple, si les mineurs peuvent théoriquement modifier les règles à la majorité simple (51%), ils sont contraints par le consentement des autres circonscriptions. S'ils agissent unilatéralement, le reste des participants peut simplement refuser de les suivre, maintenant l'activité économique sur une chaîne minoritaire. Sans activité économique (transactions, marchands, portefeuilles, échanges), les mineurs exploiteront une pièce sans valeur avec des blocs vides. Cette diffusion du pouvoir signifie que tous les participants doivent se coordonner, sinon aucun changement ne peut être apporté. Le statu quo est l'état stable de ce système

avec seulement quelques changements possibles s'il existe un consensus fort à une très large majorité. Le seuil de 95% pour les fourches souples reflète cette réalité.

Il est important de reconnaître qu'il n'existe pas de solution parfaite pour le développement d'un consensus. Les fourches dures et les fourches souples impliquent des compromis. Pour certains types de changements, les fourches souples peuvent être un meilleur choix ; pour d'autres, les fourches dures peuvent être un meilleur choix. Il n'y a pas de choix parfait ; les deux comportent des risques. La seule caractéristique constante du développement de logiciels consensuels est que le changement est difficile et que le consensus force le compromis.

Certains y voient une faiblesse des systèmes de consensus. Avec le temps, vous pourrez peut-être le voir comme moi, comme la plus grande force du système.

12

Sécurité Bitcoin

Sécuriser le bitcoin est un défi car le bitcoin n'est pas une référence abstraite à la valeur, comme un solde sur un compte bancaire. Le Bitcoin ressemble beaucoup à l'argent ou à l'or numérique. Vous avez probablement entendu l'expression : « La possession est les neuf dixièmes de la loi ». Eh bien, en bitcoin, la possession représente dix dixièmes de la loi. La possession des clés pour déverrouiller le bitcoin équivaut à la possession d'argent liquide ou d'un morceau de métal précieux. Vous pouvez le perdre, l'égarer, le faire voler ou donner accidentellement le mauvais montant à quelqu'un. Dans chacun de ces cas, les utilisateurs n'ont aucun recours, comme s'ils déposaient de l'argent sur un trottoir public.

Cependant, le bitcoin a des capacités que les comptes en espèces, en or et en banque n'ont pas. Un portefeuille Bitcoin, contenant vos clés, peut être sauvegardé comme n'importe quel fichier. Il peut être stocké en plusieurs copies, même imprimé sur papier pour une sauvegarde papier. Vous ne pouvez pas "sauvegarder" des comptes en espèces, en or ou en banque.

Bitcoin est suffisamment différent de tout ce qui est arrivé auparavant que nous devons également penser à la sécurité Bitcoin d'une manière nouvelle.

Principes de sécurité

Le principe de base de Bitcoin est la décentralisation et cela a des implications importantes pour la sécurité. Un modèle centralisé, comme une banque traditionnelle ou un réseau de paiement, dépend du contrôle d'accès et de la vérification pour garder les mauvais acteurs hors du système. En comparaison, un système décentralisé comme Bitcoin pousse la responsabilité et le contrôle aux utilisateurs. Étant donné que la sécurité du réseau est basée sur la preuve de travail et non sur le contrôle d'accès, le réseau peut être ouvert et aucun cryptage n'est requis pour le trafic Bitcoin.

Sur un réseau de paiement traditionnel, tel qu'un système de carte de crédit, le paiement est illimité car il contient l'identifiant privé de l'utilisateur (le numéro de carte de crédit). Après la facturation initiale, toute personne ayant accès à l'identifiant peut « extraire » des fonds et facturer le propriétaire encore et encore. Ainsi, le réseau de paiement doit être sécurisé de bout en bout par cryptage et doit garantir qu'aucune écoute ou intermédiaire ne puisse compromettre le trafic de paiement, en transit ou lorsqu'il est stocké (au repos). Si un mauvais acteur accède au système, il peut compromettre les transactions en cours *et les* jetons de paiement qui peuvent être utilisés pour créer de nouvelles transactions. Pire encore, lorsque les

données des clients sont compromises, les clients sont exposés au vol d'identité et doivent prendre des mesures pour empêcher l'utilisation frauduleuse des comptes compromis.

Le Bitcoin est radicalement différent. Une transaction Bitcoin n'autorise qu'une valeur spécifique à un destinataire spécifique et ne peut pas être falsifiée ou modifiée. Il ne révèle aucune information privée, telle que l'identité des parties, et ne peut être utilisé pour autoriser des paiements supplémentaires. Par conséquent, un réseau de paiement Bitcoin n'a pas besoin d'être crypté ou protégé contre les écoutes clandestines. En fait, vous pouvez diffuser des transactions Bitcoin sur un canal public ouvert, tel que le WiFi non sécurisé ou Bluetooth, sans perte de sécurité.

Le modèle de sécurité décentralisé de Bitcoin met beaucoup de pouvoir entre les mains des utilisateurs. Avec ce pouvoir vient la responsabilité de maintenir le secret des clés. Pour la plupart des utilisateurs, ce n'est pas facile à faire, en particulier sur les appareils informatiques à usage général tels que les smartphones ou les ordinateurs portables connectés à Internet. Bien que le modèle décentralisé de Bitcoin empêche le type de compromis de masse observé avec les cartes de crédit, de nombreux utilisateurs ne sont pas en mesure de sécuriser adéquatement leurs clés et de se faire pirater, un par un.

Développer des systèmes Bitcoin en toute sécurité

Le principe le plus important pour les développeurs Bitcoin est la décentralisation. La plupart des développeurs seront familiarisés avec les modèles de sécurité centralisés et pourraient

être tentés d'appliquer ces modèles à leurs applications Bitcoin, avec des résultats désastreux.

La sécurité de Bitcoin repose sur un contrôle décentralisé des clés et sur une validation indépendante des transactions par les mineurs. Si vous souhaitez tirer parti de la sécurité de Bitcoin, vous devez vous assurer de rester dans le modèle de sécurité Bitcoin. En termes simples : ne retirez pas le contrôle des clés aux utilisateurs et ne retirez pas les transactions de la blockchain.

Par exemple, de nombreux premiers échanges de bitcoins ont concentré tous les fonds des utilisateurs dans un seul portefeuille « chaud » avec des clés stockées sur un seul serveur. Une telle conception supprime le contrôle des utilisateurs et centralise le contrôle des clés dans un seul système. De nombreux systèmes de ce type ont été piratés, avec des conséquences désastreuses pour leurs clients.

Une autre erreur courante est de retirer les transactions "de la blockchain" dans un effort malavisé pour réduire les frais de transaction ou accélérer le traitement des transactions. Un système « hors blockchain » enregistrera les transactions sur un registre interne centralisé et ne les synchronisera qu'occasionnellement avec la blockchain Bitcoin. Cette pratique, encore une fois, remplace la sécurité Bitcoin décentralisée par une approche propriétaire et centralisée. Lorsque les transactions sont hors blockchain, des registres centralisés mal sécurisés peuvent être falsifiés, détournant des fonds et épuisant les réserves, sans que l'on s'en aperçoive.

À moins que vous ne soyez prêt à investir massivement dans la sécurité opérationnelle, les multiples niveaux de contrôle d'accès et les audits (comme le font les banques traditionnelles), vous devez réfléchir très attentivement avant de prendre des fonds en dehors du contexte de sécurité décentralisé de Bitcoin. Même si vous disposez des fonds et de la discipline nécessaires pour mettre en œuvre un modèle de sécurité robuste, une telle conception ne fait que reproduire le modèle fragile des réseaux financiers traditionnels, en proie au vol d'identité, à la corruption et au détournement de fonds. Pour tirer parti du modèle de sécurité décentralisé unique de Bitcoin , vous devez éviter la tentation d'architectures centralisées qui peuvent sembler familières mais qui finissent par compromettre la sécurité de Bitcoin.

La racine de la confiance

L'architecture de sécurité traditionnelle est basée sur un concept appelé *racine de confiance* , qui est un noyau de confiance utilisé comme base pour la sécurité du système ou de l'application dans son ensemble. L'architecture de sécurité est développée autour de la racine de la confiance sous la forme d'une série de cercles concentriques, comme des couches dans un oignon, étendant la confiance vers l'extérieur depuis le centre. Chaque couche s'appuie sur la couche interne plus fiable à l'aide de contrôles d'accès, de signatures numériques, de chiffrement et d'autres primitives de sécurité. À mesure que les systèmes logiciels deviennent plus complexes, ils sont plus susceptibles de contenir des bogues, ce qui les rend vulnérables aux compromissions de sécurité. En conséquence, plus un système logiciel devient complexe, plus il est difficile à sécuriser. Le concept de racine

de confiance garantit que la plus grande partie de la confiance est placée dans la partie la moins complexe du système, et donc la moins vulnérable, du système, tandis que des logiciels plus complexes sont superposés. Cette architecture de sécurité est répétée à différentes échelles, établissant d'abord une racine de confiance au sein du matériel d'un seul système, puis étendant cette racine de confiance à travers le système d'exploitation aux services système de niveau supérieur, et enfin à de nombreux serveurs disposés en cercles concentriques de diminution de la confiance.

L'architecture de sécurité Bitcoin est différente. Dans Bitcoin, le système de consensus crée un registre public de confiance qui est complètement décentralisé. Une blockchain correctement validée utilise le bloc de genèse comme racine de confiance, construisant une chaîne de confiance jusqu'au bloc actuel. Les systèmes Bitcoin peuvent et doivent utiliser la blockchain comme racine de confiance. Lors de la conception d'une application Bitcoin complexe composée de services sur de nombreux systèmes différents, vous devez examiner attentivement l'architecture de sécurité afin de déterminer où la confiance est placée. En fin de compte, la seule chose à laquelle on devrait explicitement faire confiance est une blockchain entièrement validée. Si votre application investit explicitement ou implicitement la confiance dans autre chose que la blockchain, cela devrait être une source de préoccupation car cela introduit une vulnérabilité. Une bonne méthode pour évaluer l'architecture de sécurité de votre application consiste à prendre en compte chaque composant individuel et à évaluer un scénario hypothétique où ce composant est complètement compromis et sous le contrôle d'un acteur malveillant. Prenez

chaque composant de votre application, à tour de rôle, et évaluez les impacts sur la sécurité globale si ce composant est compromis. Si votre application n'est plus sécurisée lorsque des composants sont compromis, cela montre que vous avez perdu la confiance dans ces composants. Une application Bitcoin sans vulnérabilités ne devrait être vulnérable qu'à un compromis du mécanisme de consensus Bitcoin, ce qui signifie que sa racine de confiance est basée sur la partie la plus forte de l'architecture de sécurité Bitcoin.

Les nombreux exemples d'échanges de bitcoins piratés servent à souligner ce point car leur architecture et leur conception de sécurité échouent même sous le contrôle le plus informel. Ces implémentations centralisées avaient investi explicitement la confiance dans de nombreux composants en dehors de la blockchain Bitcoin, tels que les portefeuilles actifs, les bases de données de grand livre centralisées, les clés de cryptage vulnérables et des schémas similaires.

Meilleures pratiques de sécurité des utilisateurs

Les humains utilisent des contrôles de sécurité physique depuis des milliers d'années. En comparaison, notre expérience de la sécurité numérique a moins de 50 ans. Les systèmes d'exploitation modernes à usage général ne sont pas très sécurisés et pas particulièrement adaptés au stockage de monnaie numérique. Nos ordinateurs sont constamment exposés à des menaces externes via des connexions Internet toujours actives. Ils exécutent des milliers de composants logiciels de centaines d'auteurs, souvent avec un accès sans restriction aux fichiers

de l'utilisateur. Un seul logiciel malveillant, parmi les milliers installés sur votre ordinateur, peut compromettre votre clavier et vos fichiers, en volant tout bitcoin stocké dans les applications de portefeuille. Le niveau de maintenance informatique requis pour garder un ordinateur exempt de virus et de chevaux de Troie dépasse le niveau de compétence de tous, sauf une infime minorité d'utilisateurs d'ordinateurs.

Malgré des décennies de recherche et de progrès dans le domaine de la sécurité de l'information, les actifs numériques sont toujours terriblement vulnérables à un adversaire déterminé. Même les systèmes les plus protégés et les plus restreints, dans les sociétés de services financiers, les agences de renseignement et les entreprises de défense, sont fréquemment violés. Bitcoin crée des actifs numériques qui ont une valeur intrinsèque et peuvent être volés et détournés vers de nouveaux propriétaires instantanément et irrévocablement. Cela crée une incitation massive pour les pirates. Jusqu'à présent, les pirates devaient convertir les informations d'identité ou les jetons de compte, tels que les cartes de crédit et les comptes bancaires, en valeur après les avoir compromis. Malgré la difficulté de clôturer et de blanchir les informations financières, nous avons assisté à une augmentation constante des vols. Bitcoin aggrave ce problème car il n'a pas besoin d'être clôturé ou blanchi ; c'est la valeur intrinsèque d'un actif numérique.

Heureusement, le bitcoin crée également des incitations à améliorer la sécurité informatique. Alors qu'auparavant le risque de compromission informatique était vague et indirect, Bitcoin rend ces risques clairs et évidents. La détention de bitcoin sur un ordinateur permet de concentrer l'esprit de

l'utilisateur sur la nécessité d'améliorer la sécurité informatique. En conséquence directe de la prolifération et de l'adoption accrue du bitcoin et d'autres monnaies numériques, nous avons assisté à une escalade des techniques de piratage et des solutions de sécurité. En termes simples, les pirates informatiques ont désormais une cible très juteuse et les utilisateurs ont clairement intérêt à se défendre.

Au cours des trois dernières années, en conséquence directe de l'adoption du bitcoin, nous avons assisté à d'énormes innovations dans le domaine de la sécurité de l'information sous la forme de cryptage matériel, de stockage de clés et de portefeuilles matériels, de technologie multisignature et de séquestre numérique . Dans les sections suivantes, nous examinerons diverses meilleures pratiques pour la sécurité pratique des utilisateurs.

Stockage Bitcoin physique

Étant donné que la plupart des utilisateurs sont beaucoup plus à l'aise avec la sécurité physique que la sécurité des informations, une méthode très efficace pour protéger les bitcoins consiste à les convertir en forme physique. Les clés Bitcoin ne sont rien de plus que de longs nombres. Cela signifie qu'ils peuvent être stockés sous une forme physique, telle que imprimée sur papier ou gravée sur une pièce de monnaie en métal. La sécurisation des clés devient alors aussi simple que la sécurisation physique de la copie imprimée des clés bitcoin. Un ensemble de clés bitcoin imprimées sur papier est appelé « portefeuille papier », et il existe de nombreux outils gratuits qui peuvent être utilisés pour les créer. Personnellement, je garde la grande majorité

de mes bitcoins (99% ou plus) stockés sur des portefeuilles en papier, cryptés avec BIP-38, avec plusieurs copies verrouillées dans des coffres-forts. Garder le bitcoin hors ligne s'appelle *le stockage à froid* et c'est l'une des techniques de sécurité les plus efficaces. Un système de stockage à froid est un système dans lequel les clés sont générées sur un système hors ligne (jamais connecté à Internet) et stockées hors ligne sur papier ou sur un support numérique, comme une clé USB.

Portefeuilles matériels

À long terme, la sécurité Bitcoin prendra de plus en plus la forme de portefeuilles matériels inviolables. Contrairement à un smartphone ou à un ordinateur de bureau, un portefeuille matériel bitcoin n'a qu'un seul objectif : conserver le bitcoin en toute sécurité. Sans logiciel polyvalent à compromettre et avec des interfaces limitées, les portefeuilles matériels peuvent offrir un niveau de sécurité presque infaillible aux utilisateurs non experts. Je m'attends à voir les portefeuilles matériels devenir la méthode prédominante de stockage Bitcoin. Pour un exemple d'un tel portefeuille matériel, voir le Trezor.

Équilibrer les risques

Bien que la plupart des utilisateurs soient à juste titre préoccupés par le vol de bitcoins, le risque est encore plus grand. Les fichiers de données sont perdus tout le temps. S'ils contiennent du bitcoin, la perte est beaucoup plus douloureuse. Dans l'effort de sécuriser leurs portefeuilles Bitcoin, les utilisateurs doivent faire très attention à ne pas aller trop loin et finir par perdre le Bitcoin. En juillet 2011, un projet bien connu de sensibilisation

et d'éducation sur les bitcoins a perdu près de 7 000 bitcoins . Dans leurs efforts pour empêcher le vol, les propriétaires avaient mis en place une série complexe de sauvegardes cryptées. En fin de compte, ils ont accidentellement perdu les clés de cryptage, rendant les sauvegardes sans valeur et perdant une fortune. Comme cacher de l'argent en l' enterrant dans le désert, si vous sécurisez trop bien votre bitcoin, vous ne pourrez peut-être pas le retrouver.

Diversifier les risques

Porteriez-vous l'intégralité de votre valeur nette en espèces dans votre portefeuille ? La plupart des gens considéreraient que les utilisateurs imprudents, mais les utilisateurs de bitcoins gardent souvent tous leurs bitcoins dans un seul portefeuille. Au lieu de cela, les utilisateurs devraient répartir le risque entre plusieurs portefeuilles Bitcoin divers. Les utilisateurs prudents ne conserveront qu'une petite fraction, peut-être moins de 5%, de leur bitcoin dans un portefeuille en ligne ou mobile en tant que « monnaie de poche ». Le reste doit être réparti entre quelques mécanismes de stockage différents, tels qu'un portefeuille de bureau et hors ligne (stockage à froid).

Multisig et gouvernance

Chaque fois qu'une entreprise ou un individu stocke de grandes quantités de bitcoin, il doit envisager d'utiliser une adresse bitcoin multisignature . Les adresses multi-signatures sécurisent les fonds en exigeant un nombre minimum de signatures pour effectuer un paiement. Les clés de signature doivent être stockées dans un certain nombre d' endroits différents et sous

le contrôle de différentes personnes. Dans un environnement d'entreprise, par exemple, les clés doivent être générées indépendamment et détenues par plusieurs dirigeants de l'entreprise, afin qu'aucune personne ne puisse compromettre les fonds. Les adresses multi-signatures peuvent également offrir une redondance, où une seule personne détient plusieurs clés qui sont stockées à différents endroits.

Survivabilité

Une considération de sécurité importante qui est souvent négligée est la disponibilité, en particulier dans le contexte de l'incapacité ou du décès du détenteur de la clé. Les utilisateurs de Bitcoin doivent utiliser des mots de passe complexes et garder leurs clés sécurisées et privées, sans les partager avec qui que ce soit. Malheureusement, cette pratique rend presque impossible pour la famille de l'utilisateur de récupérer des fonds si l'utilisateur n'est pas disponible pour les déverrouiller. Dans la plupart des cas, en fait, les familles d'utilisateurs de Bitcoin peuvent ne pas être complètement au courant de l'existence des fonds Bitcoin.

Si vous avez beaucoup de bitcoins, vous devriez envisager de partager les détails d'accès avec un parent de confiance ou un avocat. Un système de survie plus complexe peut être mis en place avec un accès multi-signature et une planification successorale par l'intermédiaire d'un avocat spécialisé en tant que « liquidateur d'actifs numériques ».

Conclusion

Bitcoin est une technologie complètement nouvelle, sans précédent et complexe. Au fil du temps, nous développerons de meilleurs outils et pratiques de sécurité qui seront plus faciles à utiliser par des non-experts. Pour l'instant, les utilisateurs de Bitcoin peuvent utiliser de nombreux conseils discutés ici pour profiter d'une expérience Bitcoin sécurisée et sans problème.

13

Applications de la blockchain

Développons maintenant notre compréhension du bitcoin en le considérant comme une *plate-forme d'application* . De nos jours, de nombreuses personnes utilisent le terme «blockchain» pour désigner toute plate-forme d'application qui partage les principes de conception du bitcoin. Le terme est souvent mal utilisé et appliqué à de nombreuses choses qui ne fournissent pas les fonctionnalités principales fournies par la blockchain de Bitcoin.

Dans ce chapitre, nous examinerons les fonctionnalités offertes par la blockchain Bitcoin, en tant que plate-forme d'application. Nous examinerons les *primitives de* construction d'application , qui forment les blocs de construction de toute application blockchain. Nous examinerons plusieurs applications importantes qui utilisent ces primitives, telles que les canaux de paiement (état) et les canaux de paiement acheminés (Lightning Network).

Introduction

Le système Bitcoin a été conçu comme un système de paiement et de monnaie décentralisé. Cependant, la plupart de ses fonctionnalités sont dérivées de constructions de niveau beaucoup plus bas qui peuvent être utilisées pour des applications beaucoup plus larges. Bitcoin n'a pas été construit avec des composants tels que des comptes, des utilisateurs, des soldes et des paiements. Au lieu de cela, il utilise un langage de script transactionnel avec des fonctions cryptographiques de bas niveau, comme nous l'avons vu dans [transactions] . Tout comme les concepts de plus haut niveau des comptes, des soldes et des paiements peuvent être dérivés de ces primitives de base, de nombreuses autres applications complexes le peuvent également. Ainsi, la blockchain Bitcoin peut devenir une plateforme d'applications offrant des services de confiance aux applications, telles que les contrats intelligents, dépassant de loin l'objectif initial de la monnaie numérique et des paiements.

Blocs de construction (primitifs)

Lorsqu'il fonctionne correctement et sur le long terme, le système Bitcoin offre certaines garanties, qui peuvent être utilisées comme blocs de construction pour créer des applications. Ceux-ci inclus :

Pas de double-dépense

La garantie la plus fondamentale de l'algorithme de consensus

décentralisé de Bitcoin garantit qu'aucun UTXO ne peut être dépensé deux fois.

Immutabilité

Une fois qu'une transaction est enregistrée dans la blockchain et qu'un travail suffisant a été ajouté avec les blocs suivants, les données de la transaction deviennent immuables. L'immuabilité est garantie par l'énergie, car la réécriture de la blockchain nécessite la dépense d'énergie pour produire une preuve de travail. L'énergie requise et donc le degré d'immuabilité augmentent avec la quantité de travail engagé au-dessus du bloc contenant une transaction.

Neutralité

Le réseau Bitcoin décentralisé propage des transactions valides indépendamment de l'origine ou du contenu de ces transactions. Cela signifie que n'importe qui peut créer une transaction valide avec des frais suffisants et avoir confiance en sa capacité à transmettre cette transaction et à l'inclure dans la blockchain à tout moment.

Horodatage sécurisé

Les règles de consensus rejettent tout bloc dont l'horodatage est trop éloigné dans le passé ou le futur. Cela garantit que les horodatages sur les blocs peuvent être fiables. L'horodatage sur un bloc implique une garantie non dépensée avant pour les entrées de toutes les transactions incluses.

Autorisation

Les signatures numériques, validées dans un réseau décentralisé, offrent des garanties d'autorisation. Les scripts qui

contiennent une exigence de signature numérique ne peuvent pas être exécutés sans l'autorisation du détenteur de la clé privée impliquée dans le script.

Auditabilité

Toutes les transactions sont publiques et peuvent être auditées. Toutes les transactions et tous les blocs peuvent être liés dans une chaîne ininterrompue au bloc de genèse.

Comptabilité

Dans toute transaction (à l'exception de la transaction coinbase), la valeur des entrées est égale à la valeur des sorties plus les frais. Il n'est pas possible de créer ou de détruire de la valeur bitcoin dans une transaction. Les sorties ne peuvent pas dépasser les entrées.

Nonexpiration

Une transaction valide n'expire pas. S'il est valable aujourd'hui, il le sera dans un proche avenir, tant que les contributions ne seront pas dépensées et que les règles du consensus ne changeront pas.

Intégrité

Une transaction bitcoin signée avec SIGHASH_ALL ou des parties d'une transaction signée par un autre type SIGHASH ne peut pas être modifiée sans invalider la signature, invalidant ainsi la transaction elle-même.

Atomicité de transaction

Les transactions Bitcoin sont atomiques. Ils sont valides et confirmés (minés) ou non. Les transactions partielles ne

peuvent pas être exploitées et il n'y a pas d'état provisoire pour une transaction. À tout moment, une transaction est soit minée, soit non.

Unités de valeur discrètes (indivisibles)

Les sorties de transaction sont des unités de valeur discrètes et indivisibles. Ils peuvent être dépensés ou non, en totalité. Ils ne peuvent être divisés ou partiellement dépensés.

Quorum de contrôle

Les contraintes de multi-signature dans les scripts imposent un quorum d'autorisation, prédéfini dans le schéma de multi-signature. L'exigence M-of-N est appliquée par les règles de consensus.

Timelock / Vieillissement

Toute clause de script contenant un blocage temporel relatif ou absolu ne peut être exécutée qu'une fois que son âge a dépassé le temps spécifié.

Réplication

Le stockage décentralisé de la blockchain garantit que lorsqu'une transaction est minée, après des confirmations suffisantes, elle est répliquée sur le réseau et devient durable et résiliente face aux coupures de courant, aux pertes de données, etc.

Protection contre la falsification

Une transaction ne peut dépenser que des sorties existantes et validées. Il n'est pas possible de créer ou de contrefaire de la valeur.

Cohérence

En l'absence de partitions de mineur, les blocs enregistrés dans la blockchain sont sujets à une réorganisation ou à un désaccord avec une probabilité décroissante exponentiellement, en fonction de la profondeur à laquelle ils sont enregistrés. Une fois profondément enregistrés, le calcul et l'énergie nécessaires pour changer rendent le changement pratiquement impossible.

Enregistrement de l'état externe

Une transaction peut valider une valeur de données, via OP_RETURN, représentant une transition d'état dans une machine à états externe.

Émission prévisible

Moins de 21 millions de bitcoins seront émis, à un rythme prévisible.

La liste des blocs de construction n'est pas complète et d'autres sont ajoutés à chaque nouvelle fonctionnalité introduite dans Bitcoin.

Applications des blocs de construction

Les blocs de construction offerts par Bitcoin sont des éléments d'une plateforme de confiance qui peuvent être utilisés pour composer des applications. Voici quelques exemples d'applications qui existent aujourd'hui et les blocs de construction qu'elles utilisent :

Preuve d'existence (notaire numérique)

Immuabilité + horodatage + durabilité. Une empreinte numérique peut être validée avec une transaction sur la blockchain, prouvant qu'un document existait (horodatage) au moment où il a été enregistré. L'empreinte digitale ne peut pas être modifiée ex post facto (Immuabilité) et la preuve sera conservée de manière permanente (Durabilité).

Kickstarter (phare)

Cohérence + Atomicité + Intégrité. Si vous signez une entrée et la sortie (intégrité) d'une transaction de collecte de fonds, d'autres peuvent contribuer à la collecte de fonds, mais elles ne peuvent pas être dépensées (atomicité) tant que l'objectif (valeur de sortie) n'est pas financé (cohérence).

Canaux de paiement

Quorum de contrôle + Timelock + Pas de double dépense + Nonexpiration + Résistance à la censure + Autorisation. Un multisig 2 sur 2 (Quorum) avec un délai (Timelock) utilisé comme transaction de « règlement » d'un canal de paiement peut être détenu (Non-expiration) et dépensé à tout moment (Résistance à la censure) par l'une ou l'autre des parties (Autorisation). Les deux parties peuvent alors créer des transactions d'engagement qui double-dépensent (pas de double-dépense) le règlement sur un délai plus court (Timelock).

Contrepartie

Counterparty est une couche de protocole construite sur Bitcoin. Le protocole Counterparty offre la possibilité de créer

et d'échanger des actifs virtuels et des jetons. En outre, Counterparty propose un échange d'actifs décentralisé. Counterparty met également en œuvre des contrats intelligents, basés sur la machine virtuelle Ethereum (EVM).

Counterparty intègre des métadonnées dans les transactions Bitcoin, en utilisant l'opcode OP_RETURN ou des adresses multisignatures 1-of-N qui codent des métadonnées à la place des clés publiques. En utilisant ces mécanismes, Counterparty implémente une couche de protocole encodée dans les transactions Bitcoin. La couche de protocole supplémentaire peut être interprétée par des applications prenant en charge les contreparties, telles que les portefeuilles et les explorateurs de chaînes de blocs, ou toute application créée à l'aide des bibliothèques de contrepartie.

La contrepartie peut être utilisée comme plate-forme pour d'autres applications et services, à leur tour. Par exemple, Tokenly est une plate-forme construite sur Counterparty qui permet aux créateurs de contenu, aux artistes et aux entreprises d'émettre des jetons qui expriment la propriété numérique et peuvent être utilisés pour louer, accéder, échanger ou acheter du contenu, des produits et des services. D'autres applications exploitant Counterparty incluent des jeux (Spells of Genesis) et des projets de calcul de grille (Folding Coin).

Plus de détails sur Counterparty sont disponibles sur https ://counterparty.io . Le projet open source peut être trouvé à https ://github.com/CounterpartyXCP .

Canaux de paiement et canaux d'état

Les canaux de paiement sont un mécanisme sans confiance pour échanger des transactions Bitcoin entre deux parties, en dehors de la blockchain Bitcoin. Ces transactions, qui seraient valides si elles étaient réglées sur la blockchain Bitcoin, sont plutôt tenues hors chaîne, agissant comme *des billets à ordre* pour un éventuel règlement par lots. Étant donné que les transactions ne sont pas réglées, elles peuvent être échangées sans la latence de règlement habituelle, ce qui permet un débit de transaction extrêmement élevé, une latence faible (submilliseconde) et une granularité fine (niveau satoshi).

En fait, le terme *canal* est une métaphore. Les canaux d'état sont des constructions virtuelles représentées par l'échange d'état entre deux parties, en dehors de la blockchain. Il n'y a pas de "canaux" en soi et le mécanisme de transport de données sous-jacent n'est pas le canal. Nous utilisons le terme canal pour représenter la relation et l'état partagé entre deux parties, en dehors de la blockchain.

Pour expliquer plus en détail ce concept, pensez à un flux TCP. Du point de vue des protocoles de niveau supérieur, il s'agit d'une « prise » reliant deux applications sur Internet. Mais si vous regardez le trafic réseau, un flux TCP n'est qu'un canal virtuel sur des paquets IP. Chaque point de terminaison des séquences de flux TCP et assemble des paquets IP pour créer l'illusion d'un flux d'octets. En dessous, ce sont tous des paquets déconnectés. De même, un canal de paiement n'est qu'une série de transactions. S'ils sont correctement séquencés et connectés,

ils créent des obligations remboursables auxquelles vous pouvez faire confiance même si vous ne faites pas confiance à l'autre côté du canal.

Dans cette section, nous examinerons différentes formes de canaux de paiement. Tout d'abord, nous examinerons les mécanismes utilisés pour construire un canal de paiement unidirectionnel (unidirectionnel) pour un service de micro-paiement à compteur, tel que la vidéo en streaming. Ensuite, nous développerons ce mécanisme et introduirons des canaux de paiement bidirectionnels. Enfin, nous verrons comment les canaux bidirectionnels peuvent être connectés de bout en bout pour former des canaux multi-sauts dans un réseau routé, d'abord proposé sous le nom de *Lightning Network* .

Les canaux de paiement font partie du concept plus large d'un *canal d'état* , qui représente une modification d'état hors chaîne, sécurisée par un éventuel règlement dans une blockchain. Un canal de paiement est un canal d'état où l'état en cours de modification est le solde d'une monnaie virtuelle.

Canaux d'état - Concepts de base et terminologie

Un canal d'état est établi entre deux parties, via une transaction qui verrouille un état partagé sur la blockchain. C'est ce qu'on appelle la *transaction de financement* ou *transaction d'ancrage* . Cette transaction unique doit être transmise au réseau et exploitée pour établir le canal. Dans l'exemple d'un canal de paiement, l'état verrouillé est le solde initial (en devise) du canal.

Les deux parties échangent alors des transactions signées,

appelées *transactions d'engagement* , qui modifient l'état initial. Ces transactions sont des transactions valides en ce qu'elles *peuvent* être soumises pour règlement par l'une ou l'autre des parties, mais sont à la place tenues hors chaîne par chaque partie en attendant la fermeture du canal. Les mises à jour d'état peuvent être créées aussi rapidement que chaque partie peut créer, signer et transmettre une transaction à l'autre partie. En pratique, cela signifie que des milliers de transactions par seconde peuvent être échangées.

Lors de l'échange de transactions d'engagement, les deux parties invalident également les états précédents, de sorte que la transaction d'engagement la plus à jour est toujours la seule qui puisse être remboursée. Cela empêche l'une ou l'autre des parties de tricher en fermant unilatéralement la chaîne avec un état antérieur expiré qui leur est plus favorable que l'état actuel. Nous examinerons les différents mécanismes qui peuvent être utilisés pour invalider l'état antérieur dans la suite de ce chapitre.

Enfin, le canal peut être fermé soit de manière coopérative, en soumettant une *transaction de règlement* finale à la blockchain, soit unilatéralement, par l'une ou l'autre des parties soumettant la dernière transaction d'engagement à la blockchain. Une option de fermeture unilatérale est nécessaire au cas où l'une des parties se déconnecterait de manière inattendue. La transaction de règlement représente l'état final du canal et est réglée sur la blockchain.

Pendant toute la durée de vie du canal, seules deux transactions doivent être soumises pour l'exploitation minière sur la blockchain : les transactions de financement et de règlement.

Entre ces deux États, les deux parties peuvent échanger un nombre illimité de transactions d'engagement qui ne sont jamais vues par personne d'autre, ni soumises à la blockchain.

Un canal de paiement entre Bob et Alice, montrant les transactions de financement, d'engagement et de règlement, illustre un canal de paiement entre Bob et Alice, montrant les transactions de financement, d'engagement et de règlement.

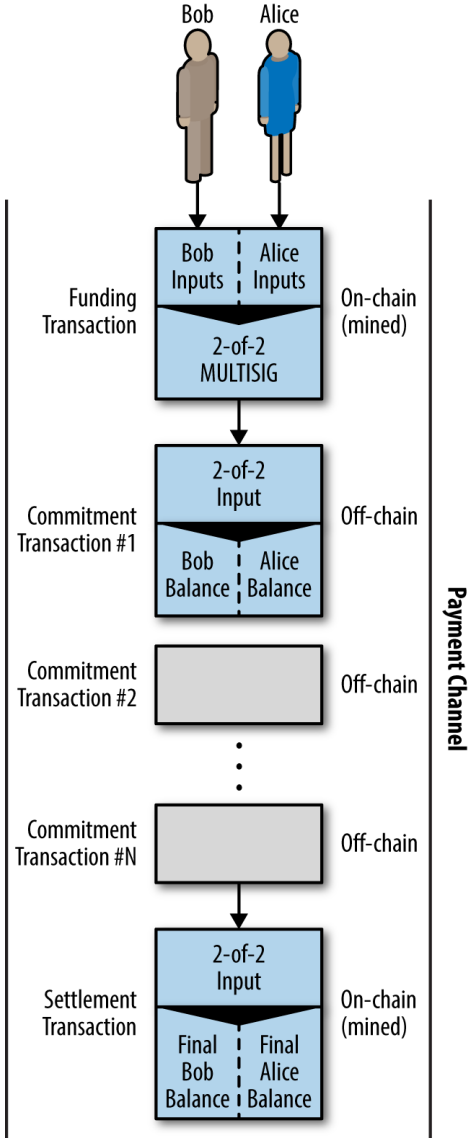


Figure 1. Un canal de paiement entre Bob et Alice, montrant les transactions de financement, d'engagement et de règlement

Exemple de canal de paiement simple

Pour expliquer les canaux d'état, nous commençons par un exemple très simple. Nous démontrons un canal à sens unique, ce qui signifie que la valeur circule dans une seule direction. Nous commencerons également par l'hypothèse naïve que personne n'essaie de tricher, pour garder les choses simples. Une fois que nous aurons expliqué l'idée de base de la chaîne, nous examinerons ce qu'il faut pour la rendre sans confiance afin qu'aucune des parties *ne puisse* tricher, même si elles essaient de le faire.

Pour cet exemple, nous supposerons deux participants : Emma et Fabian. Fabian propose un service de streaming vidéo facturé à la seconde via un canal de micropaiement. Fabian facture 0,01 millibit (0,00001 BTC) par seconde de vidéo, ce qui équivaut à 36 millibits (0,036 BTC) par heure de vidéo. Emma est un utilisateur qui achète ce service de vidéo en streaming auprès de Fabian. Emma achète une vidéo en streaming à Fabian avec un canal de paiement, payant pour chaque seconde de vidéo montre Emma achetant le service de streaming vidéo à Fabian en utilisant un canal de paiement.

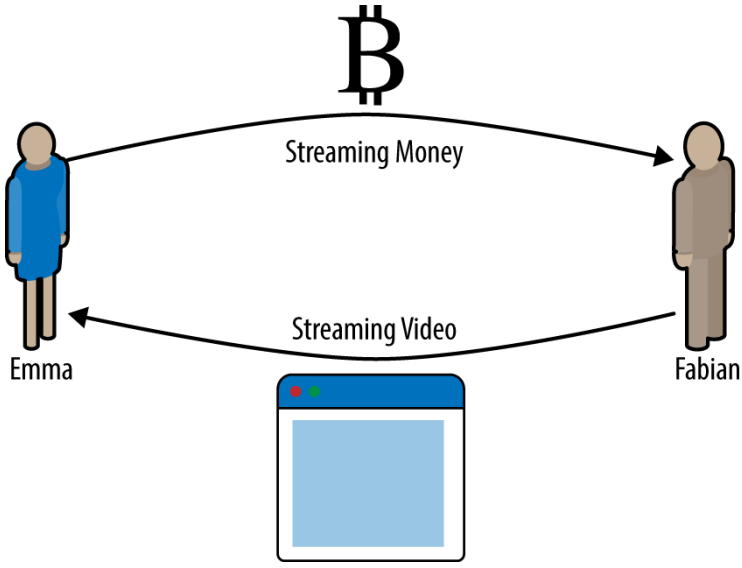


Figure 2. Emma achète une vidéo en streaming à Fabian avec un canal de paiement, en payant pour chaque seconde de vidéo

Dans cet exemple, Fabian et Emma utilisent un logiciel spécial qui gère à la fois le canal de paiement et le streaming vidéo. Emma exécute le logiciel dans son navigateur, Fabian l'exécute sur un serveur. Le logiciel comprend les fonctionnalités de base du portefeuille Bitcoin et peut créer et signer des transactions Bitcoin. Tant le concept que le terme « canal de paiement » sont complètement cachés aux utilisateurs. Ce qu'ils voient, c'est une vidéo payée à la seconde.

Pour configurer le canal de paiement, Emma et Fabian établissent une adresse multisignature 2 sur 2, chacun d'eux détenant l'une des clés. Du point de vue d'Emma, le logiciel

de son navigateur présente un code QR avec une adresse P2SH (commençant par « 3 »), et lui demande de soumettre un « dépôt » pour jusqu'à 1 heure de vidéo. L'adresse est ensuite financée par Emma. La transaction d'Emma, payant à l'adresse multisignature, est la transaction de financement ou d'ancrage pour le canal de paiement.

Pour cet exemple, disons qu'Emma finance la chaîne avec 36 millibits (0,036 BTC). Cela permettra à Emma de consommer *jusqu'à* 1 heure de vidéo en streaming. Dans ce cas, la transaction de financement définit le montant maximum qui peut être transmis dans ce canal, en définissant la *capacité* du *canal* .

La transaction de financement consomme une ou plusieurs entrées du portefeuille d'Emma, source des fonds. Il crée une sortie d'une valeur de 36 millibits payée à l'adresse multisignature 2 sur 2 contrôlée conjointement par Emma et Fabian. Il peut avoir des sorties supplémentaires pour revenir au portefeuille d'Emma.

Une fois la transaction de financement confirmée, Emma peut commencer à diffuser la vidéo. Le logiciel d'Emma crée et signe une transaction d'engagement qui modifie le solde du canal pour créditer 0,01 millibit à l'adresse de Fabian et rembourser 35,99 millibits à Emma. La transaction signée par Emma consomme les 36 millibits de sortie créés par la transaction de financement et crée deux sorties : l'une pour son remboursement, l'autre pour le paiement de Fabian. La transaction n'est que partiellement signée - elle nécessite deux signatures (2 sur 2), mais n'a que la signature d'Emma. Lorsque le serveur de Fabian reçoit cette transaction, il ajoute

la deuxième signature (pour l'entrée 2 sur 2) et la renvoie à Emma avec 1 seconde de vidéo. Désormais, les deux parties ont une transaction d'engagement entièrement signée qui peut être échangée, ce qui représente le solde à jour correct du canal.

Lors du cycle suivant, le logiciel d'Emma crée et signe une autre transaction d'engagement (engagement n ° 2) qui consomme le *même résultat* 2 sur 2 de la transaction de financement. La deuxième transaction d'engagement alloue une sortie de 0,02 millibits à l'adresse de Fabian et une sortie de 35,98 millibits à l'adresse d'Emma. Cette nouvelle transaction est le paiement de deux secondes cumulées de vidéo. Le logiciel de Fabian signe et renvoie la deuxième transaction d'engagement, avec une autre seconde de vidéo.

De cette manière, le logiciel d'Emma continue d'envoyer des transactions d'engagement au serveur de Fabian en échange d'une vidéo en streaming. L'équilibre de la chaîne s'accumule progressivement en faveur de Fabian, car Emma consomme plus de secondes de vidéo. Disons qu'Emma regarde 600 secondes (10 minutes) de vidéo, créant et signant 600 transactions d'engagement. La dernière transaction d'engagement (# 600) aura deux sorties, divisant le solde du canal, 6 millibits à Fabian et 30 millibits à Emma.

Enfin, Emma sélectionne "Arrêter" pour arrêter la diffusion de la vidéo. Fabian ou Emma peuvent désormais transmettre la transaction d'état finale pour règlement. Cette dernière transaction est la *transaction de règlement* et paie à Fabian toute la vidéo consommée par Emma, remboursant le reste de la transaction de financement à Emma.

Le canal de paiement d'Emma avec Fabian, montrant les transactions d'engagement qui mettent à jour le solde du canal, montre le canal entre Emma et Fabian et les transactions d'engagement qui mettent à jour le solde du canal.

Au final, seules deux transactions sont enregistrées sur la blockchain : la transaction de financement qui a établi le canal et une transaction de règlement qui répartit correctement le solde final entre les deux participants.

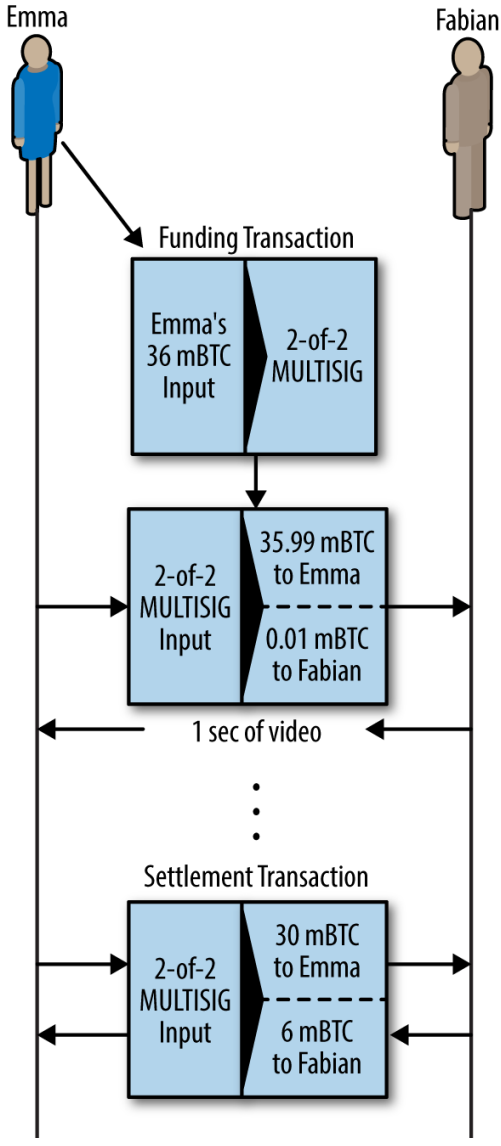


Figure 3. Canal de paiement d'Emma avec Fabian, montrant les transactions d'engagement qui mettent à jour le solde du canal

Créer des canaux sans confiance

La chaîne que nous venons de décrire fonctionne, mais seulement si les deux parties coopèrent, sans échec ni tentative de triche. Examinons certains des scénarios qui cassent ce canal et voyons ce qui est nécessaire pour les résoudre :

- Une fois la transaction de financement effectuée, Emma a besoin de la signature de Fabian pour récupérer son argent. Si Fabian disparaît, les fonds d'Emma sont bloqués dans un 2 sur 2 et effectivement perdus. Ce canal, tel qu'il est construit, conduit à une perte de fonds si l'une des parties se déconnecte avant qu'il y ait au moins une transaction d'engagement signée par les deux parties.

- Pendant que le canal fonctionne, Emma peut prendre n'importe laquelle des transactions d'engagement que Fabian a contresignées et en transmettre une à la blockchain. Pourquoi payer 600 secondes de vidéo, si elle peut transmettre la transaction d'engagement n ° 1 et ne payer que 1 seconde de vidéo ? La chaîne échoue car Emma peut tricher en diffusant un engagement préalable qui est en sa faveur.

Ces deux problèmes peuvent être résolus avec des timelocks - voyons comment nous pourrions utiliser des timelocks au niveau des transactions (nLocktime).

Emma ne peut pas risquer de financer un multisig 2 sur 2 à moins d'avoir un remboursement garanti. Pour résoudre ce

problème, Emma construit les opérations de financement et de remboursement en même temps. Elle signe la transaction de financement mais ne la transmet à personne. Emma transmet uniquement la transaction de remboursement à Fabian et obtient sa signature.

La transaction de remboursement agit comme la première transaction d'engagement et son échéancier établit la limite supérieure pour la durée de vie de la chaîne. Dans ce cas, Emma pourrait définir le nLocktime sur 30 jours ou 4320 blocs dans le futur. Toutes les transactions d'engagement ultérieures doivent avoir un délai plus court, afin qu'elles puissent être échangées avant la transaction de remboursement.

Maintenant qu'Emma a une transaction de remboursement entièrement signée, elle peut transmettre en toute confiance la transaction de financement signée en sachant qu'elle peut éventuellement, après l'expiration du délai, racheter la transaction de remboursement même si Fabian disparaît.

Chaque transaction d'engagement que les parties échangeront pendant la durée de vie de la chaîne sera verrouillée dans le futur. Mais le délai sera légèrement plus court pour chaque engagement, de sorte que l'engagement le plus récent pourra être remboursé avant que l'engagement précédent ne soit invalidé. En raison du nLockTime, aucune des parties ne peut propager avec succès l'une des transactions d'engagement jusqu'à ce que leur délai expire. Si tout se passe bien, ils coopéreront et fermeront le canal gracieusement avec une transaction de règlement, rendant inutile la transmission d'une transaction d'engagement intermédiaire. Sinon, la transaction

d'engagement la plus récente peut être propagée pour régler le compte et invalider toutes les transactions d'engagement antérieures.

Par exemple, si la transaction d'engagement n ° 1 est verrouillée dans le temps à 4320 blocs dans le futur, alors la transaction d'engagement n ° 2 est verrouillée dans le temps à 4319 blocs dans le futur. La transaction d'engagement n ° 600 peut être dépensée 600 blocs avant que la transaction d'engagement n ° 1 ne devienne valide.

Chaque engagement fixe un délai plus court, lui permettant d'être dépensé avant que les engagements précédents ne deviennent valides , montre que chaque transaction d'engagement définit un délai plus court, ce qui lui permet d'être dépensé avant que les engagements précédents ne deviennent valides.

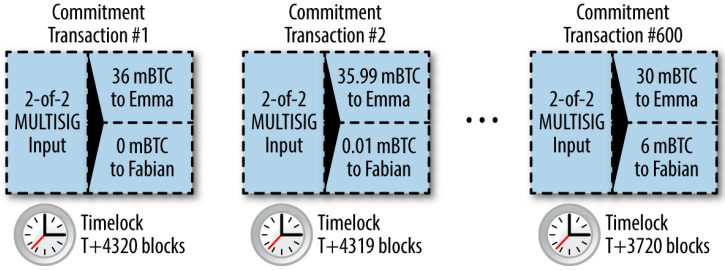


Figure 4. Chaque engagement fixe un délai plus court, ce qui lui permet d'être dépensé avant que les engagements précédents ne deviennent valides

Chaque transaction d'engagement ultérieure doit avoir un délai

plus court pour pouvoir être diffusée avant ses prédécesseurs et avant la transaction de remboursement. La possibilité de diffuser un engagement plus tôt garantit qu'il sera en mesure de dépenser le produit du financement et empêchera toute autre transaction d'engagement d'être remboursée en dépensant le produit. Les garanties offertes par la blockchain bitcoin, empêchant les doubles dépenses et imposant des délais, permettent effectivement à chaque transaction d'engagement d'invalider ses prédécesseurs.

Les chaînes d'État utilisent des délais pour appliquer des contrats intelligents dans une dimension temporelle. Dans cet exemple, nous avons vu comment la dimension temporelle garantit que la transaction d'engagement la plus récente devient valide avant tout engagement antérieur. Ainsi, la transaction d'engagement la plus récente peut être transmise, dépensant les entrées et annulant les transactions d'engagement antérieures. L'application de contrats intelligents avec des délais absolus protège contre la tricherie par l'une des parties. Cette implémentation ne nécessite rien de plus que des délais absolus au niveau des transactions (nLocktime). Ensuite, nous verrons comment les timelocks au niveau du script, CHECKLOCKTIMEVERIFY et CHECKSEQUENCEVERIFY, peuvent être utilisés pour construire des canaux d'état plus flexibles, utiles et sophistiqués.

La première forme de canal de paiement unidirectionnel a été présentée comme un prototype d'application de streaming vidéo en 2015 par une équipe de développeurs argentins.

Les délais ne sont pas le seul moyen d'invalider les transactions d'engagement antérieures. Dans les sections suivantes, nous

verrons comment une clé de révocation peut être utilisée pour obtenir le même résultat. Les timelocks sont efficaces mais ils présentent deux inconvénients distincts. En établissant un délai maximal lors de la première ouverture du canal, ils limitent la durée de vie du canal. Pire encore, ils obligent les implémentations de chaînes à trouver un équilibre entre l'autorisation des chaînes à longue durée de vie et le fait de forcer l'un des participants à attendre très longtemps un remboursement en cas de fermeture prématurée. Par exemple, si vous autorisez la chaîne à rester ouverte pendant 30 jours, en définissant le délai de remboursement sur 30 jours, si l'une des parties disparaît immédiatement, l'autre partie doit attendre 30 jours pour un remboursement. Plus le point final est éloigné, plus le remboursement est éloigné.

Le deuxième problème est que, puisque chaque transaction d'engagement ultérieure doit décrémenter le délai, il existe une limite explicite sur le nombre de transactions d'engagement qui peuvent être échangées entre les parties. Par exemple, un canal de 30 jours, définissant un délai de 4320 blocs dans le futur, ne peut accepter que 4320 transactions d'engagement intermédiaire avant de devoir être clôturé. Il existe un danger en définissant l'intervalle de transaction d'engagement de verrouillage temporel à 1 bloc. En définissant l'intervalle de temps entre les transactions d'engagement à 1 bloc, un développeur crée un fardeau très lourd pour les participants au canal qui doivent être vigilants, rester en ligne et regarder, et être prêts à transmettre la bonne transaction d'engagement à tout moment.

Maintenant que nous comprenons comment les délais peuvent

être utilisés pour invalider des engagements antérieurs, nous pouvons voir la différence entre fermer la chaîne en coopération et la fermer unilatéralement en diffusant une transaction d'engagement. Toutes les transactions d'engagement sont verrouillées dans le temps, par conséquent, la diffusion d'une transaction d'engagement impliquera toujours d'attendre l'expiration du délai. Mais si les deux parties s'entendent sur le solde final et savent qu'elles détiennent toutes deux des transactions d'engagement qui finiront par faire de ce solde une réalité, elles peuvent construire une transaction de règlement sans délai représentant ce même solde. Lors d'une clôture coopérative, l'une ou l'autre des parties prend la transaction d'engagement la plus récente et construit une transaction de règlement qui est identique à tous points de vue sauf qu'elle omet le délai. Les deux parties peuvent signer cette transaction de règlement en sachant qu'il n'y a aucun moyen de tricher et d'obtenir un solde plus favorable. En signant et en transmettant la transaction de règlement en coopération, ils peuvent fermer le canal et rembourser immédiatement leur solde. Dans le pire des cas, l'une des parties peut être mesquine, refuser de coopérer et forcer l'autre partie à conclure unilatéralement la transaction d'engagement la plus récente. Mais s'ils le font, ils doivent également attendre leurs fonds.

Engagements révocables asymétriques

Une meilleure façon de gérer les états d'engagement précédents consiste à les révoquer explicitement. Cependant, cela n'est pas facile à réaliser. Une caractéristique clé du bitcoin est qu'une fois qu'une transaction est valide, elle reste valide et n'expire pas. La seule façon d'annuler une transaction est de doubler ses

entrées avec une autre transaction avant qu'elle ne soit minée. C'est pourquoi nous avons utilisé des délais dans l'exemple de canal de paiement simple ci-dessus pour nous assurer que les engagements plus récents pourraient être dépensés avant que les engagements plus anciens ne soient valides. Cependant, la chronologie des engagements crée un certain nombre de contraintes qui rendent les canaux de paiement difficiles à utiliser.

Même si une transaction ne peut pas être annulée, elle peut être construite de manière à la rendre indésirable à utiliser. La façon dont nous faisons cela est de donner à chaque partie une *clé de révocation* qui peut être utilisée pour punir l'autre partie si elle essaie de tricher. Ce mécanisme de révocation des transactions d'engagement antérieures a d'abord été proposé dans le cadre du Lightning Network.

Pour expliquer les clés de révocation, nous allons construire un canal de paiement plus complexe entre deux échanges gérés par Hitesh et Irene. Hitesh et Irene gèrent des échanges de bitcoins en Inde et aux États-Unis, respectivement. Les clients de la bourse indienne de Hitesh envoient souvent des paiements aux clients de la bourse américaine d'Irene et vice versa. Actuellement, ces transactions se produisent sur la blockchain Bitcoin, mais cela signifie payer des frais et attendre plusieurs blocs pour les confirmations. La mise en place d'un canal de paiement entre les bourses réduira considérablement le coût et accélérera le flux de transaction.

Hitesh et Irene lancent le canal en construisant en collaboration une transaction de financement, chacun finançant le canal avec

5 bitcoins. Le solde initial est de 5 bitcoins pour Hitesh et de 5 bitcoins pour Irene. La transaction de financement verrouille l'état du canal dans un multisig 2 sur 2, tout comme dans l'exemple d'un canal simple.

La transaction de financement peut avoir une ou plusieurs entrées de Hitesh (ajoutant jusqu'à 5 bitcoins ou plus), et une ou plusieurs entrées d'Irene (ajoutant jusqu'à 5 bitcoins ou plus). Les entrées doivent légèrement dépasser la capacité du canal afin de couvrir les frais de transaction. La transaction a une sortie qui verrouille le total de 10 bitcoins à une adresse multisig 2 sur 2 contrôlée à la fois par Hitesh et Irene. La transaction de financement peut également avoir une ou plusieurs sorties renvoyant le changement à Hitesh et Irene si leurs contributions dépassaient leur contribution de canal prévue. Il s'agit d'une transaction unique avec des entrées proposées et signées par deux parties. Il doit être construit en collaboration et signé par chaque partie avant d'être transmis.

Désormais, au lieu de créer une seule transaction d'engagement que les deux parties signent, Hitesh et Irene créent deux transactions d'engagement différentes qui sont *asymétriques* .

Hitesh a une transaction d'engagement avec deux sorties. La première sortie paie à Irene les 5 bitcoins qui lui sont dus *immédiatement* . La deuxième sortie paie à Hitesh les 5 bitcoins qui lui sont dus, mais seulement après un délai de 1000 blocs. Les sorties de transaction ressemblent à ceci :

```
Entrée: sortie de financement 2 sur 2, signée par
Irene
Sortie 0 <5 bitcoin>:
    <Clé publique d'Irene> CHECKSIG
Sortie 1 <5 bitcoin>:
    <1000 blocs>
    VÉRIFIER LA SÉQUENCE VÉRIFIER
    TOMBER
    <Clé publique de Hitesh> CHECKSIG
```

Irene a une transaction d'engagement différente avec deux sorties. La première sortie paie à Hitesh les 5 bitcoins qui lui sont dus immédiatement. La deuxième sortie paie à Irene les 5 bitcoins qui lui sont dus, mais seulement après un délai de 1000 blocs. La transaction d'engagement qu'Irene détient (signée par Hitesh) ressemble à ceci :

```
Entrée: résultat de financement 2 sur 2, signé par
Hitesh
Sortie 0 <5 bitcoin>:
    <Clé publique de Hitesh> CHECKSIG
Sortie 1 <5 bitcoin>:
    <1000 blocs>
    VÉRIFIER LA SÉQUENCE VÉRIFIER
    TOMBER
    <Clé publique d'Irene> CHECKSIG
```

De cette façon, chaque partie a une transaction d'engagement, dépensant la production de financement 2 sur 2. Cette entrée est signée par l'*autre* partie. À tout moment, la partie qui détient

la transaction peut également signer (compléter le 2 sur 2) et diffuser. Cependant, s'ils diffusent la transaction d'engagement, cela paie immédiatement l'autre partie alors qu'ils doivent attendre l'expiration d'un délai. En imposant un délai sur le rachat de l'une des sorties, nous mettons légèrement chaque partie dans une situation désavantageuse lorsqu'elle choisit de diffuser unilatéralement une transaction d'engagement. Mais un délai à lui seul ne suffit pas pour encourager une conduite équitable.

Deux transactions d'engagement asymétriques avec paiement différé pour la partie détenant la transaction montrent deux transactions d'engagement asymétriques, où la sortie payant le détenteur de l'engagement est retardée.

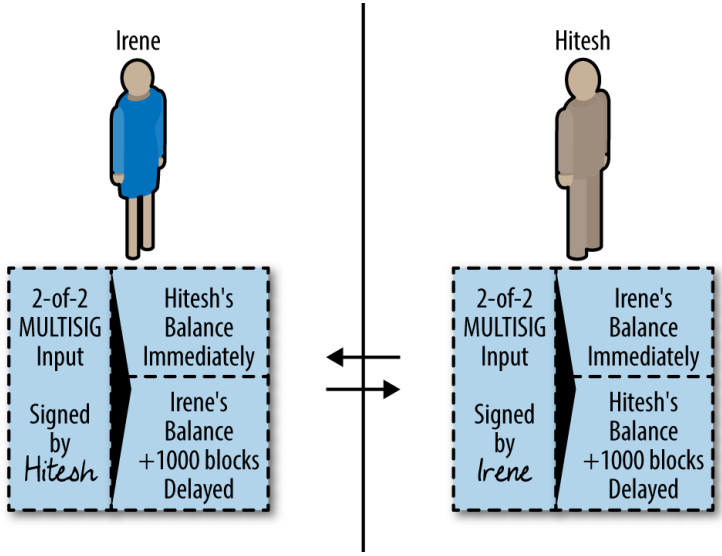


Figure 5. Deux transactions d'engagement asymétriques avec retard de paiement pour la partie détenant la transaction

Maintenant, nous introduisons le dernier élément de ce schéma : une clé de révocation qui empêche un tricheur de diffuser un engagement expiré. La clé de révocation permet à la partie lésée de punir le tricheur en prenant tout le solde de la chaîne.

La clé de révocation est composée de deux secrets, chacun étant généré indépendamment par chaque participant au canal. Il est similaire à un multisig 2 sur 2, mais construit en utilisant l'arithmétique de la courbe elliptique, de sorte que les deux parties connaissent la clé publique de révocation mais que chaque partie ne connaît que la moitié de la clé secrète de révocation.

À chaque tour, les deux parties révèlent leur moitié du secret de révocation à l'autre partie, donnant ainsi à l'autre partie (qui a maintenant les deux moitiés) les moyens de réclamer la sortie de pénalité si cette transaction révoquée est jamais diffusée.

Chacune des transactions d'engagement a une sortie « retardée ». Le script de remboursement de cette sortie permet à une partie de la récupérer après 1000 blocs, *ou à* l'autre de la récupérer si elle dispose d'une clé de révocation, ce qui pénalise la transmission d'un engagement révoqué.

Ainsi, quand Hitesh crée une transaction d'engagement à signer par Irene, il rend la deuxième sortie payable à lui-même après 1000 blocs, ou à la clé publique de révocation (dont il ne connaît que la moitié du secret). Hitesh construit cette transaction. Il ne révélera sa moitié du secret de révocation à Irene que lorsqu'il sera prêt à passer à un nouvel état de chaîne et voudra révoquer cet engagement.

Le script de la deuxième sortie ressemble à ceci :

```
Sortie 0 <5 bitcoin>:
    <Clé publique d'Irene> CHECKSIG
Sortie 1 <5 bitcoin>:
SI
    # Sortie de pénalité de révocation
    <Clé publique de révocation>
AUTRE
    <1000 blocs>
    VÉRIFIER LA SÉQUENCE VÉRIFIER
    TOMBER
    <Clé publique de Hitesh>
```

```
FIN SI
CHECKSIG
```

Irene peut signer cette transaction en toute confiance, car si elle est transmise, elle lui paiera immédiatement ce qui lui est dû. Hitesh détient la transaction, mais sait que s'il la transmet dans un canal unilatéral se fermant, il devra attendre 1000 blocs pour être payé.

Lorsque le canal passe à l'état suivant, Hitesh doit *révoquer* cette transaction d'engagement avant qu'Irene n'accepte de signer la prochaine transaction d'engagement. Pour ce faire, il lui suffit d'envoyer sa moitié de la *clé de révocation* à Irène. Une fois qu'Irène a les deux moitiés de la clé secrète de révocation pour cet engagement, elle peut signer le prochain engagement en toute confiance. Elle sait que si Hitesh essaie de tricher en publiant l'engagement précédent, elle peut utiliser la clé de révocation pour récupérer la sortie retardée de Hitesh. *Si Hitesh triche, Irene obtient les DEUX sorties*. Pendant ce temps, Hitesh n'a que la moitié du secret de révocation pour cette clé publique de révocation et ne peut pas utiliser la sortie avant 1000 blocs. Irene pourra récupérer la sortie et punir Hitesh avant que les 1000 blocs ne se soient écoulés.

Le protocole de révocation est bilatéral, ce qui signifie qu'à chaque tour, à mesure que l'état du canal est avancé, les deux parties échangent de nouveaux engagements, échangent des secrets de révocation pour les engagements précédents et signent les nouvelles transactions d'engagement de l'autre. En

acceptant un nouvel état, ils rendent l'état antérieur impossible à utiliser, en se donnant mutuellement les secrets de révocation nécessaires pour punir toute tricherie.

Regardons un exemple de comment cela fonctionne. L'un des clients d'Irene souhaite envoyer 2 bitcoins à l'un des clients de Hitesh. Pour transmettre 2 bitcoins à travers le canal, Hitesh et Irene doivent faire avancer l'état du canal pour refléter le nouvel équilibre. Ils s'engageront dans un nouvel état (état numéro 2) où les 10 bitcoins de la chaîne sont partagés, 7 bitcoins vers Hitesh et 3 bitcoins vers Irene. Pour faire progresser l'état du canal, ils créeront chacun de nouvelles transactions d'engagement reflétant le nouveau solde du canal.

Comme précédemment, ces transactions d'engagement sont asymétriques de sorte que la transaction d'engagement que chaque partie détient les oblige à attendre si elles la rachètent. Surtout, avant de signer de nouvelles transactions d'engagement, ils doivent d'abord échanger des clés de révocation pour invalider l'engagement précédent. Dans ce cas particulier, les intérêts de Hitesh sont alignés sur l'état réel de la chaîne et il n'a donc aucune raison de diffuser un état antérieur. Cependant, pour Irene, l'état numéro 1 lui laisse un solde plus élevé que l'état 2. Quand Irene donne à Hitesh la clé de révocation pour sa transaction d'engagement antérieure (état numéro 1), elle révoque effectivement sa capacité à tirer profit de la régression du canal à un niveau antérieur. state car avec la clé de révocation, Hitesh peut racheter les deux sorties de la transaction d'engagement précédente sans délai.

Surtout, la révocation ne se produit pas automatiquement. Alors

que Hitesh a la capacité de punir Irene pour avoir triché, il doit surveiller la blockchain avec diligence pour détecter des signes de triche. S'il voit une transaction d'engagement antérieure diffusée, il a 1000 blocs pour agir et utiliser la clé de révocation pour contrecarrer la triche d'Irène et la punir en prenant tout le solde, les 10 bitcoins.

Les engagements révocables asymétriques avec des verrous de temps relatif (CSV) sont un bien meilleur moyen de mettre en œuvre des canaux de paiement et une innovation très significative dans cette technologie. Avec cette construction, le canal peut rester ouvert indéfiniment et peut avoir des milliards de transactions d'engagement intermédiaires. Dans les implémentations prototypes de Lightning Network, l'état d'engagement est identifié par un index de 48 bits, permettant plus de 281 trillions ($2{,}8 \times 10^{14}$) de transitions d'état dans un seul canal !

Contrats de verrouillage du temps de hachage (HTLC)

Les canaux de paiement peuvent être encore étendus avec un type spécial de contrat intelligent qui permet aux participants d'engager des fonds dans un secret remboursable, avec un délai d'expiration. Cette fonctionnalité s'appelle un *contrat Hash Time Lock* , ou *HTLC* , et est utilisée à la fois dans les canaux de paiement bidirectionnels et acheminés.

Expliquons d'abord la partie "hash" du HTLC. Pour créer un HTLC, le destinataire prévu du paiement créera d'abord un secret R. Il calculera ensuite le hachage de ce secret H :

```
H = Hash (R)
```

Cela produit un hachage H qui peut être inclus dans le script de verrouillage d'une sortie. Quiconque connaît le secret peut l'utiliser pour récupérer la sortie. Le secret R est également appelé *pré-image* de la fonction de hachage. La pré-image est juste les données qui sont utilisées comme entrée dans une fonction de hachage.

La deuxième partie d'un HTLC est le composant « time lock ». Si le secret n'est pas révélé, le payeur du HTLC peut obtenir un « remboursement » après un certain temps. Ceci est réalisé avec un verrouillage de temps absolu à l'aide de CHECKLOCKTIMEVERIFY.

Le script implémentant un HTLC peut ressembler à ceci :

```
SI
    # Paiement si vous avez le secret R
    HASH160 <H> EQUALVERIFY
AUTRE
    # Remboursement après expiration du délai.
    <heure de verrouillage> CHECKLOCKTIMEVERIFY DROP
    <Clé publique du payeur> CHECKSIG
FIN SI
```

Quiconque connaît le secret R, qui, lorsqu'il est haché est égal à H, peut récupérer cette sortie en exerçant la première clause du flux IF.

Si le secret n'est pas révélé et le HTLC réclamé, après un certain

nombre de blocs, le payeur peut réclamer un remboursement en utilisant la deuxième clause du flux IF.

Il s'agit d'une implémentation de base d'un HTLC. Ce type de HTLC peut être utilisé par *quiconque* possède le secret R. Un HTLC peut prendre de nombreuses formes différentes avec de légères variations du script. Par exemple, l'ajout d'un opérateur CHECKSIG et d'une clé publique dans la première clause limite le rachat du hachage à un destinataire nommé, qui doit également connaître le secret R.

Canaux de paiement acheminés (Lightning Network)

Le Lightning Network est un réseau routé proposé de canaux de paiement bidirectionnels connectés de bout en bout. Un réseau comme celui-ci peut permettre à n'importe quel participant d'acheminer un paiement d'un canal à l'autre sans faire confiance à aucun des intermédiaires. Le Lightning Network a été décrit pour la première fois par Joseph Poon et Thadeus Dryja en février 2015 , en s'appuyant sur le concept de canaux de paiement tel que proposé et développé par de nombreux autres.

« Lightning Network » fait référence à une conception spécifique d'un réseau de canaux de paiement acheminé, qui a maintenant été implémentée par au moins cinq équipes open source différentes. Les implémentations indépendantes sont coordonnées par un ensemble de normes d'interopérabilité décrites dans l' article *BOLT (Basics of Lightning Technology)* .

Des implémentations prototypes de Lightning Network ont été publiées par plusieurs équipes.

Le Lightning Network est un moyen possible de mettre en œuvre des canaux de paiement acheminés. Il existe plusieurs autres modèles qui visent à atteindre des objectifs similaires, tels que Teechan et Tumblebit.

Exemple de base de Lightning Network

Voyons comment cela fonctionne.

Dans cet exemple, nous avons cinq participants : Alice, Bob, Carol, Diana et Eric. Ces cinq participants ont ouvert des canaux de paiement les uns avec les autres, par paires. Alice a un canal de paiement avec Bob. Bob est connecté à Carol, Carol à Diana et Diana à Eric. Pour simplifier, supposons que chaque canal est financé avec 2 bitcoins par chaque participant, pour une capacité totale de 4 bitcoins dans chaque canal.

Une série de canaux de paiement bidirectionnels liés pour former un Lightning Network qui peut acheminer un paiement d'Alice à Eric montre cinq participants dans un Lightning Network, connectés par des canaux de paiement bidirectionnels qui peuvent être liés pour effectuer un paiement d'Alice à Eric (Canaux de paiement acheminés (Lightning Network)).

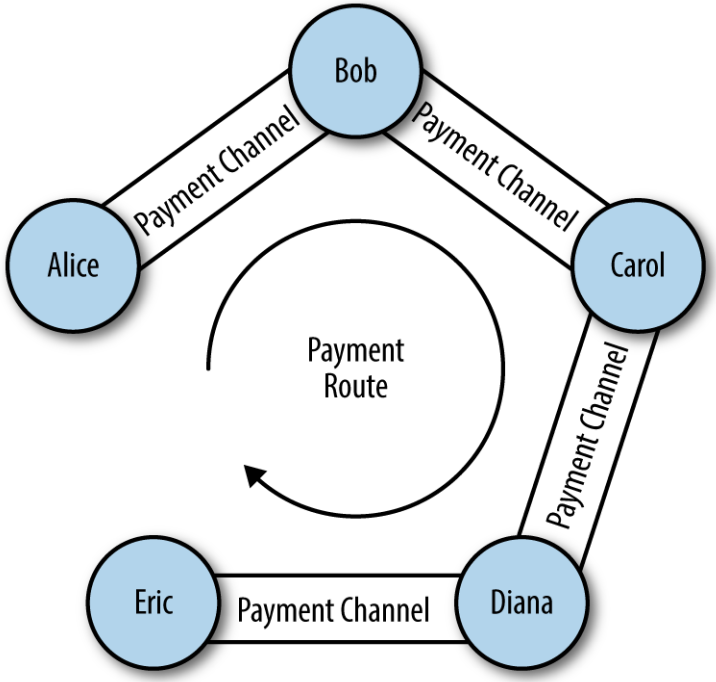


Figure 6. Une série de canaux de paiement bidirectionnels liés pour former un Lightning Network qui peut acheminer un paiement d'Alice vers Eric

Alice veut payer 1 bitcoin à Eric. Cependant, Alice n'est pas connectée à Eric par un canal de paiement. La création d'un canal de paiement nécessite une transaction de financement, qui doit être engagée dans la blockchain Bitcoin. Alice ne veut pas ouvrir un nouveau canal de paiement et engager davantage de ses fonds. Y a-t-il moyen de payer Eric, indirectement ?

L'acheminement des paiements étape par étape via un réseau

Lightning montre le processus étape par étape d'acheminement d'un paiement d'Alice vers Eric, à travers une série d'engagements HTLC sur les canaux de paiement reliant les participants.

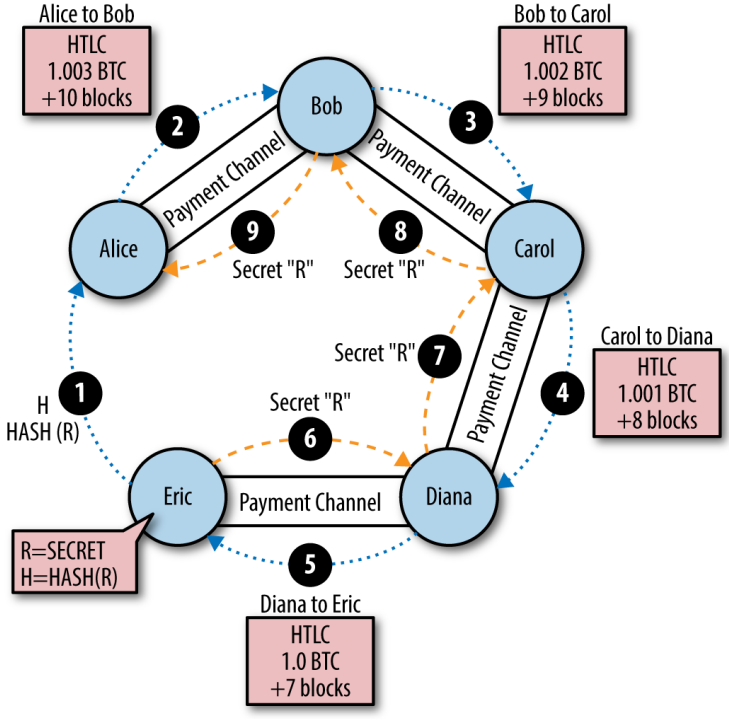


Figure 7. Acheminement des paiements étape par étape via un Lightning Network

Alice exécute un nœud Lightning Network (LN) qui suit son canal de paiement vers Bob et a la capacité de découvrir des itinéraires entre les canaux de paiement. Le nœud LN d'Alice a également la capacité de se connecter via Internet au nœud LN d'Eric. Le nœud LN d'Eric crée un secret R en utilisant un générateur de nombres aléatoires. Le nœud d'Eric ne révèle ce secret à personne. Au lieu de cela, le nœud d'Eric calcule un hachage H du secret R et transmet ce hachage au nœud d'Alice (voir Acheminement des paiements étape par étape via un Lightning Network étape 1).

Maintenant, le nœud LN d'Alice construit une route entre le nœud LN d'Alice et le nœud LN d'Eric. L'algorithme de routage utilisé sera examiné plus en détail plus tard, mais pour l'instant supposons que le nœud d'Alice puisse trouver une route efficace.

Le nœud d'Alice construit ensuite un HTLC, payable au hachage H, avec un délai de remboursement de 10 blocs (bloc actuel + 10), pour un montant de 1,003 bitcoin (voir Acheminement des paiements étape par étape via un Lightning Network étape 2). Le 0,003 supplémentaire sera utilisé pour compenser les nœuds intermédiaires pour leur participation à cette voie de paiement. Alice offre ce HTLC à Bob, en déduisant 1,003 bitcoin de son solde de canal avec Bob et en l'engageant au HTLC. Le HTLC a la signification suivante : *"Alice engage 1,003 du solde de sa chaîne à payer à Bob si Bob connaît le secret, ou remboursée au solde d'Alice si 10 blocs se sont écoulés."* L'équilibre de canal entre Alice et Bob s'exprime désormais par des transactions d'engagement avec trois sorties : solde de 2 bitcoins à Bob, solde de 0,997 bitcoin à Alice, 1,003 bitcoin engagé dans le HTLC d'Alice. Le solde

d'Alice est réduit du montant engagé au HTLC.

Bob a maintenant un engagement que s'il est capable d'obtenir le secret R dans les 10 prochains blocs, il peut réclamer le 1.003 verrouillé par Alice. Avec cet engagement en main, le nœud de Bob construit un HTLC sur son canal de paiement avec Carol. Le HTLC de Bob engage 1,002 bitcoin dans le hachage H pour 9 blocs, que Carol peut échanger si elle a un R secret (voir Routage des paiements étape par étape via un réseau Lightningétape 3). Bob sait que si Carol peut réclamer son HTLC, elle doit produire R. Si Bob a R en neuf blocs, il peut l'utiliser pour lui réclamer le HTLC d'Alice. Il gagne également 0,001 bitcoin pour avoir engagé son solde de canal pour neuf blocs. Si Carol est incapable de réclamer son HTLC et qu'il ne peut pas réclamer le HTLC d'Alice, tout revient aux soldes de canaux précédents et personne n'est perdu. L'équilibre des canaux entre Bob et Carol est maintenant : 2 à Carol, 0,998 à Bob, 1,002 engagé par Bob au HTLC.

Carol s'est maintenant engagée à ce que si elle obtient R dans les neuf blocs suivants, elle puisse réclamer 1,002 bitcoin verrouillé par Bob. Elle peut désormais s'engager sur sa chaîne HTLC avec Diana. Elle commet un HTLC de 1,001 bitcoin au hachage H, pour huit blocs, que Diana peut échanger si elle a un R secret (voir Acheminement des paiements étape par étape via un Lightning Network étape 4). Du point de vue de Carol, si cela fonctionne, elle est mieux lotie à 0,001 bitcoin et si ce n'est pas le cas, elle ne perd rien. Son HTLC à Diana n'est viable que si R est révélé, auquel point elle peut réclamer le HTLC de Bob. L'équilibre des canaux entre Carol et Diana est maintenant : 2 à Diana, 0,999 à Carol, 1,001 engagé par Carol au HTLC.

Enfin, Diana peut proposer un HTLC à Eric, en engageant 1 bitcoin pour sept blocs dans le hachage H (voir Routage de paiement étape par étape via un Lightning Network étape 5). L'équilibre des canaux entre Diana et Eric est maintenant : 2 à Eric, 1 à Diana, 1 engagé par Diana au HTLC.

Cependant, à ce saut de la route, Eric *a le* secret R. Il peut donc revendiquer le HTLC proposé par Diana. Il envoie R à Diana et réclame le 1 bitcoin, en l'ajoutant au solde de son canal (voir Acheminement des paiements étape par étape via un Lightning Network étape 6). La balance des canaux est maintenant : 1 pour Diana, 3 pour Eric.

Maintenant, Diana a le secret R. Par conséquent, elle peut maintenant réclamer le HTLC de Carol. Diana transmet R à Carol et ajoute le bitcoin 1,001 à son solde de canal (voir Acheminement des paiements étape par étape via un Lightning Network étape 7). Maintenant, l'équilibre des canaux entre Carol et Diana est : 0,999 pour Carol, 3,001 pour Diana. Diana a « gagné » 0,001 en participant à cette voie de paiement.

Revenant sur l'itinéraire, le secret R permet à chaque participant de réclamer les HTLC en circulation. Carol réclame 1,002 à Bob, définissant le solde de son canal sur : 0,998 à Bob, 3,002 à Carol (voir Acheminement des paiements étape par étape via un Lightning Network, étape 8). Enfin, Bob réclame le HTLC à Alice (voir Acheminement des paiements étape par étape via un Lightning Network, étape 9). Leur équilibre de canal est mis à jour comme suit : 0,997 à Alice, 3,003 à Bob.

Alice a payé 1 bitcoin à Eric sans ouvrir de chaîne à Eric. Aucune

des parties intermédiaires de la voie de paiement ne devait se faire confiance. Pour l'engagement à court terme de leurs fonds dans le canal, ils peuvent gagner une somme modique, le seul risque étant un petit retard dans le remboursement si le canal a été fermé ou si le paiement acheminé a échoué.

Transport et routage Lightning Network

Toutes les communications entre les nœuds LN sont cryptées point à point. De plus, les nœuds disposent d'une clé publique à long terme qu'ils utilisent comme identifiant et pour s'authentifier mutuellement.

Chaque fois qu'un nœud souhaite envoyer un paiement à un autre nœud, il doit d'abord construire un *chemin* via le réseau en connectant des canaux de paiement avec une capacité suffisante. Les nœuds publient des informations de routage, y compris les canaux qu'ils ont ouverts, la capacité de chaque canal et les frais qu'ils facturent pour acheminer les paiements. Les informations de routage peuvent être partagées de différentes manières et différents protocoles de routage sont susceptibles d'émerger à mesure que la technologie Lightning Network progresse. Certaines implémentations Lightning Network utilisent le protocole IRC comme un mécanisme pratique permettant aux nœuds d'annoncer les informations de routage. Une autre implémentation de la découverte de routes utilise un modèle P2P où les nœuds propagent les annonces de canaux à leurs pairs, dans un modèle « d'inondation », similaire à la façon dont le bitcoin propage les transactions. Les plans futurs incluent une proposition appelée Flare, qui est un modèle de routage hybride avec des « voisinages » de nœuds locaux et des nœuds

de balise à plus longue portée.

Dans notre exemple précédent, le nœud d'Alice utilise l'un de ces mécanismes de découverte d'itinéraire pour trouver un ou plusieurs chemins reliant son nœud au nœud d'Eric. Une fois que le nœud d'Alice a construit un chemin, elle initialisera ce chemin à travers le réseau, en propageant une série d'instructions cryptées et imbriquées pour connecter chacun des canaux de paiement adjacents.

Surtout, ce chemin n'est connu que du nœud d'Alice. Tous les autres participants à l'itinéraire de paiement ne voient que les nœuds adjacents. Du point de vue de Carol, cela ressemble à un paiement de Bob à Diana. Carol ne sait pas que Bob transmet en fait un paiement d'Alice. Elle ne sait pas non plus que Diana transmettra un paiement à Eric.

Il s'agit d'une fonctionnalité essentielle de Lightning Network, car elle garantit la confidentialité des paiements et rend très difficile l'application de la surveillance, de la censure ou des listes noires. Mais comment Alice établit-elle ce chemin de paiement, sans rien révéler aux nœuds intermédiaires ?

Le Lightning Network implémente un protocole routé en oignon basé sur un schéma appelé Sphinx . Ce protocole de routage garantit qu'un expéditeur de paiement peut construire et communiquer un chemin via Lightning Network de telle sorte que :

- Les nœuds intermédiaires peuvent vérifier et déchiffrer leur partie des informations d'itinéraire et trouver le

prochain saut.

- À part les sauts précédent et suivant, ils ne peuvent pas connaître les autres nœuds faisant partie du chemin.

- Ils ne peuvent pas identifier la longueur du chemin de paiement ou leur propre position sur ce chemin.

- Chaque partie du chemin est chiffrée de telle manière qu'un attaquant au niveau du réseau ne peut pas associer les paquets de différentes parties du chemin les uns aux autres.

- Contrairement à Tor (un protocole d'anonymisation routé par l'oignon sur Internet), il n'y a pas de « nœuds de sortie » qui peuvent être placés sous surveillance. Les paiements n'ont pas besoin d'être transmis à la blockchain Bitcoin ; les nœuds mettent simplement à jour les équilibres des canaux.

En utilisant ce protocole routé par l'oignon, Alice enveloppe chaque élément du chemin dans une couche de cryptage, en commençant par la fin et en travaillant en arrière. Elle crypte un message à Eric avec la clé publique d'Eric. Ce message

est enveloppé dans un message chiffré à Diana, identifiant Eric comme le prochain destinataire. Le message à Diana est enveloppé dans un message chiffré sur la clé publique de Carol et identifiant Diana comme le prochain destinataire. Le message à Carol est chiffré sur la clé de Bob. Ainsi, Alice a construit cet "oignon" multicouche crypté de messages. Elle envoie ceci à Bob, qui ne peut déchiffrer et dérouler que la couche externe. À l'intérieur, Bob trouve un message adressé à Carol qu'il peut transmettre à Carol mais ne peut pas se déchiffrer. En suivant le chemin, les messages sont transférés, décryptés, transférés, etc., jusqu'à Eric.

Chaque élément du chemin contient des informations sur le HTLC qui doivent être étendues au prochain saut, le montant qui est envoyé, les frais à inclure et l'expiration du temps de verrouillage CLTV (en blocs) du HTLC. Au fur et à mesure que les informations de route se propagent, les nœuds prennent des engagements HTLC vers le saut suivant.

À ce stade, vous vous demandez peut-être comment il est possible que les nœuds ne connaissent pas la longueur du chemin et leur position dans ce chemin. Après tout, ils reçoivent un message et le transmettent au saut suivant. Cela ne raccourcit-il pas, leur permettant de déduire la taille du chemin et leur position ? Pour éviter cela, le chemin est toujours fixé à 20 sauts et rempli de données aléatoires. Chaque nœud voit le saut suivant et un message chiffré de longueur fixe à transférer. Seul le destinataire final voit qu'il n'y a pas de prochain saut. Pour tout le monde, il semble qu'il y ait toujours 20 sauts de plus à faire.

Avantages de Lightning Network

Un Lightning Network est une technologie de routage de deuxième couche. Il peut être appliqué à n'importe quelle blockchain prenant en charge certaines fonctionnalités de base, telles que les transactions multisignatures, les délais et les contrats intelligents de base.

Si un Lightning Network est superposé au réseau Bitcoin, le réseau Bitcoin peut gagner une augmentation significative de la capacité, de la confidentialité, de la granularité et de la vitesse, sans sacrifier les principes de fonctionnement sans confiance sans intermédiaires :

Intimité

Les paiements Lightning Network sont beaucoup plus privés que les paiements sur la blockchain Bitcoin, car ils ne sont pas publics. Bien que les participants à un itinéraire puissent voir les paiements propagés sur leurs canaux, ils ne connaissent ni l'expéditeur ni le destinataire.

Fongibilité

Un Lightning Network rend beaucoup plus difficile l'application de la surveillance et des listes noires sur Bitcoin, augmentant la fongibilité de la monnaie.

La vitesse

Les transactions Bitcoin utilisant Lightning Network sont réglées en millisecondes, plutôt qu'en minutes, car les HTLC sont effacées sans engager de transactions dans un bloc.

Granularité

Un Lightning Network peut permettre des paiements au moins aussi petits que la limite de "poussière" de bitcoin, peut-être même plus petite. Certaines propositions permettent des incréments de subsatoshi.

Capacité

Un Lightning Network augmente la capacité du système Bitcoin de plusieurs ordres de grandeur. Il n'y a pas de limite supérieure pratique au nombre de paiements par seconde qui peuvent être acheminés sur un Lightning Network, car cela dépend uniquement de la capacité et de la vitesse de chaque nœud.

Fonctionnement sans confiance

Un Lightning Network utilise des transactions Bitcoin entre des nœuds qui fonctionnent comme des pairs sans se faire confiance. Ainsi, un Lightning Network préserve les principes du système Bitcoin, tout en élargissant considérablement ses paramètres de fonctionnement.

Bien entendu, comme mentionné précédemment, le protocole Lightning Network n'est pas le seul moyen de mettre en œuvre des canaux de paiement acheminés. Les autres systèmes proposés incluent Tumblebit et Teechan. Pour le moment, cependant, Lightning Network a déjà été déployé sur testnet. Plusieurs équipes différentes ont développé des implémentations concurrentes de LN et travaillent à l'élaboration d'une norme d'interopérabilité commune (appelée BOLT). Il est probable que Lightning Network sera le premier réseau de canaux de paiement acheminé à être déployé en production.

Conclusion

Nous n'avons examiné que quelques-unes des applications émergentes qui peuvent être créées en utilisant la blockchain Bitcoin comme plate-forme de confiance. Ces applications élargissent la portée du bitcoin au-delà des paiements et au-delà des instruments financiers, pour englober de nombreuses autres applications où la confiance est essentielle. En décentralisant la base de la confiance, la blockchain Bitcoin est une plate-forme qui engendrera de nombreuses applications révolutionnaires dans une grande variété d'industries.