Papier Blanc pour Ethereum

Plateforme de contrats autonomes et d'applications décentralisées de nouvelle génération

Le développement de Bitcoin par Satoshi Nakamoto en 2009 a souvent été salué comme une évolution radicale de la monnaie, premier exemple d'un actif numérique qui n'est adossé à aucun autre actif ni n'a de « valeur intrinsèque », ni émetteur centralisé ou régulateur. Cependant, une autre partie sans doute plus importante de l'expérimentation Bitcoin est la technologie blockchain sous-jacente en tant qu'outil de consensus distribué et l'attention est rapidement en train de se porter sur cet autre aspect de Bitcoin. D'autres applications de la technologie blockchain fréquemment citées comprennent l'utilisation d'actifs numériques sur la blockchain pour représenter des monnaies personnalisées et des produits financiers (« colored coins ») , la propriété d'un bien physique (« smart property »), des actifs non fongibles tels que les noms de domaine (« Namecoin »), de même que des applications plus complexes où des actifs numériques sont directement contrôlés par un bout de code exécutant des règles diverses (« smart contracts »), ou même encore des organisations autonomes décentralisées basées sur la blockchain « decentralized autonomous organizations » ou DAOs. Ce qu'Ethereum entend fournir est une blockchain avec un langage de programmation intégré, Turing-complet, qui peut être utilisé pour créer des « contrats » susceptibles de coder des fonctions de transition d'état quelconques, permettant aux utilisateurs de créer n'importe lequel des systèmes décrits ci-dessus ainsi que beaucoup d'autres que nous n'avons pas encore imaginés, tout ceci en quelques lignes de code.

Table des matières

•Introduction à Bitcoin et aux concepts existants

•Histoire

•Bitcoin en tant que système de transition d'état

•Minage

•Les arbres de Merkle

•Autres applications de la blockchain

•Scripting

•Ethereum

•Les comptes Ethereum

•Messages et transactions

•Fonction de transition d'état Ethereum

•Exécution de code

•Blockchain et minage

•Applications

•Systemes de jetons

•Produits dérivés financiers et monnaies à valeur stable

•Systèmes d'identité et de réputation

•Stockage de fichiers réparti

•Organisations Autonomes Décentralisées

•Autres Applications

•Miscellanées et préoccupations

•L'implémentation modifiée de GHOST

•Les frais

•Calcul et Turing-complétude

•Monnaie et émission

•Concentration du minage

•Passage à l’échelle

•Conclusion

•Références et suggestions de lecture

Introduction à Bitcoin et aux concepts existants

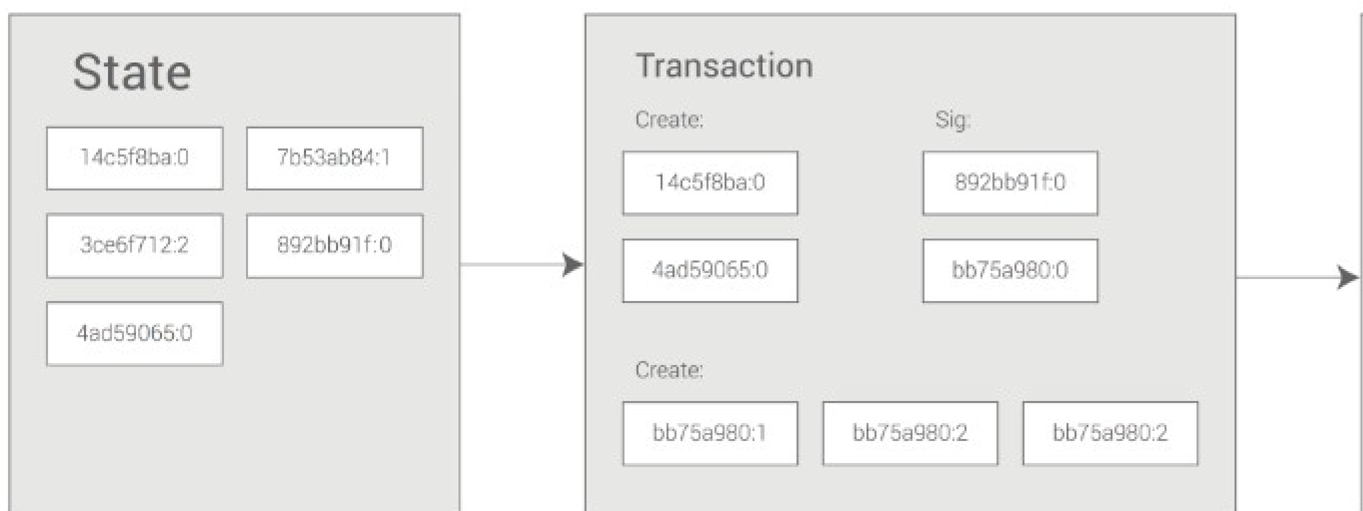
Histoire

Le concept de monnaie numérique décentralisée ainsi que d’autres applications comme les registres de propriété ont été étudiés depuis des décennies. Les protocoles anonymes de monnaie électronique des années 1980 et 1990 étaient principalement fondés sur une primitive cryptographique connue sous le nom Chaumian Blinding. Le Chaumian Blinding garantissait à ces nouvelles monnaies un degré élevé de protection de la vie privée, mais leurs protocoles sous-jacents ont largement échoué à se populariser en raison de leur dépendance à l’égard d’un intermédiaire centralisé. En 1998, [b-money] de Wei Dai (http://www.weidai.com/bmoney.txt) fut la première proposition visant à introduire l’idée de créer l’argent à travers la résolution de puzzles cryptographiques ainsi que de consensus décentralisé, mais la proposition était avare en détails quant à la façon dont le consensus décentralisé pouvait effectivement être mis en œuvre. En 2005, Hal Finney présenta le concept de « preuves de travail réutilisables », un système reprenant les idées de b-money et les puzzles cryptographiques Hashcash d’Adam Back, coûteux en ressources informatiques, pour en arriver au concept d’une crypto-monnaie, mais sans arriver à une solution idéale puisque nécessitant un tiers de confiance comme backend. En 2009, une monnaie décentralisée était pour la première fois mise en œuvre par Satoshi Nakamoto en combinant des primitives reconnues pour la gestion de la propriété grâce à la cryptographie asymétrique avec un algorithme de consensus pour garder la trace de qui possède combien, connu sous le nom de « preuve de travail ».

Le mécanisme de la preuve de travail fut une découverte capitale car il a résolu simultanément deux problèmes. Tout d’abord, il fournit un algorithme de consensus simple et assez efficace, ce qui permet à des nœuds du réseau de se mettre collectivement d’accord sur un ensemble de mises à jour quant à l’état du registre Bitcoin.

Deuxièmement, il fournit un mécanisme permettant facilement de participer au processus de consensus, résolvant le problème politique de décider qui a le droit d'influencer le consensus, tout en empêchant les attaques de type Sybil. Il s'agit de substituer un obstacle formel à la participation, comme l'obligation d'être enregistré en tant qu'entité unique sur une liste donnée, par une barrière économique – le poids d'un seul nœud dans le processus de vote par consensus est directement proportionnel à la puissance de calcul que celui-ci apporte. Depuis lors, une approche alternative a été proposée appelée proof of stake (preuve de possession ou preuve d'enjeu), calculant le poids d'un nœud proportionnellement à ses avoirs en devises et non à ses ressources en calcul. La discussion sur les mérites relatifs des deux approches est au-delà de la portée de ce document mais il convient de noter que les deux approches peuvent être utilisées pour servir de colonne vertébrale à une crypto-monnaie.

Bitcoin en tant que système de transition d'état



D'un point de vue technique, le registre d'une crypto-monnaie telle que Bitcoin peut être considéré comme un système de transition d'état où il y a un « état » consistant en l'état de la propriété de tous les bitcoins existants et une « fonction de transition d'état » qui prend un état et une transaction, et en fait résulter un nouvel état. Dans un système bancaire classique par exemple, l'état est un bilan, une transaction est une demande de déplacer X $ de A à B, et la fonction de transition d'état réduit la valeur du compte A de X $ et augmente la valeur du compte B de X $. Si au départ le compte A a moins de X $, la fonction de transition d'état renvoie une erreur. Par conséquent, on peut formellement définir :

```
APPLY(S,TX) -> S' or ERROR
```

Dans le système bancaire défini ci-dessus:

```
APPLY({ Alice: $50, Bob: $50 },"send $20 from Alice to Bob") = { Alice: $30, Bob: $70 }
```

Mais:

```
APPLY({ Alice: $50, Bob: $50 },"send $70 from Alice to Bob") = ERROR
```

L'« état » dans Bitcoin est l'ensemble de toutes les unités de compte (techniquement, des « unspent transaction outputs» ou UTXO, sorties de transaction non dépensées) qui ont été frappées et non encore dépensées, chaque UTXO ayant une valeur nominale et un propriétaire (défini par une adresse de 20 octets qui est essentiellement une clé cryptographique publique [1]). Une transaction comporte une ou plusieurs entrées, chaque entrée contenant une référence à un UTXO existant et une signature cryptographique produite par la clé privée associée à l'adresse du propriétaire, et une ou plusieurs sorties, chaque sortie contenant un nouvel UTXO qui sera ajouté à l'état.

La fonction de transition d'état APPLY(S,TX) -> S' peut grossièrement être définie comme suit:

1.Pour chaque entrée dans TX:

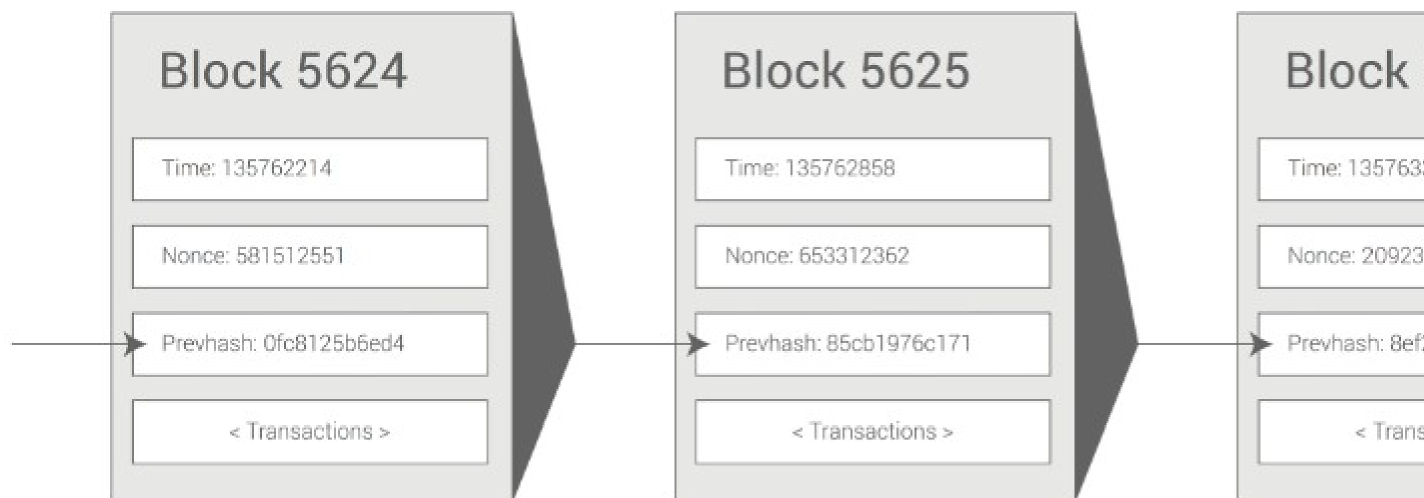
•Si l'UTXO référencé n'est pas S, on renvoie une erreur.
•Si la signature fournie ne correspond pas à celle du propriétaire de l'UTXO, on renvoie une erreur.

2.Si la somme de tous les UTXO d'entrée est inférieure à la somme de tous les UTXO de sortie, on renvoie une erreur.

3.On renvoie S' avec tous les UTXO d'entrée supprimés et tous les UTXO de sortie ajoutés.

La première moitié de la première étape empêche les expéditeurs de transaction de dépenser des unités de compte qui n'existent pas, la seconde moitié de la première étape empêche les expéditeurs de transaction de dépenser les unités de compte d'autres personnes, et la deuxième étape impose la conservation de la valeur. Afin d'utiliser cela comme moyen de paiement, le protocole est le suivant. Supposons qu'Alice veuille envoyer 11,7 BTC à Bob. Tout d'abord, Alice va chercher un ensemble disponible d'UTXO qu'elle possède qui s'élève au moins à 11,7 BTC. En réalité, Alice ne sera pas en mesure d'obtenir exactement 11,7 BTC ; disons que la plus petite somme qu'elle peut obtenir est 6+4+2=12. Elle crée alors une transaction avec ces trois entrées et deux sorties. La première sortie sera 11,7 BTC avec l'adresse de Bob comme propriétaire, et la seconde sortie sera les 0,3 BTC de monnaie restante. Si Alice n'essaye pas de récupérer la monnaie en l'envoyant à une adresse qu'elle possède, le mineur sera en mesure de la réclamer.

Minage



Si nous avions accès à un service centralisé digne de confiance, ce système serait trivial à mettre en œuvre ; il pourrait être codé exactement comme il a été décrit plus haut, en utilisant le disque dur d'un serveur centralisé pour garder une trace de l'état. Cependant, avec Bitcoin, nous essayons de construire un système monétaire décentralisé et nous avons donc besoin de combiner le système de transition d'état avec un système de consensus afin d'assurer que tout le monde est d'accord sur l'ordre des transactions. Le processus de consensus décentralisé de Bitcoin nécessite que les nœuds du réseau tentent en continu de produire des paquets de transactions appelés « blocs ». Le réseau est destiné à créer un bloc environ toutes les dix minutes, chaque bloc contenant un horodatage, un nonce, une référence au bloc précédent (c.à.d. son empreinte) et une liste de toutes les transactions qui ont eu lieu depuis le bloc précédent. Au fil du temps, cela crée une « blockchain » persistante, sans cesse croissante, qui se met continuellement à jour pour représenter le dernier état du registre Bitcoin.

L'algorithme utilisé pour vérifier si un bloc est valide, exprimé dans ce paradigme, est le suivant:

1.On vérifie si le précédent bloc référencé par le bloc existe et est valide.

2.On vérifie que l'horodatage du bloc est supérieur à celui du bloc précédent[2] et inférieur à 2 heures dans l'avenir.

3.On vérifie que la preuve de travail du bloc est valide.

4.Soit S[0] l’état à la fin du bloc précédent.

5.On suppose que TX est la liste des transactions du bloc avec n transactions. Pour tout i dans 0 ... n-1, alors S [i+1] = APPLY(S[i],TX [i]). Si l’application renvoie une erreur, on sort et on renvoie faux.

6.Retour vrai, et enregistre S[n] comme état à la fin de ce bloc.

Fondamentalement, chaque transaction dans le bloc doit fournir une transition d’état valide vers un nouvel état à partir de ce qui était l’état canonique avant que la transaction n’ait été exécutée. On note que l’état n’est pas encodé dans le bloc de quelque façon que ce soit ; ce n’est qu’une abstraction dont le nœud qui valide doit se souvenir et il ne peut être calculé (en toute sécurité) pour tout bloc qu’en partant de l’état d’origine et en y appliquant séquentiellement chaque transaction dans chaque bloc. En outre, on note que l’ordre dans lequel le mineur inclut les transactions dans le bloc a de l’importance ; s’il y a deux transactions A et B dans un bloc tel que B dépense un UTXO créé par A, alors le bloc sera valable si A précède B, mais pas d’une autre manière.

La seule condition de validité présente dans la liste ci-dessus qui ne figure pas dans d’autres systèmes est le besoin de « preuve de travail ». La condition précise est que la double empreinte SHA256 de chaque bloc, traitée comme un nombre de 256 bits, doit être inférieure à une cible ajustée dynamiquement qui, au moment de la rédaction de ce document, est d’environ 2187. Son but est de rendre la création de bloc difficile en terme de calculs, ce qui empêche les attaquants exerçant une attaque de type Sybil de recréer toute la blockchain en leur faveur. Étant donné que SHA256 est conçu comme une fonction pseudo-aléatoire complètement imprévisible, la seule façon de créer un bloc valide consiste en essais et en erreurs, en incrémentant le nonce de façon répétée pour voir si chaque nouvelle empreinte correspond.

À l’objectif actuel de ~2187, le réseau doit faire une moyenne de ~269 essais avant de trouver un bloc valide ; en général, la cible est recalibrée par le réseau tous les 2016 blocs de sorte que, en moyenne, un nouveau bloc est produit par un noeud du réseau toutes les dix minutes. Afin de compenser le travail de calcul des mineurs, celui qui trouve un bloc est en droit d’inclure une transaction qui lui donne 25 BTC venant de nulle part. En outre, si une transaction a une valeur totale supérieure dans ses entrées et dans ses sorties, la différence va aussi au mineur en tant que « transaction fee » (frais de transaction). Soit dit en passant, il s’agit également du seul mécanisme par lequel les BTC sont émis ; l’état de genèse ne contenait pas du tout d’unités de compte.

Afin de mieux comprendre le but du minage, nous allons examiner ce qui se passe dans le cas d’un attaquant malveillant. Puisque la cryptographie sous-jacente à Bitcoin est connue pour être sécurisée, l’attaquant va cibler la partie du système Bitcoin qui n’est pas directement protégée par la cryptographie : l’ordre des transactions. La stratégie de l’attaquant est simple :

1.Envoyer 100 BTC à un marchand en échange d'un produit (de préférence un bien numérique en livraison rapide).

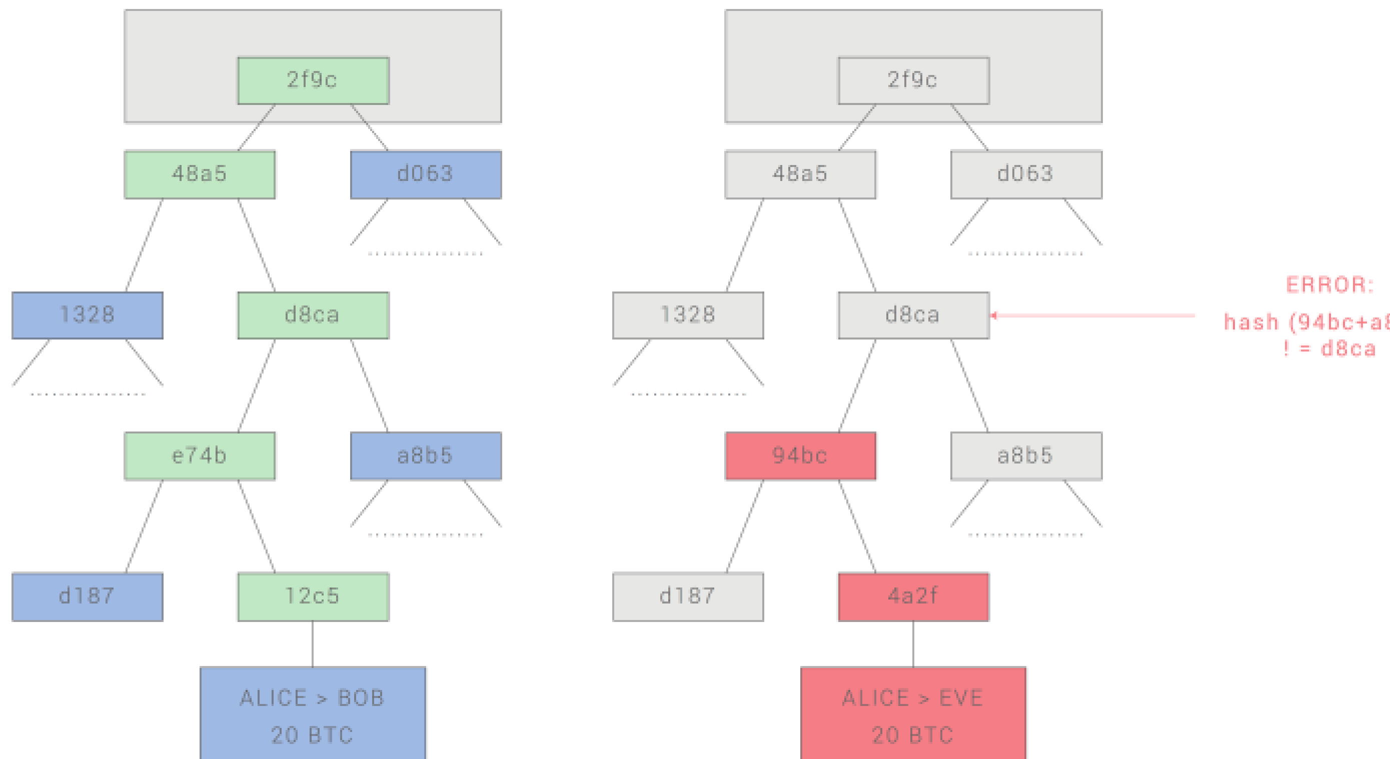
2.Attendre la livraison du produit.

3.Produire une autre transaction envoyant 100 BTC à lui-même.

4.Essayer de convaincre le réseau que sa transaction à lui-même était celle qui est venue en premier.

Une fois que l'étape (1) a eu lieu, après quelques minutes un mineur inclura la transaction dans un bloc, par exemple le bloc numéro 270000. Après environ une heure, cinq autres blocs auront été ajoutés à la chaîne après ce bloc, avec chacun de ces blocs pointant indirectement à la transaction et donc la « confirmant ». À ce stade, le marchand acceptera le paiement comme finalisé et livrera le produit ; puisque nous supposons que cela est un bien numérique, la livraison est instantanée. Maintenant, l'attaquant crée une autre transaction envoyant 100 BTC à lui-même. Si l'attaquant l'envoie simplement dans la nature, la transaction ne sera pas traitée ; les mineurs vont tenter d'exécuter APPLY(S, TX) et verront que TX consomme un UTXO qui n'est plus dans l'état. Ainsi, au lieu de cela, l'attaquant crée un « fork » de la blockchain en commençant par miner une autre version du bloc 270000 pointant vers le même bloc 269999 parent, mais avec la nouvelle transaction à la place de l'ancienne. Comme les données du bloc sont différentes, cela nécessite de refaire la preuve de travail. En outre, la nouvelle version du bloc 270000 de l'attaquant a une empreinte différente, donc les blocs d'origine de 270001 à 270005 ne « pointent » pas vers lui ; ainsi, la chaîne d'origine et la nouvelle chaîne de l'attaquant sont complètement séparées. La règle est que, lors d'une scission, la plus longue blockchain est considérée comme la vraie, et ainsi les mineurs légitimes travaillent sur la chaîne 270005 tandis que l'attaquant travaille seul sur la chaîne 270000. Pour que la blockchain de l'attaquant soit la plus longue, il aurait besoin d'avoir plus de puissance de calcul que le reste du réseau combiné afin de rattraper son retard (d'où l'attaque dite 51%).

Les arbres de Merkle



À gauche : il suffit de ne présenter qu’un petit nombre de nœuds d’un arbre de Merkle pour donner la preuve de la validité d’une branche

À droite : toute tentative de modifier quelque partie que ce soit d’un arbre de Merkle mènera fatalement à une incohérence dans la chaîne

Une fonctionnalité importante de Bitcoin, en ce qui concerne le passage à l’échelle, est que le bloc est stocké dans une structure de données multi-niveaux. Le hash (l’empreinte) d’un bloc n’est en fait que l’empreinte de l’en-tête du bloc, environ 200 octets de données contenant l’horodatage, le nonce, l’empreinte du bloc précédent et l’empreinte d’une structure de données appelée arbre de Merkle qui stocke toutes les transactions dan bloc. Un arbre de Merkle est un type d’arbre binaire composé d’un ensemble de nœuds avec un grand nombre de nœuds terminaux au bas de l’arbre, contenant les données sous-jacentes, un ensemble de nœuds intermédiaires où chacun est l’empreinte de ses deux fils et, finalement, un nœud racine unique, également formé à partir de l’empreinte de ses deux fils, qui représente le « haut » de l’arbre. Le but de l’arbre de Merkle est de permettre aux données d’un bloc d’être fournies au coup par coup : un nœud peut ne télécharger que l’en-tête d’un bloc depuis une source et la petite partie pertinente d’un arbre depuis une autre, tout en restant assuré de l’intégrité des données. Tout ceci fonctionne parce que les empreintes se propagent vers le haut : si un utilisateur malveillant tente d’altérer une transaction au bas de l’arbre de Merkle, cette modification en provoque une autre au

nœud du dessus, puis une autre au nœud encore au-dessus, pour finir par modifier la racine de l'arbre et donc l'empreinte du bloc, ce qui fait que le protocole le prend en compte comme un bloc complètement différent (presque certainement avec une preuve de travail invalide).

Le protocole de l'arbre de Merkle est indiscutablement essentiel pour une viabilité à long terme. Un full node (nœud complet) du réseau Bitcoin, qui stocke et traite l'intégralité de chaque bloc, prenait environ 15 Go d'espace disque en avril 2014 et croît d'environ un gigaoctet par mois. La situation actuelle est viable pour certains ordinateurs de bureau, pas pour les téléphones, et, plus tard, seuls les entreprises et les passionnés seront en mesure de participer. Un protocole connu sous le nom de simplified payment verification (SPV) ou vérification simplifiée de paiement permet l'existence d'une autre catégorie de nœuds que l'on appelle light nodes (nœuds légers ou clients légers). Ils téléchargent les en-têtes de blocs, vérifient la preuve de travail de ces en-têtes, puis ne téléchargent que les « branches » associées aux transactions qui les concernent. Cela permet aux clients légers de déterminer avec un vraie garantie de sécurité quel est le statut d'une transaction Bitcoin quelconque, ainsi que son solde actuel, en ne téléchargeant qu'une très petite partie de la blockchain.

Autres applications de la blockchain

L'application de l'idée de la blockchain à d'autres concepts a aussi une longue histoire. En 2005, Nick Szabo esquissa le concept de « secure property titles with owner authority » (titres de propriété sécurisés avec preuve de possession), un document décrivant de « nouvelles avancées dans la technologie des bases de données répliquées » permettant à un système fondé sur la blockchain de stocker une sorte de cadastre et créant un framework élaboré qui comprenait des concepts comme l'établissement sur un terrain, la prescription acquisitive et la taxe foncière de Géorgie. Il n'existait malheureusement pas de système de base de données répliquées à ce moment et le protocole n'a donc jamais été implémenté dans la pratique. Cependant, après 2009, une fois développé le consensus décentralisé de Bitcoin, un certain nombre d'autres applications commencèrent rapidement à émerger.

•**Namecoin** – créé en 2010, Namecoin peut être décrit comme une base de données décentralisée d'enregistrement de noms. Dans les protocoles décentralisés comme Tor, Bitcoin et BitMessage, il doit y avoir un moyen d'identifier les comptes afin que d'autres personnes puissent interagir avec eux, mais dans toutes les solutions existantes, le seul type d'identificateur disponible est une empreinte pseudo-aléatoire comme par exemple 1LW79wp5ZBqaHW1jL5TCiBCrhQYtHagUWy. Idéalement, on souhaiterait être en mesure d'avoir un compte avec un nom comme « george ». Il y a cependant un problème si une personne peut créer un compte nommé « george », puis que quelqu'un d'autre utilise le même processus pour enregistrer à nouveau « george » afin d'usurper l'identité du premier. La seule solution est d'appliquer le principe du premier arrivé, premier servi, où le premier enregistrement réussit tandis que le second échoue – un problème parfaitement adapté au protocole de consensus Bitcoin. Namecoin est la mise en œuvre la plus ancienne et la plus réussie d'un système d'enregistrement de nom basée sur une telle idée.

•**Colored coins** – Le but des colored coins est de servir de protocole permettant de créer sa propre monnaie numérique ; ou, dans le cas basique mais important d'une monnaie avec une seule unité, des tokens (jetons) numériques, sur la blockchain de Bitcoin. Dans le protocole des colored coins, on « émet » une nouvelle monnaie en assignant publiquement une couleur à un UTXO de Bitcoin, et ce protocole donne récursivement cette couleur aux autres UTXO créés par les transactions qui en dépensent (certaines règles spécifiques sont appliquées dans le cas des entrées de couleurs différentes). Cela permet aux utilisateurs de conserver des portefeuilles ne contenant que des UTXO d'une couleur donnée et de les envoyer comme des bitcoins normaux, en remontant l'historique de la blockchain pour déterminer la couleur d'un UTXO reçu.

•**Metacoins** – L'idée de base du metacoin consiste à superposer un protocole à Bitcoin, en utilisant les transactions de Bitcoin pour stocker les transactions metacoin, mais avec une fonction de transition d'état différente, APPLY'. Comme le protocole metacoin ne peut pas empêcher des transactions invalides d'apparaître dans la blockchain de Bitcoin, une règle a été ajoutée selon laquelle si APPLY'(S,TX) renvoie une erreur, le protocole donne par défaut APPLY'(S,TX) = S. Cela facilite le mécanisme de création d'un protocole de crypto-monnaie comprenant potentiellement des fonctionnalités avancées qui ne peuvent être implémentées à l'intérieur de Bitcoin lui-même, mais avec un coût de développement très bas puisque les complexités du minage et du réseau sont déjà gérées par le protocole de Bitcoin. Les Metacoins ont été utilisés pour implémenter certains types de contrats financiers, d'enregistrements de nom et de bourses d'échange décentralisées.

On connaît donc en général deux approches pour bâtir un protocole de consensus : soit un réseau indépendant, soit un protocole au-dessus de Bitcoin. La première approche, bien qu'elle soit raisonnablement efficace dans le cas d'applications comme Namecoin, est difficile à implémenter ; chaque implémentation doit recommencer à zéro une chaîne indépendante, et nécessite l'écriture et les tests de tout le code de transition d'état et de réseau. De plus, nous prédisons que l'ensemble des applications de la technologie de consensus décentralisé suivra une distribution en loi de puissance où la très grande majorité des applications seront trop peu importantes pour justifier leur propre blockchain. Remarquons enfin qu'il existe des types importants d'applications décentralisées, notamment des organisations autonomes décentralisées, qui devront interagir les unes avec les autres.

L'approche fondée sur Bitcoin, en revanche, a l'inconvénient de ne pas hériter des fonctionnalités de vérification simplifiée de paiement (SPV) de Bitcoin. Le SPV fonctionne pour Bitcoin en utilisant la profondeur de la blockchain pour transmettre la validité ; à un certain point, une fois que l'on est remonté assez loin dans les ancêtres d'une transaction, on peut légitimement affirmer qu'ils font partie de l'état. D'un autre côté, les méta-protocoles fondés sur la blockchain ne peuvent pas forcer celle-ci à ne pas inclure de transactions qui ne sont pas valides dans le contexte de leur propre protocole. Donc, une implémentation totalement sûre du méta-protocole de SPV devrait examiner l'intégralité de la blockchain de Bitcoin jusqu'au début pour déterminer si certaines transactions sont ou non valides. Actuellement, toutes les implémentations « légères » des méta-protocoles fondés sur Bitcoin se fient à un serveur de confiance pour fournir les données, un résultat à l'évidence insatisfaisant surtout quand le but premier d'une crypto-monnaie est d'éliminer le besoin de confiance.

Scripting

Même sans extensions, le protocole Bitcoin offre une version allégée du concept de « smart contracts » (contrats autonomes). Les UTXO dans Bitcoin peuvent non seulement appartenir à une clé publique, mais aussi à un script plus complexe exprimé dans un langage simple de programmation à pile. Dans ce paradigme, une transaction dépensant cet UTXO doit fournir les données qui conviennent au script. En effet, même le simple mécanisme d'appartenance d'une clé publique est implémenté via un un script : le script prend une signature de courbe elliptique en entrée, vérifie sa concordance avec la transaction et l'adresse propriétaire de l'UTXO, et retourne 1 si la vérification a réussi ou 0 dans le cas contraire. D'autres scripts plus compliqués existent pour d'autres cas d'usage. Par exemple, on peut concevoir un script qui requiert les signatures de deux clés privées sur trois pour être validé (« multisig »), un mécanisme utile pour des comptes d'entreprise, des comptes d'épargne sécurisés ou certains cas de dépôts commerciaux sous séquestre. Les scripts peuvent aussi être utilisés pour offrir des récompenses à la résolution de problèmes mathématiques, et on peut même concevoir un script qui déclarerait « cet UTXO Bitcoin est le vôtre si vous pouvez fournir une preuve SPV que vous m'avez envoyé une transaction Dogecoin de telle valeur », permettant ainsi la création d'une bourse d'échange entre différentes crypto-monnaies.

Le langage de script implémenté dans Bitcoin a cependant d'importantes limites :

•**Langage non Turing-complet** – Bien qu'il existe un vaste sous-ensemble de calculs permis par le langage de script de Bitcoin, il est très loin d'en permettre la totalité. La principale catégorie manquante est celle des boucles. Cela empêche les boucles infinies pendant la vérification d'une transaction ; théoriquement, c'est un obstacle surmontable pour les programmeurs de scripts car toute boucle peut être simulée en répétant simplement le code à plusieurs reprises avec une instruction if, mais elle conduit à des scripts qui sont très volumineux. Par exemple, la mise en œuvre d'un algorithme de signature de courbe elliptique nécessiterait probablement 256 itérations de multiplications déclarées à chaque fois dans le code.

•**Ignorance de la valeur** – Un script UTXO n'a aucun moyen de contrôler finement le montant à retirer. Par exemple, un cas d'utilisation intéressant pour un contrat d'oracle serait un contrat de couverture du risque où A et B investiraient pour 1000 $ de BTC et, après 30 jours, le script enverrait 1000 $ de BTC à A et le reste à B. La détermination de la valeur du BTC en USD demanderait un oracle mais il s'agirait tout de même d'une énorme amélioration en termes de confiance et d'infrastructure par rapport aux solutions totalement centralisées disponibles aujourd'hui. Cependant, comme les UTXO sont en tout-ou-rien, la seule manière d'y arriver passe par un bricolage peu efficient où l'on dispose de nombreux UTXO de diverses dénominations (c'est-à-dire un UTXO de 2k pour k allant de 1 à 30) et où l'on choisit quel UTXO envoyer à A et lequel envoyer à B.

•**Manque d'état** – Les UTXO peuvent être dépensés ou non ; il n'y a pas de possibilité de contrats en plusieurs étapes ou de scripts qui conservent un état interne plus complexe. Cela complique les contrats d'options en plusieurs étapes, les bourses d'échange

décentralisées ou les protocoles d’engagement cryptographiques en deux étapes (nécessaires pour les primes calculées sécurisées). Cela signifie également que les UTXO ne peuvent être utilisés que pour bâtir des contrats simples, à usage unique, et non des contrats « stateful » plus complexes comme les organisations décentralisées, et cela rend l’implémentation des méta-protocoles difficile. L’état binaire combiné à l’ignorance de la valeur rend impossible une autre application importante, les limites de retrait.

•**Ignorance de la blockchain** – Les UTXO ignorent certaines données de la blockchain comme le nonce et l’empreinte du bloc précédent. Cela limite sévèrement les applications de jeux d’argent ainsi que d’autres catégories en privant le langage de script d’une précieuse source de hasard.

Nous discernons ainsi trois approches pour construire des applications évoluées sur la base d’une crypto-monnaie : la construction d’une nouvelle blockchain, l’utilisation de scripts reposant sur Bitcoin, et la construction d’un méta-protocole reposant sur Bitcoin. La création d’une nouvelle blockchain offre une liberté illimitée pour la construction d’un ensemble de fonctionnalités, moyennant un important coût en temps de développement, en effort de mise en service et en sécurisation. La solution des scripts est simple à mettre en œuvre et à normaliser, mais très limitée en terme de fonctionnalités, et les méta-protocoles, bien que simples à implémenter, souffrent de défauts de passage à l’échelle. Avec Ethereum, nous avons l’intention de construire un framework alternatif qui offre des perspectives bien plus importantes tant au niveau de la simplicité de développement que du potentiel des clients légers, tout en permettant aux applications de profiter d’un environnement économique et de la sécurité d’une blockchain.

Ethereum

Le but d’Ethereum est de créer un protocole alternatif pour construire des applications décentralisées, fournissant un ensemble différent de compromis que nous pensons être très utile pour une vaste classe d’applications décentralisées, avec un accent particulier sur les situations où le développement rapide, la sécurité des petites applications rarement utilisées et la possibilité pour les différentes applications d’interagir ensemble de manière très efficace sont importants. Ethereum fait cela en construisant ce qui est essentiellement la couche fondamentale abstraite ultime : une blockchain intégrant un langage de programmation Turing-complet, permettant à quiconque de rédiger des smart contracts(contrats autonomes) et des applications décentralisées où l’on peut créer ses propres règles concernant la propriété, les formats de transaction et les fonctions de transition d’état. Une version dépouillée de Namecoin peut être écrite en deux lignes de code et d’autres protocoles comme les devises et les systèmes de réputation peuvent être développés en moins de vingt lignes. Des contrats autonomes, des « boîtes » cryptographiques qui contiennent une valeur et ne se déverrouillent que si certaines conditions sont remplies, peuvent également être bâtis au-dessus de la plate-forme avec beaucoup plus de puissance que celle offerte par le langage de script de Bitcoin en raison des capacités supplémentaires qu’offre la Turing-complétude, la connaissance de la valeur, la connaissance de la blockchain et l’état.

Les comptes Ethereum

Dans Ethereum, l’état est composé d’objets appelés « comptes », chaque compte ayant une adresse sur 20 octets et les transitions d’état étant des transferts directs de valeur et d’information entre les comptes. Un compte Ethereum contient quatre champs :

•Le **nonce**, un compteur utilisé pour s’assurer que chaque transaction ne peut être traitée qu’une seule fois ;

•Le **solde en ether** actuell du compte ;

•Le **code du contrat** du compte, s’il est présent ;

•Le **storage** ou mémoire de stockage du compte (vide par défaut).

L’« Ether » est le principal crypto-carburant interne d’Ethereum et est utilisé pour payer les frais de transaction. D’une manière générale, il existe deux types de comptes: les **comptes de détenteur externe**, contrôlés par des clés privées, et les **comptes de contrat**, commandés par le code du contrat. Un compte de détenteur externe n’a pas de code et on peut envoyer des messages à partir de celui-ci en créant et en signant une transaction ; dans un compte de contrat, chaque fois que le compte de contrat reçoit un message, son code s’active, ce qui lui permet de lire et écrire dans la mémoire de stockage interne et d’envoyer d’autres messages ou de créer des contrats à son tour.

Notons que les « contrats » dans Ethereum ne doivent pas être considérés comme devant être « accomplis » ou « respectés » ; ils ressemblent davantage à des « agents autonomes » qui vivent à l’intérieur de l’environnement d’exécution Ethereum, en exécutant toujours un bout de code spécifique lorsqu’ils sont appelés par un message ou une transaction, et en conservant le contrôle direct de leur propre solde d’éther et de leur propre collection de clés/valeurs pour garder une trace des variables persistantes.

Messages et transactions

Le terme « transaction » est utilisé dans Ethereum pour se référer à un paquet de données signé qui stocke un message à envoyer à partir d’un compte de détenteur externe. Les transactions contiennent :

•Le destinataire du message ;

•Une signature identifiant l’expéditeur ;

•La quantité d’ether à transférer de l’expéditeur au destinataire ;

•Un champ optionnel de données ;

•Une valeur STARTGAS représentant le nombre maximum d’étapes de calcul autorisé pour l’exécution de la transaction ;

•Une valeur GASPRICE représentant les frais que l’expéditeur paie par étape de calcul.

Les trois premiers sont des champs standards attendus dans toutes les crypto-monnaies. Le champ de données n’a pas de fonction par défaut, mais la machine virtuelle a un opcode avec lequel un contrat peut accéder aux données; un exemple de cas d’utilisation serait un contrat qui fonctionne comme un service d’enregistrement de domaine sur la blockchain, il voudra peut-être interpréter les données passées comme contenant deux « champs », le premier champ étant un domaine à enregistrer et le second champ étant l’adresse IP. Le contrat lira ces valeurs à partir des données du message et les stockera de manière appropriée.

Les champs STARTGAS et GASPRICE sont cruciaux pour le modèle anti-déni de service d’Ethereum. Afin d’éviter les boucles infinies accidentelles ou hostiles, ou encore d’autres gaspillages de calcul dans le code, il est nécessaire pour chaque transaction de limiter le nombre d’étapes de calcul dans l’exécution du code. L’unité fondamentale de calcul est le « gas» (gaz) ; généralement, une étape de calcul coûte 1 gaz, mais certaines opérations coûtent davantage car elles sont plus coûteuses en calcul ou elles augmentent la quantité de données devant être stockées dans l’état. Il y a aussi une taxe de 5 gaz pour chaque octet de données de transaction. Le but de ce système de frais est d’exiger d’un attaquant qu’il paie proportionnellement chaque ressource qu’il consomme, ceci comprenant le calcul, la bande passante et le stockage ; par conséquent, toute transaction qui conduit le réseau à consommer une plus grande quantité de l’une de ces ressources doit payer des frais en gaz à peu près proportionnels à cette augmentation.

Messages

Les contrats ont la capacité d’envoyer des « messages » à d’autres contrats. Les messages sont des objets virtuels qui ne sont jamais sérialisés et n’ont d’existence qu’au sein de l’environnement d’exécution Ethereum. Un message contient :

•L’expéditeur du message (implicite) ;

•Le destinataire du message ;

•La quantité d’ether à transférer avec le message ;

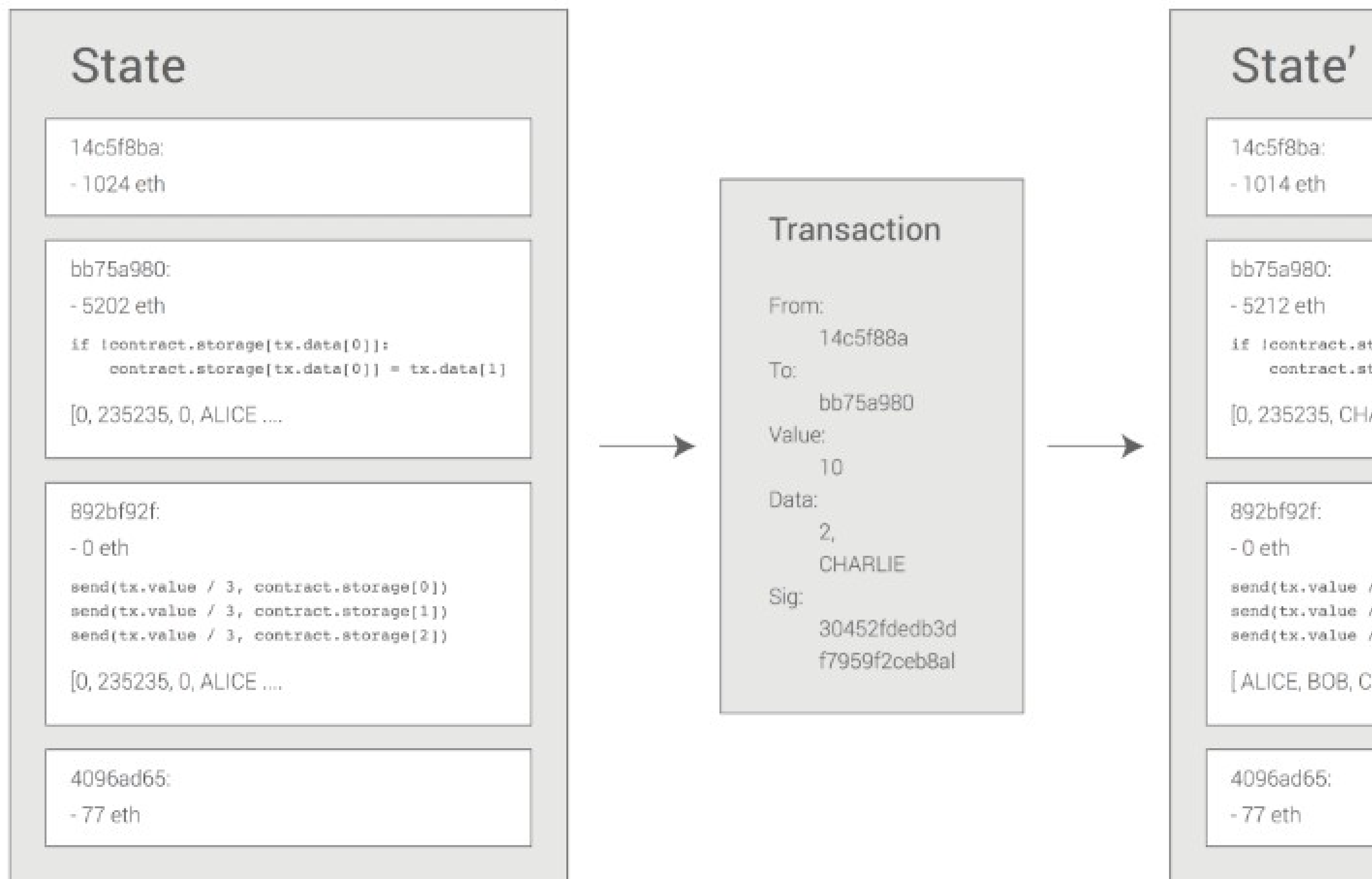
•Un champ optionnel de données ;

•Une valeur STARTGAS.

Fondamentalement, un message est similaire à une transaction, à la différence qu’il est émis par un contrat et non par un acteur externe. Un message est créé lorsqu’un contrat en cours d’exécution fait appel à l’opcode CALL, qui produit et exécute un message. De même qu’une transaction, un message provoque l’exécution du code du compte destinataire. Ainsi, les contrats peuvent interagir avec d’autres contrats de la même manière que le peuvent des acteur externes.

Il faut noter que l’allocation en gaz assignée par une transaction ou un contrat concerne la totalité du gaz consommé par cette transaction et toutes les sous-exécutions. Par exemple, si un acteur externe A envoie une transaction vers B accompagnée de 1000 gaz, que B consomme 600 gaz avant d’envoyer un message à C, et que l’exécution de C consomme 300 gaz, B pourra encore consommer 100 gaz avant de se retrouver en panne sèche.

Fonction de transition d'état Ethereum



La fonction de transition d'état Ethereum, APPLY(S,TX) -> S' peut être définie de la manière suivante :

1.On vérifie que la transaction est bien formée (c.à.d. qu'elle a le bon nombre de valeurs), que la signature est valide et que le nonce correspond au nonce du compte de l'expéditeur. Sinon, on renvoie une erreur.

2.On calcule les frais de transaction correspondant à STARTGAS * GASPRICE et on détermine l'adresse d'envoi en fonction de la signature. On déduit les frais du solde du compte de l'expéditeur et on incrémente le nonce de l'expéditeur. Si le solde est insuffisant, on renvoie une erreur.

3.On initialise GAS = STARTGAS et on enlève une certaine quantité de gaz par octet correspondant aux frais à payer pour les octets constituant la transaction.

4.On transfère la valeur de la transaction du compte de l’expéditeur vers celui du destinataire. Si le compte destinataire n’existe pas encore, on le crée. Si le compte de destination est un contrat, on exécute le code du contrat, soit jusqu’à son terme, soit jusqu’à ce que l’exécution ait épuisé le gaz.

5.Si le transfert de valeur échoue parce que l’expéditeur n’a pas assez d’argent ou que l’exécution du code est à court de gaz, on annule tous les changements d’état à l’exception du paiement des frais et on ajoute les frais sur le compte du mineur.

6.Dans le cas contraire, on rembourse les frais correspondant au gaz restant à l’expéditeur et on envoie au mineur les frais correspondant au gaz consommé.

Par exemple, supposons que le code du contrat soit :

```
if !self.storage[calldataload(0)]:


   self.storage[calldataload(0)] = calldataload(32)
```

Notons qu’en réalité, le code du contrat est écrit en code EVM bas niveau ; cet exemple est écrit en Serpent, un de nos langages de haut niveau, pour plus de clarté, qui peut être compilé en code EVM. Supposons que l’espace de stockage du contrat soit initialement vide et qu’une transaction soit émise avec un valeur de 10 ether, 2000 gaz, un coût unitaire en gaz de 0,001 ether et 64 octets de données, avec les octets 0 à 31 représentant le nombre 2 et les octets 32 à 63 représentant la chaîne CHARLIE. Le déroulement de la fonction de transition d’état dans ce cas est la suivante :

1.On vérifie que la transaction est valide et bien formée.

2.On vérifie que l’expéditeur de la transaction dispose d’au moins 2000 * 0,001 = 2 ether. Si c’est le cas, on déduit 2 ether du compte de l’expéditeur.

3.On initialise gaz = 2000 ; en supposant que la transaction est de 170 octets et que le coût par octet est de 5, on soustrait 850 de sorte qu’il reste 1150 gaz.

4.On déduit 10 ether supplémentaires du compte de l’expéditeur pour les ajouter au compte du contrat.

5.On exécute le code. Dans ce cas précis, c’est simple : il vérifie si l’index 2 de l’espace du stockage du contrat est utilisé, remarque qu’il ne l’est pas, et il insère donc la

valeur CHARLIE à l’index 2 de l’espace de stockage. Supposons que cela consomme 187 gaz, la quantité de gaz restante est alors de 1150-187 = 963.

6.On rend 963 * 0.001 = 0.963 ether au compte de l’expéditeur, et on renvoie l’état résultant.

S’il n’y avait pas de contrat à l’autre bout de la transaction, le coût total de la transaction serait égal au GASPRICE fourni multiplié par la longueur de la transaction en octets, et les données associées à la transaction seraient ignorées.

On note que les messages fonctionnent de la même manière que les transactions en terme d’invalidation : si l’exécution d’un message est à court de gaz, alors l’exécution de ce message ainsi que toutes les autres exécutions déclenchées par celle-ci reviennent à leur état initial, mais les exécutions mères n’ont pas à être invalidées. Cela signifie qu’il est « sans danger » pour un contrat d’appeler un autre contrat, car si A appelle B avec G gaz, alors l’exécution de A ne consommera jamais plus de G gaz. Enfin, notez qu’il existe un opcode, CREATE, qui crée un contrat ; son principe de fonctionnement est globalement similaire à CALL mais la valeur de retour de l’exécution détermine le code d’un nouveau contrat.

Exécution de code

Le code dans les contrats Ethereum est écrit dans un langage bas niveau à bytecode orienté pile, appelé « code Ethereum Virtual Machine » ou « code EVM ». Le code se compose d’une série d’octets où chaque octet représente une opération. En général, l’exécution de code est une boucle infinie qui consiste à effectuer l’opération présente au compteur de programme actuel (qui commence à zéro), puis à incrémenter le compteur de programme jusqu’à la fin du code, une erreur ou la détection d’une instruction STOP ou RETURN. Les opérations ont accès à trois types d’espace pour stocker des données :

•La **stack** (pile), un conteneur premier-entré-premier-sorti auquel on peut ajouter et retirer des valeurs ;

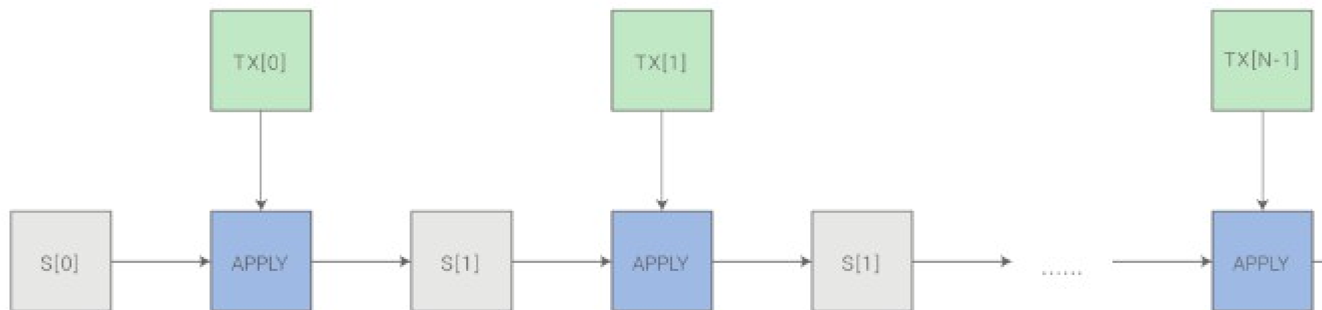
•La **memory** (mémoire), un tableau d’octets extensible à l’infini ;

•Le **storage** (stockage) à long terme du contrat, un tableau de clés/valeurs. Contrairement à la pile et à la mémoire, qui sont réinitialisées après exécution, le stockage est conservé dans le temps.

Le code peut aussi accéder à la valeur, à l’expéditeur et aux données du message reçu, ainsi qu’aux données des en-têtes de bloc. Le code peut aussi renvoyer des données sous forme d’un tableau d’octets en sortie.

Le modèle d’exécution formelle du code EVM est étonnamment simple. Alors que la machine virtuelle Ethereum est en cours d’exécution, son état interne complet peut être défini par le tuple (block_state, transaction, message, code, memory, stack, pc, gas), où block_state est l’état global contenant tous les comptes et comprenant les soldes et le stockage. Au début de chaque étape d’exécution, l’instruction en cours est trouvée en lisant l’octet à la position pc du code (ou 0 si pc >= len(code)), et chaque instruction a sa propre définition quant à la manière d’affecter le tuple. Par exemple, ADD extrait deux éléments de la pile et insère leur somme, réduit gasde 1 et incrémente pc de 1, tandis que SSTORE extrait les deux éléments supérieurs de la pile et insère le deuxième élément dans l’espace de stockage du contrat à l’index spécifié par le premier élément. Bien qu’il existe de nombreuses façons d’optimiser l’exécution de la machine virtuelle Ethereum via la compilation à la volée, une implémentation de base d’Ethereum peut être réalisée en quelques centaines de lignes de code.

Blockchain et minage



La blockchain Ethereum est en de nombreux points similaire à celle de Bitcoin, bien qu’elle présente certaines différences. La principale différence entre Ethereum et Bitcoin, en ce qui concerne l’architecture de la blockchain est que, contrairement à Bitcoin, les blocs Ethereum contiennent à la fois une copie de la liste des transactions et de l’état le plus récent. Hormis cela, deux autres valeurs, le numéro de bloc et la difficulté, sont aussi stockés dans le bloc. L’algorithme essentiel de validation de bloc Ethereum est le suivant :

1.On vérifie que le précédent bloc référencé existe et qu’il est valide.

2.On vérifie que l'horodatage du bloc est supérieur à celui du précédent bloc référencé et qu'il n'excède pas les 15 minutes dans l'avenir.

3.On vérifie que le numéro de bloc, la difficulté, la racine de transaction, la racine oncle et la limite de gaz (plusieurs concepts de bas niveau spécifiques à Ethereum) sont valides.

4.On vérifie que la preuve de travail sur le bloc est valide.

5.Soit S[0] l'état final du bloc précédent.

6.Soit TX la liste des n transactions du bloc. Pour tout i parmi 0...n-1, on définit S[i+1] = APPLY(S[i],TX[i]). Si une application renvoie une erreur, ou si la totalité du gaz consommé jusqu'à ce point du bloc dépasse le GASLIMIT, on renvoie une erreur.

7.Soit S_FINAL égal à S[n], mais en ajoutant la récompense de bloc versée au mineur.

8.On vérifie si la racine de l'arbre de Merkle de l'état S_FINAL est égale à la racine de l'état final fournie dans l'en-tête de bloc. Si c'est le cas, le bloc est valide ; sinon, il est invalide.

L'approche peut sembler très inefficace à première vue car elle nécessite de stocker l'état complet à chaque bloc mais, en réalité, l'efficacité devrait être comparable à celle de Bitcoin. Cela vient du fait que l'état est stocké dans la structure en arbre, et qu'après chaque bloc, seule une petite partie de l'arbre a besoin d'être modifiée. Ainsi, en général, la grande majorité de l'arbre devrait être identique entre deux blocs adjacents. Les données peuvent par conséquent être stockées une fois et référencés deux fois en utilisant des pointeurs (c.à.d. des empreintes de sous-arbres). Un type particulier d'arbre connu sous le nom d'« arbre Patricia » est utilisé pour ce faire. C'est un dérivé du concept d'arbre de Merkle qui permet d'insérer et de supprimer des nœuds, et pas seulement de les modifier, de manière efficace. En outre, puisque toutes les informations d'état font partie du dernier bloc, il n'y a pas besoin de stocker tout l'historique de la blockchain – une stratégie qui, si elle pouvait être appliquée à Bitcoin, réduirait les besoins de stockage d'un facteur 5 à 20.

Une question fréquemment posée est de savoir « où » le code d'un contrat est exécuté, en termes de matériel physique. La réponse est simple : le processus d'exécution du code d'un contrat fait partie de la définition de la fonction de transition d'état, qui fait partie de l'algorithme de validation de bloc. Donc si une transaction est ajoutée dans le bloc B, l'exécution de code engendrée par cette opération est exécutée par tous les nœuds, actuels et futurs, qui téléchargent et valident le bloc B.

Applications

En général, il existe trois types d’applications sur Ethereum. La première catégorie est celle des applications financières, fournissant aux utilisateurs des moyens plus puissants de gérer et conclure des contrats en utilisant leur argent. On y inclut les sous-monnaies, les instruments financiers dérivés, les contrats de couverture, les portefeuilles d’épargne, les testaments, et même certaines catégories de contrats de travail. La deuxième catégorie est celle des applications semi-financières, où de l’argent est impliqué, mais où le traitement comporte aussi un important aspect non monétaire ; un exemple parfait est celui des primes automatiquement accordées pour des solutions à des problèmes de calcul. Enfin, il existe des applications telles que le vote en ligne et la gouvernance décentralisée qui n’ont aucun aspect financier.

Systemes de jetons

Les systèmes de tokens (jetons) sur une blockchain ont de nombreuses applications allant des sous-monnaies représentant des actifs tels que des dollars US ou de l’or jusqu’au actions d’entreprise, en passant par des jetons individuels représentant des titres de propriété, des coupons sécurisés infalsifiables, et même des systèmes de jetons sans lien avec la moindre valeur conventionnelle, utilisés comme mesure d’incitation sous forme de points. Les systèmes de jetons sont étonnamment faciles à mettre en œuvre sur Ethereum. Le point essentiel est que tout monnaie, ou système de jetons, est essentiellement une base de données proposant une seule opération : prendre X unités de A et donner ces X unités à B, à condition que (1) A dispose d’au moins X unités avant la transaction et (2) que la transaction soit approuvée par A. Il suffit d’implémenter cette logique dans un contrat pour pouvoir mettre en œuvre un système de jetons.

Le code de base d’une telle implémentation en Serpent ressemble au suivant :

```
def send(to, value):

    if self.storage[msg.sender] >= value:

        self.storage[msg.sender] = self.storage[msg.sender] - value

        self.storage[to] = self.storage[to] + value
```

Il s’agit fondamentalement d’une implémentation littérale de la fonction de transition d’état de « système bancaire » décrite plus haut dans ce document. Quelques lignes de code supplémentaires sont nécessaires pour fournir l’étape initiale pré-requise de distribution des unités monétaires et quelques autres cas particuliers, et idéalement une fonction

serait ajoutée pour permettre à d'autres contrats d'obtenir le solde d'une adresse. Mais rien de plus. En théorie, les systèmes de jetons basés sur Ethereum fonctionnant comme des sous-monnaies peuvent potentiellement présenter une autre caractéristique importante qui manque aux méta-monnaies implémentées sur la blockchain Bitcoin : la capacité de payer les frais de transaction directement dans cette monnaie. Pour mettre cela en œuvre, le contrat gère un solde en ether avec lequel il rembourse l'expéditeur du montant en ether utilisé pour payer les frais, et rééquilibre son solde en collectant des unités monétaires internes pour les revendre dans une vente aux enchères permanente. Les utilisateurs ont donc besoin d'« activer » leurs comptes avec de l'ether, mais peuvent par la suite réutiliser cette somme puisqu'elle est à chaque fois remboursée par le contrat.

Produits dérivés financiers et monnaies à valeur stable

Les produits dérivés sont l'application la plus commune d'un contrat autonome et l'une des plus simples à implémenter en programmation. La principale difficulté dans l'implémentation de contrats financiers est qu'ils nécessitent pour la plupart une référence à une cotation externe ; une application très intéressante serait par exemple un contrat qui lutterait contre la volatilité de l'ether (ou de toute autre crypto-monnaie) par rapport au dollar US, mais cela impliquerait que le contrat connaisse le taux de change ETH/USD. La méthode la plus simple pour ce faire serait d'utiliser un contrat de « flux de données » géré par une entité définie (par exemple NASDAQ) conçu pour que cette entité puisse le mettre à jour quand cela est nécessaire, et fournissant une interface permettant à d'autres contrats de lui envoyer un message afin d'obtenir une réponse leur fournissant le taux de change.

Si cet ingrédient essentiel existait, le contrat serait le suivant :

1.On attend que le tiers A fournisse 1000 ether.

2.On attend que le tiers B fournisse 1000 ether.

3.On enregistre dans l'espace de stockage du contrat la valeur en dollars US de 1000 ether, calculée en interrogeant de contrat de flux de données. Notons cette valeur $x.

4.Après 30 jours, on autorise A ou B à « réactiver » le contrat afin d'envoyer l'équivalent de $x en ether (calculé en interrogeant à nouveau le contrat de flux de données afin d'obtenir le nouveau taux) à A et le reste à B.

Ce type de contrat aurait un potentiel significatif dans le crypto-commerce. L'un des principaux problèmes évoqués concernant les crypto-monnaies est leur volatilité ; bien que de nombreux utilisateurs et marchands puissent être intéressés par la fiabilité et la simplicité d'usage des actifs cryptographiques, il ne souhaitent sans doute pas risquer de

perdre 23% de leurs fonds en une seule journée. Jusqu'à présent, la solution la plus souvent proposée consistait en des actifs soutenus par l'émetteur. L'idée est qu'un émetteur crée une sous-monnaie dont il aurait les droits d'émission et de révocation, et fournisse une unité de monnaie à toute personne lui fournissant (hors-ligne) une unité d'un actif sous-jacent donné (par exemple : or, dollar US). L'émetteur promet ensuite de fournir une unité de l'actif sous-jacent à toute personne qui lui envoie une unité de l'actif cryptographique. Ce mécanisme permet à n'importe quel actif non cryptographique d'être « transformé » en actif cryptographique, si l'on peut faire confiance à l'émetteur.

En pratique, cependant, les émetteurs ne sont pas toujours fiables, et dans certains cas l'infrastructure bancaire est trop fragile, ou trop hostile, pour permettre l'existence de tels services. Les produits dérivés fournissent une alternative. Ici, au lieu d'avoir un unique émetteur fournissant les fonds pour soutenir un actif, un marché de spéculateurs décentralisé, pariant sur la hausse du prix d'un actif cryptographique de référence (par exemple ETH) remplit ce rôle. Contrairement aux émetteurs, les spéculateurs n'ont pas la possibilité de faire défaut car le contrat spéculatif conserve leurs fonds sous séquestre. Il faut noter que cette approche n'est pas totalement décentralisée car une source de confiance est toujours nécessaire pour fournir le taux de change, bien que cela représente tout de même une importante amélioration en terme de réduction de besoins d'infrastructure (au contraire d'un émetteur, la publication d'un taux de change ne nécessite aucun permis et peut très bien être assimilée à de la liberté d'expression) et de réduction de vecteurs de fraude.

Systèmes d'identité et de réputation

La première crypto-monnaie alternative, Namecoin, a tenté d'utiliser une blockchain similaire à celle de Bitcoin pour fournir un système d'enregistrement de noms, où les utilisateurs peuvent enregistrer leurs noms dans une base de données publique, associés à d'autres données. Le principal cas d'usage évoqué est celui d'un DNS, qui associe des noms de domaine comme « bitcoin.org » (ou, dans le cas de Namecoin, « bitcoin.bit ») à une adresse IP. D'autres cas d'usage incluent l'authentification email et potentiellement des systèmes de réputation plus perfectionnés. Voici un contrat élémentaire permettant de fournir un système d'enregistrement de nom similaire à celui de Namecoin sur Ethereum :

```
def register(name, value):

    if !self.storage[name]:

        self.storage[name] = value
```

Le contrat est très simple ; ce n'est qu'une base données au sein du réseau Ethereum, à laquelle on peut ajouter des données, mais sans possibilité de modification ni suppression. N'importe qui peut enregistrer un nom et une valeur et cet enregistrement demeure à

jamais. Un contrat d'enregistrement de nom plus sophistiqué aura une « clause fonction » permettant à d'autres contrats de l'interroger, ainsi qu'un mécanisme permettant au « propriétaire » (c.a.d. le premier déposant) d'un nom d'en changer les données ou d'en transférer la propriété. On peut même rajouter à cela une fonctionnalité de réputation et de réseau de confiance.

Stockage de fichier réparti

Au cours des dernières années, plusieurs startups dans le secteur du stockage de fichier en ligne ont vu le jour, la plus importante étant Dropbox. Leur objectif est de permettre aux utilisateurs d'uploader une sauvegarde de leur disque dur, d'en assurer le stockage et de permettre aux utilisateurs d'y accéder, moyennant un abonnement mensuel. Cependant, le marché du stockage de fichiers en ligne est à ce jour relativement inadapté ; Un rapide survol des solutions existantes met en évidence, particulièrement dans la « vallée dérangeante » allant de 20 à 200 Go où il n'existe ni offre gratuite ni offre professionnelle abordable, des coûts mensuels supérieurs à celui d'un disque dur de taille équivalente. Les contrats Ethereum peuvent permettre le développement d'un écosystème de stockage de fichiers décentralisé où chaque utilisateur pourrait être rémunéré en louant l'espace inutilisé de ses disques durs, réduisant davantage le coût du stockage de fichiers.

La clé de voûte d'un tel système serait ce que nous avons appelé le « contrat Dropbox réparti ». Ce contrat fonctionne de la manière suivante. Premièrement, les données concernées sont divisées en blocs, chaque bloc étant chiffré pour en assurer la confidentialité, et sont utilisées pour la construction d'un arbre de Merkle. Un contrat est ensuite créé avec une règle stipulant que, à chaque N blocs, le contrat choisira un index aléatoire de l'arbre de Merkle (en utilisant l'empreinte du bloc précédent, accessible depuis le code du contrat, comme source aléatoire) et donnera X ether à la première entité qui fournira une transaction avec une preuve de possession du bloc situé à cet l'index dans l'arbre de Merkle. Lorsqu'un utilisateur souhaite re-télécharger un des ses fichiers, il peut utiliser un protocole de canal de micropaiement (par exemple payer 1 zsabo par 32 kilo-octets) pour récupérer le fichier ; l'approche la plus économique au niveau des frais consiste pour le payeur à ne pas publier la transaction avant la fin, en remplaçant plutôt la transaction par une autre plus lucrative avec un nonce identique après chaque 32 kilo-octets.

Une fonctionnalité importante du protocole est que, bien qu'il semble qu'on fasse confiance à plusieurs nœuds aléatoires pour qu'ils ne perdent pas le fichier, on peut quasiment supprimer ce risque en divisant le fichier en de nombreux morceaux par partage secret et en observant les contrats pour vérifier que chaque morceau est toujours hébergé par un ou plusieurs nœuds. Le fait qu'un contrat continue à effectuer un paiement est la preuve cryptographique que quelqu'un héberge toujours le fichier.

Organisations Autonomes Décentralisées

Le concept général d'une « organisation autonome décentralisée » (decentralized autonomous organization, DAO) est celui d'une entité virtuelle constituée d'un certain nombre de membres ou actionnaires qui, par exemple avec une majorité de 67%, ont le droit de dépenser les fonds de l'entité et de modifier son code. Les membres peuvent collectivement décider de la façon dont l'organisation doit allouer ses fonds. Les méthodes d'attribution des fonds d'une DAO peuvent aller des primes et des salaires à des mécanismes plus exotiques, comme une monnaie interne pour récompenser un travail effectué. Ceci réplique essentiellement les attributs juridiques d'une entreprise traditionnelle ou à but non lucratif en s'appuyant uniquement sur la technologie cryptographique de la blockchain. Jusqu'à présent, les conversations autour des DAO se sont focalisées sur le modèle « capitaliste » d'une « société autonome décentralisée » avec des actionnaires percevant des dividendes et des actions échangeables ; une alternative pouvant être décrite comme une « communauté autonome décentralisée » confèrerait à tous ses membres une part égale dans la prise de décision et exigerait que 67% des membres donnent leur accord pour inviter ou exclure un membre. La règle selon laquelle une personne ne pourrait avoir qu'une adhésion devrait alors être assurée collectivement par le groupe.

Un aperçu général de la façon de coder une DAO est la suivante. Le modèle le plus simple est tout simplement un bout de code auto-modifiable qui évolue si deux tiers des membres approuvent un changement. Bien que le code soit théoriquement immuable, on peut facilement contourner ce problème et obtenir une mutabilité de facto en ayant des morceaux du code dans des contrats distincts, et en enregistrant les adresses des contrats à appeler dans un espace de stockage modifiable. Dans une mise en œuvre simple d'un tel contrat de DAO, il y aurait trois types de transactions distinguées par les données fournies à la transaction :

•[0,i,K,V] pour enregistrer à l'index i une proposition de modifications de l'adresse à stockée à l'index K à la valeur V ;

•[0,i] pour enregistrer un vote en faveur de la proposition i ;

•[2,i] pour finaliser la proposition i si suffisamment de votes ont été enregistrés ;

Le contrat aurait alors des clauses pour chacune d'elles. Il tiendrait un registre de tous les changements potentiels de l'espace de stockage, ainsi qu'une liste des personnes y ayant apporté un vote favorable. Il disposerait également d'une liste de tous les membres. Dès lors qu'une proposition d'altération de l'espace de stockage recueillerait deux tiers de votes favorables, une opération de finalisation pourrait appliquer le changement. Une architecture plus sophistiquée intégrerait également un système de vote pour des opérations telles que l'envoi d'une transaction, l'ajout de membres et la révocation des membres, et pourrait même prévoir une délégation de vote dans l'esprit de la Démocratie Liquide(chacun pourrait ainsi désigner quelqu'un pour voter en son nom, permettant, si A

donne procuration à B et que B donne procuration à C, à C de déterminer le vote de A, la procuration étant transitive). Cette conception permettrait à la DAO de croître organiquement comme une communauté décentralisée, permettant finalement aux membres de déléguer la tâche de filtrage des adhésions à des spécialistes, bien que contrairement au « système actuel », des spécialistes pourraient facilement être désignés et révoqués au fil du temps au gré des changements d'opinion des membres individuels de la communauté.

Un modèle alternatif est celui d'une société décentralisée, où tout compte pourrait détenir zéro actions ou plus, les deux tiers des actions étant nécessaires pour prendre une décision. Un prototype complet nécessiterait une fonctionnalité de gestion d'actifs, la capacité de proposer une offre d'achat ou de vente de titres, et la capacité d'accepter des offres (de préférence avec un mécanisme d'appariement d'ordres au sein du contrat). Une possibilité de délégation dans l'esprit de la Démocratie Liquide serait aussi disponible, implémentant le concept d'un « conseil d'administration ».

Autres Applications

1. Compte d'épargne. Supposons qu'Alice souhaite garder ses fonds en sécurité, mais craigne de perdre ou de se faire pirater sa clé privée. Elle place des ether dans un contrat avec Bob, une banque, de la manière suivante :

•Alice seule ne peut retirer au maximum que 1% des fonds chaque jour.

•Bob seul ne peut retirer au maximum que 1% des fonds chaque jour, mais Alice a la possibilité d'effectuer une transaction avec sa clé pour bloquer cette possibilité.

•Alice et Bob peuvent conjointement retirer n'importe quelle somme.

En principe, 1% par jour est suffisant pour Alice et, si Alice souhaite retirer davantage elle peut demander l'aide de Bob. Si la clé d'Alice est piratée, elle prévient Bob pour qu'il déplace les fonds vers un nouveau contrat. Si elle perd sa clé, Bob a la possibilité de sortir les fonds du contrat. Si Bob s'avère malveillant, Alice peut décider de révoquer son droit de retrait.

2. Assurance récolte. On peut facilement faire un contrat de produits dérivés financiers, mais en utilisant un flux de données météo au lieu d'un indice de prix. Si un agriculteur dans l'Iowa achète un dérivé dont les paiements sont inversement proportionnels aux précipitations dans l'Iowa et qu'une sécheresse se produit, l'agriculteur recevra automatiquement de l'argent. S'il y a suffisamment de pluie, l'agriculteur sera satisfait car ses cultures se porteront bien. Cela peut être étendu aux assurances contre les catastrophes naturelles en général.

3. Un flux de données décentralisé. A l’inverse, pour les contrats financiers, il est effectivement possible de décentraliser le flux de données via un protocole appelé « SchellingCoin ». SchellingCoin fonctionne essentiellement de la manière suivante : N parties insèrent dans le système la valeur d’une donnée spécifique (par exemple le taux ETH/USD), les valeurs sont triées, et les sources de toutes les données comprises entre le 25ème et le 75ème centile sont récompensées par un jeton. Chacun a intérêt à fournir la réponse que tout le monde fournira, et la seule valeur sur laquelle un grand nombre d’acteurs peuvent raisonnablement se mettre d’accord est la valeur par défaut évidente : la vérité. Cela crée un protocole décentralisé qui peut théoriquement fournir un certain nombre de valeurs, y compris le taux ETH/USD, la température à Berlin ou même le résultat d’un calcul complexe spécifique.

4. Séquestre autonome à signatures multiples. Bitcoin autorise des contrats de transaction à signatures multiples où, par exemple, trois clés sur cinq sont nécessaires pour dépenser les fonds. Ethereum permet une granularité plus fine ; par exemple, quatre clés sur cinq donnent accès à la totalité des fonds, trois sur cinq peuvent dépenser jusqu’à 10% par jour, et deux sur cinq peuvent dépenser jusqu’à 0,5% par jour. En outre, la signature multiple sur Ethereum est asynchrone – deux parties peuvent enregistrer leurs signatures sur la blockchain à différents moments et la dernière signature émettra automatiquement la transaction.

5. Calcul réparti. La technologie EVM peut également être utilisée pour créer un environnement de calcul vérifiable, permettant à des utilisateurs de demander à d’autres de réaliser des calculs et de réclamer éventuellement à certains points de contrôle choisis au hasard des preuves que les calculs ont été effectués correctement. Cela permet la création d’un marché de calcul réparti où tout utilisateur peut participer avec son ordinateur de bureau, ordinateur portable ou serveur spécialisé, et au sein duquel des vérifications ponctuelles associées à des dépôts de garantie sont mises en place pour assurer que le système est digne de confiance (les nœuds ne pourraient pas tricher avec profit). Bien qu’un tel système puisse ne pas convenir à toutes les tâches – les tâches qui nécessitent un haut niveau de communication inter-processus, par exemple, ne peuvent pas être facilement réalisées entre de nombreux nœuds – d’autres sont beaucoup plus faciles à paralléliser ; des projets comme SETI@home, Folding@home et des algorithmes génétiques peuvent aisément être mis en œuvre sur une telle plate-forme.

6. Jeux d’argent en pair-à-pair. Un nombre infini de protocoles de jeux d’argent en pair-à-pair, tels que Cyberdice par Frank Stajano et Richard Clayton, peut être mis en œuvre sur la blockchain Ethereum. Le protocole de jeu plus simple est tout simplement un contrat s’appuyant sur la différence de l’empreinte de bloc suivante, des protocoles plus complexes pouvant être élaborés à partir de là, permettant la création de services de jeux d’argent avec des frais proches de zéro, sans possibilité de triche.

7. Marchés de prédiction. Grâce à un oracle ou à SchellingCoin, les marchés de prédiction sont également faciles à mettre en œuvre. Les marchés de prédiction ainsi que SchellingCoin peuvent se révéler être la première application grand public d’une futarchie comme protocole de gouvernance pour des organisations décentralisées.

8. Places de marché décentralisées, s’appuyant sur un système d’identité et de réputation.

Miscellanées et préoccupations

L’implémentation modifiée de GHOST

Le protocole « Greedy Heaviest Observed Subtree » (GHOST) est une innovation introduite par Yonatan Sompolinsky et Aviv Zohar en Décembre 2013. La motivation derrière GHOST est que les blockchains avec des temps de confirmation rapides souffrent actuellement d’une faible sécurité en raison d’un taux élevé de blocs dépréciés – comme les blocs prennent un certain temps pour se propager à travers le réseau, si un mineur A mine un bloc et si un mineur B arrive à miner un autre bloc avant que le bloc du mineur A ne se propage à B, le bloc du mineur B sera perdu et ne contribuera pas à la sécurité du réseau. En outre, il y a un problème de centralisation : si le mineur A est une coopérative de minage avec 30% du hashpower (de la puissance de calcul du réseau) et que B n’en a que 10%, A aura un risque de produire un bloc déprécié 70% du temps (puisque l’autre 30% du temps A produit le dernier bloc et obtient donc les données de minage immédiatement) alors que B aura un risque de produire un bloc déprécié 90% du temps. Ainsi, si l’intervalle de bloc est suffisamment court pour que le taux de blocs dépréciés soit élevé, A sera nettement plus efficace simplement en raison de sa taille. Ces deux effets combinés font que les blockchains qui produisent rapidement des blocs sont très susceptibles de conduire à une coopérative de minage ayant un assez large pourcentage de la puissance de calcul du réseau et d’avoir de facto un contrôle sur le processus de minage.

Comme l’ont décrit Sompolinsky et Zohar, GHOST résout le premier problème de perte de sécurité du réseau en incluant les blocs dépréciés dans le calcul de la longueur de la chaîne ; c’est-à-dire, non seulement le parent et les ancêtres suivant d’un bloc, mais aussi les descendants dépréciés de l’ancêtre du bloc (en jargon Ethereum, les « uncles » ou oncles) sont ajoutés au calcul permettant d’établir le bloc qui a la plus grande somme de preuve de travail. Pour résoudre la deuxième question du biais de centralisation, nous allons au-delà du protocole décrit par Sompolinsky et Zohar et nous donnons une récompense aux blocs dépréciés : un bloc déprécié reçoit 87,5% de sa récompense de base, et le neveu qui inclut le bloc déprécié reçoit les 12,5% restant. Les frais de transaction, eux, ne sont pas donnés aux oncles.

Ethereum implémente une version simplifiée de GHOST qui ne descend que de sept niveaux. Plus précisément, cela est définit comme suit :

•Un bloc doit spécifier un parent, et il doit spécifier 0 oncle ou plus ;

•Un oncle inclus dans le bloc B doit avoir les propriétés suivantes :

•Il doit être un enfant direct de la k-ième génération d’ancêtre de B, où 2 <= k <= 7 :
•Il ne peut pas être un ancêtre de B ;
•Un oncle doit être un en-tête de bloc valide mais n’a pas besoin d’être un bloc préalablement vérifié ou même valide ;
•Un oncle doit être différent de tous les oncles inclus dans les blocs précédents et de tous les autres oncles inclus dans le même bloc (pas de double inclusion) ;

•Pour chaque oncle U dans le bloc B, le mineur de B reçoit un montant supplémentaire de 3,125% ajouté à sa récompense de coinbase (la première transaction du bloc pour le mineur) et le mineur de U obtient 93,75% d’une récompense de coinbase standard.

Cette version limitée de GHOST, avec des oncles pouvant être inclus jusqu’à 7 générations, a été employée pour deux raisons. Tout d’abord, l’utilisation de GHOST sans limitation compliquerait trop le calcul de la validité des oncles pour un bloc donné. Deuxièmement, un GHOST sans limite avec compensation dans Ethereum supprimerait l’incitation du mineur de miner sur la chaîne principale et non la chaîne d’un attaquant public.

Les frais

Comme chaque transaction publiée dans la blockchain impose au réseau le coût de son téléchargement et de sa vérification, il y a nécessité d’un mécanisme de régulation, impliquant généralement des frais de transaction, pour prévenir les abus. L’approche par défaut, utilisé dans Bitcoin, est d’avoir des frais purement volontaires, en se reposant sur les mineurs pour agir comme des gardiens et définir des minimums dynamiques. Cette approche a été reçue très favorablement dans la communauté Bitcoin notamment en raison de sa philosophie « basée sur le marché », où l’offre et la demande entre les mineurs et les expéditeurs de transactions détermine le prix. Cependant, le problème avec cette façon de raisonner est que le traitement des transactions n’est pas un marché ; bien qu’il soit intuitivement attrayant d’interpréter le traitement des transactions en tant que service que le mineur offre à l’expéditeur, en réalité, chaque transaction qu’un mineur inclut devra être traitée par chaque nœud du réseau, de sorte que la grande majorité des coûts de traitement de transaction sera prise en charge par des tiers et non par le mineur qui prend la décision de l’inclure ou non. Par conséquent, des problèmes de « tragédie des biens communs » sont très susceptibles de se produire.

Cependant, il se trouve que la faille de ce mécanisme fondé sur le marché s’annule comme par magie lorsque l’on emploie une hypothèse de simplification inexacte. L’argument est le suivant. Supposons que :

1.Une transaction mène à k opérations, offrant la récompensekR à tout mineur qui l’inclut où R est fixé par l’expéditeur et k et R sont (approximativement) visibles par le mineur au préalable.

2.Une opération a un coût de traitement C pour chaque nœud (c.à.d. que tous les nœuds ont une efficacité égale).

3.Il y a N nœuds de minage, chacun possédant une puissance de traitement exactement égale (c.à.d. 1/N du total).

4.Il n'existe aucun nœud complet qui ne mine pas.

Un mineur est disposé à traiter une transaction si la récompense attendue est supérieure au coût. Ainsi, la récompense attendue est kR/N puisque le mineur a 1/N chance de traiter le bloc suivant, et le coût de traitement pour le mineur est tout simplement kC. Par conséquent, les mineurs vont inclure les transactions où kR/N > kC ou bien R > NC. On note que R représente les frais par opération fournis par l'expéditeur, par conséquent une borne inférieure sur l'avantage que l'expéditeur retire de la transaction, et NC est le coût pour l'ensemble du réseau de traiter une opération. Par conséquent, les mineurs sont incités à inclure uniquement les transactions pour lesquelles le bénéfice utilitaire total excède le coût.

Cependant, il y a en réalité plusieurs écarts importants par rapport à ces hypothèses :

1.Le mineur paie un coût plus élevé pour traiter la transaction que les autres nœuds la vérifiant puisque les retards de vérification supplémentaires bloquent la propagation et augmentent ainsi les chances du bloc de devenir déprécié.

2.Il existe des nœuds complets qui ne minent pas.

3.La répartition de la puissance de minage peut devenir radicalement inégalitaire dans la pratique.

4.Il existe des spéculateurs, des ennemis politiques et des cinglés dont la principale fonction consiste notamment à causer des dommages au réseau. Tous ceux-ci peuvent habilement mettre en place des contrats dont le coût qu'ils assument est beaucoup plus faible que le coût payé par d'autres nœuds de vérification.

(1) fournit une incitation pour le mineur d'inclure moins de transactions, et (2) augmente NC ; par conséquent, ces deux effets se neutralisent mutuellement au moins partiellement. (3) et (4) posent les plus gros problèmes ; pour les résoudre, nous établissons simplement une limite flottante : aucun bloc peut avoir plus d'opérations que BLK_LIMIT_FACTOR fois la moyenne mobile exponentielle à long terme. Plus précisément :

```
blk.oplimit = floor((blk.parent.oplimit * (EMAFACTOR - 1) + floor(parent.opcount *
BLK_LIMIT_FACTOR)) / EMA_FACTOR)
```

BLK_LIMIT_FACTOR et EMA_FACTOR sont des constantes qui sont établies à 65536 et 1,5 pour le moment, mais seront fortement susceptibles de changer après de plus amples analyses.

Il existe un autre facteur décourageant les blocs de grande taille dans Bitcoin : les blocs de grandes tailles prendront plus de temps à se propager et auront donc une plus grande probabilité de devenir dépréciés. Dans Ethereum, des blocs très consommateurs de gaz peuvent également prendre plus de temps à se propager à la fois parce qu'ils sont physiquement plus grand et parce qu'il faut plus de temps pour y traiter les transitions d'état de transactions pour les valider. Ce retard dissuasif est une considération importante dans Bitcoin, mais moins dans Ethereum en raison du protocole GHOST ; par conséquent, des limites régulées de blocs procurent une base plus stable.

Calcul et Turing-complétude

Il est important de noter que la machine virtuelle Ethereum est Turing-complète ; cela signifie que le code EVM permet de programmer tout calcul qui peut théoriquement être réalisé, y compris des boucles infinies. Le code EVM permet d'effectuer des boucles de deux manières. Premièrement, l'instruction JUMP permet au programme de revenir à un point précédent dans le code, tandis que l'instruction JUMPI permet d'effectuer un saut conditionnel, permettant des déclarations comme `while x < 27: x = x * 2`. Deuxièmement, les contrats peuvent appeler d'autres contrats, permettant potentiellement une boucle à travers la récursivité. Cela conduit naturellement à un problème : des utilisateurs malveillants peuvent-ils faire planter les mineurs et les nœuds complets en les forçant à entrer dans une boucle infinie ? La question se pose en raison d'un problème informatique connu sous le nom de « problème de l'arrêt » : il n'y a aucun moyen de dire, dans un cas général, si oui ou non un programme donné finira par s'arrêter.

Comme on l'a vu dans la section sur la transition d'état, notre solution fonctionne en exigeant d'une transaction qu'elle fixe un nombre maximum d'étapes de calcul qu'elle est autorisée à effectuer. Si ce nombre est dépassé, le calcul est annulé mais les frais sont tout de même payés. Les messages fonctionnent de la même manière. Pour comprendre les motivations derrière notre solution, considérons les exemples suivants :

•Un attaquant crée un contrat qui exécute une boucle infinie et envoie ensuite une opération d'activation de cette boucle au mineur. Le mineur traite la transaction, exécutant la boucle infinie, et attend qu'elle soit à court de gaz. Même si la transaction est à court de gaz et qu'elle s'arrête à mi-parcours, la transaction est toujours valide et le mineur reçoit de l'attaquant les frais correspondant à chaque étape de calcul.

•Un attaquant crée une boucle infinie avec l’intention de forcer le mineur à continuer son calcul pendant une durée suffisamment longue pour que d’ici la fin du calcul, un ou plusieurs blocs aient eu le temps d’apparaître, empêchant le mineur d’inclure le transaction pour réclamer le paiement. Cependant, l’attaquant est tenu d’indiquer une valeur pour STARTGAS limitant le nombre d’étapes de calcul que l’exécution peut effectuer, de sorte que le mineur saura à l’avance que le calcul effectuera un trop grand nombre d’étapes.

•Un attaquant repère un contrat dont le code ressemble à send(A,contract.storage[A]); contract.storage[A] = 0, et envoie une transaction avec juste assez de gaz pour exécuter la première étape, mais pas la seconde (c.à.d. effectuer un retrait, mais ne pas laisser la balance se décrémenter). L’auteur du contrat n’a pas à se soucier de se protéger contre de telles attaques, car si l’exécution est interrompue, les changements sont annulés.

•Un contrat financier prend la médiane de neuf flux de données propriétaires afin de minimiser les risques. Un attaquant prend de contrôle d’un des flux de données, qui est conçu pour être modifiable via le mécanisme d’appel par adresse variable décrit dans la section sur les DAO, et l’altère pour qu’il exécute une boucle infinie, tentant ainsi de forcer à court de gaz toute tentative de réclamer des fonds du contrat financier. Cependant, le contrat financier peut fixer une limite de gaz sur le message pour éviter ce problème.

Le pendant d’un langage Turing-complet est un langage Turing-incomplet, où JUMP etJUMPI n’existent pas et qui ne permet l’existence que d’une seule copie de chaque contrat dans la pile d’exécution à un moment donné. Avec ce système, le système de frais décrit et les incertitudes autour de l’efficacité de notre solution n’auraient pas lieu d’être, car le coût d’exécution d’un contrat serait défini par sa taille. En outre, la non Turing-complétude n’est pas en soi une limitation importante ; sur tous les exemples de contrats que nous avons conçus en interne jusqu’à présent, un seul nécessitait une boucle, qui pouvait d’ailleurs être éliminée en faisant 26 répétitions successives d’un morceau de code d’une ligne. Étant donné les graves conséquences de la Turing-complétude, et l’avantage limité, pourquoi ne pas se contenter d’un langage non Turing-complet ? En pratique, un langage non Turing-complet est loin d’être une solution élégante au problème. Pour comprendre pourquoi, considérons les contrats suivants :

```
C0: call(C1); call(C1);

C1: call(C2); call(C2);

C2: call(C3); call(C3);

...

C49: call(C50); call(C50);
```

C50: (exécute une étape d'un programme et enregistre le changement dans l'espace de stockage)

Maintenant, on envoie une transaction à A. En 51 transactions, nous avons ainsi un contrat qui effectue 250 étapes de calcul. Les mineurs pourraient essayer de détecter de telles bombes logiques à l’avance par le maintien d’une valeur associé à chaque contrat spécifiant le nombre maximum d’étapes de calcul qu’il est autorisé à effectuer, en calculant aussi cette valeur pour les contrats appelant d’autres contrats récursivement, mais il faudrait pour cela que les mineurs interdisent les contrats qui créent d’autres contrats (puisque la création et l’exécution de l’ensemble des 26 contrats ci-dessus pourraient facilement être programmés dans un unique contrat). Un autre point problématique est que le champ d’adresse d’un message est une variable. Il n’est donc même pas garanti qu’il soit possible de déterminer à l’avance quels autres contrats un contrat donné sera susceptible d’appeler. Par conséquent, au final, nous arrivons à une conclusion surprenante : un langage Turing-complet est étonnamment facile à gérer, alors qu’un langage Turing-incomplet est tout aussi étonnamment difficile à gérer à moins que les mêmes contrôles précis soient mis en place – mais dans ce cas, pourquoi ne pas simplement choisir un protocole Turing-complet ?

Monnaie et émission

Le réseau Ethereum inclut sa propre monnaie, l’ether, qui sert le double objectif de fournir une couche de liquidité primaire pour permettre un échange efficace entre les différents types d’actifs numériques et, plus important encore, de fournir un mécanisme pour le paiement des frais de transaction. Pour plus de simplicité et pour éviter d’éventuels conflits (voir le débat mBTC/uBTC/satoshi actuel au sujet de Bitcoin), les dénominations seront prédéfinies :

•1 : wei

•1012 : szabo

•1015 : finney

•1018 : ether

If faut considérer cela comme une version étendue du concept de « dollars » et « cents » ou de « BTC » et « satoshi ». Dans un futur proche, nous nous attendons à ce que « ether » soit utilisé pour des transactions ordinaires, « finney » pour des micro-transactions et « szabo » et « wei » pour des discussions techniques à propos de frais et d’implémentation de protocole ; les dénominations restantes pourraient devenir utiles dans le futur et ne doivent pas être incluses dans les clients pour le moment.

Le protocole d'émission sera le suivant :

•De l'ether sera distribué par une vente de monnaie au prix de 1000 à 2000 ether par BTC, un mécanisme visant à financer l'organisation Ethereum et payer les développements qui a été utilisé avec succès par d'autres plateformes telles que Mastercoin et NXT. Les premiers acheteurs bénéficieront de remises plus importantes. Les BTC reçus seront utilisés en totalité pour payer les salaires et les primes des développeurs et à investir dans divers projets, à buts lucratifs ou non, de l'écosystème Ethereum et des crypto-monnaies en général.

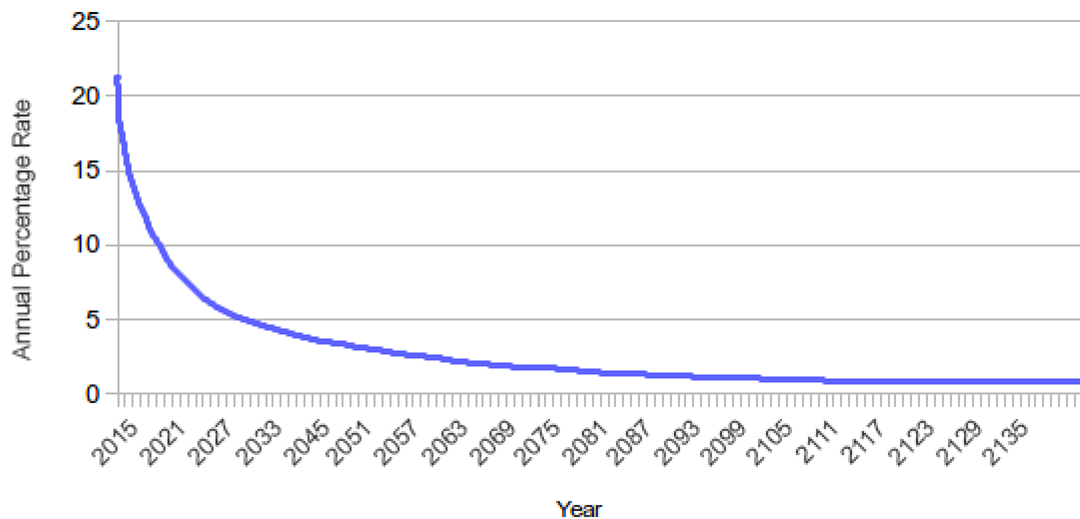
•9,9M du montant total vendu (60 102 216 ETH) seront alloués à l'organisation pour rétribuer les contributeurs initiaux et régler les dépenses liées à ETH effectuées en amont du genesis block, le bloc de genèse.

•9,9% du montant total vendu sera conservé comme réserve de fonds à long terme.

•26% du montant total vendu sera chaque année alloué aux mineurs, sans limite dans le temps.

Groupe	Au lancement	Après 1 an	Après 5 ans
Unités monétaires	1,198x	1x458x	2,498x
Acheteurs	83,5%	68,6%	40,0%
Réserve dépensée avant vente	8,26%	6,79%	3,96%
Réserve dépensée après vente	8,26%	6,79%	3,96%
Mineurs	0%	17,8%	52,0%

Taux de croissance de l'approvisionnement à long terme (pourcentages)



Malgré l’émission de monnaie linéaire, la progression de l’approvisionnement, tout comme avec Bitcoin, tend vers zéro

Les deux principaux choix dans le modèle ci-dessus sont (1) l’existence et la dimensionnement d’un fond de dotation, et (2) l’existence d’un approvisionnement linéaire permanent, différent de l’approvisionnement plafonné de Bitcoin. La justification du fond de dotation est la suivante. Si le fond de dotation n’existait pas, et que l’émission linéaire était réduite à 0,217x afin de fournir le même taux d’inflation, la quantité totale d’ether serait réduite de 16,5%, sa valeur augmentant chaque année de 19,8%. Par conséquent, à l’équilibre, 19,8% d’ether supplémentaires seraient achetés durant la vente, conférant à chaque unité la même valeur qu’auparavant. L’organisation aurait alors également 1,198 fois plus de BTC, qui pourraient être divisé en deux tranches : les BTC d’origine et les 19,8% supplémentaires. Par conséquent, cette situation serait exactement équivalente à la dotation, mais avec une différence importante : l’organisation détenant uniquement des BTC, elle ne serait pas incitée à soutenir la valeur de l’ether.

Le modèle de croissance de l’émission linéaire permanente réduit le risque de concentration excessive de la richesse telle que certains l’observent dans Bitcoin, et offre à chacun, aujourd’hui ou demain, la possibilité d’acquérir des unités monétaires, tout en conservant une forte incitation à acquérir et conserver des éther puisque le « taux de croissance de l’approvisionnement » en pourcentage tend toujours vers zéro au fil du temps. Nous pouvons aussi prévoir cela car puisque des pièces sont forcément perdues au fil du temps en raison de négligences, décès, etc. et comme la perte de pièces peut être modélisée en tant que pourcentage de l’émission annuelle totale, l’offre totale de monnaie en circulation se stabilisera finalement à une valeur égale à l’émission annuelle divisée par le taux de perte (par exemple à un taux de perte de 1%, une fois que l’émission aura atteint 26x alors 0,26x seront minés et 0,26x perdus chaque année, créant ainsi un équilibre).

Notons qu’un jour, il est probable qu’Ethereum passe à un modèle de proof-of-stake (preuve d’enjeu) pour des raisons de sécurité, réduisant l’exigence d’émission à une fourchette allant de zéro à 5% chaque année. Dans le cas où l’organisation Ethereum perdrait son financement ou viendrait à disparaître, nous proposons le « contrat social » suivant : toute personne a le droit de créer une nouvelle version candidate d’Ethereum, à l’unique condition de limiter la quantité totale d’ether à 60102216 * (1.198 + 0,26 * n) où n représente le nombre d’années après le bloc de genèse. Les créateurs sont libres de procéder à la vente ou d’assigner une partie ou la totalité de la différence entre l’expansion de l’émission en preuve d’enjeu et l’expansion de l’émission maximale afin de financer le développement. Les mises à jour ne respectant pas ce contrat social pourront être légitimement scindées vers des versions conformes.

Concentration du minage

L’algorithme de minage de Bitcoin fonctionne en faisant calculer aux mineurs des empreintes SHA256 sur des versions légèrement modifiées de l’en-tête de bloc, des millions de fois, sans relâche, jusqu’à ce que un nœud obtienne finalement une version dont l’empreinte est inférieure à la cible (actuellement autour de 2192). Cet algorithme de minage est cependant vulnérable à deux formes de centralisation. Tout d’abord, l’écosystème du minage a fini par être dominé par les ASIC (application-specific integrated circuits ou circuits intégrés spécialisés), des puces informatiques conçues spécifiquement pour le minage de Bitcoin et donc des milliers de fois plus performantes. Cela signifie que le minage de Bitcoin n’est plus vraiment une quête décentralisée et égalitaire puisqu’il nécessite des millions de dollars d’investissement pour une participation efficace. Deuxièmement, la plupart des mineurs Bitcoin n’effectuent pas la validation des blocs localement. Ils dépendent d’un pool centralisé (coopérative de mineurs) pour leur fournir les en-têtes de blocs. Ce problème est sans doute le pire : au moment de la rédaction de ce document, les trois principales coopératives contrôlent indirectement environ 50% de la puissance de calcul du réseau Bitcoin, bien que cela soit atténué par le fait que les mineurs puissent passer à d’autres coopératives au cas où l’une d’elles ou une coalition tente une attaque des 51%.

Ethereum a actuellement l’intention d’utiliser un algorithme de minage obligeant les mineurs à extraire des données aléatoires de l’état, à calculer certaines transactions choisies au hasard parmi les N derniers blocs dans la blockchain, et à renvoyer l’empreinte du résultat. Cela présente deux avantages importants. Tout d’abord, les contrats Ethereum peuvent comporter tous types de calculs, donc un ASIC Ethereum serait essentiellement un ASIC généraliste – c.à.d. un processeur plus performant. Deuxièmement, le minage nécessite l’accès à l’ensemble de la blockchain, obligeant les mineurs à la stocker en totalité et à être au moins capable de vérifier chaque transaction. Cela élimine le besoin de recourir à des coopératives de minage centralisées ; bien que celles-ci puissent toujours servir à atténuer le caractère aléatoire des récompenses, cette fonction peut tout à fait être assurée par des coopératives en pair-à-pair sans contrôle centralisé.

Ce modèle n’a pas été testé et l’on peut rencontrer des difficultés pour éviter certaines optimisations ingénieuses lors de l’exécution d’un contrat en tant qu’algorithme de minage.

Cet algorithme présente cependant une caractéristique intéressante qui permet à quiconque d'« empoisonner le puits », en créant un grand nombre de contrats dans la blockchain spécifiquement conçus pour contrecarrer certains ASIC. Les incitations économiques existantes peuvent pousser les fabricants d'ASIC à utiliser ces techniques pour s'attaquer mutuellement. Ainsi, la solution que nous développons est finalement une solution adaptative économique humaine plutôt que purement technique.

Passage à l'échelle

Un sujet d'inquiétude courant à propos d'Ethereum est la question de son passage à l'échelle. Comme Bitcoin, Ethereum souffre de ce que chaque transaction nécessite d'être traitée par tous les nœuds du réseau. Avec Bitcoin, la taille actuelle de la blockchain est d'environ 15 Go, augmentant d'environ 1 Mo par heure. Si le réseau Bitcoin devait traiter les 2000 transactions par seconde de Visa, elle augmenterait de 1 Mo toutes les trois secondes (1 Go par heure, 8 To par an). Ethereum est susceptible de pâtir d'un modèle de croissance similaire, aggravé par le fait qu'il y aura de nombreuses applications sur la blockchain Ethereum et non uniquement une monnaie comme c'est le cas avec Bitcoin, mais atténué par le fait que les nœuds complet Ethereum ne doivent stocker que l'état et pas l'intégralité de l'historique de la blockchain.

Le problème avec une blockchain d'aussi grande taille est le risque de centralisation. Si la taille de la blockchain atteint, disons, 100 To, le scénario probable serait que seul un très petit nombre de grandes entreprises ferait tourner des nœuds complets, avec tous les utilisateurs ordinaires utilisant des nœuds SPV légers. Dans une telle situation, survient le problème potentiel que les nœuds complets pourraient se regrouper et se mettre d'accord pour tricher dans un intérêt lucratif (par exemple en changeant la récompense de bloc, en s'attribuant des BTC). Les nœuds légers n'auraient aucun moyen de le détecter immédiatement. Bien sûr, il existerait probablement au moins un nœud complet honnête et, après quelques heures, l'information au sujet de la fraude finirait par atteindre des canaux comme Reddit, mais il serait déjà trop tard : les utilisateurs ordinaires devraient s'organiser pour mettre sur liste noire les blocs concernés, une tâche de coordination aussi massive et probablement aussi infaisable que la réalisation d'une attaque à 51%. Dans le cas de Bitcoin, c'est actuellement un problème, mais il existe une modification de la blockchain suggérée par Peter Todd qui permettra d'atténuer ce problème.

Dans un futur proche, Ethereum va mettre en place deux stratégies supplémentaires pour faire face à ce problème. La première imposera, du fait des algorithmes de minage basés sur la blockchain, qu'au minimum tout mineur soit un nœud complet, créant un plancher quant au nombre de nœuds complets. La deuxième, et certainement la plus importante, consistera à inclure une racine d'état intermédiaire dans la blockchain après le traitement de chaque transaction. Même si la validation des blocs est centralisée, tant qu'un seul nœud de vérification existe, le problème de la centralisation peut être contourné grâce à un protocole de vérification. Si un mineur publie un bloc invalide, c'est que ce bloc est soit mal formaté, soit que l'état `S[n]` est incorrect. Puisque `S[0]` est connu comme étant correct, il doit exister un premier état `S[i]` qui est incorrect où `S[i-1]` est correct. Le noeud de vérification fournirait l'indice `i`, avec une « preuve d'invalidité » constituée par le sous-ensemble de nœuds d'arbre Patricia nécessaire pour traiter `APPLY(S[i-1],TX[i]) -> S[i]`.

Les nœuds (complets) seraient en mesure d'utiliser ces noeuds (de vérification) pour exécuter cette partie du calcul, et de voir que le S[i] généré ne correspond pas au S[i] fourni.

Un autre attaque, plus sophistiquée, impliquerait que les mineurs malveillants publient des blocs incomplets, et qu'ainsi l'information complète n'existe nulle part pour déterminer si oui ou non les blocs sont valides. La solution à cela est un protocole de challenge-response (défi-réponse) : les nœuds de vérification émettent des « défis » sous la forme d'indices de transaction cible, et lors de la réception d'un bloc, un nœud léger traite ce bloc comme suspect jusqu'à ce qu'un autre nœud, que ce soit celui du mineur ou d'un autre vérificateur, fournisse un sous-ensemble de nœuds Patricia comme preuve de validité.

Conclusion

Le protocole Ethereum a été initialement conçu comme une crypto-monnaie améliorée offrant des fonctionnalités avancées sur la blockchain telles que dépôts sous séquestre, limites de retrait, contrats financiers, plateformes de pari, etc. via un langage de programmation très généraliste. Le protocole Ethereum ne « supporte » aucune de ces applications directement mais l'existence d'un langage de programmation Turing-complet permet théoriquement la création de contrats quelconques pour tout type de transaction ou d'application. Le plus intéressant dans Ethereum est que son protocole va bien au delà de la monnaie. Les protocoles en rapport avec le stockage de fichier décentralisé, le calcul décentralisé et les marchés de prédiction décentralisés, parmi des dizaines d'autres concepts du même ordre, ont le potentiel d'augmenter considérablement la productivité de l'industrie informatique, et de stimuler massivement d'autres protocoles pair-à-pair en y ajoutant pour la première fois une couche économique. Enfin, il existe aussi toute une gamme d'applications importantes n'ayant aucun rapport avec l'argent.

Le concept d'une fonction arbitraire de transition d'état telle qu'elle est implémentée par le protocole Ethereum offre une plateforme avec un potentiel unique ; plutôt que d'être un protocole fermé à usage unique destiné à une gamme spécifique d'applications dans le stockage de données, le jeu ou la finance, Ethereum est ouvert par conception, et nous croyons qu'il est extrêmement bien adapté pour servir de socle à un très grand nombre de protocoles financiers et non financiers dans les années à venir.

Références et suggestions de lecture

Références

1.Un lecteur averti notera qu'en pratique, une adresse Bitcoin est l'empreinte de la clé publique de courbe elliptique, et non la clé publique elle-même. Cependant, il est

parfaitement admis en terminologie cryptographique d'assimiler l'empreinte de la clé publique à la clé publique elle-même. En effet, la cryptographie Bitcoin peut être considérée comme un algorithme de signature numérique particulier, où la clé publique est constituée de l'empreinte de la clé publique ECC, où la signature représente la clé publique ECC concaténée avec la signature ECC, et où l'algorithme de vérification consiste à valider la clé publique ECC dans la signature avec l'empreinte de la clé publique ECC fournie en tant que clé publique pour ensuite valider la signature ECC avec la clé publique ECC.

2.D'un point de vue technique, la médiane des 11 blocs précédents.

3.En interne, 2 et « CHARLIE » sont tous deux des nombres, le second étant représenté en base 256 gros-boutien (big-endian). Les nombres peuvent aller de 0 à 2256-1.

Suggestions de lecture

1.Valeur intrinsèque : http://bitcoinmagazine.com/8640/an-exploration-of-intrinsic-value-what-it-is-why-bitcoin-doesnt-have-it-and-why-bitcoin-does-have-it/

2.Smart property : https://en.bitcoin.it/wiki/Smart_Property

3.Smart contracts: https://en.bitcoin.it/wiki/Contracts

4.B-money : http://www.weidai.com/bmoney.txt

5.Preuves de travail réutilisables : http://www.finney.org/~hal/rpow/

6.Titres de propriété sécurisés avec preuve de possession : http://szabo.best.vwh.net/securetitle.html

7.Livre blanc Bitcoin : http://bitcoin.org/bitcoin.pdf

8.Namecoin : https://namecoin.org/

9.Triangle de Zooko : http://en.wikipedia.org/wiki/Zooko's_triangle

10. Livre blanc colored coins : https://docs.google.com/a/buterin.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE/edit

11. Livre blanc Mastercoin : https://github.com/mastercoin-MSC/spec

12. Sociétés autonomes décentralisées, Bitcoin Magazine : http://bitcoinmagazine.com/7050/bootstrapping-a-decentralized-autonomous-corporation-part-i/

13. Vérification de paiement simplifiée : https://en.bitcoin.it/wiki/Scalability#Simplifiedpaymentverification

14. Arbres de Merkle : http://en.wikipedia.org/wiki/Merkle_tree

15. Arbres Patricia : http://en.wikipedia.org/wiki/Patricia_tree

16. GHOST : http://www.cs.huji.ac.il/~avivz/pubs/13/btc_scalability_full.pdf

17. StorJ and Autonomous Agents, Jeff Garzik : http://garzikrants.blogspot.ca/2013/01/storj-and-bitcoin-autonomous-agents.html

18. Mike Hearn, Smart Property, Turing Festival : http://www.youtube.com/watch?v=Pu4PAMFPo5Y

19. RLP Ethereum : https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-RLP

20. Arbres de Merkle et Patricia dans Ethereum : https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-Patricia-Tree

21. Peter Todd à propos des arbres de Merkle additifs : http://sourceforge.net/p/bitcoin/mailman/message/31709140/

Document d’origine : Vitalik Buterin – https://github.com/ethereum/wiki/wiki/White-Paper

Traduction : Stéphane Roche, Jean Zundel, Frédéric Jacquot, Alexandre Kurth et Etienne Jouin – https://github.com/asseth/whitepaper