Comprendre Bitcoin
Rapport de projet INFRES357

Pierre Galland, Benoît de Laitre

3 mai 2015

Table des matières

1 Le protocole Bitcoin

1.1 Briques de base

Le protocole de Bitcoin utilise plusieurs briques de bases de la cryptographie : le chiffrement asymétrique, la signature et le hachage. C'est d'ailleurs là que réside l'intérêt de ce protocole, c'est un assemblage astucieux de briques qui existent depuis de nombreuses années et qui sont très bien connues, pourtant cet assemblage crée une technologie totalement nouvelle.

1.1.1 Chiffrement asymétrique

Le principe du chiffrement asymétrique est que les clefs vont par paire, une clef publique et une clef privé (K_{Pub}, K_{Pri}). On peut voir les clefs comme des suites finies de caractères ou de chiffres, et les message aussi. Il est possible de chiffrer un message avec la clef privé et alors le message chiffré ne pourra être déchiffré qu'avec la clef publique correspondante. De même on peut chiffrer un message avec la clef publique et alors il ne pourra être déchiffré qu'avec la clef privé correspondante. Considérons l'exemple classique de Bob et Alice, ils possède chacun une paire clef publique/clef privée. La paire de clefs de Bob est (K^B_{Pub}, K^B_{Pri}) et celle d'Alice est (K^A_{Pub}, K^A_{Pri}).

Si Bob veut envoyer un message $mess$ à Alice qu'elle seule pourra lire, alors il chiffre son message avec le clef publique d'Alice, et il obtient le message chiffré $*mess*$:

$$chiffrer(mess,\ K^A_{Pub}) \rightarrow *mess*$$

Il envoie alors ce message $*mess*$ à Alice, qui quand elle le reçoit le déchiffre avec sa clef privée K^A_{Pri} :

$$déchiffrer(*mess*,\ K^A_{Pri}) \rightarrow mess$$

Si Alice essayait de déchiffrer le message $*mess*$ avec une autre clef que sa clef privée K^A_{Pri} alors cela ne marcherait pas, elle obtiendrait n'importe quoi (une suite de symbole qui n'a rien à voir avec le message $mess$). Donc comme on considère qu'Alice est la seule à connaître sa clef privée, elle est la seule à pouvoir déchiffrer le message.

$$déchiffrer(*mess*,\ K^{autre}_{Pri}) \rightarrow n'importe\ quoi$$

On peut également utiliser la paire clef publique/clef privée dans l'autre sens. Si Bob chiffre un message avec sa clef privée K^B_{Pri} alors on ne pourra le déchiffrer qu'avec sa clef publique K^B_{Pub}. En essayant de le déchiffrer avec une autre clef K^{autre}_{Pub} on obtiendrait n'importe quoi.

Ainsi si Alice veut que Bob prouve qu'il est bien Bob donc qu'il connaît la clef privée K^B_{Pri}, sans qu'il ait à révéler cette clef, alors elle peut créer un message $mess2$ et demander à Bob de le chiffrer avec sa clef K^B_{Pri} et de lui envoyer le résultat $*mess2*$. Alice va ensuite déchiffrer $*mess2*$ avec la clef publique de Bob K^B_{Pub} et si elle retombe bien sur $mess2$ cela prouve que Bob est bien Bob (qu'il connaît la clef K^B_{Pri}). Ce mécanisme sera très utile pour la signature, décrite plus loin.

1.1.2 Hachage cryptographique

On considère toujours nos messages comme des suites finies de caractères ou de chiffres (d'ailleurs in fine sur un ordinateur toute donnée est une suite finie de chiffres). Le but d'une fonction de hachage cryptographique h est de transformer chaque message de taille inférieure à T (où T est très très grand) en un message de taille beaucoup plus petite que T (par exemple dans Bitcoin en message de longueur 256 chiffres). Si j'ai un message $mess$ assez long alors $h(mess)$ sera beaucoup plus petit que $mess$. On appellera $h(mess)$ le hash de $mess$.

Prenons un exemple avec la fonction de hachage SHA-256, qui est utilisée dans Bitcoin. Je considère le message $mess$ suivant et je calcule son $h(mess)$:

$$mess = Un\ message\ pas\ très\ long,\ juste\ pour\ faire\ un\ exemple$$

$$h(mess) \rightarrow 6423e8584411fee28e5064799d8a230c64f999c448c769ab7d309baba9a33f42$$

Le but de la fonction de hachage est également que le hash d'un message puisse l'identifier de manière presque unique. Ainsi à priori si je considère deux messages différents alors leurs hashs sont également différents.

$$mess1 \neq mess2 \Rightarrow presque\ toujours\ h(mess1) \neq h(mess2)$$

Un autre point important est que si je connais seulement le hash du message $h(mess)$ ainsi que la fonction de hachage h mais que je ne connais pas le message $mess$ alors je ne puisse pas retrouver quel est le message $mess$. Il est impossible (cela demande trop de puissance de calcul) de retrouver le message à partir de son hash.

$$h\ et\ h(mess) \nrightarrow mess$$

Enfin si je considère un message $mess1$ et ma fonction de hachage h, il m'est impossible (une fois encore cela demande trop de puissance de calcul) de trouver un message $mess2$ tel que $h(mess1) = h(mess2)$.

Les fonctions de hachage jouent un rôle très important dans le protocol Bitcoin !

1.1.3 Signature

Le but de la signature électronique est de s'assurer que le message que l'on reçoit a bien été envoyé par la personne qui signe, et que le message n'a pas été modifié entre le moment de la signature et sa réception. Reprenons l'exemple de Bob et Alice, avec leurs paires de clefs respectives (K^B_{Pub}, K^B_{Pri}) et (K^A_{Pub}, K^A_{Pri}). Pour qu'Alice puisse être certaine que le message qu'elle reçoit de Bob est bien de lui et qu'il n'a pas été modifié en cours de route il faut que Bob signe son message :

Après avoir écrit son message $mess$, Bob calcul son hash $h(mess)$

$$mess \rightarrow h(mess)$$

Puis il utilise chiffre ce hash avec sa clef privée :

$$chiffrer(h(mess),\ K^B_{Pri}) \rightarrow *h(mess)*$$

Ce $*h(mess)*$ constitue la signature du message. Bob envoie à Alice le message avec sa signature : $(mess,\ *h(mess)*)$. Lorsque Alice reçoit le message et la signature elle vérifie que la signature est valide et qu'elle provient bien de Bob.

Alice reçoit donc $(mess1,\ *h(mess)*)$, on écrit $mess1$ car pour le moment on ne sait pas s'il s'agit bien du message original de Bob (il a peut-être été modifié par quelqu'un l'ayant intercepté). Alice calcule le hash de $mess1$:

$$mess1 \ \rightarrow \ h(mess1)$$

Puis elle déchiffre $*h(mess)*$ avec la clef publique de Bob. En le déchiffrant avec la clef publique de Bob K^B_{Pub} elle s'assure que $*h(mess)*$ a bien été chiffré avec la clef privé de Bob K^B_{Pri}. En effet s'il avait été chiffré avec une autre clef privée alors en le déchiffrant avec K^B_{Pub} on obtiendrait n'importe quoi. Comme à priori on n'est pas encore sûr que le $*h(mess)*$ que l'on déchiffre a bien été chiffré avec la clef privée de Bob, on note $?h(mess)?$ ce que l'on obtient :

$$déchiffrer(*h(mess)*,\ K^B_{Pub}) \ \rightarrow ?h(mess)?$$

Enfin on compare $h(mess1)$ et $?h(mess)?$, s'ils sont égaux alors on est sûr (modulo la solidité de la fonction de hachage et de l'algorithme de chiffrement) que le message est bien de Bob et qu'il n'a pas été modifié, s'ils sont différents alors on ne peut pas faire confiance à ce message.

On peut se convaincre que c'est la cas en reprenant le processus et en imaginant qu'un attaquant modifie le message et/ou la signature en cours de route, ou qu'il essaie d'envoyer le message avec sa paire de clefs. On verra qu'à la fin le test que fait Alice donne bien $h(mess1) \ \neq \ ?h(mess)?$ (en gardant à l'esprit qu'Alice connaît la clef publique de Bob).
En revanche si un attaquant connaît la clef privée de Bob alors il peut très bien se faire passer pour lui !

1.2 Une transaction Bitcoin

1.2.1 Identité et adresses

Dans le protocole Bitcoin la notion d'identité repose sur le chiffrement asymétrique donc un individu est une paire clef publique/clef privée. Sachant que l'on peut générer la clef publique à partir de la clef privée, un individu est une clef privée. On parle ici d'individus dans le protocole, car une personne réelle peut très bien avoir plusieurs clefs privées et donc correspondre à plusieurs individus dans le protocole. C'est d'ailleurs normalement le cas. Chaque personne réelle possédant des bitcoins a au moins une centaine de clefs privées, donc correspond à de nombreux individus théoriques dans le protocole.
Les individus sont identifiés dans le protocole par leur adresse. Une adresse est un hash de clef publique. Par exemple 3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy est une adresse. Pour vérifier qu'une adresse $addr$ correspond bien à la clef publique K_{Pub} on a juste à vérifier que $h(K_{Pub}) \ == \ addr$ où h est la fonction de hachage définie par le protocole.

1.2.2 Structure d'une transaction

Tout d'abord on va considérer qu'il existe un "livre de compte" partagé entre tous les utilisateurs de Bitcoin et que chaque transaction valide est inscrite dans ce livre de compte de manière chronologique. On verra par la suite quel procédé permet l'existence de ce livre de compte, qu'on

appelle la Blockchain.

Chacun peut donc accéder à l'historique de toutes les transactions valides qui ont été effectuées depuis la création de Bitcoin. Il est important de comprendre qu'il n'existe pas de pièce ou de billet de bitcoin, même virtuels. Il n'y a pas de fichiers qui représentent des bitcoins. C'est seulement la Blockchain (le livre de compte partagé) qui indique le nombre de bitcoins associé à chaque adresse. Même cette vision n'est pas tout à fait exacte. La Blockchain n'est que l'historique de toutes les transactions valides effectuées. Il est donc plus correct de dire qu'on peut calculer le nombre de bitcoin de chaque adresse à partir de la Blockchain.

Prenons un exemple : on suppose que Bob veut envoyer 1 btc à Alice (on utilisera l'abréviation btc pour désigner l'unité de monnaie bitcoin). Bob et Alice ont toujours chacun une clef privée, respectivement K^B_{Pri} et K^A_{Pri}, et ils ont également une adresse bitcoin : $adr_B = h(K^B_{Pub})$ et $adr_A = h(K^A_{Pub})$.

Pour que Bob puisse envoyer 1 btc, il faut qu'il dispose d'au moins 1 btc. Cela signifie qu'il faut qu'il existe dans la Blockchain une transaction qui envoie au moins 1 btc à l'adresse adr_B. Supposons que ce soit le cas, il y a dans la Blockchain une transaction Tx_1 qui envoie 1 btc à l'adresse adr_B. Bob va donc retransférer vers Alice l'argent qu'il a reçu de Tx_1. La transaction Tx de Bob contient donc les éléments suivants :

- Input
  - ⋆ $h(Tx_1)$
  - ⋆ K^B_{Pub}
  - • $signature(K^B_{Pri}, Tx^*)$
- Output
  - ⋆ value : 1 btc
  - ⋆ adr_A

FIGURE 1 – Transaction Tx

La transaction contient donc 5 champs :

Dans la partie Input la transaction indique quelle transaction précédente est utilisée pour récupérer les bitcoins. Ici c'est la transaction Tx_1, c'est pour cela que le premier champ est $h(Tx_1)$. Le deuxième champ est la clef publique de Bob. En effet Bob veut utiliser les bitcoins donnés à adr_B par Tx_1, il doit donc prouver qu'il possède bien l'adresse adr_B. Pour cela il fournit K^B_{Pub} et $signature(K^B_{Pri}, Tx^*)$. On peut donc vérifier d'une part que $h(K^B_{Pub}) == adr_B$, donc que l'adresse adr_B correspond bien à la clef K^B_{Pub}, et d'autre part que $signature(K^B_{Pri}, Tx^*)$ est valide, donc que celui qui émet le message possède bien la clef privée K^B_{Pri}. Ce qui est signé est Tx^* c'est à dire la transaction Tx sans la signature (les éléments précédés d'une étoile sur la figure).

Dans la partie Output Bob indique qu'il souhaite donner 1 btc (champ value) à l'adresse adr_A, soit à Alice.

1.2.3 Sécurité de la transaction

Une fois que Bob a écrit sa transaction, il la diffuse sur le réseau Bitcoin afin qu'elle soit intégrée dans la Blockchain et devienne ainsi une transaction valide, faisant partie de l'historique des transactions Bitcoin. Nous verrons par la suite que le réseau Bitcoin est un réseau peer-to-peer, donc lorsque Bob diffuse sa transaction, n'importe qui peut y avoir accès. La transaction est diffusée en clair sur le réseau.

Cependant pour pouvoir être intégrée dans la Blockchain une transaction doit être valide. Tous les champs seront vérifiés, il sera également vérifié que l'output de la transaction précédente (Tx_1 dans notre exemple) est bien dirigé vers l'adresse émettrice de la nouvelle transaction ($adr_B = h(K^B_{Pub})$ dans notre exemple), et qu'il est suffisant (supérieur à 1 btc dans notre exemple).

Il est donc primordial que si un attaquant entre en possession de la transaction il ne puisse pas la modifier afin par exemple de s'attribuer les fonds plutôt qu'à Alice, ou de changer le montant de la transaction, etc. Explorons quelques tentatives d'attaques et les raisons pour lesquelles elles échouent :

Changer le destinataire d'une transaction

Imaginons le scénario suivant : Bob a créé sa transaction, il la diffuse sur le réseau, en particulier Eve obtient une copie de la transaction. Eve veut essayer de modifier la transaction pour s'attribuer le 1 btc que Bob envoie à Alice, puis rediffuser cette nouvelle transaction modifiée sur le réseau, en espérant qu'elle soit intégrée dans la Blockchain. Eve possède l'adresse adr_E et la clef privée K^E_{Pri}.

Eve fait un premier essai, elle crée la transaction Tx' en remplaçant l'adresse adr_A par son adresse adr_E dans l'output :

- Input
  - ⋆ $h(Tx_1)$
  - ⋆ K^B_{Pub}
  - • $signature(K^B_{Pri}, Tx^*)$
- Output
  - ⋆ value : 1 btc
  - ⋆ adr_E

FIGURE 2 – Transaction Tx'

Mais comme la signature $signature(K^B_{Pri}, Tx^*)$ est faite sur Tx^* et que Eve a changé un champs faisant partie de Tx^*, la signature n'est plus valide. Par conséquent cette transaction ne pourra jamais être acceptée dans la Blockchain. Dans un deuxième essai, Eve décide donc de changer la signature. Cependant comme elle ne connaît pas la clef privée de Bob, elle ne peut pas signer avec. Elle décide donc de signer avec sa clef privée K^E_{Pri}, mais à ce moment là elle doit aussi changer le deuxième champs qui contient K^B_{Pub} sinon la signature sera reconnue comme invalide. Elle va donc remplacer K^B_{Pub} par sa clef publique. Elle crée ainsi une transaction Tx''

- Input
  - ⋆ $h(Tx_1)$
  - ⋆ K^E_{Pub}
  - • $signature(K^E_{Pri}, Tx''^*)$
- Output
  - ⋆ value : 1 btc
  - ⋆ adr_E

FIGURE 3 – Transaction Tx''

Cette fois-ci la signature dans Tx'' est valide, car Eve a signé sur Tx''^*. Cependant il subsiste un problème et la transaction ne sera pas acceptée par la Blockchain. En effet cette transaction veut utiliser les fonds fournis à l'adresse adr_B par la transaction Tx_1, or il va être vérifié que c'est bien la clef publique derrière adr_B qui utilise ces fonds. On vérifiera donc que $h(K^E_{Pub}) == adr_B$ ce qui n'est pas le cas. Si Eve laissait la clef K^B_{Pub} dans le deuxième champs alors c'est la signature qui serait invalide cette fois, puisqu'elle n'est pas faite avec la clef privée correspondant à K^B_{Pub}. Eve est donc bloquée, elle ne peut pas détourner la transaction de Bob !

Changer le montant d'une transaction

Cette fois-ci Eve veut essayer de changer le montant de la transaction, le remplacer par 0.5 btc par exemple. Mais cette attaque échouera également car le champs value fait parti de Tx^* et est donc signé avec K^B_{Pub}, on ne peut pas le changer sans rendre la signature invalide. Or on a vu dans l'exemple précédent que l'on ne peut pas changer la signature. Cette attaque échoue donc également.

1.2.4 Une transaction réelle

Les utilisateurs de Bitcoin dispose en réalité d'une multitude de paires clefs privées/clef publiques, donc d'une multitude d'adresse Bitcoin. Cela permet d'effectuer des transactions avec plusieurs inputs et plusieurs outputs. La majorité des transactions Bitcoin sont de ce type. Supposons donc que Bob dispose des clefs privées K^B_{Pri}, K^{B1}_{Pri}, K^{B2}_{Pri} et qu'Alice et Eve aient toujours les même clefs. On suppose de plus qu'il existe dans la blockchain les transactions $Tx1$ et $Tx2$ suivantes :

- Input
  - ⋆ ...
- Output
  - ⋆ value : 1 btc
  - ⋆ adr_{B1}

FIGURE 4 – Transaction $Tx1$

- Input
  - ⋆ ...
- Output
  - ⋆ ...
  - ⋆ value : 0.7 btc
  - ⋆ adr_{B2}

FIGURE 5 – Transaction $Tx2$

La transaction $Tx1$ envoie 1 btc à l'adresse adr_{B1} et elle ne compte qu'un seul output (donc output d'index 0). la transaction $Tx2$ compte plusieurs outputs et son troisième output envoie 0.7 btc à l'adresse adr_{B2} (output d'index 2). Bob peut donc créer la transaction Tx qui utilisent ces deux outputs et attribue ensuite 0.5 btc à Alice, 0.5 btc à Eve et se rend 0.6 btc sur les 0.7 qu'il reste.

- Input
  - ⋆ $h(Tx1)$ idx : 0
  - ⋆ K^{B1}_{Pub} / $sign(K^{B1}_{Pri}, Tx^*)$
  - ⋆ $h(Tx2)$ idx : 2
  - ⋆ K^{B2}_{Pub} / $sign(K^{B2}_{Pri}, Tx^*)$
- Output
  - ⋆ adr_A 0.5 btc
  - ⋆ adr_E 0.5 btc
  - ⋆ adr_B 0.6 btc

FIGURE 6 – Transaction Tx

Pour chaque input dans Tx il faut fournir la clef publique associée à l'adresse à laquelle était destiné l'output utilisé et fournir la signature sur Tx' afin de prouver que l'on possède la clef privée associée. Par exemple pour $h(Tx1)$ idx : 0, comme l'output utilisé est destiné à adr_{B1} on fournit la clef K^{B1}_{Pub} et la signature $sign(K^{B1}_{Pri}, Tx^*)$.

1.3 Blockchain et mineurs

2 Le minage

2.1 La parade au double-spend

3 Les différents types de clients

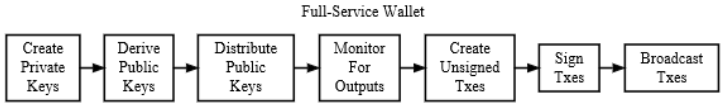
3.1 Découpage des fonctions

Avec ce qu'on a vu auparavant, on peut déclarer que le protocole Bitcoin est sûr pourvu que :
— les clefs secrètes restent secrètes
— on a accès au consensus atteint par le réseau, c'est-à-dire à la *blockchain*
Ces deux conditions sont malheureusement assez antagonistes : l'accès à la *blockchain* suppose une connectivité avec l'extérieur, ce qui est nécessairement un facteur de risque pour la sécurité des clefs.

Cependant, l'architecture de Bitcoin rend possible de séparer les différentes fonctions d'un portefeuille, et en particulier d'isoler la partie signature. De façon intéressante, cette séparation peut même se faire entre différents acteurs, le détenteur d'une adresse délégant une partie de l'ensemble des fonctions d'un portefeuille Bitcoin à une autre entité.

Dans cette section, nous exposons les différents types de clients Bitcoin qui existent à ce jour (en nous référant à des implémentations existantes) et proposons une première analyse de leurs avantages (en termes de simplicité ou de légèreté) et des changements, en bien ou en mal, qu'ils induisent en termes de sécurité.

3.2 Full clients



Ce genre de clients implémente la totalité du protocole Bitcoin, et garde en mémoire une copie de la totalité de la *blockchain*. Leurs fonctionnalités peuvent se diviser en deux parties :

- *publique*
  - Découvrir d'autres nœuds et communiquer avec eux
  - envoyer et recevoir des transactions et des blocs
  - enregistrer tous les blocs valides
  - vérifier toutes les transactions reçues et *broadcaster* toutes celles qui sont valides
- pour l'utilisateur
  - stocker son historique de transactions et les clefs privées pour le "portefeuille"
  - offrir une interface (GUI ou ligne de commande) pour afficher les avoirs et l'historique des transactions, et générer de nouvelles transactions

Ces clients renforcent la solidité du réseau et le rendent moins vulnérable aux attaques, mais sont coûteux en termes de mémoire, d'opérations (ils vérifient la validité de transactions qui ne concernent pas l'utilisateur) et de frais liés au réseau (ils reçoivent et émettent des transactions et des blocs en permanence).

En termes de sécurité, l'appareil sur lequel le client est implémenté étant connecté à l'extérieur (et échangeant de plus un volume considérable d'informations), une première mesure évidente est de chiffrer le dossier contenant les clefs privées.

3.3 Headers-only clients

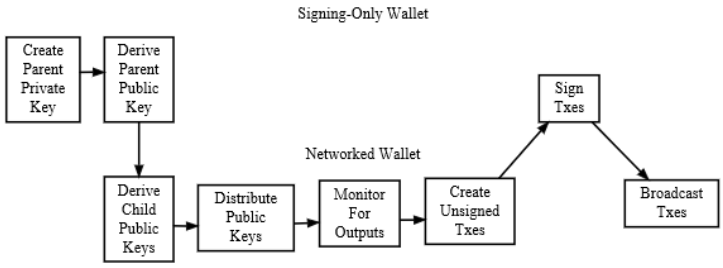
Pour les appareils limités en mémoire, ou selon la volonté de l'utilisateur, un deuxième type de client existe, qui ne garde pas en mémoire toute la *blockchain* mais uniquement les *headers* des blocs (ce qui en mai 2012 représentait 14 Mo, à comparer avec les 2 Go de la totalité de la chaîne). Ils ne téléchargent la totalité d'un bloc que s'ils attendent une transaction, ou s'ils cherchent dans l'ensemble de la chaîne des transactions liées à des clefs qu'ils détiennent. Pour vérifier la validité d'une transaction les concernant, les nœuds utilisant ce client allégé se reposent sur la procédure *Simplified Payment Verification*, décrite dès le papier fondateur. Dans la suite, nous décrivons le modèle utilisé par *bitcoinj*.

Cette procédure simplifiée repose sur les nœuds mettant en œuvre la totalité du protocole (i.e. ayant en mémoire la totalité de la chaîne). Si le client bitcoinj se connecte à l'un d'entre eux, il ne sera informé d'une transaction que si celle-ci est jugée valide par ce nœud. Une fois cette transaction incorporée dans un nouveau bloc (ce que le client bitcoinj peut vérifier s'il récupère ce bloc et contrôle sa validité), la profondeur de cette transaction dans la chaîne passe à 1, et le client bitcoinj ne récupère plus que les *headers* des blocs suivants (qui à chaque fois contiennent une référence au bloc précédent), augmentant à chaque fois la profondeur de la transaction qui l'intéresse de 1. Il considère la transaction comme sûre une fois qu'elle est à une profondeur suffisante dans la chaîne (la valeur typiquement retenue étant de 6), c'est à dire une fois qu'il estime suffisamment peu probable que cette branche de la chaîne ne soit pas la branche finalement retenue.

On voit que tout repose sur la confiance que le client léger accorde au nœud auquel il se connecte : la seule raison que l'on a de considérer une transaction comme légitime (i.e. celui qui paie est bien le propriétaire de ces bitcoins) - avant même les considérations de profondeur - est le fait qu'elle a été relayée, ce que n'aurait pas fait un nœud complet honnête si la transaction n'était pas légitime. Pour s'assurer d'être connecté à des nœuds honnêtes, le client bitcoinj se connecte à des nœuds choisis aléatoirement [1], ce qui rend difficile pour un attaquant de contrôler la connectivité du client léger pour essayer de l'induire en erreur. Une fois que la transaction a été relayée par une fraction importante de nos pairs, on peut être assez sûr qu'elle se propage bien à travers tout le réseau et donc qu'elle est très probablement valide.

L'attaque principale liée à cette question de confiance (appelée *Sybil attack*) a lieu si l'attaquant a la possibilité de contrôler entièrement notre connexion à Internet, et ainsi de nous présenter un faux réseau dans lequel tous les nœuds acceptent cette transaction pour nous faire croire qu'elle est valide, puis d'ajouter des blocs pour finir par nous faire croire qu'elle est définitive. Pour contrecarrer cela, il est envisagé de mettre en place des clients qui mesurent la vitesse à laquelle de nouveaux blocs sont minés : en effet, dans le cas où l'attaquant n'est pas dans une configuration de puissance de type 51%, cette vitesse décroîtrait brutalement avec l'usurpation, et le client pourrait ainsi signaler à l'utilisateur que la situation est douteuse.

3.4 Signing-only clients



Les *signing-only clients* vont encore une étape plus loin dans la délégation de compétence : le client ne télécharge pas la blockchain, ni même les headers. Il se base sur un autre client pour lui fournir les transactions qui concernent l'utilisateur et la situation financière de ce dernier, et ne garde en propre que la partie signature. Ce cas de figure a deux fonctions principales :

— sécuriser les clefs : un utilisateur peut être prêt à installer sur un de ses appareils un client Bitcoin complet, mais estimer que conserver ses clefs sur un appareil connecté à l'extérieur

1. Il se connecte à un serveur DNS qui lui fournit une souche aléatoire qui varie tous les jours

n'est pas raisonnable ; il découpe ainsi son client en deux parties, l'une participant au réseau et lui indiquant notamment les transactions qu'il reçoit, vérifiant leur validité et leur inclusion dans la chaîne,... et l'autre, sur un appareil distinct et isolé de l'extérieur, contenant ses clefs privées et étant donc la seule à pouvoir dépenser ses bitcoins. Il y a plusieurs possibilités pour implémenter cette séparation :
  - Faire des transferts avec par exemple une clef USB entre les deux équipements, ce qui est contraignant
  - Acheter une implémentation physique de cette partie signature, par exemple un appareil doté d'une connexion USB, sur lequel peut se trouver un dispositif de type code PIN,... pour plus de sécurité.
- confier à un serveur le soin de suivre les transactions nous concernant et de nous indiquer nos avoirs. Cela suppose un certain niveau de confiance dans le serveur, qui peut nous induire en erreur en nous indiquant de fausses transactions que l'on aurait reçues, nous trompant ainsi sur la quantité de bitcoins que nous détenons. Ce genre d'attaque est d'une importance limitée, puisque le serveur ne peut en aucun cas dépenser lui-même les bitcoins que nous possédons (nous faire signer des transactions contre notre volonté). Il est de plus possible d'éviter cette attaque en nous connectant à plusieurs serveurs indépendants, ou même à notre propre serveur [2].

Le client peut être une application PC, mobile ou web ; dans le cas d'une application Web, la signature est faite dans le browser, et les clefs ne sont donc pas envoyées.

3.5 Thin clients

Mt.Gox,... Le client ne détient pas ses clefs et ne signe pas lui-même ses transactions, il demande à un serveur distant de faire cela pour lui. En ce sens, c'est exactement la même chose qu'une banque et toute la motivation de base du Bitcoin - se passer de tiers-partie - disparaît. Il convient aussi de souligner que ces acteurs sont loin du niveau de garantie offert par les banques.

4 Sécurité des clefs

4.1 Appareils connectés à l'extérieur

4.2 Perte/casse/vol

4.3 Brain wallets

4.4 Une solution :m-of-n signature

Concept beaucoup plus général et intéressant (arbitres,...), mais qui peut être mis à profit ici.

Références

[1] Bitcoin stackexchange. `http://bitcoin.stackexchange.com`.

[2] Bitcoin wiki. `https://en.bitcoin.it/wiki`.

2. Il existe des logiciels open source permettant de réaliser ce genre de serveurs. L'intérêt de cette démarche est de pouvoir utiliser un client complet sur un appareil contraint (type mobile), en utilisant un serveur externe pour réaliser la partie la plus exigeante.