

# Bitcoin

## Éléments de compréhension technique

Jean-Luc Parouty  
Institut de Biologie Structurale (IBS)  
*Jean-Luc.Parouty<at>ibs.fr*





### Résumé

Ce document est un complément technique à l'article « Bitcoin, Principes généraux, limites et perspectives », disponible sur [docproof.org](http://docproof.org).

Si une compréhension globale de l'architecture Bitcoin est amplement suffisante pour une utilisation normale, une connaissance plus fine devient rapidement nécessaire dès lors que l'on souhaite mettre en œuvre ou intégrer certaines fonctionnalités.

Cette partie est ainsi destinée à toutes celles et à ceux qui souhaitent comprendre et approfondir l'architecture interne de Bitcoin.

### Licence

Creative Commons BY-NC-SA -    

### Mots-clefs

Bitcoin, cryptographie, crypto-monnaie, preuve, blockchain, p2p, minage, ECDSA, SHA256, Base58Check, RIPMD160

### Table des matières

1/ Boîte à outils .....	2
1.1 Interrogation par API .....	2
1.2 Posséder son propre nœud Bitcoin .....	2
1.3 Bitcoin en mode « bac à sable » .....	2
1.4 Utilitaires divers .....	3
2/ La cryptographie dans Bitcoin .....	4
2.1 Fonctions de Hachage .....	4
2.2 Signature électronique .....	5
2.3 Paradoxe de la signature Bitcoin .....	5
3/ Clefs privées, publiques et adresses .....	7
3.1 Construction des adresses .....	7
3.2 Formats d'encodage .....	8
4/ Locks scripts (scriptPubkey) .....	9
4.1 « Script » langage des transactions Bitcoin .....	9
4.2 Scripts « standards » .....	10
5/ Unlock scripts (scriptSig) .....	14
6/ Preuve par le travail .....	16
6.1 Principe .....	16
6.2 Evolution de la difficulté .....	16
7/ Simple Verification Payment (SPV) .....	17
7.1 Appartenance d'un bloc à la chaîne .....	17
7.2 Vérification de la présence d'une transaction au sein d'un block .....	17
8/ Normalisation et organisation .....	19
8.1 Standards Track .....	19
8.2 Informational BIPs .....	19
8.3 Process BIPs .....	19
8.4 La difficile quête du consensus .....	20
8.5 Documentation et sources d'information .....	20
Bibliographie .....	21

## 1/ Boîte à outils

Avant de se plonger plus avant, il est intéressant de passer en revue quelques outils que nous allons utiliser.

### 1.1 Interrogation par API

La blockchain étant publique, il est possible d'accéder à celle-ci via un certain nombre de sites qui proposent des **API** librement accessibles. Cette manière de faire est de loin la plus simple.

Exemple de requêtes pour accéder à la transaction d'Alice :

	API Blockchain.info	API Blockexplorer.com
Lisible humain.	<a href="https://blockchain.info/tx/b3(...).de">https://blockchain.info/tx/b3(...).de</a>	<a href="https://blockexplorer.com/tx/b3(...).de">https://blockexplorer.com/tx/b3(...).de</a>
Détaillée (JSON)	<a href="https://blockchain.info/rawtx/b3(...).de?format=json">https://blockchain.info/rawtx/b3(...).de?format=json</a>	<a href="https://blockexplorer.com/api/tx/b3(...).de">https://blockexplorer.com/api/tx/b3(...).de</a>
Complète (Hexa)	<a href="https://blockchain.info/rawtx/b3(...).de?format=hex">https://blockchain.info/rawtx/b3(...).de?format=hex</a>	<a href="https://blockexplorer.com/api/rawtx/b36e7de(...).de">https://blockexplorer.com/api/rawtx/b36e7de(...).de</a>
Description des API	<a href="https://blockchain.info/fr/api/blockchain_api">https://blockchain.info/fr/api/blockchain_api</a>	<a href="https://blockexplorer.com/api">https://blockexplorer.com/api</a>

### 1.2 Posséder son propre nœud Bitcoin

Se doter de son **propre nœud Bitcoin**, avec une **instance locale de la blockchain** est certainement la solution la plus intéressante, mais elle peut également être assez contraignante, compte-tenu des ressources nécessaires :

- **L'espace nécessaire** à l'hébergement de la blockchain peut être important en fonction des environnements. Il est possible d'estimer cet espace à partir des statistiques de *blockchain.info* [1], auxquelles il faut rajouter au moins 15%, pour les bases d'index.
- **L'intégration réseau et la bande passante nécessaire.**  
Un nœud Bitcoin, participant au réseau *p2p Bitcoin*, a besoin d'être accessible depuis l'Internet. Sa localisation réseau doit donc être en DMZ.

Le protocole de récupération de la blockchain consiste à récupérer et vérifier chaque bloc et chaque transaction, le temps de construction peut donc être relativement long (surtout si l'on est pressé !). Compter plusieurs jours.

Cela tant, la mise en œuvre de l'implémentation de référence (Bitcoin Core) est relativement simple et bien documentée [2] et ne présente pas de difficultés particulières.

### 1.3 Bitcoin en mode « bac à sable »

Utiliser un véritable nœud Bitcoin pose un autre problème non négligeable : celui de devoir faire des tests avec un véritable réseau de transaction et de vrais bitcoins...

Afin de permettre le développement d'applications deux solutions sont proposées :

- Utilisation du **réseau testnet** [3]  
Ce réseau est une alternative au réseau Bitcoin, destiné aux tests et développement,
- Travailler en **mode regtest** [4] (*Bitcoin Core's regression test mode*)  
Ce mode permet de faire fonctionner un nœud Bitcoin en mode autonome, déconnecté du réseau Bitcoin. Ce mode est parfaitement adapté aux tests et développements dès lors que les interactions avec le réseau Bitcoin ne sont pas nécessaires.

Pour effectuer des développements ou effectuer des tests, le mode *regtest* est généralement le plus intéressant. Il ne demande pas de ressources particulières, est extrêmement simple à mettre en œuvre... et ne présente aucun risque pour ses « vrais » bitcoins !

## 1.4 Utilitaires divers

Interroger le réseau Bitcoin, calculer des hachages, visualiser des transactions, etc. peut se faire à travers divers outils et API, notamment :

- Les **commandes en ligne** et **RPC** de **Bitcoin Core** [5], qui permettent d'accéder à l'ensemble des fonctionnalités du nœud (interrogation de la blockchain, gestion du portefeuille, des adresses, transactions, etc.)
- **Bitcoin Explorer**, alias **bx** [6], qui offre, via une interface en mode ligne de commande et sa librairie *libbitcoin*, la possibilité de développer des applications complètes (gestion de portefeuilles, transactions, etc.) ainsi que d'effectuer tous les calculs usuels possibles (hachage, encodage, etc.).

## 2/ La cryptographie dans Bitcoin

La cryptographie est omniprésente dans l'architecture Bitcoin et deux grandes familles de fonctions cryptographiques sont massivement utilisées ; les fonctions de hachage et la signature électronique.

### 2.1 Fonctions de Hachage

Le principe d'une fonction de hachage, telle que SHA-256, est de générer une « empreinte » à partir des données fournies en entrées. L'empreinte étant caractéristique de ces données et de taille réduite et fixe.

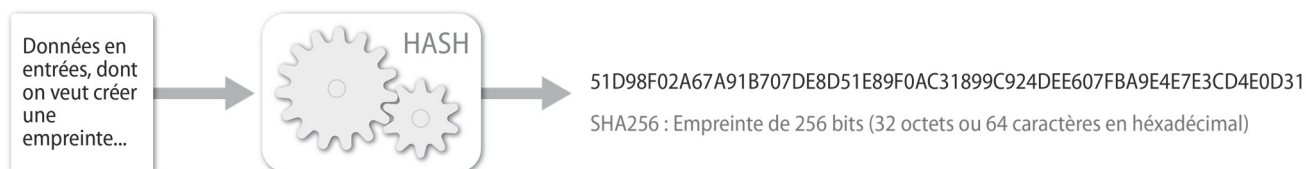


Figure 1 - Fonction de hachage SHA256

Ces fonctions possèdent deux caractéristiques essentielles :

- Toute modification des données initiales induira une empreinte différente,
- Construire un document susceptible de fournir une empreinte donnée est « réputé extrêmement difficile ».

Une empreinte peut donc être tout à la fois un **identifiant**, une **garantie d'intégrité**, une **preuve d'existence** ou encore servir

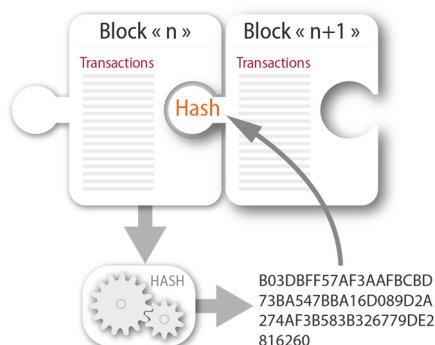


Figure 2 - Principe du chaînage

de fonction de base dans la réalisation des **preuves de travail**. Bitcoin va exploiter l'ensemble de ces possibilités.

Les empreintes sont également utilisées pour constituer des chaînes infalsifiables telles que la blockchain ou les arbres de Merkle. L'empreinte d'un bloc n est intégrée dans le bloc n+1, rendant impossible toute modification du bloc n, sans devoir modifier également le bloc n+1, puis n+2, n+3, etc.

Deux fonctions de hachage sont massivement utilisées dans l'architecture Bitcoin : SHA-256 [7] et RIPEMD-160 [8].

## 2.2 Signature électronique

Une signature électronique permet de vérifier deux choses fondamentales :

- L'intégrité du document signé,
- L'identité du signataire

Le principe repose sur un système à double clef, l'une « privée », restera propriété exclusive du signataire, l'autre « publique » sera utilisée pour vérifier la signature :

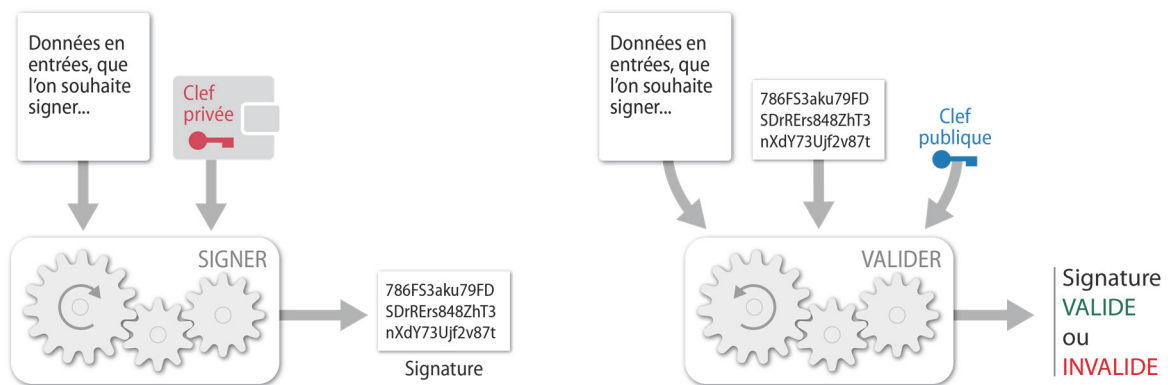


Figure 3 - Signature électronique classique

L'algorithme utilisé dans Bitcoin pour la signature électronique est **ECDSA** (*Elliptic Curve Digital Signature Algorithm*) [9] [10], dont la robustesse repose sur la difficulté de calculer le logarithme discret d'un grand nombre entier.

Connaissant  $a = b^n$  retrouver  $n = \log_b a$ , le logarithme de  $a$  en base  $b$ , est réputé extrêmement difficile.

## 2.3 Paradoxe de la signature Bitcoin

Dans l'architecture Bitcoin, la signature électronique est très souvent utilisée pour prouver que l'on est propriétaire d'une adresse. Pour être plus précis : que l'on possède bien la clef privée associée à une adresse.

La signature est bien effectuée avec la clef privée, mais la validation se fait avec **l'adresse** :

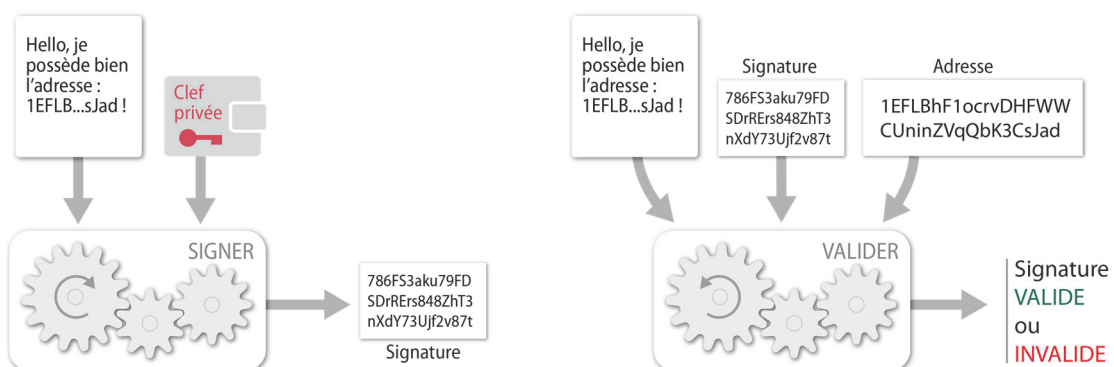


Figure 4 - Figure 1 - Vérification paradoxale de la signature Bitcoin

Or une **adresse Bitcoin**, comme nous allons le voir par la suite, n'est que le **hachage de la clef publique** !

Pour aussi paradoxal que cela puisse paraître, l'explication est en fait totalement rationnelle et est liée à l'algorithme ECDSA, qui permet de **retrouver la clef publique à partir des éléments de la signature** [10](page 47). Il n'y a pas de magie, juste des mathématiques...

Exemple de signature avec la commande *bx* :

- Ma clef privée est : L3uwZmP2RqxAG3bmQfgrAsS5ndnpEbm6pa5iw1TsnfLc49iECqW
- Ma clef publique est : 030cc1e32ce146bdc2dc4727d35ffe52c3dda78ee0ead9bc490c42e260fa57060c
- L'adresse associée est : 1EFLBhF1ocrvDHFWWCUninZVqQbK3CsJad
- Mon message est : "Bob Watson est toujours là !"

Signature du message :

```
# bx message-sign L3uwZmP2RqxAG3bmQfgrAsS5ndnpEbm6pa5iw1TsnfLc49iECqW "Bob Watson est toujours là !"
IBQBhppFGANaSo0BC+BeJcm9VXVM0pMZ07rLhEDCoZzQC64p889jBR1e6je/admNZEfDQaQziul9vs9niy30riU=
```

Vérification de la signature :

```
# bx message-validate
1EFLBhF1ocrvDHFWWCUninZVqQbK3CsJad
IBQBhppFGANaSo0BC+BeJcm9VXVM0pMZ07rLhEDCoZzQC64p889jBR1e6je/admNZEfDQaQziul9vs9niy30riU=
"Bob Watson est toujours là !"
The signature is valid.
```

### 3/ Clefs privées, publiques et adresses

Les adresses sont au cœur de l'architecture Bitcoin. La quasi-totalité des transactions consiste à transférer des bitcoins depuis des adresses sources (*inputs*) vers des adresses destination (*outputs*).

Une adresse bitcoin est dérivée d'une clef publique ECDSA. Lorsqu'une adresse est demandée au « portefeuille », celui-ci va commencer par générer un couple de clef, publique et privée :

- La clef privée est un nombre aléatoire de 256 bits,
- La clef publique est dérivée de la clef privée.

#### 3.1 Construction des adresses

Une fois la clef publique disponible, l'adresse va pouvoir être générée. La construction d'une adresse comprend deux phases, le

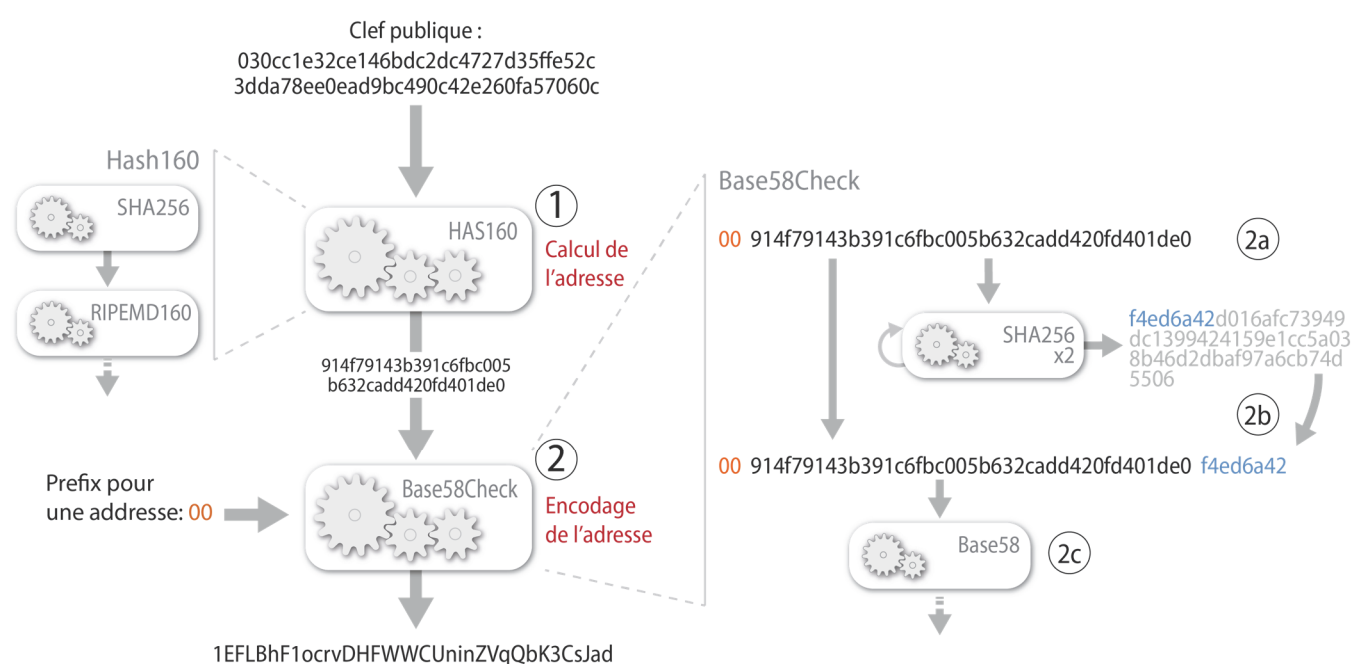


Figure 5 - Construction d'une adresse Bitcoin

calcul de l'adresse proprement dite, puis l'encodage de celle-ci :

- (1) **Calcul de l'adresse** : La clef publique est hachée une première fois avec un **SHA256**, puis une seconde fois avec un **RIPEMD160**. Ce double hachage est usuellement appelé **HASH160**.
- (2) **Encodage** : Le résultat est ensuite encodé avec le format **Base58Check**, consistant à ajouter un **préfix** un **checksum**. Le préfix permet de spécifier le type d'objet et le checksum de limiter les risques d'erreur. Des bitcoins enregistrés avec une adresse erronée seraient perdus.
  - (2a) Le préfix 0x00, correspondant à un objet de type adresse, est ajouté à l'empreinte issue du HASH160,
  - (2b) Le résultat est haché 2 fois avec un **SHA256** et les 4 premiers octets de l'empreinte obtenue sont ajoutés,
  - (2c) Le résultat : préfix + HASH160(clef publique) + checksum est finalement encodé en **base58**

Base58 est inspiré de base64 et reprend le même principe, avec une base restreinte de caractères afin de limiter les risques de confusion. Ont été retirés : 0 (zéro), O (o majuscule), l (L minuscule), I (i majuscule), +, / et \.

À noter que le préfix 0 est devenu un 1 à l'issue de l'encodage base58.

## 3.2 Formats d'encodage

Les objets cryptographiques de Bitcoin ; adresses, clefs publiques ou privées, peuvent être **encodés** de trois façons :

- Hexadécimal
- WIF (*Wallet Import Format*)
- WIF-Compressed

L'encodage WIF permet les exportations et importations avec les portefeuilles. Il correspond à un encodage **Base58Check**. L'encodage WIF-Compressed est utilisé pour les clefs publiques.

Le préfix change lors de l'encodage : Adresse 0x00 devient 1, clef privée 0x08 devient 5, K ou L, etc.

**Exemple de génération de clef et d'encodage** avec Bitcoin Explorer [6].

Génération d'une clef privée :

```
# bx seed
4673a170206edfe6656c0343ed2c7023
# bx ec-new 4673a170206edfe6656c0343ed2c7023
8a6723c3436ec4d5f49fc729e70a566bbaf52c57b7f24eff5dd0c503f1b0d15b
```

Encodage WIF et WIF-Compressed de la clef privée :

```
# ec-to-wif --uncompressed 8a6723c3436ec4d5f49fc729e70a566bbaf52c57b7f24eff5dd0c503f1b0d15b
5JsEu1f8TwwVGKcNBfNKwRoxvZxGW6RBPmJiFpLK7s4fs3cTviR
# bx ec-to-wif 8a6723c3436ec4d5f49fc729e70a566bbaf52c57b7f24eff5dd0c503f1b0d15b
L1rkLZuHwwVVkd96W4M8bEWHqSSwRQuisUzB6PvEyHyCYWwS76uy
```

Génération de la clef publique associée :

```
# bx ec-to-public 8a6723c3436ec4d5f49fc729e70a566bbaf52c57b7f24eff5dd0c503f1b0d15b
0369fd6ffbd1c2130c56bf2a0ec9a0ddad319c8b6a4b40afc4f81ce230f2960d9a
```

Génération de l'adresse associée :

```
# bx ec-to-address 0369fd6ffbd1c2130c56bf2a0ec9a0ddad319c8b6a4b40afc4f81ce230f2960d9a
1FmFARQBTHb97Kouw73KhpPThGmvXUXaRX
```

Décodage d'une adresse :

```
# bx address-decode 1FmFARQBTHb97Kouw73KhpPThGmvXUXaRX
wrapper
{
  checksum 3984863329
  payload a1efe2af6a61f745fe432a9acc3816cd400fec2b
  version 0
}
```



## 4/ Locks scripts (scriptPubkey)

Nous avons vu que les bitcoins étaient enregistrés dans la blockchain avec des **verrous** et qu'ils ne pouvaient être dépensés que si l'on déverrouillait ces derniers.

Ces verrous ne sont pas de simples conditions cryptographiques passives, mais de véritables **scripts programmables**, offrant une souplesse potentiellement illimitée dans l'élaboration des transactions.

Ce mécanisme de verrous est l'une des grandes richesses de Bitcoin.

### 4.1 « Script » langage des transactions Bitcoin

Le langage utilisé dans les *outputs* des transactions Bitcoin est un langage minimaliste, basé sur une pile et qui ne comporte pas de boucles [11]. Il n'est donc **pas Turing-complet**.

L'interpréteur suit les règles suivantes :

- La lecture du script est effectuée de la gauche vers la droite,
- Les constantes sont empilées lorsqu'elles sont rencontrées,
- Les instructions (OPerators) peuvent empiler ou dépiler un ou plusieurs paramètres, les traiter et empiler un résultat,
- Les opérateurs logiques évaluent une condition et empilent le résultat : 1=TRUE ou 0=FALSE,

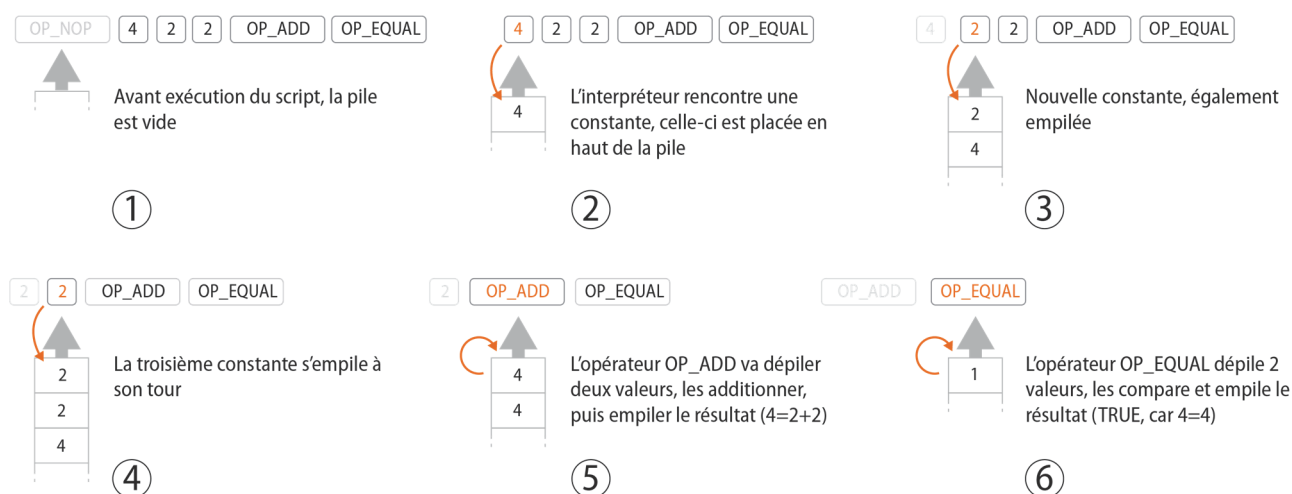


Figure 6- Exemple de Script Bitcoin

L'exemple suivant effectue une addition  $2 + 2$  et compare le résultat à `4` :

Dans la gestion des verrous Bitcoin, deux scripts sont utilisés :

- L'un pour **verrouiller** (*scriptPubkey*),
- L'autre pour effectuer le **déverrouillage** (*scriptSig*)

Lors de la phase de validation, l'interpréteur commencera par exécuter le script de déverrouillage (*scriptSig*), puis ensuite le script de verrouillages (*scriptPubkey*). A l'issue de l'exécution le haut de la pile doit être à **TRUE** pour que la dépense soit **autorisée**.

## 4.2 Scripts « standards »

Les possibilités offertes par ce mécanisme de scripts sont potentiellement sans limites. Pour des questions de sécurité, un certain nombre de « garde-fous » ont été intégrés et seuls **5 types de scripts sont acceptés** par l'implémentation de références (**Bitcoin Core**).

Cette limitation n'étant qu'un choix d'implémentation, certains mineurs peuvent librement déroger à cette restriction.

Ces cinq scripts « **standards** » sont les suivants :

<i>Scripts</i>	<i>Description</i>
<b>P2PKH</b>	<p>Pay-to-Public-Key-Hash ou paiement contre le hash d'une clef publique (adresse). Ces scripts sont aujourd'hui utilisés dans la majorité des transactions.</p> <p>Loc script :      OP_DUP OP_HASH160 &lt;Address&gt; OP_EQUALVERIFY OP_CHECKSIG Unlock script :    &lt;signature&gt; &lt;Public Key&gt;</p>
<b>P2PK</b>	<p>Pay-to-Public-Key ou Paiement contre une clef publique Version historique et plus simple du script P2PKH, mais qui nécessite davantage de place. En désuétude.</p> <p>Loc script :      &lt;Pub Key&gt; OP_CHECKSIG Unlock script :    &lt;signature&gt;</p>
<b>Multi-Signature</b>	<p>Permet de définir une condition telle que M signatures doivent être présentées sur N <i>Exemple</i> : Une sortie est protégée par 5 clefs publiques mais ne pourra être dépensée que sur présentation de 2 des 5 signatures.</p> <p>Loc script :      M &lt;public Key 1&gt; &lt;public Key 2&gt; ... &lt;public Key N&gt; N OP_CHECKMULTISIG Unlock script :    0 &lt;signature1&gt; ... &lt;signature M&gt;</p>
<b>P2SH</b>	<p>Pay-to-Script-Hash ou paiement contre le hash d'un ...script de rachat. Le script de verrouillage n'est plus enregistré dans la sortie, mais devra être envoyé lors du déverrouillage. Seule l'empreinte du script est fournie lors de la construction de la transaction, laquelle pourra être encodée dans une adresse de type P2SH [12].</p> <p>Loc script :      OP_HASH160 &lt;script Hash&gt; OP_EQUAL Unlock script :    &lt;script arguments&gt; &lt;script&gt;</p>
<b>OP_RETURN</b>	<p>L'opérateur OP-RETURN permet d'intégrer des données dans une transaction et <i>in fine</i> dans la blockchain. Une sortie comportant une instruction OP_RETURN ne peut être dépensée. La taille des données est actuellement limitée à 40 octets. Cet opérateur permet d'utiliser la blockchain à des fins de notariat électronique.</p> <p>Lock script :      OP_RETURN &lt;data&gt; Unlock script :    aucun</p>

Afin d'illustrer le fonctionnement de ces scripts, nous allons détailler l'exécution du script P2PKH, majoritairement utilisé, et OP\_RETURN, qui permet la mise en œuvre de fonction de notariat électronique.

#### 4.2.1 « Pay-to-Public-Key-Hash » (P2PKH)

L'écrasante majorité des transactions utilise cette forme de script, dont le déverrouillage requiert la présentation d'une clef publique et d'une signature effectuée avec la clef privée correspondante.

Si l'on regarde la transaction avec laquelle Alice a payé son café :

txid : b36e7dedc141675e09e2b3b6ad05a016cb2a90f540713548ca102070de74c2de  
Visible via : [https://blockexplorer.com/api/tx/b36e7ded\(...\)de74c2de](https://blockexplorer.com/api/tx/b36e7ded(...)de74c2de)

On peut voir que la sortie vers l'adresse de Bob comporte le script de verrouillage suivant :

**scriptPubkey :**

OP\_DUP OP\_HASH160 39ee7f7d4c80307d27e7657cbfef0d48d0eb84e5 OP\_EQUALVERIFY OP\_CHECKSIG

Où :

39ee7f7d4c80307d27e7657cbfef0d48d0eb84e5 (hexa)

Correspond à l'adresse de Bob :

16HKE4qxjfnVpWCpe6DL2rSZVTkpZU27UL (WIF, Base58check)

Le script de verrouillage (scriptPubkey) pourra être déverrouillé avec le scriptSig suivant :

**scriptSig :**

<signature de Bob> <clef publique de Bob>

L'exécution consécutive des deux scripts donne le résultat suivant :

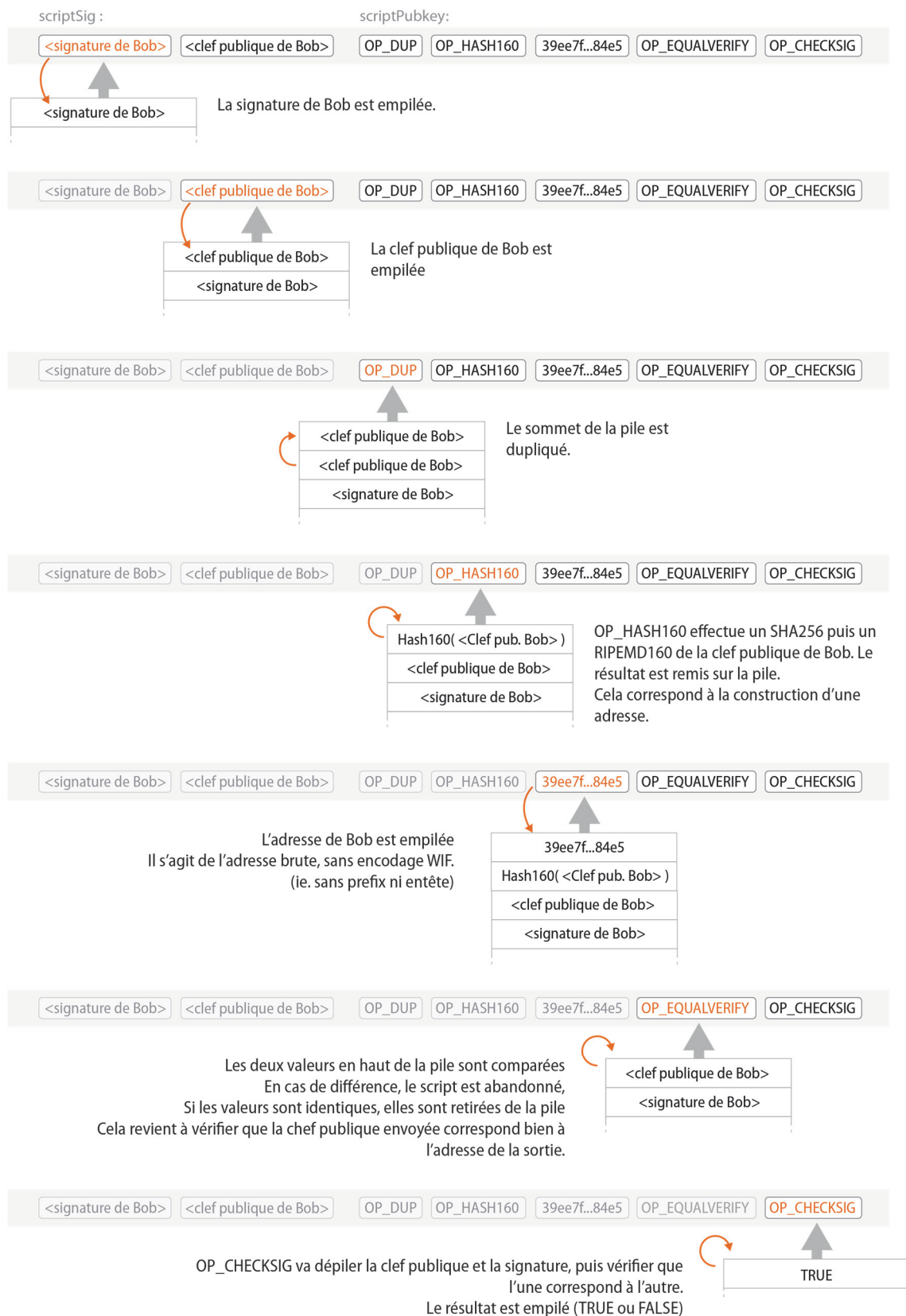


Figure 7 - Exécution d'un script P2PKH

Si à l'issue de l'exécution le résultat est TRUE, la sortie (UTXO) sera déverrouillée. Dans le cas contraire (FALSE), la sortie ne pouvant être dépensée et la transaction sera rejetée.

#### 4.2.2 OP\_RETURN

Cet opérateur permet d'intégrer des données au sein d'une transaction et donc de la blockchain.

L'usage classique consiste à créer une sortie de montant nul (0 BTC), ayant pour *scriptPubkey* :

```
OP_RETURN <data length> <data>
```

Exemple :

```
OP_RETURN 28 444f4350524f4f46 55f86e0bf0c27068b068da7bfb9240e7784c7d8e12ba12153aabe3b36cf4136a
```

Extrait la transaction :

```
8e77d41c93f53f20143cca0f72d652c5edf8c0f6893821191353c0c84c6ff0bf
```

Où :

- 28 : Est le nombre d'octets des données à insérer (40 en décimal)
- 444f4350524f4f46 : La chaîne de caractère « DOCPROOF »
- 55f86e0b...6cf4136a : Les données à insérer (hash d'un document)

Cette possibilité, exploitée par [proofofexistence.com](https://proofofexistence.com) et [docproof.org](https://docproof.org), permet d'insérer un hash dans la blockchain, offrant ainsi au document haché une preuve de son existence.

Un certain nombre de limitations sont imposées aux scripts utilisant cet opérateur :

- 40 octets maximum de données,
- Un seul opérateur OP\_RETURN par transaction,
- La sortie comportant l'opérateur OP\_RETURN ne peut être dépensée et ne sera pas indexée dans la base des UTXO (*unspent transaction output*).

## 5/ Unlock scripts (scriptSig)

Le mécanisme de **déverrouillage** est également un **script**, généralement beaucoup plus simple, comportant une ou plusieurs **signatures**.

Le rôle de ces signatures va être double :

- Apporter la **preuve de possession** de la sortie non dépensée (UTXO),
- **Protéger** tout ou partie de la transaction contre d'éventuelles tentatives de modifications.

Rappelons-nous qu'une fois construite, une transaction est librement diffusée à travers le réseau, afin de pouvoir être récupérée par un mineur. Avant son intégration au sein d'un bloc, elle est donc accessible et modifiable par tous.

La portée de ces signatures va permettre de définir la « malléabilité » de la transaction et de définir les types de modifications qu'il sera possible de lui apporter.

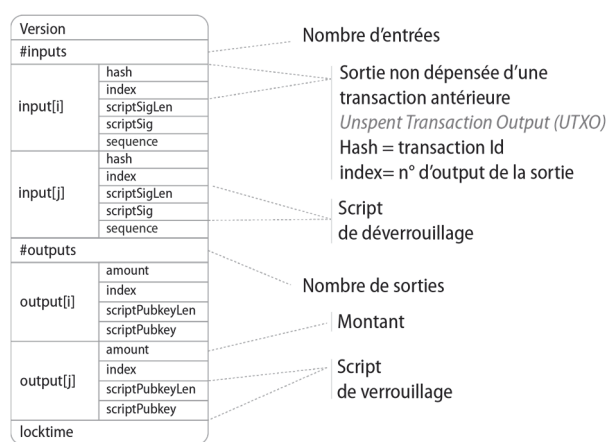


Figure 8 - Structure d'une transaction Bitcoin

Si l'on considère une transaction composée d'un certain nombre d'entrées et de sorties :

Il est possible d'apposer à chacune de ses entrées 3 types de signatures (**ALL**, **NONE**, **SINGLE**), auxquels il peut être ajouté un modificateur (**ANYONECANPAY**).

Cela donne les possibilités suivantes :

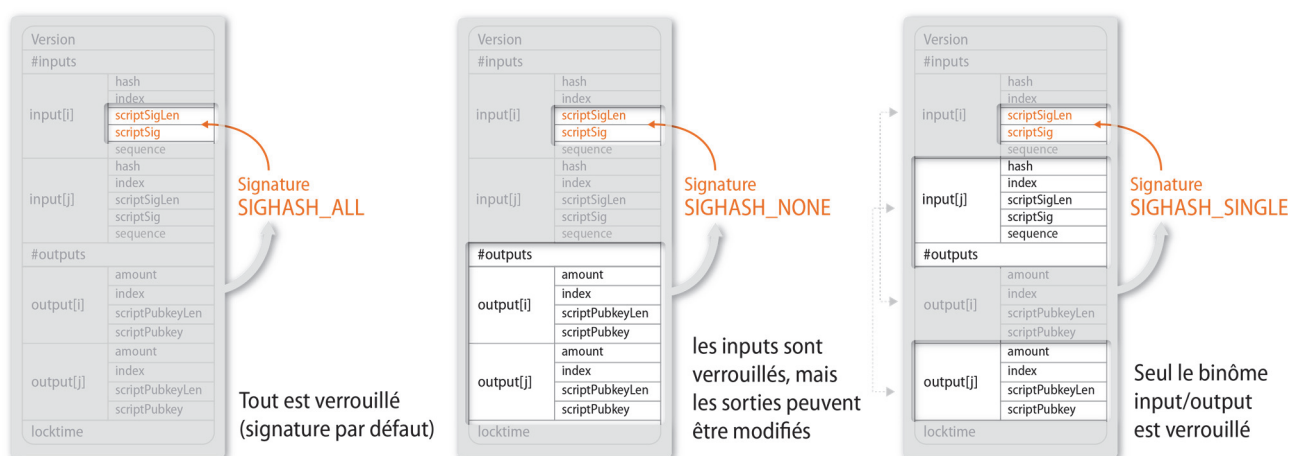


Figure 9 - Signatures possibles (ALL, NONE et SINGLE)

Et avec le modificateur **ANYONECANPAY** (« Chacun peut payer ») :



Figure 10 - Signatures avec modificateur ANYONECANPAY

Les possibilités offertes par l'utilisation de ces scripts et types de signatures sont innombrables.

Il est ainsi possible de réaliser des transactions de type collecte (**SIGHASH\_ALL & SIGHASH\_ANYONECANPAY**), dans laquelle chacun peut ajouter des entrées, mais qui ne pourront être dépensées que vers une destination définie, ou encore, des transactions « en blanc » (**SIGHASH\_NONE**), susceptibles d'être dépensées par tout le monde, etc.

## 6/ Preuve par le travail

Un autre point qui mérite quelques explications techniques est le mécanisme de preuve par le travail.

Le principe est relativement simple : Afin de rendre extrêmement difficile toute réécriture, même partielle, de la blockchain par un nœud malhonnête, on va faire en sorte que cette réécriture soit hors de sa portée en la rendant trop coûteuse à réaliser.

### 6.1 Principe

Pour être accepté par l'ensemble des nœuds, un bloc va devoir posséder une empreinte particulière, difficile à obtenir.

Cette empreinte devra, par exemple avoir une valeur inférieure à un certain seuil, que nous appellerons « difficulté ».

#### Prenons un exemple :

Considérons le document suivant :

document = « Ceci est mon document ! »

Si nous fixons notre difficulté à :

difficulty = 0000ff

Le principe de preuve par le travail va consister à trouver une « tare » (*nonce* en anglais), telle que :

SHA256( texte + tare ) < difficulty

Autrement dit, de trouver une chaîne qui, ajoutée à notre texte, permettra d'obtenir une empreinte commençant par 000.

Compte-tenu des caractéristiques d'une bonne fonction de hachage, la seule méthode possible va consister à essayer un maximum de valeurs possibles pour notre « tare »...

```
# for i in {0..10000};do s=$(echo "$t+$i" | sha256sum -);echo "$s $i"; [ "$(echo $s|cut -b -3)" == "000" ] && break ;done

99bf33aa466e1c01abe273e84cd161ee743aa4b214fffb25847345247008cb60b - 0
19f2d870bd82c481813756df8905b1b9d69a364d7ab3ca30d2b3ea5503b1d647 - 1
b3cc684841c8f93527144821e5c2208dfd2879f607ac3855eb6cd7d1449cabd0 - 2
7703903253c2579da22b1b7e417891f62882d2ce3456d6bf5a7bffc151c16480 - 3
99f58bc59f5b49c4a044e66fec809155a26bcd8532627c1cdf7ce7247485652 - 4
2d4420d4beea97475ce600daed7669fe936f0af50cfc265b6306b64229aff439 - 5
(...)
898ea41193317248a2cf21c12c13364b7c6897d7dca2da99309aedd48090efc6 - 3071
874fcdc16f84c493a1a3f8ab650093901f93b1e6a6f02bc269ef65da1ce056ad - 3072
6ebb67f2e4936e178c36ecc029cd447b6245563efc8cc4c2a91f13a77fb1bf77 - 3073
f07b19296c246bcb2eaaba7a69c53c539f4292532273723b2119e58eb99d3768 - 3074
95c9ed201543e4a5ddbfb917ed9bd49626d5ed94218f94033d35881114dc3395 - 3075
000c4a032ca7d39734759da08c672cf50e1554a3671b5a3ddccd389c44bb5db8 - 3076
#
# echo 'Ceci est mon document !+3076' | sha256sum -
000c4a032ca7d39734759da08c672cf50e1554a3671b5a3ddccd389c44bb5db8 -
```

Effectuer cela en *bash* n'est pas ce qu'il y a de plus efficace (!) et cela n'a pas d'autre intérêt ici que de servir d'illustration ;-)

Aujourd'hui, des matériels dédiés (ASIC) dépassent les 6 TH/s (6.000.000.000.000 hash/s !)

### 6.2 Evolution de la difficulté

La **difficulté** est régulièrement réévaluée [13] de manière à garantir une **durée de construction des blocs à 10 minutes** environs.

Le coût en calcul – et donc en énergie – qui est gaspillé ou investi dans ce mécanisme fait régulièrement l'objet de débats.

Certaines crypto-monnaies proposent des modèles où la preuve par le travail peut être utilisée plus utilement qu'à calculer des SHA256.



## 7/ Simple Verification Payment (SPV)

Les clients légers, ne pouvant disposer de la blockchain en entier, utilisent un protocole spécifique leur permettant de vérifier :

- La présence d'une transaction au sein d'un bloc,
- La présence du bloc au sein de la blockchain.

### 7.1 Appartenance d'un bloc à la chaîne

Les blocs sont composés d'une « entête » et d'une partie « données », comportant parfois plus d'un millier de transactions.

L'essentiel de l'espace est occupé par la partie « donnée », l'entête étant comparativement minuscule.

Un nœud léger ne conservera que l'entête des blocs, sans la partie « données ». À partir de ces entêtes, il est toutefois possible de reconstituer l'enchaînement des blocs et donc de vérifier si un bloc appartient ou non à la blockchain.

### 7.2 Vérification de la présence d'une transaction au sein d'un block

Nous avons vu que les blocs s'enchaînaient les uns aux autres, à la manière des pièces d'un puzzle, où chaque pièce est reliée à une autre par le biais d'une empreinte SHA256.

Un mécanisme analogue est utilisé au sein des blocs pour intégrer les transactions.

Ce mécanisme de chaînage, appelé arbre de Merkle, permet de rattacher chaque transaction à l'entête.

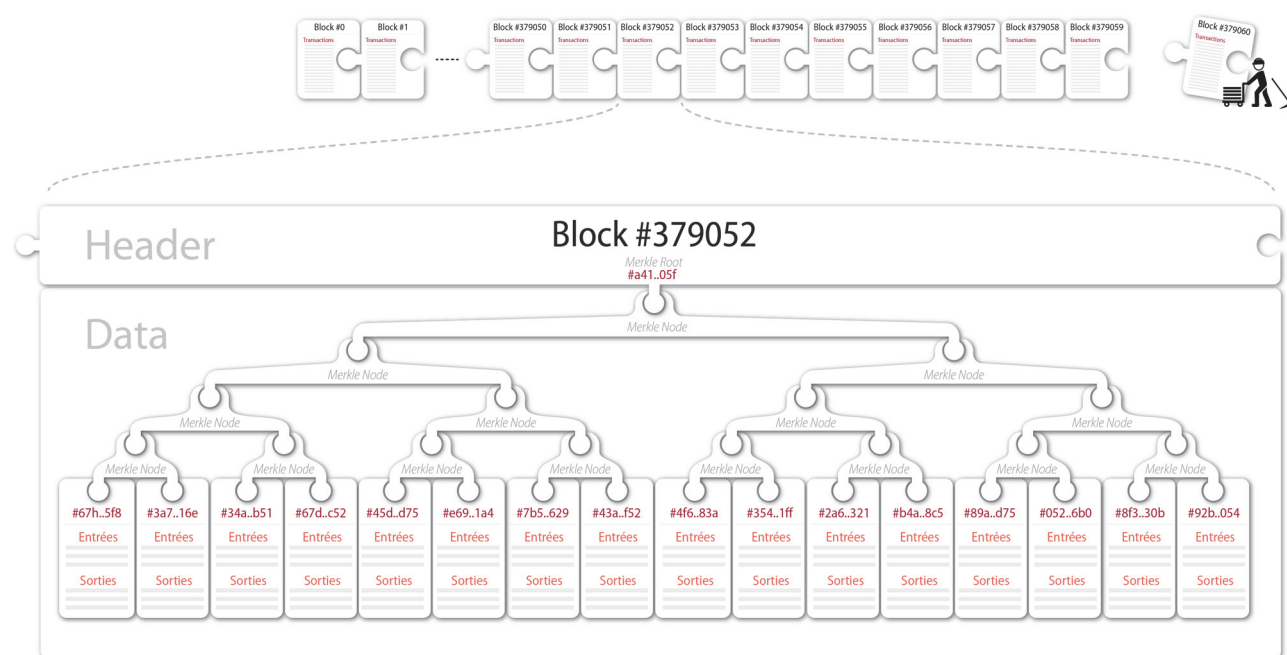


Figure 11 - Structure d'un bloc – Arbre de Merkle

Pour savoir si une transaction appartient ou non à un bloc, le nœud ira interroger le réseau pour récupérer la chaîne de Merkle, allant de l'entête du bloc jusqu'à la transaction en question.

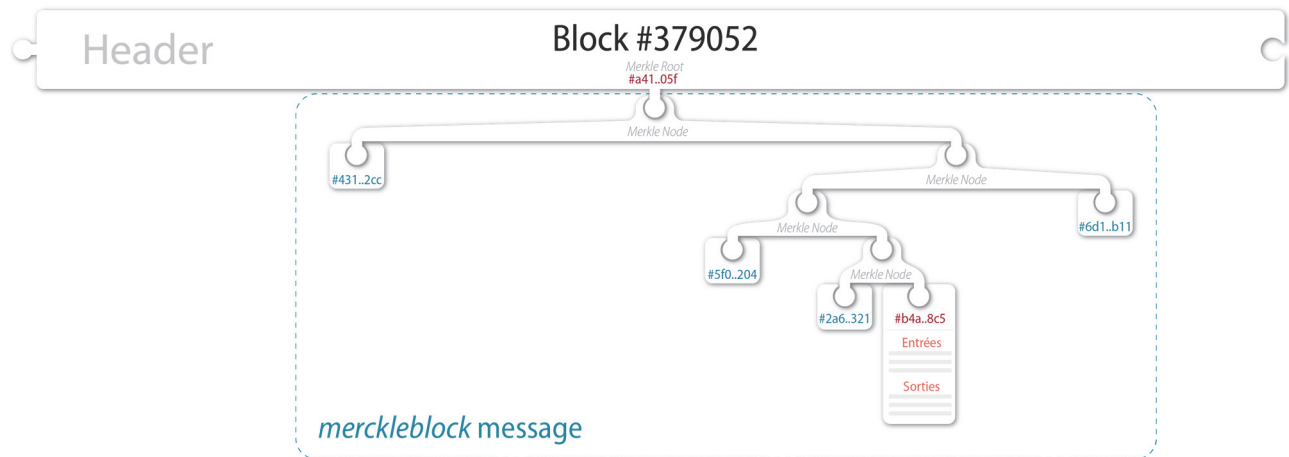


Figure 12 - Récupération d'une chaîne de Merkle via le protocole SPV

La chaîne de Merkle retournée est composée des empreintes des feuilles (en **bleu** dans la figure 12). À partir de ces éléments, le client SPV peut ensuite vérifier l'intégrité de l'arbre jusqu'à la racine de Merkle, présente dans l'entête du bloc.

Si l'intégrité de l'arbre est correcte, la preuve d'appartenance de la transaction au bloc est apportée.

L'économie en matière d'espace de stockage est considérable (facteur 1000) et rend possible l'utilisation de solutions légères (tablettes, smartphones, etc.).

Reste que si cela permet de vérifier qu'une transaction est bien intégrée dans la blockchain, cela ne donne pas d'informations concernant l'état des sorties de cette transaction.

## 8/ Normalisation et organisation

Historiquement et encore actuellement, les spécifications de Bitcoin résident en grande partie dans l'implémentation de référence *Bitcoin Core*, autrement dit, en C++ natif ;-)

Après le retrait de son fondateur, Satoshi Nakamoto, le développement de Bitcoin est devenu communautaire.

Depuis 2011, un processus de standardisation, inspiré des RFC, se met progressivement en place avec des « BIP » (*Bitcoin Improvement Proposals*) [14].

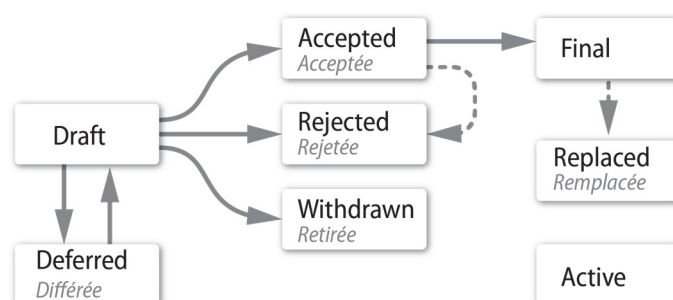


Figure 13 - Statut des Bitcoin Improvement Proposals

Trois types de documents sont définis ; *Standards Track*, *Informational BIPs* et *Process BIP*

### 8.1 Standards Track

Ces BIP ont pour vocation d'aborder tout ce qui peut **impacter la plupart ou l'ensemble des implémentations Bitcoin**. Cela peut être un changement de protocole, un changement dans les règles de validation ou de blocage des transactions, ou toute action de nature à affecter l'interopérabilité de Bitcoin,

Exemples :

- BIP 0001 : BIP Purpose and Guidelines (Active)
- BIP 0016 : Pay To Script Hash (Final)
- BIP 0021: URI Scheme (Accepted)

### 8.2 Informational BIPs

A l'image des RFC éponymes, une BIP « *informational* » a pour objectif de fournir à la communauté Bitcoin des directives générales, de **proposer des informations** ou encore de décrire certains mécanismes. Ces documents n'ont pas vocation à proposer de nouvelles fonctionnalités ni même à représenter un consensus au sein de la communauté.

Les utilisateurs et développeurs sont libres d'ignorer ces BIPs.

Exemples :

- BIP 0032 : Hierarchical Deterministic Wallets (Accepted)

### 8.3 Process BIPs

Ces BIPs décrivent ou proposent un **changement au sein des processus périphériques de Bitcoin**, c'est-à-dire qui ne sont pas de nature à impacter l'ensemble des implémentations.

Contrairement aux BIP « *informationals* », les BIP « *process* » doivent être faire l'objet d'un consensus communautaire. Elles sont davantage que des recommandations et ne peuvent être ignorées.

## 8.4 La difficile quête du consensus

L'émergence d'un consensus peut être complexe. A titre d'exemple, un débat s'est ouvert concernant l'évolution de la taille des blocs et deux propositions sont en concurrence, proposant deux approches orthogonales (BIP 100 et BIP 101).

Faute d'accord un fork s'est opéré au sein de la communauté.

Les mineurs, ont la possibilité de voter, en insérant dans les blocs minés un tag indiquant leur préférence. Le vote des mineurs peut ainsi être suivi en temps réel [15].

Reste que les autres acteurs (nœuds relais, acteurs économiques, etc.) ne sont pas représentés à travers ce processus.

## 8.5 Documentation et sources d'information

Beaucoup de choses ont été dites ou écrites sur Bitcoin, mais les articles techniques ne sont pas les plus nombreux ;-)

Il existe néanmoins d'excellentes sources d'informations :

- Bitcoin.org, *developer documentation* [16]  
<https://bitcoin.org/en/developer-documentation>
- *Bitcoin Improvement Proposals* (équivalent des RFC pour Bitcoin) [17]  
[https://en.bitcoin.it/wiki/Bitcoin\\_Improvement\\_Proposals](https://en.bitcoin.it/wiki/Bitcoin_Improvement_Proposals)
- Bitcoin Wiki, *Category : Technical* [18]  
<https://en.bitcoin.it/wiki/Category:Technical>
- Bitcoin Wiki, *Protocol documentation* [19]  
[https://en.bitcoin.it/wiki/Protocol\\_documentation](https://en.bitcoin.it/wiki/Protocol_documentation)
- *Mastering Bitcoin* (O'Reilly), d'Andreas M. Antonopoulos [20]
- *Bitcoin Developer Reference*, de Krzysztof Okupski [21]  
<https://github.com/minium/Bitcoin-Spec>

...et bien sûr, l'article fondateur de Satoshi Nakamoto, publié en 2009 :

*Bitcoin: A Peer-to-Peer Electronic Cash System* [22]

- <http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source>
- <http://www.bitcoin.org/bitcoin.pdf>
- Une traduction en français de Benkebab, Grondilu & Mackila  
<http://bitcoin.fr/Bitcoin-explique-par-son-inventeur>

## Bibliographie

- [1] Blockchain.info, «Statistiques / Taille de la blockchain,» [En ligne]. Available: <https://blockchain.info/fr/charts/blocks-size>.
- [2] Bitcoin Wiki, «Running Bitcoin,» [En ligne]. Available: [https://en.bitcoin.it/wiki/Running\\_Bitcoin](https://en.bitcoin.it/wiki/Running_Bitcoin).
- [3] Bitcoin Wiki, «Testnet, an alternative Bitcoin blockchain for testing,» [En ligne]. Available: <https://en.bitcoin.it/wiki/Testnet>.
- [4] Bitcoin.org, «Bitcoin Developer Examples / Regtest Mode,» [En ligne]. Available: <https://bitcoin.org/en/developer-examples#regtest-mode>.
- [5] Bitcoin.org, «Developer reference / RPC quick reference guide,» [En ligne]. Available: <https://bitcoin.org/en/developer-reference#rpc-quick-reference>.
- [6] E. Voskuil, «Bitcoin Explorer,» [En ligne]. Available: <https://github.com/libbitcoin/libbitcoin-explorer/wiki>.
- [7] Wikipedia, «SHA-2/SHA-256,» [En ligne]. Available: <https://fr.wikipedia.org/wiki/SHA-2#SHA-256>.
- [8] Wikipedia, «RIPEMD-160,» [En ligne]. Available: <https://fr.wikipedia.org/wiki/RIPEMD-160>.
- [9] e-ducatt.fr, «ECDSA, technologie clé de bitcoin,» [En ligne]. Available: <http://www.e-ducatt.fr/bitcoin-2/securite-signatures-ecdsa/>.
- [10] D. R. L. Brown, «SEC 1: Elliptic Curve Cryptography,» Certicom Research, 2009.
- [11] Bitcoin Wiki, «Script,» [En ligne]. Available: <https://en.bitcoin.it/wiki/Script>.
- [12] Bitcoin Wiki, «BIP 0013 - Address Format for pay-to-script-hash,» [En ligne]. Available: [https://en.bitcoin.it/wiki/BIP\\_0013](https://en.bitcoin.it/wiki/BIP_0013).
- [13] Blockchain.info, «Statistiques Bitcoin / Evolution de la difficulté,» [En ligne]. Available: [https://blockchain.info/fr/charts/difficulty?timespan=all&showDataPoints=false&daysAverageString=1&show\\_header=true&scale=0&address=](https://blockchain.info/fr/charts/difficulty?timespan=all&showDataPoints=false&daysAverageString=1&show_header=true&scale=0&address=).
- [14] Bitcoin Wiki, «Bitcoin Improvement Proposals,» [En ligne]. Available: [https://en.bitcoin.it/wiki/Bitcoin\\_Improvement\\_Proposals](https://en.bitcoin.it/wiki/Bitcoin_Improvement_Proposals).
- [15] Bitcoinity.org, «Block size vote (BIP 100 vs BIP101),» [En ligne]. Available: [https://data.bitcoinity.org/bitcoin/block\\_size\\_votes/7d?c=block\\_size\\_votes&r=hour&t=bar](https://data.bitcoinity.org/bitcoin/block_size_votes/7d?c=block_size_votes&r=hour&t=bar).
- [16] Bitcoin.org, «Developer documentation,» [En ligne]. Available: <https://bitcoin.org/en/developer-documentation>.
- [17] Communauté Bitcoin, «Bitcoin Improvement Proposals (BIP),» [En ligne]. Available: [https://en.bitcoin.it/wiki/Bitcoin\\_Improvement\\_Proposals](https://en.bitcoin.it/wiki/Bitcoin_Improvement_Proposals).
- [18] Bitcoin Wiki, «Category : Technical,» [En ligne]. Available: <https://en.bitcoin.it/wiki/Category:Technical>.
- [19] Bitcoin Wiki, «Protocol documentation,» [En ligne]. Available: [https://en.bitcoin.it/wiki/Protocol\\_documentation](https://en.bitcoin.it/wiki/Protocol_documentation).
- [20] A. M. Antonopoulos, Mastering Bitcoin, O'Reilly.
- [21] K. Okupski, Bitcoin Developer Reference, Working Paper, Technische Universiteit Eindhoven, The Netherlands, 2014.
- [22] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System,» 2009.