Portefeuilles

2022-03-06

# Portefeuilles

Le mot "portefeuille" est utilisé pour décrire différentes choses dans le bitcoin.

À un niveau élevé, un portefeuille est une application qui sert d'interface utilisateur principale. Le portefeuille contrôle l'accès à l'argent d'un utilisateur, gère les clés et les adresses, le solde et crée et signe des transactions.

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

Dans ce chapitre, nous examinerons la deuxième signification, où les portefeuilles sont des conteneurs de clés privées, généralement implémentés sous forme de fichiers structurés ou de simples bases de données.

## Présentation de la technologie de portefeuille

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

Une idée faussement courante à propos du bitcoin est que les portefeuilles bitcoin contiennent du bitcoin. En fait, le portefeuille ne contient que des clés. Les "monnaies ou coins" sont enregistrés dans la chaîne de blocs sur le réseau Bitcoin. Les utilisateurs contrôlent les pièces sur le réseau en signant des transactions avec les clés de leur portefeuille. Dans un sens, un portefeuille bitcoin est un *porte-clés*.

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

Il existe deux principaux types de portefeuilles, qui se distinguent par le fait que les clés qu'ils contiennent sont liées les unes aux autres ou non.

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

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

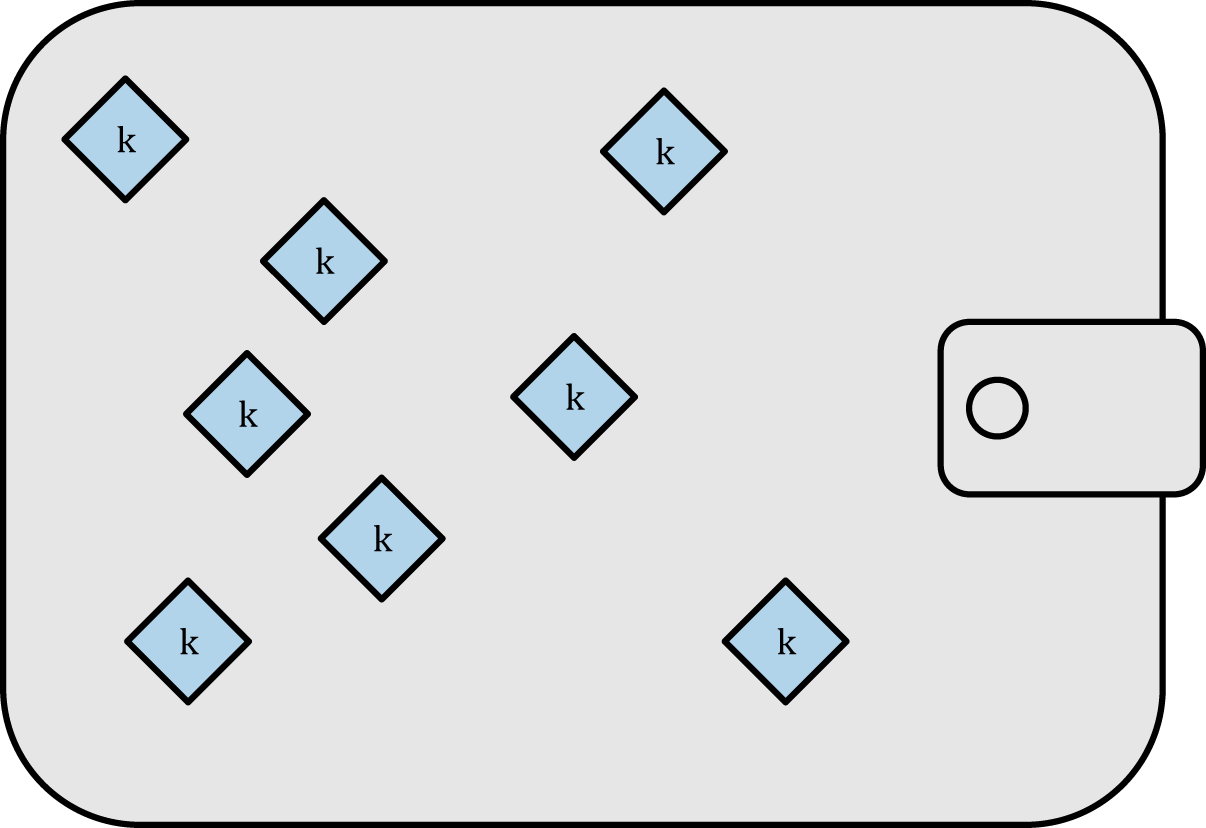
Les portefeuilles déterministes sont initialisés à partir d'une séquence aléatoire (une entropie). Pour faciliter leur utilisation, les séquences aléatoires sont encodées sous forme de mots anglais, également appelés *mots de code mnémoniques*.

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

### Portefeuilles non déterministes (aléatoires)

Dans le premier portefeuille bitcoin (maintenant appelé Bitcoin Core), les portefeuilles étaient des collections de clés privées générées aléatoirement. Par exemple, le client Bitcoin Core d'origine prégénère 100 clés privées aléatoires lors du premier démarrage et génère plus de clés si nécessaire, en utilisant chaque clé une seule fois. Ces portefeuilles sont remplacés par des portefeuilles déterministes car ils sont lourds à gérer, à sauvegarder et à importer. L'inconvénient des clés aléatoires est que si vous en générez beaucoup, vous devez conserver des copies de toutes, ce qui signifie que le portefeuille doit être sauvegardé fréquemment. Chaque clé doit être sauvegardée, sinon les fonds qu'elle contrôle sont irrévocablement perdus si le portefeuille devient inaccessible. Cela entre directement en conflit avec le principe d'éviter la réutilisation des adresses, en utilisant chaque adresse Bitcoin pour une seule transaction. La réutilisation des adresses réduit la confidentialité en associant plusieurs transactions et adresses les unes aux autres. Un portefeuille non déterministe de type-0 est un mauvais choix de portefeuille, surtout si vous voulez éviter la réutilisation des adresses car cela implique de gérer de nombreuses clés, ce qui crée le besoin de sauvegardes fréquentes. Bien que le client Bitcoin Core inclue un portefeuille de type 0, l'utilisation de ce portefeuille est déconseillée par les développeurs de Bitcoin Core. [figure\_title](#Type0_wallet) montre un portefeuille non déterministe, contenant une collection lâche de clés aléatoires.

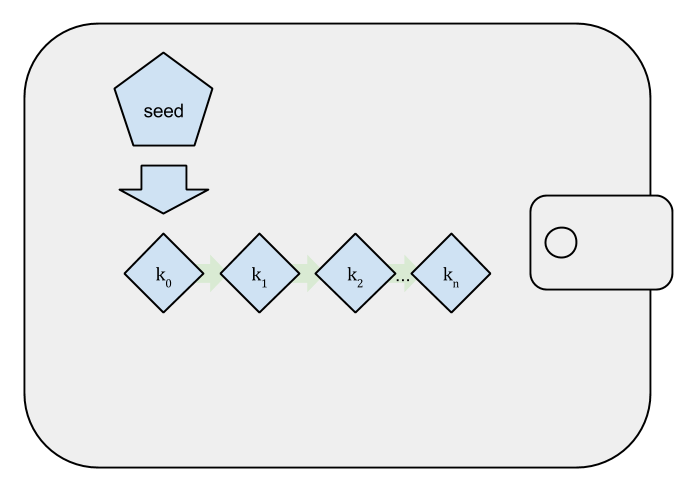
L'utilisation de portefeuilles non déterministes est déconseillée pour autre chose que de simples tests. Ils sont tout simplement trop lourds à sauvegarder et à utiliser. À la place, utilisez un *portefeuille HD* standard de l'industrie avec une séquence aléatoire *mnémonique* (entropie, ou "valeur d’amorçage") pour la sauvegarde.



Portefeuille non déterministe (aléatoire) de type-0 : une collection de clés générées aléatoirement

### Portefeuilles déterministes (ensemencés)

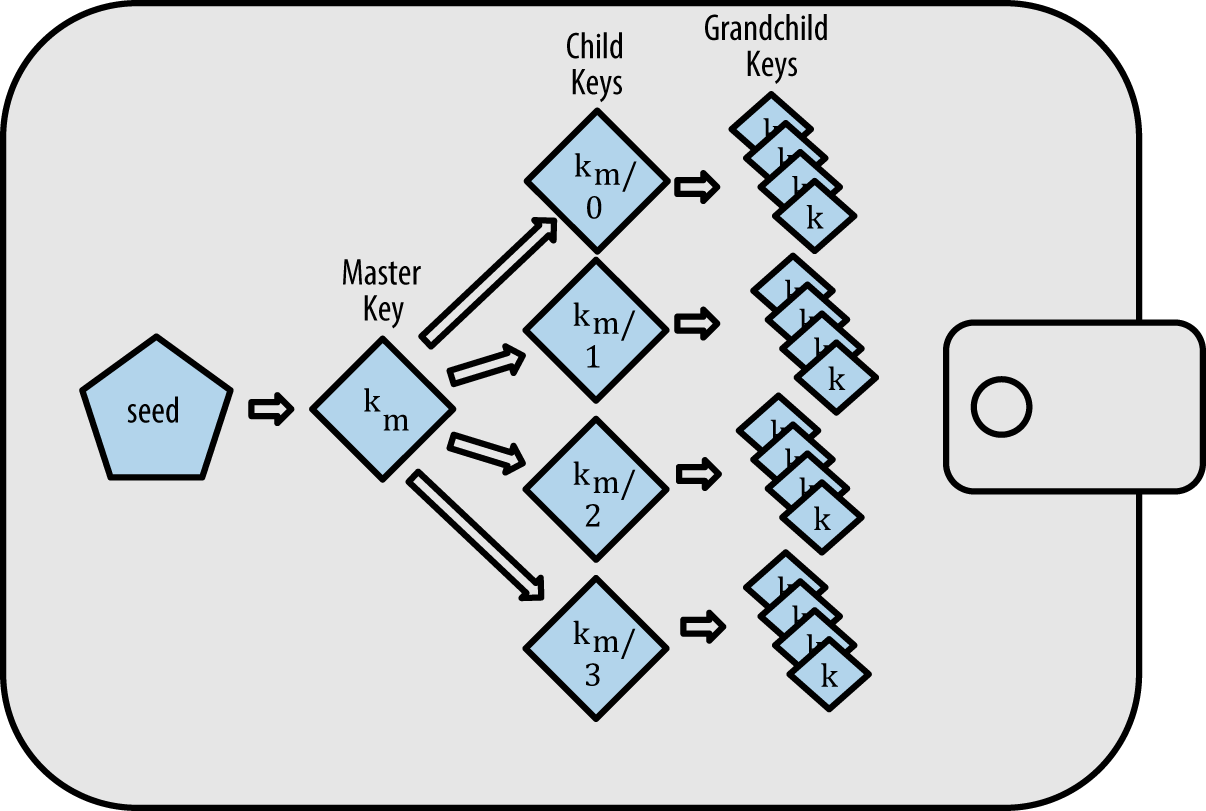
Les portefeuilles déterministes ou "ensemencés" sont des portefeuilles qui contiennent des clés privées qui sont toutes dérivées d'une valeur d’amorçage commune, grâce à l'utilisation d'un fonction de hachage à sens unique. La valeur d’amorçage est un nombre généré aléatoirement qui est combiné avec d'autres données, telles qu'un numéro d'index ou un "code de chaîne" (voir [Portefeuilles HD (BIP-32/BIP-44)](#hd_wallets)) pour dériver les clés privées. Dans un portefeuille déterministe, la valeur d’amorçage est suffisante pour récupérer toutes les clés dérivées, et donc une seule sauvegarde au moment de la création est suffisante. La valeur d’amorçage est également suffisante pour une exportation ou une importation de portefeuille, permettant une migration facile de toutes les clés de l'utilisateur entre différentes implémentations de portefeuille. [figure\_title](#Type1_wallet) montre un diagramme logique d'un portefeuille déterministe.



Portefeuille déterministe (ensemencé) de type-1 : une séquence déterministe de clés dérivée d'une valeur d’amorçage

### Portefeuilles HD (BIP-32/BIP-44)

Des portefeuilles déterministes ont été développés pour faciliter la dérivation de nombreuses clés à partir d'une seule "valeur d’amorçage". La forme la plus avancée de portefeuilles déterministes est le portefeuille HD défini par la norme BIP-32. Les portefeuilles HD contiennent des clés dérivées dans une structure arborescente, de sorte qu'une clé parent peut dériver une séquence de clés enfants, chacune pouvant dériver une séquence de clés petits-enfants, et ainsi de suite, jusqu'à une profondeur infinie. Cette arborescence est illustrée dans [figure\_title](#Type2_wallet).



Portefeuille HD de type-2 : un arbre de clés généré à partir d'une seule valeur d’amorçage

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

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

### Graines (ou valeurs d’amorçages) et codes mnémoniques (BIP-39)

Les portefeuilles HD sont un mécanisme très puissant pour gérer de nombreuses clés et adresses. Ils sont encore plus utiles s'ils sont combinés à une manière standardisée de créer des valeurs d’amorçage à partir d'une séquence de mots anglais faciles à transcrire, exporter et importer sur plusieurs portefeuilles. C'est ce qu'on appelle un *mnémonique* et la norme est définie par BIP-39. Aujourd'hui, la plupart des portefeuilles Bitcoin (ainsi que des portefeuilles pour d'autres cryptomonnaies) utilisent cette norme et peuvent importer et exporter des valeurs d’amorçage pour la sauvegarde et la récupération à l'aide de mnémoniques interopérables.

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

**Une valeur d’amorçage pour un portefeuille déterministe, en hexadécimal**

0C1E24E5917779D297E14D45F14E1A1A

**Une valeur d’amorçage pour un portefeuille déterministe, à partir d'un mnémonique de 12 mots**

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

### Meilleures procédures de portefeuille

Au fur et à mesure que les portefeuilles de Bitcoin ont atteint un degré de maturité, certaines normes industrielles mis en commun ont émergé, rendant de nos jours des portefeuilles Bitcoin largement interopérables, faciles à utiliser, sécurisés et flexibles. Ces normes communes sont :

* Mots de code mnémonique, basés sur BIP-39
* Portefeuilles HD, basés sur BIP-32
* Structure de portefeuille HD polyvalente, basée sur BIP-43
* Portefeuilles multidevises et multicomptes, basés sur BIP-44

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

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

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

Les sections suivantes examinent chacune de ces technologies en détail.

Si vous implémentez un portefeuille bitcoin, il doit être construit comme un portefeuille HD, avec une valeur d’amorçage dérivée et codée comme un code mnémonique pour la sauvegarde, suivant les normes BIP-32, BIP-39, BIP-43 et BIP-44, comme décrit dans les sections suivantes.

### Utiliser un portefeuille Bitcoin

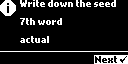
Dans [???](#user-stories) nous avons présenté Gabriel, un jeune adolescent entreprenant à Rio de Janeiro, qui gère une simple boutique en ligne qui vend des t-shirts de marque bitcoin, des tasses à café et des autocollants.

Gabriel utilise un portefeuille matériel bitcoin Trezor ([figure\_title](#a_trezor_device)) pour gérer en toute sécurité son bitcoin. Le Trezor est un simple périphérique USB à deux boutons qui stocke les clés (sous la forme d'un portefeuille HD) et signe les transactions. Les portefeuilles Trezor mettent en œuvre toutes les normes de l'industrie abordées dans ce chapitre, de sorte que Gabriel ne dépend d'aucune technologie propriétaire ou d'une solution de fournisseur unique.



Un appareil Trezor : un portefeuille bitcoin HD en matériel

Lorsque Gabriel a utilisé le Trezor pour la première fois, l'appareil a généré une séquence aléatoire (entropie), le mnémonique associé et a dérivé une valeur d’amorçage à partir d'un générateur de nombres aléatoires matériel intégré. Lors de cette phase d'initialisation, le portefeuille affichait à l'écran une séquence de mots numérotés, un par un (voir [figure\_title](#trezor_mnemonic_display)).



Trezor affichant un des mots mnémoniques

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

La sauvegarde papier de Gabriel du mnémonique

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

Un mnémonique de 12 mots est affiché dans [table\_title](#mnemonic_paper_backup), pour plus de simplicité. En fait, la plupart des portefeuilles matériels génèrent un mnémonique de 24 mots plus sécurisé. Le mnémonique est utilisé exactement de la même manière, quelle que soit sa longueur.

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

## Détails de la technologie de portefeuille

Examinons maintenant en détail chacune des normes importantes de l'industrie utilisées par de nombreux portefeuilles Bitcoin.

### Mots de code mnémonique (BIP-39)

Les mots de code mnémonique sont des séquences de mots qui représentent (encodent) un nombre aléatoire utilisé comme valeur d’amorçage pour dériver un portefeuille déterministe. La séquence de mots est suffisante pour recréer la valeur d’amorçage et à partir de là recréer le portefeuille et toutes les clés dérivées. Une application de portefeuille qui implémente des portefeuilles déterministes avec des mots mnémoniques montrera à l'utilisateur une séquence de 12 à 24 mots lors de la première création d'un portefeuille. Cette séquence de mots est à sauvegarder et est la valeur d’amorçage du portefeuille. Elle peut être utilisée pour récupérer et recréer toutes les clés dans la même application de portefeuille ou dans n'importe quelle application de portefeuille compatible. Les mots mnémotechniques facilitent la sauvegarde des portefeuilles par les utilisateurs, car ils sont faciles à lire et à transcrire correctement, par rapport à une séquence aléatoire de chiffres.

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

Les codes mnémoniques sont définis dans le BIP-39 (voir [???](#appdxbitcoinimpproposals)). Notez que BIP-39 est une implémentation d'une norme de code mnémonique. Il existe une norme différente, avec un ensemble de mots différent, utilisée par le portefeuille Electrum et antérieure à BIP-39. BIP-39 a été proposé par la société à l'origine du portefeuille matériel Trezor et est incompatible avec la mise en œuvre d'Electrum. Cependant, BIP-39 a maintenant obtenu un large soutien de l'industrie à travers des dizaines d'implémentations interopérables et devrait être considéré comme la norme de facto de l'industrie.

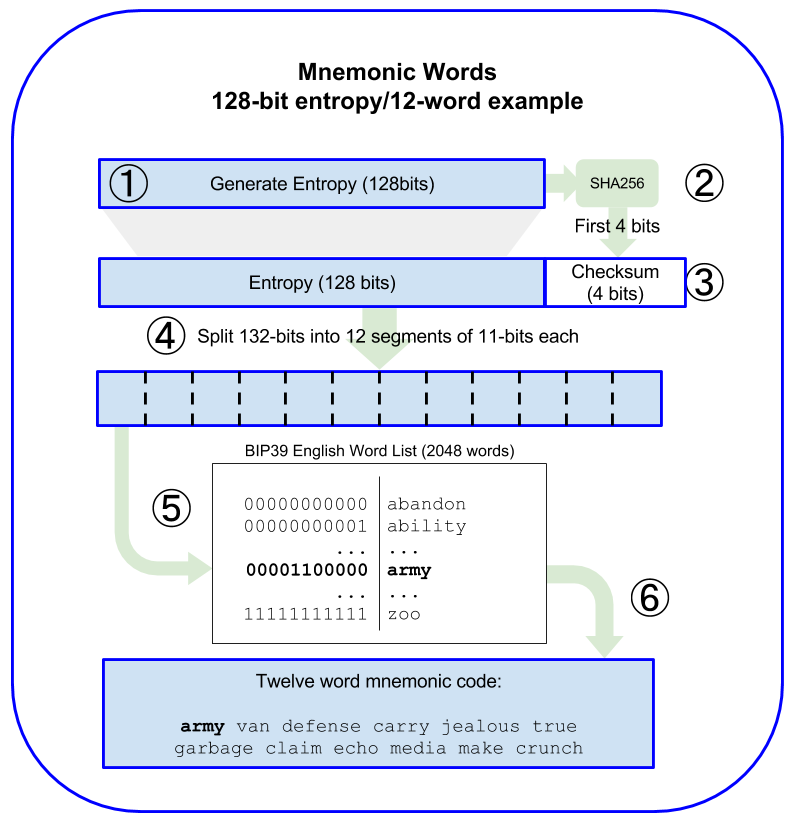
La BIP-39 définit la création d'un code mnémonique et d'une valeur d’amorçage, que nous décrivons ici en neuf étapes. Pour plus de clarté, le processus est divisé en deux parties: les étapes 1 à 6 sont présentées dans [Génération de mots mnémoniques](#generating_mnemonic_words) et les étapes 7 à 9 sont illustrées dans [Du mnémonique à la valeur d’amorçage](#mnemonic_to_seed).

#### Génération de mots mnémoniques

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

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

<<generating\_entropy\_and\_encoding> > montre comment l'entropie est utilisée pour générer des mots mnémoniques.



Génération d'entropie et encodage sous forme de mots mnémoniques

[table\_title](#table_4-5) montre la relation entre la taille des données d'entropie et la longueur des codes mnémoniques en mots.

Codes mnémoniques : entropie et longueur des mots

| Entropie (bits) | Somme de contrôle (bits) | Somme de contrôle d'entropie **+** (bits) | Longueur mnémonique (mots) |
| --- | --- | --- | --- |
| 128 | 4 | 132 | 12 |
| 160 | 5 | 165 | 15 |
| 192 | 6 | 198 | 18 |
| 224 | 7 | 231 | 21 |
| 256 | 8 | 264 | 24 |

#### Du mnémonique à la valeur d’amorçage

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

La fonction d'étirement de clé prend deux paramètres : le mnémonique et un *sel*. Le but d'un sel dans une fonction d'étirement de clé est de rendre difficile la construction d'une table de recherche permettant une attaque par force brute. Dans la norme BIP-39, le sel a un autre objectif : il permet l'introduction d'une phrase secrète qui sert de facteur de sécurité supplémentaire protégeant la valeur d’amorçage, comme nous le décrirons plus en détail dans [Phrase secrète facultative dans BIP-39](#mnemonic_passphrase).

Le processus décrit aux étapes 7 à 9 continue à partir du processus décrit précédemment dans [Génération de mots mnémoniques](#generating_mnemonic_words) :

Le premier paramètre de la fonction d'étirement de clé PBKDF2 est le

mnémonique

produit à partir de l'étape 6.

Le deuxième paramètre de la fonction d'étirement de clé PBKDF2 est un

sel

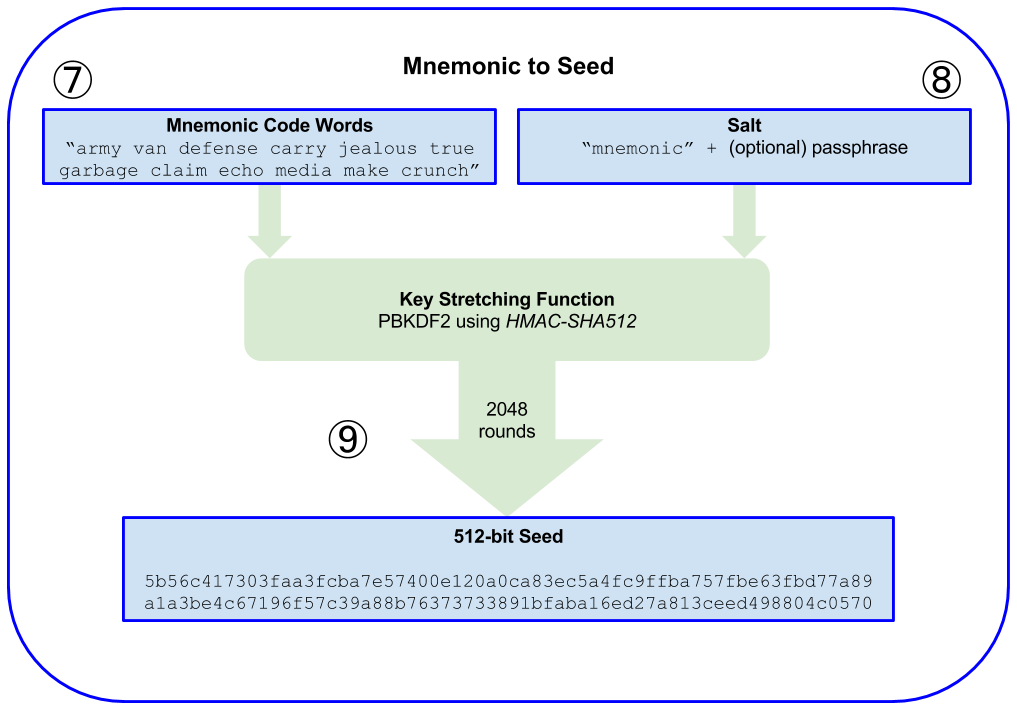
. Le sel est composé de la constante de chaîne "

mnemonique

" concaténée avec une phrase de passe facultative fournie par l'utilisateur.

PBKDF2 étend les paramètres mnémoniques et le sel en utilisant 2048 cycles de hachage avec l'algorithme HMAC-SHA512, produisant une valeur de 512 bits comme sortie finale. Cette valeur de 512 bits est la valeur d'amorçage.

[figure\_title](#fig_5_7) montre comment un mnémonique est utilisé pour générer une valeur d’amorçage.



Du mnémonique à la valeur d’amorçage

La fonction d'étirement de clé, avec ses 2048 tours de hachage, est une protection très efficace contre les attaques par force brute contre le mnémonique ou la phrase secrète. Cela rend extrêmement coûteux (en calcul) d'essayer plus de quelques milliers de combinaisons de mots de passe et de mnémoniques, alors que le nombre de graines dérivées possibles est vaste (2 ^ 512 ^).

Les tableaux #mnemonic\_128\_no\_pass, #mnemonic\_128\_w\_pass et #mnemonic\_256\_no\_pass montrent quelques exemples de codes mnémoniques et les valeurs d’amorçage qu'ils produisent (avec ou sans phrase secrète).

Code mnémonique d'entropie 128 bits, sans phrase secrète, valeur d’amorçage résultante

|  |  |
| --- | --- |
| **Entrée d'entropie (128 bits)** | 0c1e24e5917779d297e14d45f14e1a1a |
| **Mnémonique (12 mots)** | army van defense carry jealous true garbage claim echo media make crunch |
| **Phrase secrète** | (rien) |
| **Valeur d’amorçage (512 bits)** | 5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39 a88b76373733891bfaba16ed27a813ceed498804c0570 |

Code mnémonique d'entropie 128 bits, avec phrase secrète, valeur d’amorçage résultante

|  |  |
| --- | --- |
| **Entrée d'entropie (128 bits)** | 0c1e24e5917779d297e14d45f14e1a1a |
| **Mnémonique (12 mots)** | army van defense carry jealous true garbage claim echo media make crunch |
| **Phrase secrète** | SuperDuperSecret |
| **Valeur d’amorçage (512 bits)** | 3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0 715861dc8a18358f80b79d49acf64142ae57037d1d54 |

Code mnémonique d'entropie 256 bits, sans phrase secrète, valeur d’amorçage résultante

|  |  |
| --- | --- |
| **Entrée d'entropie (256 bits)** | 2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c |
| **Mnémonique (24 mots)** | cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige |
| **Phrase secrète** | (rien) |
| **Valeur d’amorçge (512 bits)** | 3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e5 5f1e0deaa082df8d487381379df848a6ad7e98798404 |

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

#### Phrase secrète facultative dans BIP-39

La norme BIP-39 permet l'utilisation d'une phrase secrète facultative dans la dérivation de la valeur d’amorçage. Si aucune phrase secrète n'est utilisée, le mnémonique est étiré avec un sel constitué de la chaîne constante &quot;mnémonique&quot;, produisant une graine spécifique de 512 bits à partir de n'importe quel mnémonique donné. Si une phrase secrète est utilisée, la fonction d'étirement produit une valeur d’amorçage *différente* à partir de ce même mnémonique. En fait, étant donné un seul mnémonique, chaque phrases secrètes possible conduit à une valeurs d’amorçage différente. Essentiellement, il n'y a pas de "mauvaise" phrase secrète. Toutes les phrases secrètes sont valides et mènent toutes à des valeurs d’amorçage différentes, formant un vaste ensemble de portefeuilles non initialisés possibles. L'ensemble des portefeuilles possibles est si grand (2 512) qu'il n'y a aucune possibilité pratique de forcer brutalement ou de deviner accidentellement celui qui est utilisé.

Il n'y a pas de "mauvaises" phrases secrètes dans BIP-39. Chaque phrase de passe mène à un portefeuille qui, à moins qu'il n'ait été utilisé auparavant, sera vide.

La phrase secrète facultative crée deux fonctionnalités importantes :

* Un deuxième facteur (quelque chose de mémorisé) qui rend un mnémonique inutile par lui-même, protégeant les sauvegardes mnémoniques de la compromission par un voleur.
* Une forme de déni plausible ou "portefeuille sous contrainte", où une phrase secrète choisie mène à un portefeuille avec une petite quantité de fonds utilisée pour distraire un attaquant du "vrai" portefeuille qui contient la majorité des fonds.

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

* Si le propriétaire du portefeuille est incapacité ou décédé et que personne d'autre ne connaît la phrase secrète, le code mnémonique est inutile et tous les fonds stockés dans le portefeuille sont perdus à jamais.
* À l'inverse, si le propriétaire sauvegarde la phrase secrète au même endroit que le code mnémonique, cela va à l'encontre de l'objectif d'un deuxième facteur.

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

#### Travailler avec des codes mnémoniques

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

[python-mnemonic](https://github.com/trezor/python-mnemonic)

L'implémentation de référence de la norme par l'équipe SatoshiLabs qui a proposé BIP-39, en Python

[bitcoinjs/bip39](https://github.com/bitcoinjs/bip39)

Une implémentation de BIP-39, dans le cadre du populaire framework bitcoinJS, en JavaScript

[libbitcoin/mnemonic](https://github.com/libbitcoin/libbitcoin/blob/master/src/wallet/mnemonic.cpp)

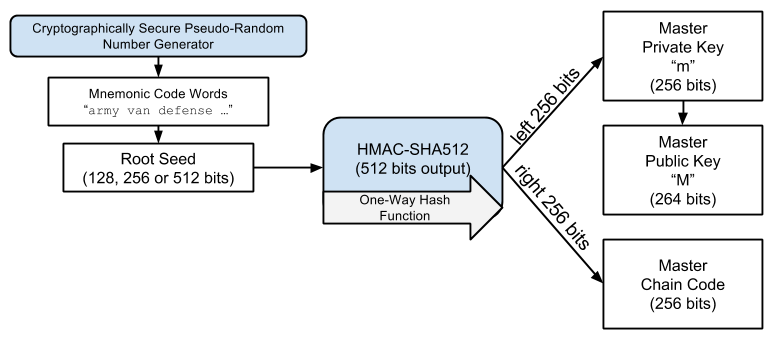
Une implémentation de BIP-39, dans le cadre du populaire framework Libbitcoin, en C++

### Créer un portefeuille HD à partir de la valeur d’amorçage

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

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

Le processus de création des clés principales et du code de chaîne principal pour un portefeuille HD est illustré dans [figure\_title](#HDWalletFromSeed).



Création de clés principales et de code de chaîne à partir d'une valeur d’amorçage racine

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

La clé privée principale (m) génère ensuite une clé publique principale correspondante (M) en utilisant le processus normal de multiplication de courbe elliptique m \* G que nous avons vu dans [???](#pubkey).

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

#### Dérivation de clé enfant privée

Les portefeuilles HD utilisent une fonction *dérivation de la clé-enfant* (CKD) pour dériver les clés enfants à partir des clés parents.

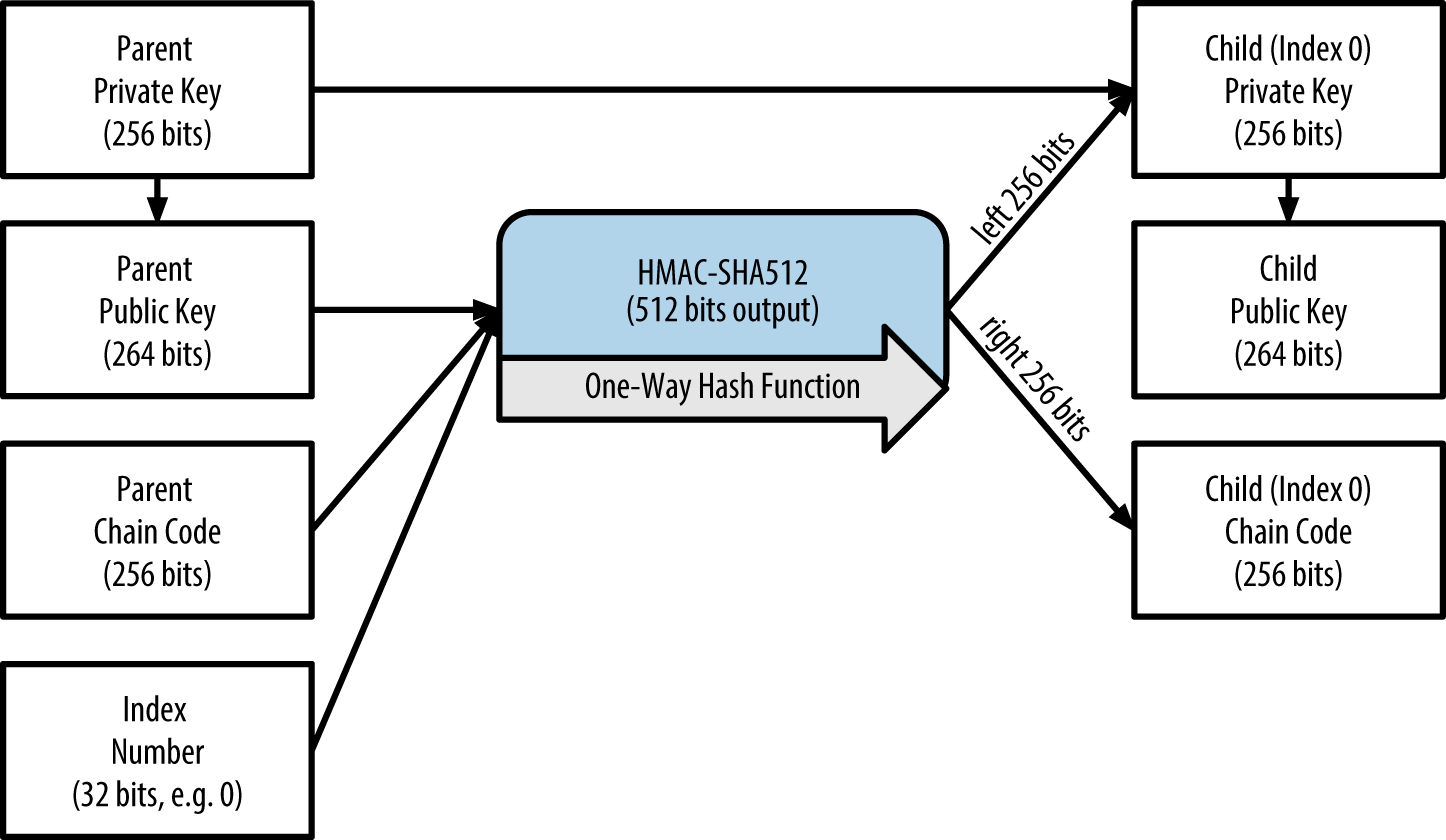
Les fonctions de dérivation de la clé-enfant sont basées sur une fonction de hachage unidirectionnelle qui combine :

* Une clé privée ou publique parent (clé compressée ECDSA)
* Une valeur d’amorçage appelée code de chaîne (256 bits)
* Un numéro d'index (32 bits)

Le code de chaîne est utilisé pour introduire des données aléatoires déterministes dans le processus, de sorte que la connaissance de l'index et d'une clé-enfant ne suffit pas pour dériver d'autres clés-enfants. Connaître une clé enfant ne permet pas de retrouver ses frères et sœurs, sauf si vous disposez également du code chaîne. La valeur d’amorçage de code de chaîne initiale (à la racine de l'arborescence) est créée à partir de la valeur d’amorçage, tandis que les codes de chaîne enfants suivants sont dérivés de chaque code de chaîne parent.

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

La clé publique parent, le code de chaîne et le numéro d'index sont combinés et hachés avec l'algorithme HMAC-SHA512 pour produire un hachage de 512 bits. Ce hachage de 512 bits est divisé en deux moitiés de 256 bits. Les 256 bits de la moitié droite de la sortie de hachage deviennent le code de chaîne pour l'enfant. Les 256 bits de la moitié gauche du hachage sont ajoutés à la clé parent pour produire la clé privée enfant. Dans [figure\_title](#CKDpriv), nous voyons cela illustré avec l'index mis à 0 pour produire l'enfant "zéro" (premier par index) du parent.



Étendre une clé privée parent pour créer une clé privée enfant

Changer l'index nous permet d'étendre le parent et de créer les autres enfants dans la séquence, par exemple, Enfant 0, Enfant 1, Enfant 2, etc. Chaque clé parent peut avoir 2 147 483 647 (231) enfants (231 est moitié de la plage entière 232 disponible car l'autre moitié est réservée à un type spécial de dérivation dont nous parlerons plus loin dans ce chapitre).

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

#### Utilisation de clés enfants dérivées

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

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

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

#### Touches étendues

Comme nous l'avons vu précédemment, la fonction de dérivation de clé peut être utilisée pour créer des enfants à n'importe quel niveau de l'arborescence, en fonction sur les trois entrées : une clé, un code chaîne et l'index de l'enfant souhaité. Les deux ingrédients essentiels sont la clé et le code de chaîne, et combinés, ils s'appellent une *clé étendue*. Le terme "clé étendue" pourrait également être considéré comme une "clé extensible" car une telle clé peut être utilisée pour dériver des enfants.

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

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

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

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

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

xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6Hx4tws2six3b9c

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

xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJunSDMstweyLXhRgPxdp14sk9tJPW9

#### Dérivation de la clé enfant publique

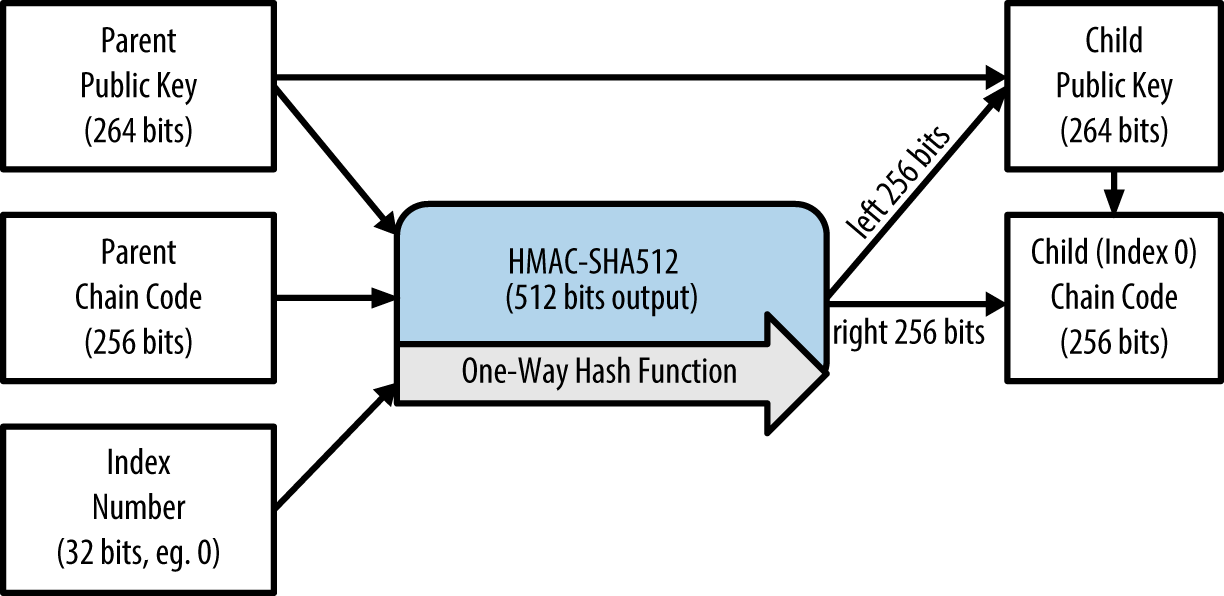
Comme mentionné précédemment, une caractéristique très utile des portefeuilles HD est la possibilité de dériver des clés publiques enfant à partir de clés publiques parent, *sans* avoir les clés privées. Cela nous donne deux façons de dériver une clé publique enfant : soit à partir de la clé privée enfant,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

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

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

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

Une autre application courante de cette solution est le stockage à froid ou les portefeuilles matériels. Dans ce scénario, la clé privée étendue peut être stockée sur un portefeuille papier ou un périphérique matériel (tel qu'un portefeuille matériel Trezor), tandis que la clé publique étendue peut être conservée en ligne. L'utilisateur peut créer des adresses "de réception" à volonté, tandis que les clés privées sont stockées en toute sécurité hors ligne. Pour dépenser les fonds, l'utilisateur peut utiliser la clé privée étendue sur un client Bitcoin de signature hors ligne ou signer des transactions sur le périphérique de portefeuille matériel (par exemple, Trezor). [figure\_title](#CKDpub) illustre le mécanisme d'extension d'une clé publique parent pour dériver des clés publiques enfants.

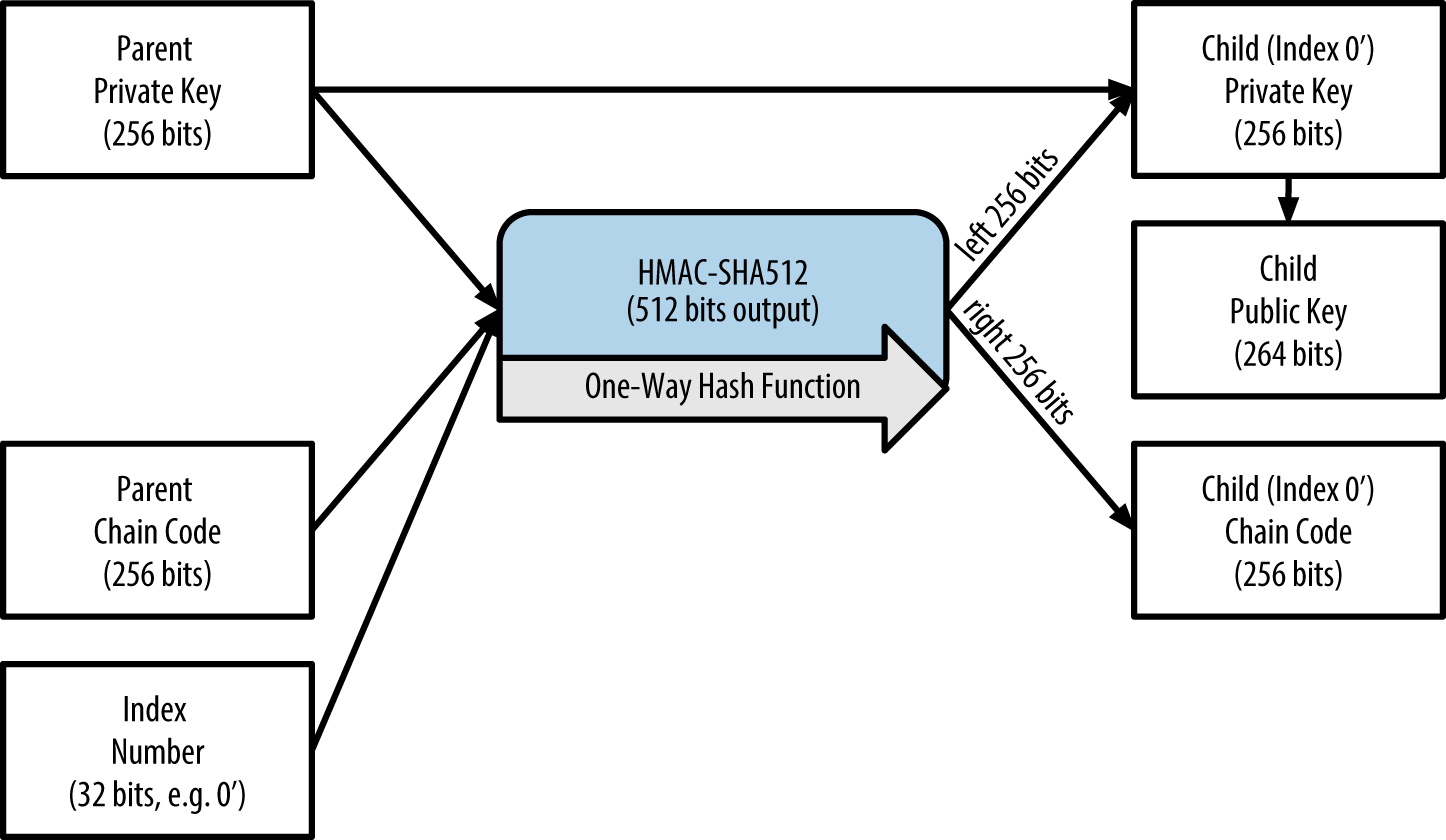


Étendre une clé publique parent pour créer une clé publique enfant

#### Dérivation de clé enfant renforcée

La possibilité de dériver une branche de clés publiques à partir d'un xpub est très utile, mais elle est potentiellement risquée. L'accès à une xpub ne donne pas accès aux clés privées enfants. Cependant, étant donné que xpub contient le code de chaîne, si une clé privée enfant est connue ou divulguée d'une manière ou d'une autre, elle peut être utilisée avec le code de chaîne pour dériver toutes les autres clés privées enfants. Une seule clé privée enfant divulguée, associée à un code de chaîne parent, révèle toutes les clés privées de tous les enfants. Pire encore, la clé privée enfant associée à un code de chaîne parent peut être utilisée pour déduire la clé privée parent.

Pour contrer ce risque, les portefeuilles HD utilisent une fonction de dérivation alternative appelée *dérivation renforcée*, qui "casse" la relation entre la clé publique parent et le code de chaîne enfant. La fonction de dérivation renforcée utilise la clé privée parent pour dériver le code de chaîne enfant, au lieu de la clé publique parent. Cela crée un "pare-feu" dans la séquence parent/enfant, avec un code de chaîne qui ne peut pas être utilisé pour compromettre une clé privée parent ou sœur. La fonction de dérivation renforcée semble presque identique à la dérivation de clé privée enfant normale, sauf que la clé privée parent est utilisée comme entrée de la fonction de hachage, au lieu de la clé publique parent, comme indiqué dans le diagramme de [figure\_title](#CKDprime).



Dérivation renforcée d'une clé enfant ; omet la clé publique parent

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

En termes simples, si vous souhaitez utiliser la commodité d'un xpub pour dériver des branches de clés publiques, sans vous exposer au risque d'une fuite de code de chaîne, vous devez le dériver d'une clé parent renforcée, plutôt que d'une clé normale (non-renforcée) clé parent. En tant que meilleure pratique, les enfants de niveau 1 des clés principales sont toujours dérivés via la dérivation renforcée, pour éviter la compromission des clés principales.

#### Numéros d'index pour dérivation normale et durcie

Le numéro d'index utilisé dans la fonction de dérivation est un entier de 32 bits. Pour distinguer facilement les clés dérivées par la fonction de dérivation normale des clés dérivées par dérivation renforcée, ce numéro d'index est divisé en deux plages. Les numéros d'index entre 0 et 231–1 (0x0 à 0x7FFFFFFF) sont utilisés *uniquement* pour la dérivation normale. Les numéros d'index entre 231 et 232–1 (0x80000000 à 0xFFFFFFFF) sont utilisés *uniquement* pour la dérivation renforcée. Donc, si l'indice est inférieur à 231, l'enfant est normal, alors que si l'indice est égal ou supérieur à 231, l'enfant est renforcé.

Pour faciliter la lecture et l'affichage du numéro d'index, le numéro d'index pour les enfants renforcés est affiché à partir de zéro, mais avec un symbole prime. La première clé enfant normale s'affiche donc sous la forme 0, tandis que la première clé enfant renforcée (index 0x80000000) s'affiche sous la forme 0&#x27;. Dans l'ordre, la deuxième clé renforcée aurait l'index 0x80000001 et serait affichée sous la forme 1&#x27;, et ainsi de suite. Lorsque vous voyez un index de portefeuille HD i&#x27;, cela signifie 231+i.

#### Identifiant de clé de portefeuille HD (chemin)

Les clés d'un portefeuille HD sont identifiées à l'aide d'une convention de dénomination "chemin", où chaque niveau de l'arborescence est séparé par une barre oblique (/) (voir [table\_title](#table_4-8)). Les clés privées dérivées de la clé privée principale commencent par "m". Les clés publiques dérivées de la clé publique principale commencent par « M ». Par conséquent, la première clé privée enfant de la clé privée principale est m/0. La première clé publique enfant est M/0. Le deuxième petit-enfant du premier enfant est m/0/1, et ainsi de suite.

L'"ascendance" d'une clé se lit de droite à gauche, jusqu'à atteindre la clé maîtresse dont elle est issue. Par exemple, l'identifiant m/x/y/z décrit la clé privée qui est le z-ième enfant de la clé privée parent m/x/y, qui est le y-ième enfant de la clé privée parent m/x, qui est le x-ième enfant de la clé privée principale parente m.

Exemples de chemin de portefeuille HD

| Chemin HD | Clé décrite |
| --- | --- |
| m/0 | La première (0) clé privée enfant de la clé privée maîtresse (m) |
| m/0/0 | La première (0) clé privée enfant du premier enfant (m/0) |
| m/0'/0 | Le premier (0) enfant normal du premier enfant *renforcé* (m/0') |
| m/1/0 | La première (0) clé privée enfant du deuxième enfant (m/1) |
| M/23/17/0/0 | La première (0) clé publique enfant du premier enfant (M/23/17/0) du 18ème enfant (M/23/17) du 24ème enfant (M/23) |

#### Naviguer dans l'arborescence du portefeuille HD

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

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

En étendant cette spécification, BIP-44 propose une structure multicompte en tant que numéro "d'objet" 44&#39; sous BIP-43. Tous les portefeuilles HD suivant la structure BIP-44 sont identifiés par le fait qu'ils n'utilisent qu'une seule branche de l'arborescence : m/44'/.

BIP-44 spécifie la structure comme étant composée de cinq niveaux d'arborescence prédéfinis :

m / but&#39; / type\_monnaie&#39; / compte&#39; / change / index\_adresse

Le "but" de premier niveau est toujours réglé sur 44&#39;. Le "type\_monnaie" de deuxième niveau spécifie le type de pièce de cryptomonnaie, permettant des portefeuilles HD multidevises où chaque devise a son propre sous-arbre sous le deuxième niveau. Il y a trois devises définies pour l'instant : Bitcoin est m/44'/0', Bitcoin Testnet est m/44&#x27;/1&#x27;, et Litecoin est m/44&#x27;/2&#x27;.

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

Au quatrième niveau, "change", un portefeuille HD a deux sous-arborescences, une pour créer des adresses de réception et une pour créer des adresses de modification. Notez qu'alors que les niveaux précédents utilisaient une dérivation renforcée, ce niveau utilise une dérivation normale. Cela permet à ce niveau de l'arborescence d'exporter des clés publiques étendues pour une utilisation dans un environnement non sécurisé. Les adresses utilisables sont dérivées par le portefeuille HD en tant qu'enfants du quatrième niveau, faisant du cinquième niveau de l'arborescence l'"index\_adresse". Par exemple, la troisième adresse de réception pour les paiements en bitcoins dans le compte principal serait M/44&#x27;/0&#x27;/0&#x27;/0/2. [table\_title](#table_4-9) montre quelques exemples supplémentaires.

Exemples de structure de portefeuille HD BIP-44

| Chemin HD | Clé décrite |
| --- | --- |
| M/44&#39;/0&#39;/0&#39;/0/2 | La troisième clé publique de réception pour le compte bitcoin principal |
| M/44&#39;/0&#39;/3&#39;/1/14 | La quinzième clé publique de changement d'adresse pour le quatrième compte bitcoin |
| m/44&#39;/2&#39;/0&#39;/0/1 | La deuxième clé privée du compte principal Litecoin, pour la signature des transactions |

### Utilisation d'une clé publique étendue sur une boutique en ligne

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

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

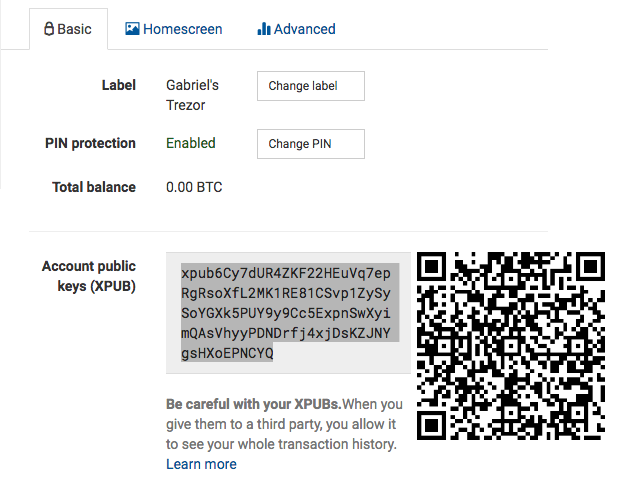
Gabriel a utilisé la première adresse Bitcoin générée par son appareil Trezor comme adresse Bitcoin principale pour son magasin. De cette façon, tous les paiements entrants seraient versés à une adresse contrôlée par son portefeuille matériel Trezor.

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

Cependant, la petite boutique en ligne a connu un certain succès et a attiré de nombreuses commandes de la communauté locale. Bientôt, Gabriel a été submergé. Avec toutes les commandes payées à la même adresse, il devenait difficile de faire correspondre correctement les commandes et les transactions, en particulier lorsque plusieurs commandes pour le même montant arrivaient à proximité.

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

Pour exporter le xpub, Gabriel utilise l'application de bureau Trezor Suite en conjonction avec le portefeuille matériel Trezor. L'appareil Trezor doit être branché pour que les clés publiques soient exportées. Notez que les portefeuilles matériels n'exporteront jamais de clés privées, celles-ci restant toujours sur l'appareil. [figure\_title](#export_xpub) montre ce que Gabriel voit dans Trezor Suite lors de l'exportation du xpub.



Exportation d'un xpub à partir d'un portefeuille matériel Trezor

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

#### Découverte et gestion de compte

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

À un moment donné, le site Web de Gabriel peut avoir un grand nombre d'adresses en attente pour les clients effectuant des achats, dont certaines peuvent rester inutilisées et éventuellement expirer. Une fois ces adresses expirées, *BTCPay Server* réutilisera ces adresses pour combler le vide dans l'*index\_adresse*, mais il devient clair qu'il peut y avoir des vides entre les feuilles d'*index\_adresse* de l'arbre déterministe hiérarchique où se trouve réellement l'argent.

Disons que Gabriel est intéressé à voir son montant total de bitcoin gagné sur un portefeuille de surveillance uniquement (celui qui vous permet de voir l'historique des transactions, mais pas de dépenser des fonds) qui est distinct du serveur BTCPay mais également conforme à la norme BIP-44 . Comment ce portefeuille séparé devrait-il rechercher des fonds dans ce vaste arbre hiérarchique, et quand devrait-il cesser de chercher ? La plupart des portefeuilles suivent généralement un processus itératif qui utilise une limite prédéfinie, connue sous le nom de *gap limit*. Si, lors de la recherche d'adresses utilisées, le portefeuille ne trouve pas d'adresses utilisées d'affilée au-delà de ce nombre limite, il arrête de rechercher la chaîne d'adresses. La limite d'écart par défaut est généralement fixée à 20. Ceci est détaillé dans [BIP-44](https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki).

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