Hashcoding et Blockchain

c.f. https://fr.wikipedia.org/wiki/Fonction_de_hachage

Définition :
Une **fonction de hachage** est une fonction *Hash* qui à un nombre entier positif pouvant être très grand (des millions de chiffres) fait correspondre un nombre entier positif inférieur à un nombre donné.
Ceci n'est que le début de la définition.

Un exemple très simple est :
Hash (*n*) = *n* % 787

n % 787 est le reste de la division du nombre *n* par 787.

Définition complétée, pour son utilisation dans une Blockchain :
Une **fonction de hachage** est une fonction *Hash* qui à un nombre entier positif pouvant être très grand (des millions de chiffres) fait correspondre un nombre entier positif inférieur à un nombre donné.
Elle doit satisfaire en plus les conditions suivantes :
1) Elle doit être "facile" à calculer.
2) pour un nombre *s* donné, il doit être très difficile de trouver une valeur *n*, telle que *Hash* (*n*) = *s*.
(*s* comme *s*ignature)

Autrement dit, calculer *Hash* doit être facile, mais aller dans l'autre sens doit être difficile.

De cette manière, une fonction de hachage peut servir de **signature d'un document numérique**.
n correspondrait au document numérique.
s = *Hash* (*n*) correspondrait à la signature du document.
° La signature "*s*" est facile à obtenir, car la fonction est "facile" à calculer.
° Créer une falsification du document numérique "n_2" ayant la même signature est difficile, car cela revient à aller dans l'autre sens.
La signature "*s*" étant donné, il est difficile de trouver n_2 tel que *Hash* (n_2) = *s*.

Pour information :
Les exemples standards de fonctions de hachage (hashcoding) utilisé pour signer des documents sont : MD5 ; SHA-1 ; SHA-256 et SHA-512.
(Les deux premières sont déconseillées de nos jours, car moins sûr que les deux suivantes.)
SHA-256 crée une signature de 256 bits, ce qui donne un nombre allant jusqu'à _____

SHA-512 crée une signature de 512 bits, ce qui donne un nombre allant jusqu'à _____

Ces fonctions sont considérées comme facile à calculer.
Regardez leur algorithme de calcul et vous verrez que "facile" n'est pas le terme qui serait utilisé par tout le monde. c.f. https://fr.wikipedia.org/wiki/SHA-2#SHA-256

En page suivante, une fonction de hachage plus simple à calculer est donnée. Son défaut est de ne pas être si difficile à inverser pour un professionnel.

Un exemple de fonction de hachage, donné sous forme d'algorithme.
Un **algorithme** est une suite d'instructions.

Soit *n* le nombre à "hacher".
Suite d'instructions dans un pseudo code :
hash = 0
Répéter 10 fois :
  chiffre = reste de la division de n par 10
  hash = (hash + chiffre) * 384049
  hash = reste de la division de hash par 7538492917
  n = arrondi vers le bas de (n / 10)

retourner hash ; *donc* hash = *Hash (n).*

Exercice 1
Allez dans : http://www.juggling.ch/gisin/program/blockly/abgex/ex2000/ex2000.html
Configuration > Divers exemples, au choix... > Choix 3
° Programmez une fonction "Hash(n)", selon l'algorithme précédent qui calcule la fonction de hachage du nombre n.
° Tester votre fonction sur : n = 314159265 vous devez obtenir Hash (n) = 3062988386
° Comparez votre fonction avec celle d'autres élèves.
° Trouvez un autre nombre n_2 tel que sa fonction de hachage Hash (n_2) = 3062988386

Blockchain

c.f. https://www.youtube.com/watch?v=SccvFbyDaUI
c.f. mieux (anglais) https://www.youtube.com/watch?v=bBC-nXj3Ng4&t=35s
Supposons avoir un nombre *Big*, qui contient une information. Il peut être très grand.
On cherche un nombre que l'on nommera "*preuve*" tel que le nombre *n* formé des chiffres de *Big* suivit des chiffres de *preuve* satisfait :
° *Hash* (*n*) donne un nombre plus petit que 10'000.
(10'000 pouvant être remplacé par n'importe quelle autre limite.)

La difficulté est de trouver un nombre "*preuve*" pour que *Hash* (*n*) soit plus petit qu'une limite donnée.
Le travail des **mineurs** est de trouver un nombre "*preuve*".

La **blockchain** est une suite de nombre (très grands) B_1 ; B_2 ; B_3 ; B_4 ; etc. (*B* pour Bloc)
Chaque nombre (Bloc) B_j est formé en mettant les chiffres des trois nombres suivants à la suite.
° Le Hash du bloc précédent, *Hash* (B_{j-1})
° Une information convertie sous forme de nombre I_j
° Une "*preuve*" de travail, qui est telle que le Hash de ce bloc B_j soit plus petit qu'une valeur donnée.

Exemple :

	Hash($B_{précédent}$)	Information	Preuve telle que Hash(Bloc) < limite	
B_1 =	0000000000	0000000000	0000000043	Hash(B_1) = 0096794702 < 10^8. C'est le premier bloc.
B_2 =	0096794702	1414213562	0000001119	Hash(B_2) = 0004889573 < 10^7.
B_3 =	0004889573	1732050808	0000006817	Hash(B_3) = 0000793420 < 10^6.
B_4 =	0000793420	0314159265	0000060600	Hash(B_4) = 0000029765 < 10^5.
B_5 =	0000029765	2236067978	0000232005	Hash(B_5) = 0000000264 < 10^4.
B_6 =	0000000264	2449489743	0017661415	Hash(B_6) = 0000000009 < 10^3.
B_7 =	0000000009	2645751311	0140302748	Hash(B_7) = 0000000004 < 10^2.
B_8 =	0000000004	2828427125	0029811788	Hash(B_8) = 0000000020 < 10^2.

Habituellement, l'information est beaucoup plus longue. Tous les nombres sont plus grands habituellement.
Les calculs de "preuve" deviennent de plus en plus longs, car la contrainte de preuve de travail est de plus en plus restrictive, il faut que le Hash soit de plus en plus petit.