# Table des matières

# Abstract

La cryptographie est un domaine extrêmement sensible. Les données envoyées et reçues se doivent d'être confidentielles au mieux possible. Toutes les fonctions de hachage sont soit lourdes à calculer même pour les plus petites données, soit non fiables et non sécurisées. Alors en m'inspirant de SHA-256, et du Multi-Layer Perceptron en Deep Learning, j'ai émis mon propre algorithme de hachage : La fonction de hachage de la chaîne d’or. Son nom est tiré d’une métaphore dans laquelle chaque bout de la chaîne est une partie du message à chiffrer, et toute la « chaîne d’or » forme une encryption sécurisée de toutes les données.

# Contexte de création

Modulo est un outil puissant en cryptographie. Dans le cas du hachage, il permet de "cacher" les entrées, en permettant cependant beaucoup de collisions.

Alors il m'a semblé évident de réduire au mieux que possible les collisions afin de créer une fonction de hachage. Ainsi je me suis pris de la façon suivante :  
Pour un texte équivalent à une matrice de 4 entiers naturels, a, b, c, d :

On a hash(a,b,c,d) = f2( f(a,b), f(c,d) ).

Pour mon cas, j'ai essayé : f(a,b) = a % b et f2(a, b) = a + b

En essayant des entrées, j'ai tout de suite compris que ma fonction de hachage comportait trop de collisions.

Ayant eu plusieurs cours d'intelligence artificielle et notamment de Deep Learning : j'ai décidé de m'en inspirer.  
Un réseau de Multi-Layer Perceptron calcule pour chaque neurone : la somme des entrées multipliées par les poids du neurones, le tout subissant une fonction d'activation.

De plus, après mon premier échec, j'ai compris qu'il était préférable de mélanger toutes les entrées ensemble : chaque entrée devrait avoir alors une position particulière par rapport aux autres.

Donc, j'ai mis alors en place un algorithme qui, prenant plusieurs entrées (nombres réels), ne renvoie qu'une sortie (nombre réel) : la fonction de hachage à la chaîne d’or.

# Développement

## Introduction

Tout d’abord, voici ci-dessous un aperçu simple des calculs de la fonction de hachage à la chaîne d’or :  
Avec a,b,c,d sont des entrées naturelles représentant l’encodage d’une lettre.  
SHF(a,b,c,d) = (a%b + a%c + a%d)(b%a + b%c + b%d)(c%a + c%b + c%d)(d%a + d%b + d%c)

Cette fonction de hachage est la plus simple qui puisse exister, et la plus légère qui soit.  
Elle permet de réduire le temps de calcul, ainsi que l’espace de stockage utilisé notamment pour des entrées relativement courtes : par exemple des mots de passe.

## Collision par similarité des entrées

Cependant, une sorte de collision se dissimule. En effet, si les entrées (lettres encodées) d’un appel de fonction sont exactement les mêmes d’un autre appel de fonction mais dans un ordre mélangé, alors le résultat en sortie est le même.  
Exemple : « code » et « deco », encodés respectivement [1,2,3,4], et [3,4,1,2]  
SHF(1,2,3,4) = 4 x 5 x 6 x 4 = 280  
SHF(3,4,1,2) = 6 x 4 x 4 x 5 = 280

Ainsi, cela peut convenir ou non selon les besoins.

Pour résoudre ce problème, il est essentiel d’ajouter une entrée unique et spécifique pour chaque appel de fonction.

Je propose ainsi un ajout d’entrée aléatoire ou mathématique ou de concaténation. L’ajout d’entrée mathématique devra être un entier naturel. Pour ma part, je valoriserais la concaténation. Cette dernière met en valeur et dévoile l’ordre des lettres. Cependant, elle résulte souvent en un nombre trop grand. Ainsi, pour la réduire pour propose de soustraire cette concaténation de chiffres (qui est un nombre) par la multiplication de toutes les entrées. Et, le tout en valeur absolue.

added\_entry = | int(concat({entries})) – (∏({entries\_k})) |

Avec cette nouvelle entrée, nous pouvons en rajouter une nouvelle, etc.

Enfin, grâce à ce calcul, nous pouvons éviter la collision par similarité des entrées.

## Collision par répétition d’entrées

Il advient de même que des collisions particulières trahissent la fonction de hachage à la chaîne d’or : si une seule même entrée compose le message, ou si cette dernière est répétée n fois. En effet, la « chaîne d’or » ne tient que sur plusieurs maillons. Or, la répétition d’une seule lettre provoque une instabilité de la chaîne ne tenant que sur un maillon. Ainsi, pour résoudre ce problème, il advient de mettre en valeur le défaut de répétition en avantage.

Pour n (entier naturel) répétition de X nombre naturel dans l’ensemble A des entrées :

On a pour k allant de 0 à n – 1 : X\_k := X\_k × k

Enfin cette affection pour chaque variable X\_k permet de supprimer toute redondance.

## Conclusion : formule complète

Ainsi, à l’aide de ces modifications, la fonction de hachage à la chaîne d’or fait bloque à toutes les collisions possibles.

La formule de cette dernière subissant toutes les modifications requises est la suivante :

Pour A un ensemble d’entiers naturels (représentant des lettres sous l’encodage voulu).

Si et seulement si A contient des mêmes entiers naturels a :

On note X l’ensemble de ces entiers naturels a :

(J’ai volontairement choisi de mélanger les lettres similaires en un même ensemble.)

n = Card(X)

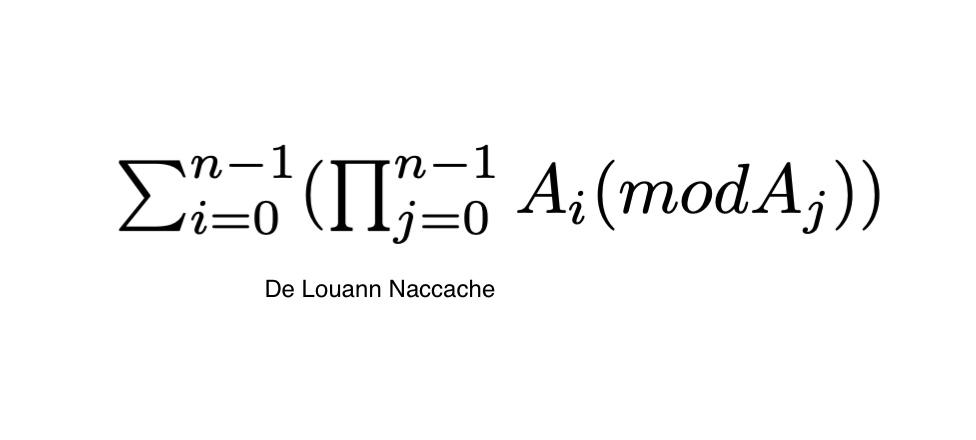
Avec l’entier naturel n, représentant la longueur de X.

Pour k allant de 0 à n – 1 : X\_k := X\_k × k

Pour k allant de 0 à n – 1 et avec j := A.index(a\_k) : A\_j := X\_k

Avec b = | int(concat(A)) – (∏(A\_k)) |

On a A := A ∪ {b}



chaîne\_d\_or =

# Conclusion finale

Ainsi la fonction de hachage à la chaîne d’or associe la rapidité de calculs et la sécurité ensemble. En effet, peu importe le nombre d’entrées, la fonction de hachage assure une sécurité sans pareille. De plus, les efforts de calculs sont variables.

Il est d’autant plus essentiel de comprendre que cette fonction de hachage s’adapte selon les besoins : si le particulier ou l’entreprise a besoin d’une sécurité, il peut alors étendre les données envoyées jusqu’à 20 ou plus (ce qui est très sécurisé). En revanche, si ce dernier préfère réduire l’espace de stockage utilisé, il peut utiliser la fonction de compression de la chaîne d’or (une fonction aussi imaginée par moi).

# Recommandations et quelques mots sur moi

Je conseille à tous d’utiliser la fonction de hachage à la chaîne d’or pour le hachage de mot de passe ou de tout texte de 6 à 12 caractères. En effet, la fonction de hachage à la chaîne d’or est à son paroxysme de performance de 6 à 12 caractères, ce qui est idéal pour les mots de passe notamment.

Pour ma part, je suis passionné d’informatique et de science des calculs. Je maîtrise la cryptographie et notamment les mathématiques qui donnent vie au RSA et AES. Évidemment, je maîtrise aussi SHA-256. Je suis des formations d’informatique et de science de calculs dont de cryptographie. Cette passion s’est déclaré en moi assez tôt, bien qu’âgé que de 16 ans.