Mémoire :
Au cœur du chiffrement avec les fonctions de hachage

Defraiteur Maxence

Décembre 2022 - Mai 2023

Sous la supervision de Volker Mayer

M1 Master MEEF

Introduction

Traditionnellement, la monnaie au format papier a besoin d'encres et de matériaux spéciaux pour éviter la contre-façon. Un problème majeur lié à l'ordinateur repose sur la copie et l'altération des données. Dans une société où les informations sont stockées et transmises électroniquement, il faut assurer la sécurité de l'information. Un outil fondamental utilisé dans la sécurité de l'information est la signature : utile pour le non-rejet, origine de l'authentification des données, identification, certificat d'authenticité. Cette signature doit tenter d'être unique. La cryptographie vise à assurer de manière sécurisée : la confidentialité, l'intégrité des données, l'authentification et le non-rejet de service via des opérations mathématiques. Les fonctions de hachage sont des fonctions répondant à ces enjeux. Elles permettent d'identifier rapidement si deux données sont identiques ou non.

J'ai choisi ce sujet de mémoire étant donné que le domaine de la cryptographie a toujours attisé ma curiosité. Je souhaitais connaître plus en profondeur les fondements et fonctionnements des algorithmes cachés dans notre vie de tous les jours.

Table des matières

1 Histoire sur l'origine du chiffrement

1.1 Généralités

La cryptographie est l'étude des techniques mathématiques en lien avec les communications secrètes. C'est un procédé qui est devenu commun dans la vie moderne, établissant une connection entre des appareils digitaux. On appelle cela le « handshaking [1] ». L'invention du *handshaking* est attribuée aux trois mathématiciens : Ron **R**ivest, Adi **S**hamir et Leonard **A**dleman. En 1977, ils développèrent l'algorithme **RSA** (cette procédure de cryptage leur fit gagner le Turing Award en 2002). On retrouve couramment la cryptographie dans le domaine des finances. Le cryptage est utilisé pour empêcher la compréhension des données par une personne tiers.

1.2 Etymologie

Le terme cryptographie vient du Grec « étude de l'écriture cachée ». Très souvent la cryptographie était utilisée pour sécuriser des messages. Dans ce cas, on peut distinguer deux types de textes : le texte brut et le texte crypté (de l'anglais *cipher*). Le code secret (cipher) est un algorithme. Par exemple, le chiffrement de César est un système de substitutions des lettres. Le chiffrement de César datant du Ier siècle av. J-C est un exemple de cryptage symétrique, de sorte que la même clé et code sont utilisés pour décrypter le message. Le code César est facilement cassable à la main ou avec un ordinateur en essayant toutes les possibilités de substitution. On appelle, ce type de décryptage « force brute ». Les codes et clés modernes rendent difficile la tâche de décryptage par force brute.

L'émergence de la cryptographie a permis de protéger des données sensibles dès l'Antiquité. À cette époque, les codes et clés étaient facilement trouvables. À travers les siècles, les codes sont devenus difficiles à casser grâce notamment aux calculs sur ordinateur. Les codes modernes sont presques impossibles à craquer. Ainsi, la cryptographie permet de transmettre des données en sécurité. Par ailleurs, les longs messages étaient vulnérables en utilisant la technique de décrytage par analyse fréquentielle. Le décryptage par analyse fréquentielle a été initié par le mathématicien arabe al-Kindi au IX$^{\text{ème}}$ s. Cette technique employait la fréquence de chaque lettre de l'alphabet les plus souvent utilisées dans le langage particulier. Cette technique de décryptage cassait tous les codes de substitutions. Pour palier à l'analyse fréquentielle, il faut dissimuler le texte brut en utilisant un « code ». Pour améliorer le niveau de sécurité, on peut utiliser un polyalphabet secret où une lettre d'un texte brut peut être substituée à plusieurs lettres d'un texte se-

1. Processus automatique pour établir l'entrée en communications de deux entités

cret. Cela supprime la possibilité des attaques par analyse fréquentielle. Ces types de codes ont été développés au XVI$^{\text{ème}}$s. mais le plus connu a été produit par la machine Enigma durant la seconde guerre mondiale. Cette machine a été utilisée par les services d'espionnage allemands entre 1923 et 1945. La machine Enigma possédait 158 962 555 217 paramétrages possibles, qui étaient changés tous les jours. Pour le cryptage symétrique, il fallait transmettre physiquement la clé rendant ainsi vulnérable la transmission du message crypté. Le cryptage asymétrique rentra en jeu dès l'émergence des ordinateurs dans les années 1960. Sur de grandes distances, les gens peuvent se transmettre des informations sans interagir entre eux physiquement. Cependant, avec Internet, le réseau est publique donc toutes les clés symétriques peuvent être interceptées. Dans les années 1970, IBM adopte le *US Federal Information Processing Standard* pour crypter les informations déclassées. En 1976, un changement majeur s'est produit lorsque Diffie et Hellman ont publié des nouvelles directions dans la cryptographie. Ils ont introduit la clé publique, étant une nouvelle méthode pour échanger les clés. Puis en 1978, Rivest, Shamir et Adleman ont découvert la première clé publique utile dans la cryptographie et la signature digitale. En 1991, la première signature digitale ISO/IED est réalisée.

1.3 Algortihmes et protocoles

1.4 Algorithmes de chiffrement faible

Les premiers algorithmes de chiffrement reposaient sur le remplacement des caractères par d'autres.

— ROT_{13} (rotation de 13 caractères, sans clé)

— Chiffre de César (décalage de 3 lettres dans l'alphabet sur la gauche)

— Chiffre de Vigenère (introduit la notion de clé)

1.5 Algorithmes de cryptographie symétrique (à clé secrète)

Principe : Une même clé pour chiffrer et déchiffrer un message. La transmission de la clé doit être faite de manière sûre.

— Chiffre de Vernam (le seul offrant une sécurité théorique absolue, à condition que la clé ait au moins la même longueur que le message à chiffrer, qu'elle ne soit utilisée qu'une seule fois et qu'elle soit aléatoire). Gilbert Vernam a publié ce chiffre en 1926. On appelle cela un « masque jetable » et il n'y a pas besoin d'ordinateur. Claude Shannon prouva plus tard en 1949 que le chiffre de Vernam est parfaitement sûr.

1.6 Algorithmes de cryptographie asymétrique (à clé publique et privée)

Les algorithmes de cryptographie asymétrique résolvent le problème de l'échange de clés. Ce type d'algorithme a émergé dans les années 1970. La clé publlique permet le chiffrement et la clé privée permet le déchiffrement. La clé privée reste confidentielle. Voici quelques exemples de ce type d'algorithme :

— Protocole d'échange de clés Diffie-Hellman (1977)

— RSA pour Rivest, Shamir et Adleman (1977)

— DSA signifiant Digital Signature Algorithm (1993)

— Cryptographie sur les courbes elliptiques (1985) (Normalisation faite devant la loi)

— Algorithme de Schoof (compte les points de courbes elliptiques sur les corps finis)

— Algorithme de LUC (suites de Lucas)

1.7 Algorithme RSA

L'algorithme RSA a permis de développer le crytage asymétrique où l'envoyeur et le receveur utilisent 2 clés.

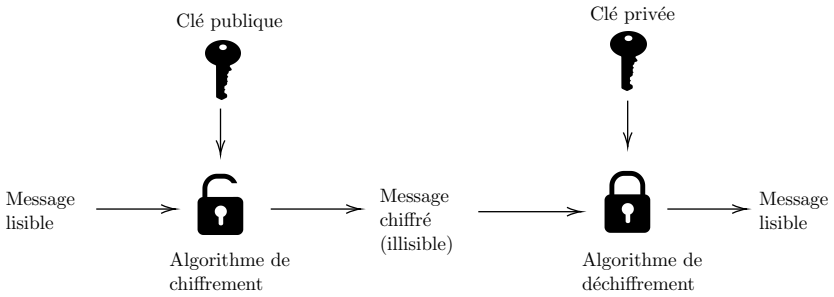

FIGURE 1 – Principe de l'agorithme RSA

Les inconvénients du chiffrement RSA et des autres algorithmes à clé publique résident dans la grande lenteur de processus par rapport aux algorithmes à clés secrètes. La plupart des algorithmes de cryptographie asymétrique sont vulnérables à des attaques utilisant un calculateur quantique, à cause de l'algorithme de Shor. La cryptographie post-quantique tend à résoudre ce problème. [2]

2. Site de la NIST (National Institute of Standards and Technology), Cryptography in the Quantum Age

2 Fonctions

2.1 Les fonctions au sens large

Définition

Une fonction $f : X \to Y$ est dite **injective** si pour tout $y \in Y$, il existe au plus un $x \in X$ tel que $f(x) = y$. Dit autrement, si

$$\forall x, x' \in X, x \neq x' \Rightarrow f(x) \neq f(x')$$

Définition

Si $f : X \to Y$ est une bijection et g est une bijection de Y vers X : pour chaque $y \in Y$ définissent $g(y) = x$ où $x \in X$ et $f(x) = y$. Cette fonction g obtenue de la fonction f est appelée la *fonction inverse* et est notée $g = f^{-1}$.

Remarque

Si f est une bijection alors f^{-1} l'est aussi. En cryptographie, les bijections sont utilisées comme outil pour crypter des messages et inverser des transformations utilisées pour le décryptage. Si les transformations ne sont pas des bijections alors il n'est pas toujours possible de décrypter un message de façon unique.

Définition

Soit une fonction $f : X \to Y$ est appelée **fonction à sens unique** si $f(x)$ est « facile » à calculer pour tous les $x \in X$ mais pas « essentiellement » pour tous les $y \in \text{Im}(f)$ dont le calcul est infaisable pour trouver un $x \in X$ vérifiant $f(x) = y$.

Définition

Une fonction $f : X \to Y$ est appelée **fonction à sens unique à trappe** (TDF [a]), si elle est une fonction à sens unique ayant la propriété de donner une information supplémentaire : *l'information à trappe* rendant faisable le calcul pour trouver n'importe quel $y \in \text{Im}(f)$, d'un $x \in X$, où $f(x) = y$.

a. Trapdoor function

Exemple

Un nombre premier est un entier naturel strictement supérieur à 1 dont les seuls diviseurs sont 1 et lui-même. Prenons deux nombres premiers $p = 23189$ et $q = 43991$. Le nombre $n = pq$ vaut 1020107299 et X est l'ensemble $X = \{1, 2, \cdots, n-1\}$. Il est relativement « facile » [a] d'inverser cette fonction pour ces nombres. Par contre, s'il on prend des nombres ayant plus de 100 chiffres, le problème est rendu impossible sur un espace de temps fini pour factoriser n avec les ordinateurs actuels. Cela s'appelle le **problème de factorisation des entiers** et est la source d'une mutltitude de fonction à sens unique à trappe. Qui plus est, on ne connaît pas actuellement d'algorithme de factorisation en temps polynomiale.

a. Le terme « facile » désigne un temps de calcul raisonnable pour une machine moderne, de complexité algorithmique polynômiale.

2.2 Fonctions de hachage

2.2.1 Introduction

Les fonctions de hachage ont une utilité importante dans la cryptographie moderne. Elles jouent un rôle essentiel dans la sécurité des données, la finance, les crypto-monnaies et le téléchargement de fichier volumineux par exemple. Le principe d'une fonction de hachage est de renvoyer un *hash-code, hash-result, hash-value, hash, haché (en français)* à partir d'un message.

Les fonctions de hachage (abbrégées **FDH** dans la suite du document) doivent être déterministes, c'est-à-dire pour une même entrée, elles doivent toujours produire le même haché.

La fonction de hachage h désigne une chaîne de n-bits de longueur arbitraire vers une longueur fixée, dites n bits[3]. Un n-bit est un élément de l'ensemble $\{0,1\}^n = R$. Une FDH associe un texte de longueur arbitraire à un hash de longueur n-bit. D représente l'ensemble des textes possibles. Introduisons la fonction $h : D \to R$ avec $|D| > R$. h est une fonction de hachage. On appelle les images par h les *hash-value, hash, haché.*

Exemples : Les fonctions MD_4, MD_5, SHA_{256} sont des fonctions de hachages.

Comme les textes de D ont des longueurs arbitrairement grandes[4] alors que R est un ensemble fini. (de 2^n éléments). La fonction h n'est pas injective.

La fonction h renvoie une « image compacte ». En d'autres termes, l'un des intérêts du hachage est que l'image texte par h, soit le haché, est une représentation condensée du texte initial. Donc le haché est une chaîne de caractères réduite par rapport au texte initial mais comme on le verra par la suite, les hachés rassemblent suffisament d'informations pour retrouver le texte initial.

La non injectivité de cette fonction implique l'existence de collisions (deux hashs identiques avec des messages différents). Des collisions peuvent être testées par le paradoxe des anniversaires (expliqué ultérieurement dans la section 2.3). Les fonctions de hachage permettent d'obtenir des *hachés* servant de représentation d'image compacte. On les appelle aussi *imprint, digital fingerprint* ou *message digest* d'une entrée de caractères et ces *hachés* peuvent être utilisées comme si

3. un bit correspond à la plus petite unité d'information en informatique pouvant avoir deux valeurs : 0 ou 1.

4. Dans certains contextes, les textes ou bases de données initiaux sont de petites tailles comme pour les mots de passe par exemple. Néanmoins, les algorithmes des fonctions de hachage comprennent une étape de normalisation (dite « padding » signifiant remplissage). L'étape de padding consiste à ajouter des bits supplémentaires à l'entrée de manière à ce que sa longueur soit un multiple de n de la taille de bloc de la fonction de hachage. Cela est généralement nécessaire pour garantir que l'entrée puisse être traitée de manière cohérente par la fonction de hachage, quelle que soit sa longueur car $|D| > R$ par définition .

elles étaient uniquement identifiables avec ce nombre fini de caractère.

Les fonctions de hachage sont utilisées pour préserver l'intégralité des données avec les procédés de signatures digitales. En pratique, un message est d'abord haché puis sa *hash-value* est signée à la place du message initial.

Illustrons ce qu'il se passe en pratique par le schéma ci-dessous.

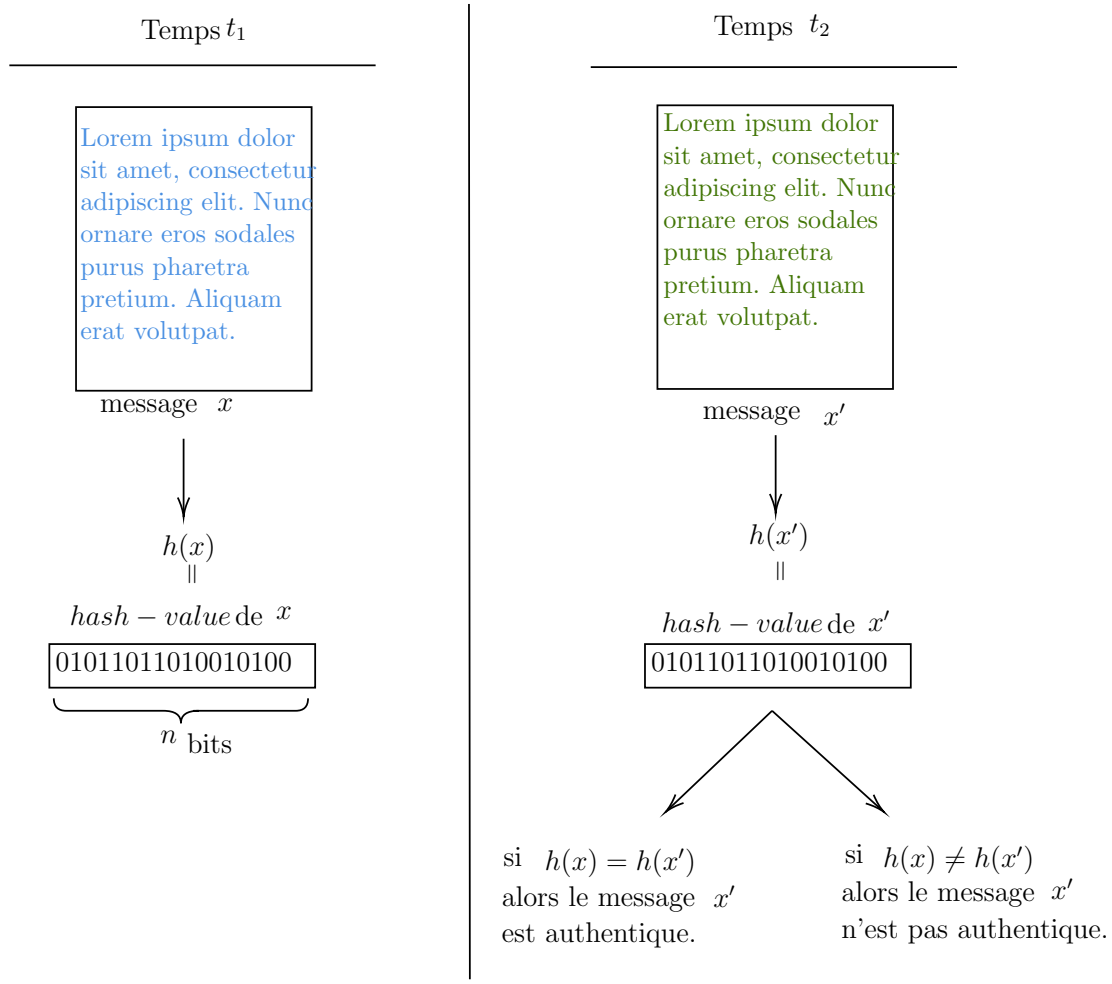

FIGURE 2 – Fonctionnement d'une fonction de hachage

Le **problème** qui se pose est la non injectivité des fonctions de hachage. Il faut ainsi trouver le n qui convient puis discuter de la taille des messages initiaux en fonction de la probabilité afin d'obtenir des collisions. On peut ainsi se demander dans quelle mesure les fonctions de hachage sont suffisament fortes pour palier au problème des collisions.

Lorsqu'une fonction de hachage produit un **hash identique**, cela ne signifie pas nécessairement que les données initiales sont identiques. Cela est dû au fait

que les fonctions de hachage sont conçues pour produire des hachés de taille fixe (n), quelle que soit la taille de l'entrée. (de longueur t). Ces collisions peuvent être intentionnelles ou accidentelles. Dans le cas des collisions intentionnelles, un attaquant peut chercher à trouver deux entrées différentes qui produisent le même haché afin de tromper le système. Dans le cas des collisions accidentelles, il s'agit simplement d'une coïncidence due à la nature probabiliste des fonctions de hachage. Bien que la probabilité de collision soit très faible, elle n'est pas nulle.

Lorsqu'une fonction de hachage renvoie des **hashs différents**, cela implique qu'il est fort probable que les textes soient différents. Une fonction de hachage est conçue pour avoir une faible probabilité que deux entrées produisent le même haché.

Exemple : Pour la fonction de hachage **SHA-256**, celle-ci produit un haché de 256 bits. Il existe donc 2^{256} car pour chaque bit, il y a deux choix possibles : 0 ou 1 (soit environ 10^{77} combinaisons possibles de hachés différents). Cette immense quantité de combinaisons rend pratiquement impossible la création de deux entrées différentes produisant le même haché.

2.2.2 Procédés mathématiques utilisés par les FDH

Les fonctions de hachage confrontent plusieurs théories mathématiques comme la théorie des groupes, la théorie des nombres, la théorie de l'information, la théorie de la complexité algorithmique et la théorie des probabilités.

Voici l'explication détaillée des concepts mathématiques sous-jacents aux FDH :

- **la théorie des groupes** : les fonctions de hachage utilisent souvent des opérations sur les groupes, telles que l'addition modulo un nombre premier, pour mélanger les bits d'un message avant de le hacher. Les groupes abéliens et non abéliens sont particulièrement utiles pour la conception de fonctions de hachage cryptographiques.
- **la théorie des nombres** : les fonctions de hachage utilisent souvent des propriétés des nombres premiers et des fonctions arithmétiques pour créer des hashs uniques. Par exemple, les fonctions de hachage cryptographiques utilisent souvent des nombres premiers aléatoires pour diviser le message en blocs et les hacher séparément.
- **la théorie de l'information** : les fonctions de hachage sont utilisées pour réduire la taille d'un message tout en préservant autant que possible l'information qu'il contient. Les fonctions de hachage sont conçues pour minimiser les collisions, c'est-à-dire les situations où deux messages différents produisent le même hash.
- **la théorie de la complexité algorithmique** : les fonctions de hachage sont conçues pour être efficaces à calculer, mais difficiles à inverser. Les fonctions de hachage sont souvent mesurées en termes de leur résistance aux attaques

de collision et aux attaques d'antécédents, qui sont toutes deux liées à la complexité des algorithmes nécessaires pour les casser.

— **la théorie des probabilités** : les fonctions de hachage sont sujettes à des collisions, c'est-à-dire des situations où deux messages différents produisent le même hash. Les fonctions de hachage sont conçues pour minimiser les collisions en utilisant des techniques telles que le mélange des bits du message, l'utilisation de nombres premiers aléatoires et la sélection d'une taille de hash appropriée.

Exemple de structure d'algorithme utilisé dans les fonctions de hachage :

— **Compression** : une FDH prend une entrée initiale qui peut être de taille variable et la comprime en une forme fixe. La compression est effectuée en plusieurs étapes : substitution, permutation, addition modulo 2.

— **Clé de hachage** : certaines FDH nécessitent une clé de hachage qui est un paramètre fixe, qui peut être utilisé pour personnaliser la FDH en fonction des besoins de l'utilisateur.

— **Bouclage** : la compression est effectuée plusieurs fois dans une boucle. Cela permet à la FDH de prendre en compte toutes les parties de l'entrée, quelque soit leur ordre.

— **Clé de hachage** : après la compression, la FDH peut appliquer une série d'opérations supplémentaires pour finaliser le haché. Cela peut inclure des opérations tel que la concaténation des bits, l'application des fonctions non linéaires et la réduction de la taille du haché.

2.2.3 Classification des familles

Une famille de ces fonctions de hachage est nommée **MACs** pour ***M**essage **A**uthentication **C**odes*, autorisant l'authentification des messages par des techniques symétriques.

Principe des algorithmes MACs :

— Entrée : un message et une clé tenue secrète.

— Sortie : une chaîne de caractère de n-bits.

Il est infaisable en pratique de produire le même output (haché) sans connaître la clé.

Exemples :

— **HMAC** (**H**ash-based **M**essage **A**uthentication **C**ode) : HMAC est une technique de calcul de code d'authentification de message qui utilise une fonction de hachage cryptographique (comme SHA-256, SHA-384 ou SHA-512) en combinaison avec une clé secrète partagée entre les parties qui communiquent. HMAC fournit à la fois l'authentification et l'intégrité des données.

— **Poly1305** : Poly1305 est une technique de calcul de code d'authentification de message qui utilise un algorithme de chiffrement à flux en combinaison avec une clé secrète partagée entre les parties qui communiquent. Poly1305 fournit à la fois l'authentification et l'intégrité des données.

— **GCM** (**G**alois/**C**ounter **M**ode) : GCM est une technique de chiffrement de blocs qui intègre également une fonction de calcul de code d'authentification de message. GCM utilise une fonction de hachage cryptographique (comme SHA-256 ou SHA-512) pour fournir l'authentification et l'intégrité des données. GCM est souvent utilisée pour les communications sécurisées via Internet (par exemple, dans les protocoles TLS/SSL).

À noter que les fonctions de hachage utilisées dans les MAC sont souvent les mêmes que celles utilisées dans les MDC (terme défini dans le paragraphe qui suit), mais les MAC utilisent ces fonctions de hachage d'une manière différente pour fournir l'authentification et l'intégrité des données.

Une autre famille est nommée **MDCs** pour ***M****odification* ***D****etection* ***C****ode*.

Exemples :

— **MD5** (**M**essage **D**igest 5) : MD5 est une FDH à sens unique et à résistance à la collision. Elle produit un hachage de 128 bits pour un message d'entrée donné. Bien que MD5 soit encore largement utilisée, elle est considérée comme peu sécurisée pour les applications critiques en raison de faiblesses découvertes dans sa conception.

— **SHA-1** (**S**ecure **H**ash **A**lgorithm 1) : SHA-1 est une fonction de hachage à sens unique et à résistance à la collision. Elle produit un hachage de 160 bits pour un message d'entrée donné. Tout comme MD5, SHA-1 est maintenant considérée comme peu sécurisée pour les applications critiques.

— **SHA-2** (**S**ecure **H**ash **A**lgorithm 2) : SHA-2 est une famille de fonctions de hachage cryptographiques à sens unique et à résistance à la collision. Elle comprend des fonctions de hachage telles que SHA-256, SHA-384 et SHA-512. Ces fonctions de hachage sont beaucoup plus sécurisées que MD5 et SHA-1 et sont couramment utilisées pour la signature numérique, la vérification d'intégrité des données, la génération de clés aléatoires, etc.

— **SHA-3** (**S**ecure **H**ash **A**lgorithm 3) : SHA-3 est une famille de fonctions de hachage cryptographiques à sens unique et à résistance à la collision. Elle est basée sur le nouveau standard de l'Institut National des Standards et de la Technologie (NIST) appelé Keccak. SHA-3 est conçue pour être plus résistante aux attaques cryptographiques que SHA-2.

Propriété

Une FDH h a au minimum les deux propriétés suivantes :

- **Compression** : h caractérise un input (un texte, une base de donnée) x arbitraire de longueur fini de bit vers un output $h(x)$ de longueur de bit n fixé.
- **Facile à calculer** : pour un h donné et un input, $h(x)$ est facile [a] à calculer.

a. Il n'y a pas de définition précise de ce que signifie qu'une fonction de hachage est « facile » à calculer, car cela dépend de nombreux facteurs tels que la puissance de calcul disponible, les attaques cryptographiques connues, la longueur de l'entrée, etc.

Cependant, on peut dire qu'une fonction de hachage est considérée comme « facile » à calculer si elle peut être évaluée en un temps raisonnable avec les ressources de calcul disponibles. Le temps raisonnable dépend de l'objectif de la fonction de hachage et du contexte dans lequel elle est utilisée. Par exemple, pour une application de vérification d'intégrité de fichiers simples, une fonction de hachage pouvant être évaluée en quelques millisecondes peut être considérée comme « facile », tandis que pour une application de sécurité critique comme la signature numérique, une fonction de hachage qui peut être brisée en quelques minutes par une attaque cryptographique puissante ne serait pas considérée comme « sûre ».

En général, les fonctions de hachage cryptographiques modernes sont conçues pour résister à de nombreuses attaques cryptographiques, telles que la cryptanalyse différentielle et la cryptanalyse par collision, et sont considérées comme sûres lorsqu'elles sont utilisées correctement. Cependant, de nouvelles attaques cryptographiques peuvent être découvertes à l'avenir, et les fonctions de hachage peuvent devenir plus vulnérables avec le temps si elles sont exposées à des ressources de calcul plus puissantes.

A propos des FDH sans clé, on peut ajouter des propriétés complémentaires :

Propriété

- **Résistance à l'antécédent** : pour la majorité de tous les outputs, il est difficile de calculer en temps polynomial un antécédent x' comme $h(x') = y$ pour un y donné. (y est l'output)
- **Résistance au second antécédent** : il est difficile de trouver x', pour un x donné, tel que $x \neq x' \Rightarrow h(x') = h(x)$.
- **Résistance au collision** : libre choix du x et x', il est difficile de trouver $x \neq x' \Rightarrow h(x) = h(x')$.

A propos des FDH avec clé, elles doivent facilement calculable, posséder les propriétés de compression et de résistance au collision.

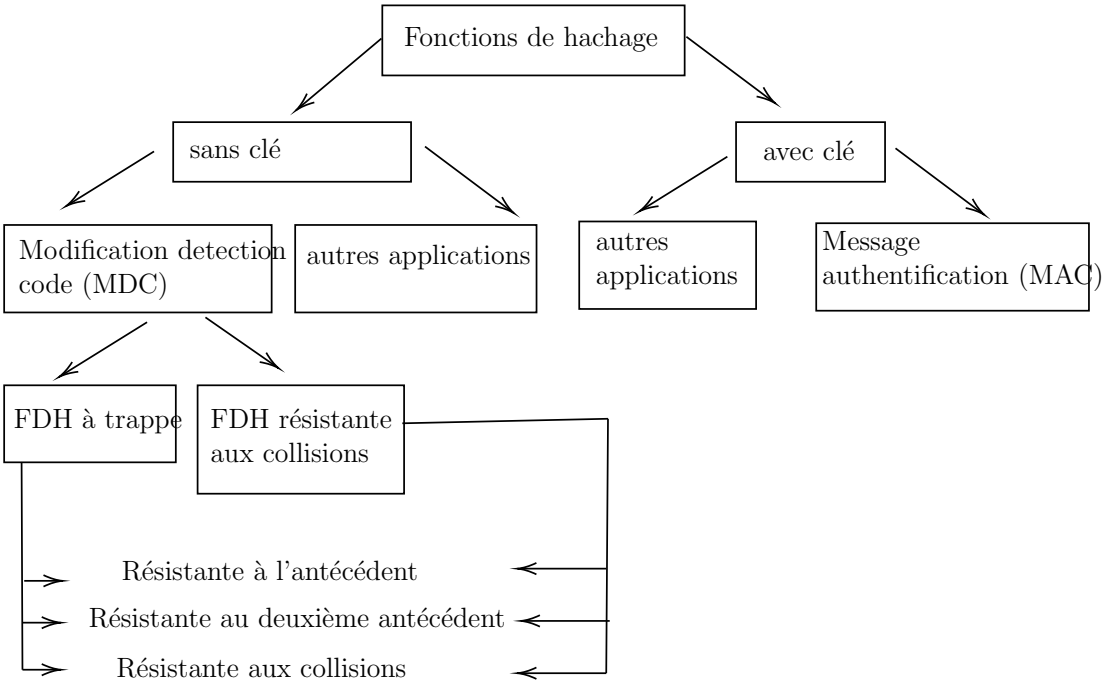

FIGURE 3 – Classification des fonctions de hachage

2.3 Attaque par le paradoxe des anniversaires

Que signifie l'attaque par le paradoxe des anniversaires et en quoi cela est-il un paradoxe ?

Une **attaque** est une tentative de compromettre la sécurité d'un système cryptographique en tentant de trouver une méthode pour déchiffrer un message ou pour découvrir la clé secrète utilisée pour chiffrer les données.

Dans le contexte des fonctions de hachage, l'attaque des anniversaires peut être utlisée pour modifier les communications entre deux personnes ou plus. L'attaque est possible grâce à la probabilité plus élevée de collisions avec des tentatives d'attaques aléatoires et un niveau fixe de permutations, comme dans le principe des tiroirs.

Le principe est de calculer la probabilité p pour que dans un groupe de n personnes deux au moins aient leur date d'anniversaire identique. (Cela correspond à une collision du haché.) Dans une année, il y a $N = 365$ jours [5].

Numérotons les personnes de 1 à n. Leurs dates d'anniversaires forment un n-uplet $(d_1, \cdots, d_n) \in \{1, \cdots, N\}^n$. Ainsi, il y a N^n possibilités. Parmi ces possibilités, seulement $N(N-1)(N-2)\cdots(N-n+1)$ ne contiennent pas deux dates identiques, c'est-à-dire toutes les dates sont différentes. [6]

La calculatrice ne renvoyant pas un résultat pour ce genre de nombre aussi grand, calculons cette probabilité grâce au code python ci-dessous pour $n = 30$.

5. En simpliant sans les années bissextiles.

6. Au rang 1, il y a N dates différentes, au rang 2, il n'y a plus que $N-1$ dates potentiellement différentes en enlevant d_1, au rang 3, on peut enlever deux dates : d_1, d_2, il reste $N-2$ dates pouvant être différentes. On procède ce raisonnement jusqu'au n-ième rang, où l'on a enlevé $n-1$ dates, il reste $N-n+1$ dates pouvant être différentes.

```
from random import *
from math import *

N = 1000000
succes = 0

# Calcul de la probabilité qu'au moins deux dates
   soient identiques.
for _ in range(N):
    annivs = [randrange(365) for __ in range(30)] # la
       valeur 30 correspond aux nombres de personnes
    if len(set(annivs)) != len(annivs):
        succes += 1
    p=succes/N
print(f"proba approchée qu'au moins deux dates sont
   identiques. {round((p)*100,3)} %")

# Calcul de la probabilité qu'au moins deux dates
   soient identiques :
print(f"proba approchée dates toutes différentes
   {round((1-p)*100,3)} %")
```

./programmes/anniv.py

Sur 10^7 essais, la probabilité vaut 70.612%.

Dressons un tableau de valeurs pour des valeurs remarquables n variant de 1 à 70 et observons ce qu'il se passe.

Nombre de personnes	**Probabilité que deux personnes soient nées le même jour**	**Probabilité que toutes les dates soient différentes**
1	0%	100%
2	0.27%	99.73%
3	0.82%	99.18%
5	2.699 %	97.301%
10	11.673%	88.327%
15	25.362%	74.638%
20	41.184%	58.816%
23	50.716 %	49.284%
25	56.957%	43.043%
30	70.596%	29.404%
40	89.136%	10.864%
50	97.027%	2.973%
70	99.917%	0.083%

On dit que c'est un paradoxe car en seulement 23 effectifs, on a une chance sur deux que deux personnes soient nées le même jour : ceci contredit l'intuition. (on penserait qu'il faudrait bien plus de personnes pour atteindre cette valeur de probabilité).

Remarque

Le calcul de la force et de la faiblesse des fonctions de hachage se base sur celui fait dans la résolution du paradoxe des anniversaires. Pour une empreinte égale à n, il faut $2^{\frac{n}{2}}$ essais pour trouver une collision au hasard.

Justifions rigoureusement la remarque précedente.

Nous allons utiliser le principe du paradoxe des anniversiares appliqué aux fonctions de hachage. Par définition une FDH h renvoie un hash de longueur n-bits où n est fixé. Comme le bit est l'unité en informatique prenant uniquement pour valeur 0 ou 1. Il y a 2^n combinaisons de hash différents. Cherchons à calculer la probabilité qu'il n'y ait pas de collision parmi k hachés. Dans un premier temps, on peut dénombrer le nombre de cas où les hashs sont tous différents :

$$2^n \cdot (2^n - 1)(2^n - 2) \times \cdots \times (2^n - (k-1))$$

Puis, le nombre de cas possibles correspond à : $\underbrace{(2^n \cdot 2^n \cdots 2^n)}_{\text{k fois}}$

En supposant qu'il y ait équiprobabilité : $p(k,n) = \dfrac{\text{nombre d'issues favorables}}{\text{nombre d'issues possibles}}$

$$\begin{aligned}
p(k,n) &= \frac{2^n \cdot (2^n - 1) \cdot (2^n - 2) \cdots (2^n - (k-1))}{2^n \cdot 2^n \cdots 2^n} \\
&= 1 \cdot \left(1 - \frac{1}{2^n}\right) \cdots \left(1 - \frac{k-1}{2^n}\right) \qquad \text{on suppose que } k \ll n \Rightarrow 1 - x \leq e^{-x} \\
&\leq 1 \times e^{-\frac{1}{2^n}} \times \cdots e^{-\frac{k-1}{2^n}} \\
&= \exp\left(-\frac{1 + \cdots + (k-1)}{2^n}\right) \\
&= \exp\left(-\frac{((k-1)+1) \cdot (k-1)}{2 \cdot 2^n}\right) \\
&= \exp\left(-\frac{-k(k-1)}{2 \cdot 2^n}\right) \\
&= \exp\left(-\frac{k^2}{2^{n+1}}\right)
\end{aligned} \tag{1}$$

Remarque

Dans la pratique, on suppose que $k \ll n$ car les valeurs de n deviennent sécurisée au-delà de n bits suffisament grands, k représente le nombre de haché en collision : les attaquants ne cherchent pas beaucoup de hachés qui soient identiques. En effet, le but est de trouver deux hachés identiques puis d'altérer la donnée initiale, ainsi $k = 2$ dans ce principe. Dans la suite du document, on discutera de quelle valeur de n pouvant convenir le mieux pour assurer la fiabilité des FDH.

D'après le paradoxe des anniversaires, nous avons obtenu deux séries de résultats dans le tableau **??** . Dans le deuxième colonne intitulée « probabilités que deux personnes soient nées le même jour », on remarque qu'à la ligne $n = 23$, on obtient une probabilité de collision de 50.716%.

Dans la suite des calculs, nous allons supposer qu'une FDH n'est pas suffisament sécurisée pour une probabilité supérieure à 0.5.

Cela revient à dire que $0.5 < p(k, n)$. Or, nous avions trouvé précédemment une majoration sur $p(k, n)$. Ainsi,

$$\begin{aligned}
0.5 < p(k,n) &\leq \exp(-\frac{k^2}{2^{n+1}}) \\
\Leftrightarrow \frac{1}{2} &\leq \exp(-\frac{k^2}{2^{n+1}}) \\
\Leftrightarrow \ln(\frac{1}{2}) &\leq -\frac{k^2}{2^{n+1}} \\
\Leftrightarrow 2^{n+1} \cdot \ln(2) &\geq k^2 \\
\Leftrightarrow \sqrt{2^{n+1} \cdot \ln(2)} &\geq k \quad \text{car } k \geq 0 \\
\Leftrightarrow \sqrt{2 \times \ln(2)} \cdot \sqrt{2^n} &\geq k \\
\Leftrightarrow \sqrt{2 \times \ln(2)} \cdot 2^{n/2} &\geq k \\
\approx 1,1 \cdot 2^{n/2} &\geq k
\end{aligned} \tag{2}$$

Par conséquent, pour un hash de taille n, il faut au moins $2^{n/2}$ essais pour trouver une collision.

Concrètement, pour des hashs de longueurs de 64 bits, une probabilité de collision supérieure à 0.5 sera trouvée en 1.1×2^{32} essais, soit moins de 5 milliards d'essais [7]. Ce nombre d'essais est largement faisable pour les machines modernes [8].

7. La valeur exacte est 4 724 464 025.6, il font donc précisément 4 724 464 026 essais.
8. TP jupyter de l'université Gustave Eiffel, Institut d'électronique et d'informatique gaspard monge.

La puissance de calcul d'un ordinateur dépend de plusieurs facteurs tels que la vitesse du processeur, la quantité de mémoire vive (RAM), la capacité de stockage, le type de carte graphique, la vitesse de l'interface d'E/S (entrée/sortie) et bien d'autres facteurs.

La mesure la plus courante pour évaluer la puissance de calcul d'un ordinateur est le nombre d'opérations à virgule flottante par seconde (**FLOPS**[9]). Cette mesure est utilisée pour évaluer la capacité de l'ordinateur à effectuer des calculs mathématiques complexes.

Le nombre de FLOPS d'un ordinateur dépend de la vitesse de son processeur, de la quantité et de la vitesse de sa mémoire RAM et de la qualité de sa carte graphique. Les ordinateurs modernes peuvent avoir une puissance de calcul allant de quelques gigaflops (milliards d'opérations à virgule flottante par seconde, c'est-à-dire de l'ordre de 10^9 FLOPS) à plusieurs téraflops (10^{12} opérations à virgule flottante par seconde) voire plus pour les supercalculateurs.

Il convient de noter que la puissance de calcul d'un ordinateur ne détermine pas nécessairement sa capacité à exécuter des tâches spécifiques. Par exemple, un ordinateur équipé d'un processeur plus lent mais avec une mémoire RAM plus rapide pourrait être plus performant pour certaines tâches que celui avec un processeur plus rapide et une mémoire RAM plus lente.

Les supercalculateurs[10] sont des ordinateurs conçus pour effectuer des calculs intensifs à grande échelle et ils sont donc beaucoup plus puissants que les ordinateurs de bureau ou les serveurs conventionnels.

Leur puissance de calcul est mesurée en petaflops (10^{15} opérations à virgule flottante par seconde). Les supercalculateurs modernes peuvent avoir une puissance de calcul allant de quelques petaflops à plusieurs exaflops.

En 2022, la barre de l'exaFLOPS (d'ordre de grandeur 10^{18}) a été dépassé par le superordinateur américain Frontier.[11]

Il convient de noter que la puissance de calcul des supercalculateurs est me-

9. Floating-Point Operations Per Second

10. Ces ordinateurs sont utilisés pour des applications qui nécessitent une grande puissance de calcul, telles que la simulation de phénomènes physiques, l'analyse de données complexes, la modélisation de systèmes météorologiques et climatiques, l'analyse de risques financiers, la conception de médicaments et d'autres applications scientifiques et industrielles. Les supercalculateurs sont généralement utilisés dans des environnements de recherche scientifique, de défense, d'industrie et de finance, où les exigences en matière de calcul intensif sont très élevées. Ils sont également utilisés dans les centres de données pour traiter de grandes quantités de données, notamment dans les domaines de l'intelligence artificielle et de l'apprentissage automatique.

11. D'après le site TOP500 (TOP500 est réalisé par Hans Meuer de l'université de Mannheim en Allemagne, Jack Dongarra de l'université du Tennessee à Knoxville, Erich Strohmaier et Horst Simon du National Energy Research Scientific Computing Center (NERSC) du Lawrence Berkeley National Laboratory (LBL)), « Frontier is the new No. 1 system in the TOP500. This HPE Cray EX system is the first US system with a peak performance exceeding one ExaFlop/s. »

surée en FLOPS et en petaflops pour évaluer leur capacité à effectuer des calculs intensifs à grande échelle. Cependant, la performance des supercalculateurs dépend également de leur architecture, de leur connectivité et de leur capacité à traiter efficacement de grandes quantités de données.

Prenons l'exemple de la fonction de hachage SHA_{256}, pour cette fonction de hachage donnant en sortie un hash de $n = 256$ bits. On a prouvé qu'il fallait $1.1 \times 2^{n/2}$ essais pour atteindre une collision. Il faut donc :

$$\begin{aligned} 1.1 \times 2^{256/2} &= 1.1 \times 2^{128} \\ &\approx 3.7 \cdot 10^{38} \text{ essais} \end{aligned}$$

Conclusion

Par conséquent, pour des attaquants utilisant des ordinateurs modernes, les puissances de calculs sont de l'ordre du téraflops (10^{12} FLOPS) et pour des attaquants utilisant des supercalculateurs, les puissances de sont de l'ordre de l'exaflops (10^{18} FLOPS). On peut remarquer que pour des FDH où $n = 64$ bits, les puissances de calculs sont de l'ordre du gigaflops (nécessitant $4 \cdot 10^9$ essais). Puis, pour les FDH à 128-bits, les puissances de calculs sont de l'ordre de 10 fois l'exaflops. Pour des ordinateurs modernes, la FDH reste suffisament sécurisée mais selon l'agence NIST [a] , elles ne le sont plus étant donnés qu'il existe des supercalculateurs capables de calculer en exaflops. Ainsi, on considère qu'à partir de 256-bits, les FDH sont suffisament sécurisées. (nécessitant $3.7 \cdot 10^{38}$ essais dépassant largement l'ordre de grandeur de l'exaflops). Exemple : La FDH $SHA - 256$ peut-être considérée comme une fonction fiable.

a. National Institute of Standards and Technology (Institut national des normes et de la technologie)

3 Applications des fonctions de hachage

On a pu voir dans les parties précédentes que les fonctions de hachages permettaient de compresser des données de grandes ou petites tailles. On peut être amener à penser que ces compressions de données peuvent jouer un rôle dans le cadre de conversion d'un fichier au format .mp4 vers .mp3. Cependant, la compression des fichiers au format MP3 ne repose pas sur une fonction de hachage mais sur un algorithme de compression audio avec pertes. L'algorithme de compression utilisé pour les fichiers MP3 est appelé **MPEG-1 Layer III** [12] . Cet algorithme utilise une technique de compression appelée « codage perceptuel ». Cela permet de supprimer certaines informations du signal audio qui ne sont pas perceptibles pour l'oreille humaine, tout en préservant les parties les plus importantes du signal.

3.1 Dans la bureautique

Les fonctions de hachages sont souvent utilisées pour sécuriser les mails en les rendant plus difficiles à falsifier ou à intercepter. Lorsqu'un courrier électronique est envoyé, il passe souvent par plusieurs serveurs avant d'arriver à sa destination finale. Chaque serveur est potentiellement vulnérable à des attaques de piratage ou de falsification, ce qui peut entraîner une altération du contenu du courrier électronique. Pour éviter cela, les fournisseurs de messagerie peuvent utiliser des fonctions de hachages pour créer une empreinte numérique (*digital fingerprint*) unique du contenu du courrier électronique. Cette empreinte est ensuite envoyée avec le courrier électronique et peut être vérifiée par le destinataire pour s'assurer que le contenu n'a pas été altéré pendant la transmission. On peut comparer la signature numérique dun e-mail à une signature manuscrite dans un courrier. Celle-ci est unique et sans équivoque. A noter, qu'une signature numérique ne chiffre pas le message mais apporte uniquement la preuve de son intégrité. Les contenus confidentiels doivent faire lobjet dun chiffrement supplémentaire. Plus généralement, ces fonctions permettent l'identification de fichiers. Elles peuvent être utilisées pour identifier de manière unique des fichiers ou des ensembles de données. Chaque fichier a une empreinte numérique unique qui peut être calculée à l'aide d'une fonction de hachage, ce qui permet de s'assurer que le fichier n'a pas été modifié ou remplacé.

12. Le format MP3 est un format propriétaire : dont les spécifications sont contrôlées par des intérêts privés.

3.2 En informatique

Les fonctions de hachage sont également utilisées pour stocker les mots de passe de manière sécurisée dans les bases de données des fournisseurs de messagerie, afin d'éviter les violations de données et les accès non autorisés aux comptes de messagerie. Au lieu de stocker les mots de passe en clair dans une base de données, les fournisseurs de services peuvent utiliser ces fonctions pour stocker des empreintes numériques des mots de passe. De cette façon, même si la base de données est compromise, les pirates informatiques ne peuvent pas facilement récupérer les mots de passe en clair.

Par ailleurs, les fonctions de hachage sont souvent utilisées pour garantir que des données n'ont pas été altérées ou corrompues. Par exemple, lorsqu'un fichier est téléchargé depuis un site web, sa somme de contrôle peut être calculée à l'aide d'une fonction de hachage et comparée à celle fournie par le site web pour s'assurer que le fichier n'a pas été modifié pendant le téléchargement.

Les fonctions de hachages sont aussi souvent utilisées dans les protocoles de cryptographie pour fournir des signatures numériques et des clés de chiffrement. Par exemple, une signature numérique peut être créée en utilisant une fonction de hachage pour créer une empreinte numérique du message, qui est ensuite chiffrée avec la clé privée de l'expéditeur.

Les fonctions de hachage peuvent être utilisées pour accélérer les opérations de recherche dans les grandes bases de données. Les valeurs de hachage peuvent être utilisées comme clés de recherche, ce qui permet de trouver rapidement les données correspondantes sans avoir à parcourir toute la base de données.

(Exemple : Les FDH sont présentes dans tous les langages de programmations modernes comme en Java, Python , C++)

Les fonctions de hachage sont des algorithmes qui permettent de transformer des données en une valeur de hachage unique et fixe. Cette valeur de hachage peut ensuite être utilisée pour vérifier l'intégrité des données, c'est-à-dire s'assurer qu'elles n'ont pas été modifiées ou altérées.

Dans le cadre de SSL [13], les fonctions de hachage sont utilisées pour garantir l'intégrité des données échangées entre le navigateur et le serveur web. Avant de chiffrer les données à l'aide de la clé de session partagée, le serveur web peut appliquer une fonction de hachage aux données. La valeur de hachage est ensuite envoyée au navigateur web, qui peut vérifier l'intégrité des données reçues en recalculant la valeur de hachage et en la comparant à celle envoyée par le serveur web.

13. SSL signifie Secure Sockets Layer est un protocole de sécurité qui permet de sécuriser les échanges de données entre un serveur web et un navigateur web. Il utilise des techniques de cryptographie pour garantir que les données échangées entre les deux parties ne peuvent être ni lues ni modifiées par des tiers malveillants.

Si la valeur de hachage calculée par le navigateur web correspond à celle envoyée par le serveur web, cela signifie que les données n'ont pas été altérées pendant le transfert et que leur intégrité est garantie.

En outre, le stockage temporaire des informations dans des caches est également chiffré à laide de hashs afin que les utilisateurs non autorisés ne puissent pas déterminer les sites internets visités et les données daccès de paiement utilisées lors de la consultation du cache.

De même, les fonctions de hachage publiques réputées fortes étant par nature à la disposition de tous, il est techniquement possible pour tout un chacun de calculer des empreintes. Aujourdhui, on trouve facilement sur internet des dictionnaires immenses dempreintes MD5 précalculées. Grâce à ces données, il est aisé de retrouver instantanément le mot de passe ayant été utilisé afin de générer ces empreintes. Afin de limiter ce risque, il est conseillé dutiliser des fonctions spécialisées appelées « fonction de dérivation de clé », telles que bscrypt [14] ou Argon2 [15] par exemple, qui sont conçues spécifiquement pour stocker des mots de passe.

14. bcrypt est une fonction de hachage créée par Niels Provos et David Mazières. Elle est basée sur l'algorithme de chiffrement Blowfish et a été présentée lors de USENIX en 1991. En plus de l'utilisation d'un sel pour se protéger des attaques par table arc-en-ciel (rainbow table), bcrypt est une fonction adaptative, c'est-à-dire que l'on peut augmenter le nombre d'itérations pour la rendre plus lente. Ainsi elle continue à être résistante aux attaques par force brute malgré l'augmentation de la puissance de calcul.

15. Argon2 est une fonction de dérivation de clé, gagnante de la Password Hashing Competition en juillet 2015. Elle a été conçue par Alex Biryukov, Daniel Dinu et Dmitry Khovratovich de l'Université de Luxembourg et publié sous licence Creative Commons CC0.

Une *rainbow table* (littéralement table arc-en-ciel) est une structure de données [16] pour retrouver un mot de passe à partir de son empreinte. La table arc-en-ciel ne dépend que de la fonction de hachage considérée. Cette table est constituée d'une grande quantité de paires de mots de passe. Pour obtenir chacune de ces paires, en partant d'un mot de passe, on calcule son empreinte. Une « fonction de réduction » (variant selon la position dans la table) permet de recréer un nouveau mot de passe à partir de cette empreinte. Après avoir effectué un nombre fixe d'itérations, on obtient un mot de passe qui forme le deuxième élément de la paire. La table est créée une fois pour toutes et permet la recherche d'un mot de passe, connaissant son empreinte par la fonction de hachage considérée.

L'algorithme développé par Oechslin optimise la détection des collisions et le parcours dans la table. Des solutions pour réduire la taille de la table ont également été proposées. Plusieurs tables peuvent être produites pour améliorer les chances de réussite.

Il existe dans le langage de programmation Python, une fonction hash() qui permet de hasher des caractères.

3.2.1 Flash Code

Les fonctions de hachage peuvent être utilisées pour générer un flash code (aussi appelé QR code), qui est un code-barres en deux dimensions qui peut être scanné à l'aide d'un smartphone ou d'un lecteur de code-barres pour accéder à des informations spécifiques.

Lorsqu'un flash code est généré, il est généralement associé à une URL ou à une autre forme de données. Pour garantir l'intégrité et la sécurité de ces données, un hachage peut être calculé à partir de ces données à l'aide d'une fonction de hachage. Le hachage est ensuite inclus dans le flash code pour permettre à l'application de vérifier que les données n'ont pas été altérées depuis leur création.

Lorsque le flash code est scanné, l'application qui le lit peut calculer le hachage des données et le comparer au hachage inclus dans le flash code. Si les deux hachages correspondent, cela indique que les données n'ont pas été altérées, et l'application peut alors afficher les informations associées au flash code. L'un des algorithmes de hachage les plus couramment utilisés dans la génération de codes QR est SHA-256. SHA-256 est considérée comme étant très résistante aux collisions et aux attaques par force brute [17], ce qui en fait un choix populaire pour la génération de codes QR sécurisés (Exemple : Les flashs-codes présent sur le pass vaccinal COVID-19). D'autres fonctions de hachage, telles que MD5 ou SHA-1, sont également utilisées pour la génération de codes QR, mais elles sont considérées comme moins sécurisées que SHA-256 et sont donc moins couramment utilisées.

16. Elle a été créée en 2003 par Philippe Oechslin de l'EPFL1.
17. Référence de la NIST

3.2.2 Dans la blockchain

SHA-256 est utilisé dans la technologie de la blockchain pour sécuriser les données et garantir l'intégrité de la chaîne de blocs.

Dans une blockchain, chaque bloc de données est associé à un hachage unique qui est généré à partir du contenu du bloc. Le hachage est une empreinte numérique de la donnée, qui peut être considérée comme une représentation unique et condensée du bloc de données. La fonction de hachage SHA-256 est utilisée pour générer ces empreintes numériques.

Le fonctionnement de la blockchain repose sur le fait que chaque bloc de données contient le hachage du bloc précédent, ce qui crée une chaîne de blocs interconnectés. Si une donnée est modifiée dans un bloc, cela entraînera un changement dans son hachage, qui se propagera à tous les blocs suivants dans la chaîne de blocs. Par conséquent, toute tentative de modification d'un bloc de données antérieur serait rapidement détectée et rejetée par le réseau, car elle ne correspondrait pas au hachage enregistré précédemment.

On peut citer la crypto-monnaie Bitcoin et aussi l'université de LILLE utilisant la blockchain pour partager de façon sûre les attestations numériques de réussite au diplôme. [18] [19]

4 Approfondissement avec la fonction de hachage SHA-256

4.1 Introduction sur la fonction de hachage SHA-256

SHA-256 (Secure Hash Algorithm 256 bits) est une fonction de hachage largement utilisée pour la sécurité des données et des communications. Elle produit une empreinte numérique unique pour chaque entrée de données, ce qui permet de vérifier l'intégrité et l'authenticité des données. La fonction SHA-256 prend en entrée des données de n'importe quelle taille et produit une empreinte numérique (hash) de 256 bits. Elle fait partie de la famille des algorithmes SHA-2 développée par la National Security Agency (NSA) et publiée par le National Institute of Standards and Technology (NIST) des États-Unis. Elle est utilisée dans les protocoles de chiffrement tels que TLS (Transport Layer Security) et SSL (Secure Sockets Layer) pour garantir la confidentialité et l'intégrité des données échangées entre un serveur et un client.

SHA-256 utilise un processus itératif pour transformer l'entrée de données en une sortie de 256 bits. Les étapes du processus incluent l'ajout de bits de remplis-

18. Livre Blanc du projet Dem-Attest-ULille
19. Attestation numérique de diplôme de Maxence Defraiteur

sage, le découpage en blocs, le traitement de chaque bloc en utilisant une série de fonctions de hachage, la concaténation des résultats et la production de la valeur finale de hachage. La sécurité de SHA-256 repose sur la difficulté de trouver deux entrées différentes qui produisent la même sortie de hachage (collision), ainsi que sur la résistance aux attaques par force brute et aux attaques par compromission partielle.

La longueur de 256 bits a été choisie pour fournir une sécurité suffisante contre les attaques cryptographiques connues à ce jour. Une sortie de hachage plus longue est généralement considérée comme plus sûre car elle augmente la probabilité que des entrées différentes produisent des sorties de hachage différentes.

SHA-256 est résistante à l'antécédent. Elle est conçue pour être résistante aux attaques par force brute, ce qui signifie qu'il est très difficile de trouver une entrée qui produit une sortie de hachage donnée sans passer par une recherche exhaustive.

Le fait que SHA-256 produise une sortie de 256 bits signifie qu'il y a 2^{256} (soit environ 10^{77}) résultats possibles. Cela rend pratiquement impossible pour un attaquant (en temps polynomial) de trouver deux entrées différentes qui produisent la même sortie de hachage (collision). En d'autres termes, la probabilité de collision est extrêmement faible, ce qui garantit une sécurité suffisante pour de nombreuses applications. Comme il y a 2^{256} combinaisons possibles pour les hachés crées, cela signifie que même une modification mineure dans l'entrée produira très probablement un haché différent.

4.2 Exemple d'exécution

```
import hashlib

# Créer un objet SHA-256
hash_object = hashlib.sha256()

# Ajouter la chaîne de caractères "pomme" à l'objet
   SHA-256
hash_object.update(b"pomme")

# Obtenir le hachage SHA-256 sous forme de chaîne de
   caractères binaire (bytes)
hash_value_bytes = hash_object.digest()

# Convertir chaque byte en une chaîne de 8 bits (un
   octet)
hash_value_bits = ''.join(format(byte, '08b') for byte
   in hash_value_bytes)

# Convertir le hachage binaire en hexadécimal pour une
   meilleure lisibilité
hash_value_hex = hash_value_bytes.hex()

# Afficher le hachage SHA-256 en bits
print("Le hachage en bits de SHA-256 de la chaîne
   'pomme' est :", hash_value_bits)

# Afficher le hachage SHA-256 en hexadécimal
print("Le hachage en hexadécimal SHA-256 de la chaîne
   'pomme' est :", hash_value_hex)
```

./programmes/exemple.sha.256.py

[20]

20. Documentation python de la fonction hashlib

Les résultats de l'exécution de ce code sont des chaînes de caractères :

— 10010001011010011011111100111110
01010000000111111110101000011001
011000010 10011001010110011010110
11010110010001101 011010100001011
01100011101010101000001000101011
11000010001101100000101001001101
10110000011010101110111001001100
11010101000001001100101101001111 (en bits : 256 caractères)

— 9169bf3e501fea19614cacd6d646b50b63aa822bc2360a4db06aee4cd504cb4f (en hexadécimal : 64 caractères pour faciliter la lecture)

Nous pouvons clarifier le lien entre les 64 caractères hexadécimaux et les 256 bits. La fonction de hachage SHA-256 produit une empreinte numérique de 256 bits, soit une séquence de 32 octets (1 octet = 8 bits). Chaque octet peut prendre une valeur hexadécimale de 00 à ff, ce qui donne une représentation hexadécimale de l'empreinte numérique de 64 caractères (32 x 2 = 64).

Le site internet sha256algorithm permet de visualiser en partie les calculs effectués par l'algorithme de la FDH SHA-256. On peut vérifier et l'exécuter avec le caractère « pomme ». Nous trouvons le même hash que la sortie du code compilé en Python précédemment.

Le processus de calcul de l'empreinte numérique peut être décrit en plusieurs étapes :

— **Prétraitement** : la chaîne de caractères d'entrée est transformée en une séquence de bits de longueur multiple de 512 bits.

— **Initialisation** : une série de constantes prédéfinies (appelées vecteurs d'initialisation) et un tableau de constantes de mots clés (appelés constantes de tour) sont définis.

— **Boucle de compression** : la chaîne de bits d'entrée est découpée en blocs de 512 bits et chaque bloc est traité par une série de transformations cryptographiques.

— **Finalisation** : une fois que tous les blocs ont été traités, une dernière transformation cryptographique est appliquée pour produire l'empreinte numérique finale.

5 Bibliographie

Références

[1] Marie-José Durand-Richard, Philippe Guillot, *Comment les mathématiques ont investi la cryptologie ?*, 15 mars 2017, Image des Maths

[2] Juan GOMEZ, *Mathématiciens, espions et pirates informatiques, Le Monde est Mathématique*, 2013

[3] Pierre-Antoine Guihéneuf, Alice Cleynen,*Le paradoxe des anniversaires*, Image des Maths CNRS,22 novembre 2022

[4] **Alfred J.Menezes, Paul C. van Oorschot et Scott A. Vanstone, *Handbook of applied criptography*, août 2001, CRC Press**

[5] Matt Parker, *The Maths Book*, publié par DK le 5 septembre 2019

[6] http://bruno.maitresdumonde.com/optinfo/ads-scie/2008.pdf, Epreuve commune de TIPE 2008, Les Fonctions de Hachage