TD 5 Fonctions de Hachage

Exercice 1: Un peut de Maths et de probas Évaluation de la résistance aux collisions

Un peu de mathématiques :

1. Concernant les fonctions de hachage : Quelle différence voyez vous entre les exigences de la cryptographie et celle des bases de données ?
2. Proposer une valeur puis un programme permettant d'évaluer la probabilité *p(n)* que *n* personnes aient chacune une date d'anniversaire différente en notant N=365.
3. Les mathématiciens ont trouvé : $p(n) \approx \prod_{i=1}^{n} e^{-\frac{k}{n}} \approx e^{-\frac{n.(n-1)}{2.N}}$ et donc $n \simeq \sqrt{2 \cdot N \ln\left(\frac{1}{1-p}\right)}$.

   En quoi cela peut-il nous être utile ?
4. On souhaite travailler avec une probabilité de collision inférieure à 1%. En déduire le nombre de donnée hachable sur 8,16,32,64 et 128 bits. Construire le graphique correspondant.

Exercice 2: Des fonctions de hachage naïve :

Les fonctions de hachage héritent-elles aussi de CodeurCA sans bien-sûr surcharger binDecode : Expliquer pourquoi.

- Proposer quelques fonctions de hachage (Une méthode binCode qui retourne un Binaire603 de taille fixe) et évaluer "à vue de nez" leurs qualités.
- pour chacune de ces fonctions proposer une méthode permettant, à partir d'un Binaire603, d'en déduire un autre ayant la même image par la fonction de hachage.
- Proposer une méthode renvoyant, à partir d'un Binaire603, un Binaire603 dont il serait l'image.
- Proposer un test aux collisions : est-il universel ?
- Proposer une méthode de test sur les mots en français.

Exercice 3: Résistances !

Montrer, par contra-posée, que la résistance aux collisions implique la résistance à la seconde pré-image qui implique la résistance à la pré-image.

Exercice 4: Fonction de hachage à partir d'un CodeurCA

- Programmer une fonction de Hachage recevant en paramètre un CodeurCA et hachant selon la construction de Davies-Meyer
- Faire de même selon une construction de Miyaguchi-Prenel.
- Vérifier la qualité de ces constructions avec le "test" des bijections ainsi que celui des anniversaires.

Exercice 5: Une mauvaise fonction de hachage

Soit f $: F_2^{\,m} \mapsto F_2^{\,m}$ une fonction quelconque.

Soit H la fonction de hachage résultat de l'itération de l'itération de la fonction $\begin{array}{c} g: F_2^{\,2m} \to F_2^{\,m} \\ x=(x_l, x_r) \mapsto f(x_r \oplus x_l) \end{array}$

- Vérifier la qualité de cette fonction avec le "test" des bijections ainsi que celui des anniversaires.
- Montrer que H n'est pas résistante à la seconde pré-image.

Exercice 6: (si vous avez fini ;) Hachage par le logarithme discret Voir Codage par B.Martin p234]

Soit la fonction h définie pour p premier et q=frac{p-1}2 premier aussi,

alpha et beta deux éléments primitifs de F_p (c'est à dire d'ordre p) par : $h : F_q \times F_p \rightarrow F_p^* \quad (x_1, x_2) \rightarrow \alpha^{x_1} \beta^{x_2}$

- Déterminer le premier couple (p,q) pour p>40.
- Implémenter cette fonction et vérifier ses qualités de cette fonction avec le "test" des bijections ainsi que celui des anniversaires.

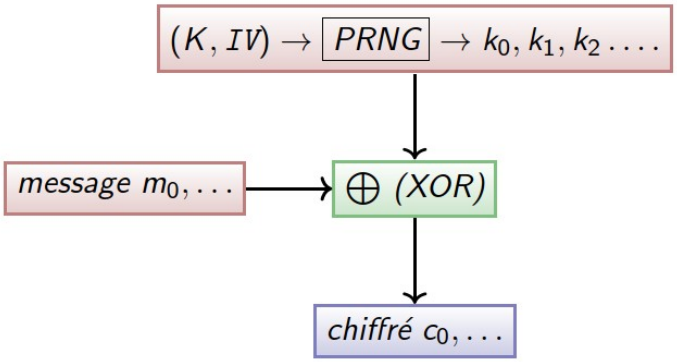
Exercice 7: Un petit retour : Chiffrement par bloc

Écrire la classe ChiffreurParBloc reprenant le principe suivant :

« On utilise un générateur de nombre pseudo-aléatoire (PRNG) : mais on peut prendre aussi les valeurs d'une « bonne » fonction de hachage.

La clé de chiffrement est alors utilisé comme graine.

Le message est découpé en bloc et chaque bloc est chiffré par un XOR avec un nombre généré par le PRNG



Exercice 8: Piratage

Proposer quelques petits programmes python permettant de récupérer des mots de passes en clair à partir d'un fichier crypté de login volé par piratage.

Exercice 9: BlockChain

J'ai compris :

- Où sont les données de la BlockChain
- Le parcours d'une transaction
- Pourquoi la BlockChain est « infalsifiable »
- Pourquoi les acteurs « jouent le jeu »
- Ce que font ces immenses fermes de serveur et à quel point c'est désolant et tout l'intérêt de la POS (hors programme)
- Pourquoi une puissance de calcul d'un demi (un peu moins en fait) ou une faille dans SHA256 détruit la BlockChain.

2. Solution

Exercice 1:

Exercice 2:

Exercice 3:

1. Il est facile de trouver des x tel que y = H(x). Donc si l'on connait x, on calcule y = H(x), puis on trouve un autre x' tel que y = H(x').

2. Il est facile de trouver x' connaissant x, donc on prend un x quelconque au hasard et on génère x'.

Exercice 4:

```
def binCode(self,monBinD:Binaire603)->Binaire603:
                    nbBitsq=round(log(self.q,2))+1 #Nombre de bits de p
                    nbBitsp=round(log(self.p,2))+1
                    nbOctetsALire=((nbBitsq*2)-nbBitsp)//8
                    Mk=self.init
                    #Construction de  Merkle-Damgård
                    k,pos=0,0
                    while pos<len(monBinD):
                    x=Mk.a//self.q #Par def de q x est dans [0..2]
                    Mk=ElmtZnZ(Mk,self.q)
                    i=0
                    while i<nbOctetsALire and pos<len(monBinD):
                    oc,pos=monBinD.lisOctet(pos)
                    x=x*256+oc  #Inférieur à q car q>2*256+255
                    i+=1
                    Mk=(self.alpha**x)*(self.beta**Mk.a)
                    monBinC=Binaire603([])
                    monBinC.ajouteLongueValeur(Mk.a)
                    return Binaire603(monBinC[1:])
```

Exercice 5:

Prenons par exemple $x'_h=(x_h \oplus x_l)$ et $x'_l=0$, alors on a $x_h \oplus x_l = x'_h \oplus x'_l$ et à fortiori également après l'application de f.

Exercice 6: Piratage

On parcourt les mots puis des combinaisons. On fait un ensemble de ces mots pour accélérer la recherche.