Corrigé : chiffrement par blocs (X mp 2009)

Partie I. Approche naïve

Question 1.

```python
def decomposerBase(N, k):
    a = [0] * N
    for i in range(N):
        a[i] = k % N
        k //= N
        if k == 0:
            break
    return a
```

Question 2. Commençons par faire quelques observations :

- $a_0 \in [\![0,1[\![$ donc $a_0 = 0$.
- $a_1 \in [\![0,2[\![$ et $k - a_1$ est pair donc $k \equiv a_1 \mod (2)$.
- $a_2 \in [\![0,3[\![$ et $\dfrac{k-a_1}{2} - a_2$ est divisible par 3 donc $\dfrac{k-a_1}{2} \equiv a_2 \mod (3)$.

Plus généralement, $\dfrac{1}{i!}\Big(k - \sum_{j=0}^{i-1} a_j j!\Big) \equiv a_i \mod (i+1)$, d'où la fonction :

```python
def decomposerFact(N, k):
    a = [0] * N
    for i in range(1, N):
        a[i] = k % (i + 1)
        k //= (i + 1)
        if k == 0:
            break
    return a
```

Question 3.

```python
def ecrirePermutation(N, k):
    a = decomposerFact(N, k)
    ell = [i for i in range(N)]
    sigma = [None] * N
    for i in range(N):
        sigma[i] = ell[a[N-i-1]]
        del ell[a[N-i-1]]
    return sigma
```

La fonction `del` supprime un élément d'un tableau donné par son rang.

Question 4. La fonction de chiffrement est immédiate :

```python
def chiffrer(N, k, b):
    sigma = ecrirePermutation(N, k)
    return sigma[b]
```

La fonction de déchiffrement l'est tout autant si on connait la méthode `index(x)` qui, appliquée à une liste, renvoie l'indice correspondant à la première occurrence de `x` dans celle-ci :

```python
def dechiffrer(N, k, b):
    sigma = ecrirePermutation(N, k)
    return sigma.index(b)
```

Partie II. Réseau de Feistel

On notera que le choix de $N = 2^{64}$ n'est pas anodin : les entiers étant codés en complément à deux sur 64 bits, le quotient de la division euclidienne par 2^{32} est égal au 32 premiers bits de cette décomposition et le reste aux 32 derniers.

Question 5.

```
def feistelTour(k, b):
    q, r = divmod(b, 2**32)
    return r * 2**32 + (q ^ f(k, r))
```

`q, r = divmod(a, b)` est équivalent à `q, r = a // b, a % b` (mais ne réalise qu'une seule fois le calcul de la division euclidienne).

Question 6. La relation $(a \oplus b) \oplus b = a$ permet d'établir les équivalences suivantes :

$$\begin{cases} r_{i+1} = q_i \oplus F_{k_i}(r_i) \\ q_{i+1} = r_i \end{cases} \iff \begin{cases} q_i = r_{i+1} \oplus F_{k_i}(r_i) \\ r_i = q_{i+1} \end{cases} \iff \begin{cases} q_i = r_{i+1} \oplus F_{k_1}(q_{i+1}) \\ r_i = q_{i+1} \end{cases}$$

```
def feistelInverseTour(k, b):
    q, r = divmod(b, 2**32)
    return (r ^ f(k, q)) * 2**32 + q
```

Question 7.

```
def feistel(K, b):
    for k in K:
        b = feistelTour(k, b)
    return b
```

Question 8.

```
def feistelInverse(K, b):
    for k in reversed(K):
        b = feistelInverseTour(k, b)
    return b
```

Partie III. Vérification de propriétés statistiques

Question 9. Contrairement à la question 1, il s'agit cette fois d'obtenir la décomposition en base 2 avec le chiffre de poids le plus significatif en premier, ce qui impose de remplir chaque tranche de 64 bits de la droite vers la gauche :

```
def sequence(n):
    S = [0] * n
    for k in range(n // 64):
        x = sigma(k)
        for i in range(64):
            S[64 * k + 63 - i] = x % 2
            x //= 2
    return S
```

Question 10.

```python
def calculerV1(n):
    S = sequence(n)
    n0 = n1 = 0
    for b in S:
        if b == 0:
            n0 += 1
        else:
            n1 += 1
    return (n0 - n1)**2 / n
```

Question 11.

```python
def calculerV2(n):
    s = sequence(n)
    n0 = n1 = 0
    for b in s:
        if b == 0:
            n0 += 1
        else:
            n1 += 1
    n00 = n01 = n10 = n11 = 0
    for i in range(len(s)-1):
        k = 2 * s[i] + s[i+1]
        if k == 0:
            n00 += 1
        elif k == 1:
            n01 += 1
        elif k == 2:
            n10 += 1
        else:
            n11 += 1
    return 4*(n00**2+n01**2+n10**2+n11**2)/(n-1)-2*(n0**2+n1**2)/n+1
```