Utilisation d’un automate cellulaire en cryptographie

Louis Jacotot

Table des matières

Est-il possible de généraliser la méthode de chiffrement de Vernam à un automate cellulaire de dimension 2 ?

Introduction

Chiffrement

- Confidentialité
- Support non sécurisé

Figure 1 – Chiffrement symétrique

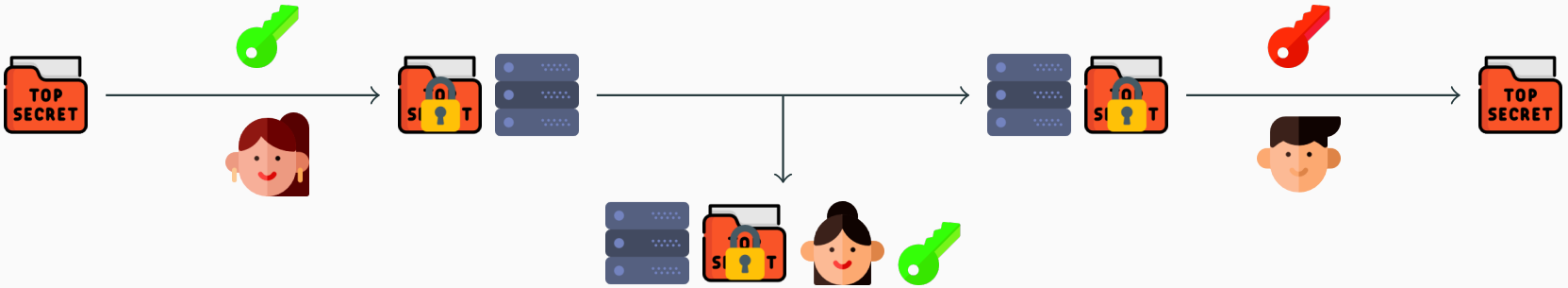


Figure 2 – Chiffrement asymétrique

Chiffrement

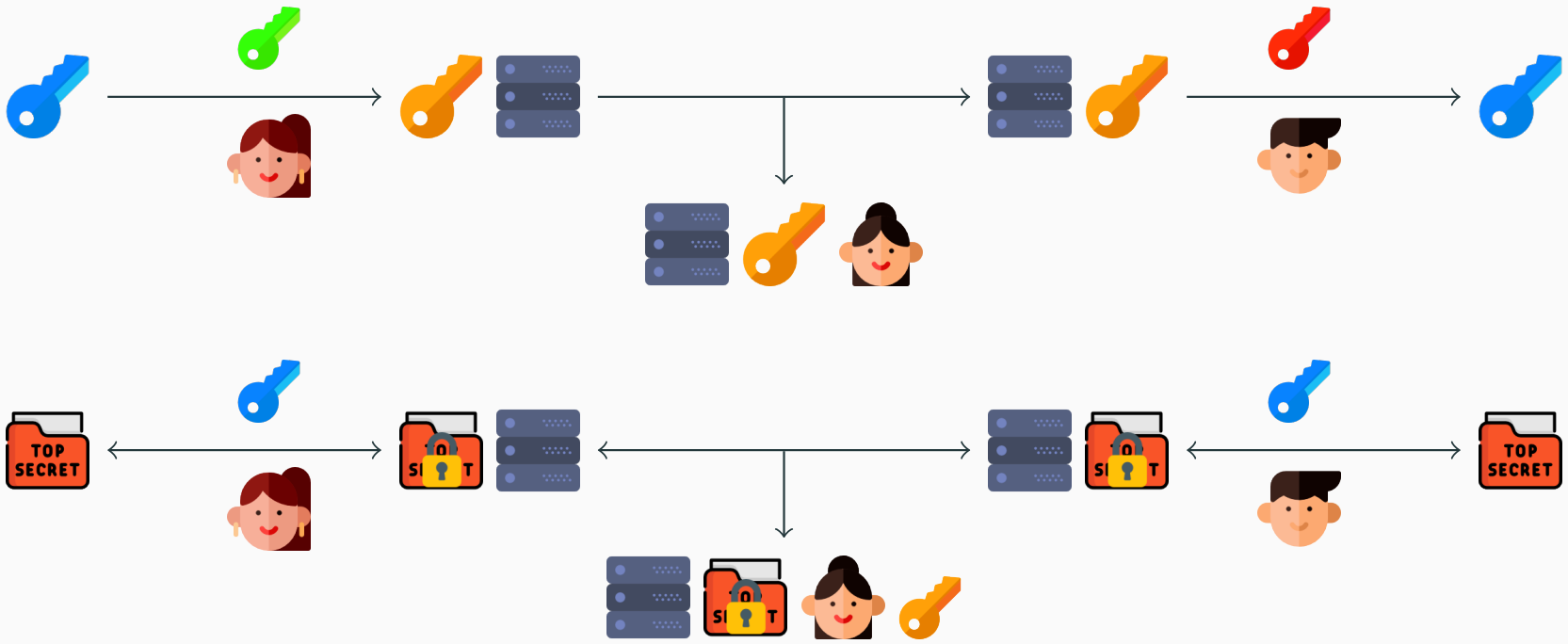


Figure 3 – Chiffrement hybride simplifié

Exemple

Protocoles TLS/SSL

Automate cellulaire

Définition : Automate cellulaire

- Cellules
- États
- Voisinage
- Règle de transition

- Théorie du chaos
- Émergence

Exemple

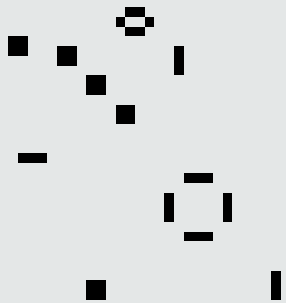


FIGURE 4 – *Jeu de la vie*

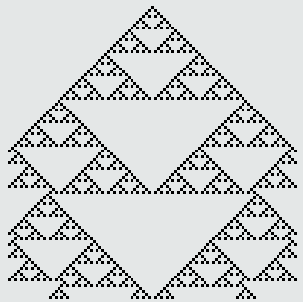


FIGURE 5 – *Règle 90*

Méthode envisagée

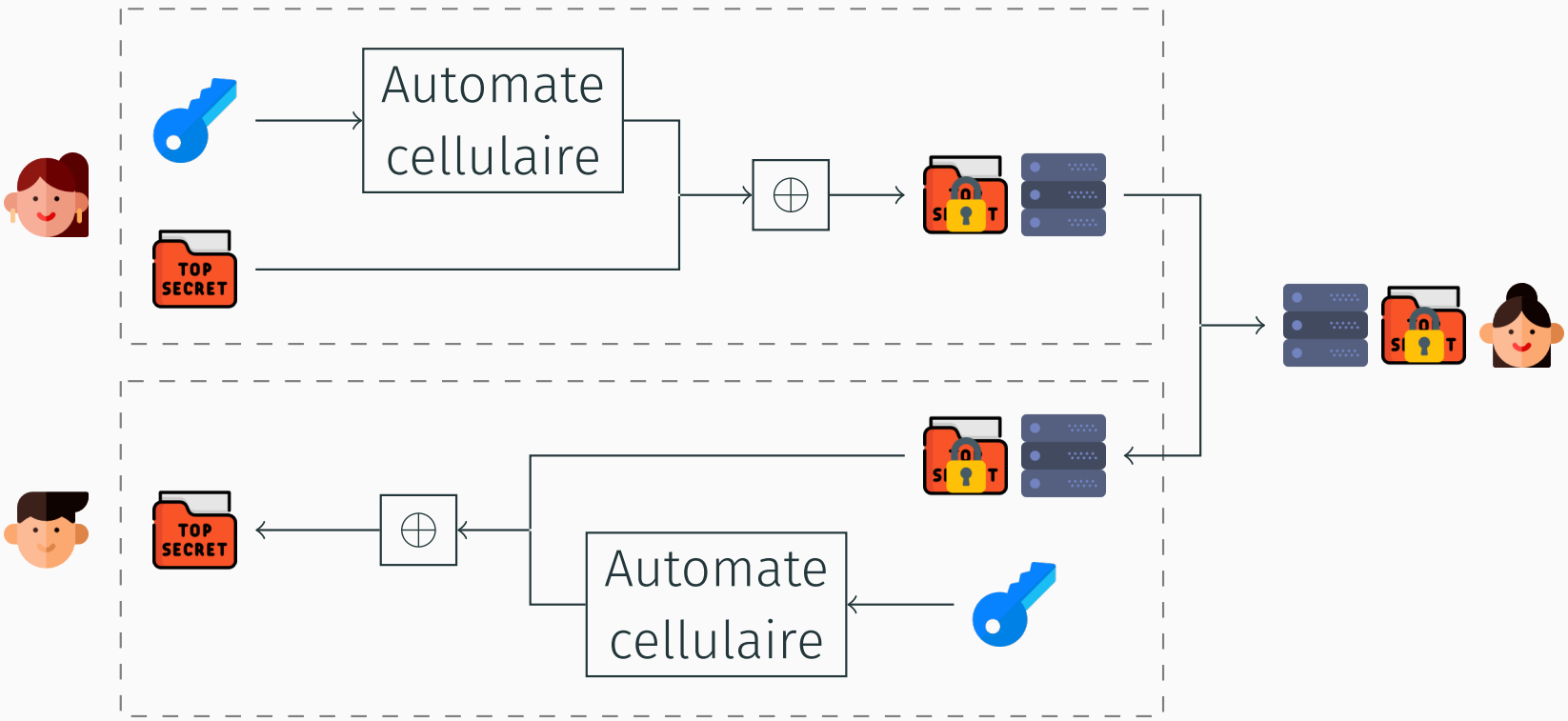


Figure 6 – Principe de fonctionnement

- Méthode de Vernam-Shannon
- Opérateur symétrique : $\oplus$

Automate cellulaire « Fourmi de Langton »

Fourmi de Langton

Définition : Automate cellulaire « Fourmi de Langton » (FDL)

- Bidimensionnel
- Cellules noires et blanches
- État initial quelconque et fourmi au centre
- À chaque étape :
  - Case noire $\Longrightarrow$ rotation de $\frac{\pi}{2}$
  - Case blanche $\Longrightarrow$ rotation de $-\frac{\pi}{2}$
  - Inversion de la couleur et la fourmi avance

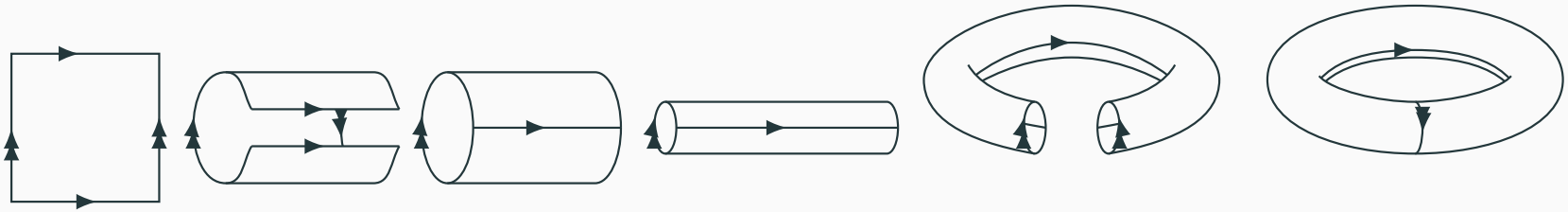


Figure 7 – Topologie d'un tore défini sur un réseau

Chaos et émergence

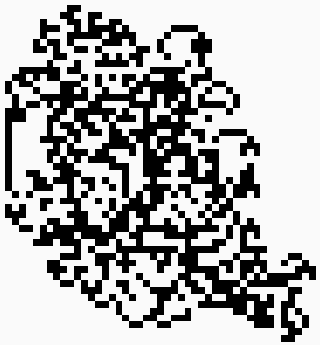


Figure 8 – FDL(60,10000)

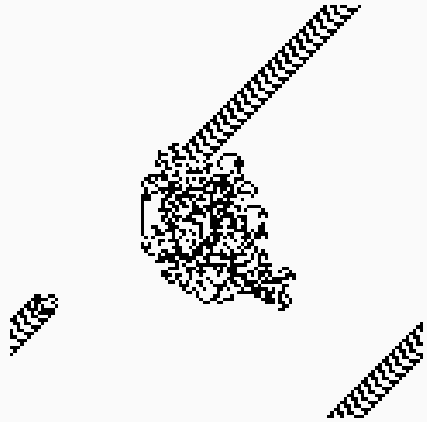


Figure 9 – FDL(120,14000)

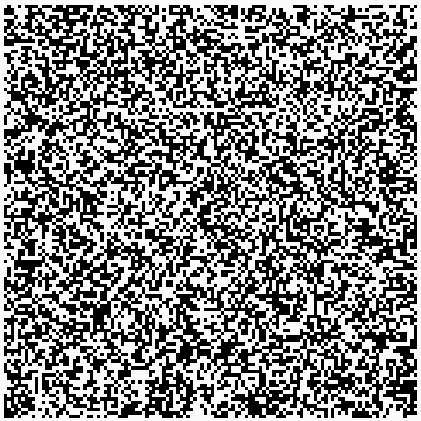


Figure 10 – FDL(120,864000)

FDL(l, n) : Configuration sur le réseau $[\![0, l-1]\!]^2$ après n étapes

Introduction d'une variable aléatoire discrète

Définition

Soit $\mathcal{X}$: « Nombre de passages de la fourmi sur une cellule ».

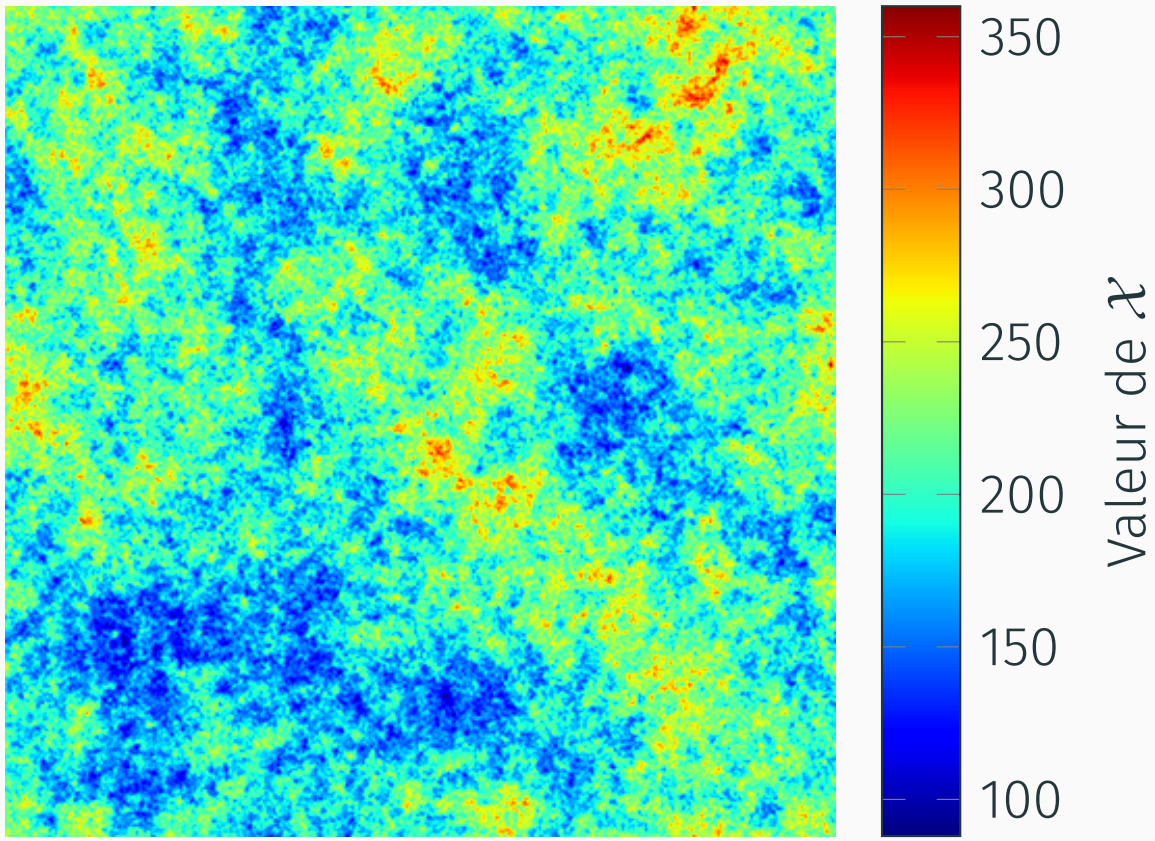


Figure 11 – Représentation de $\mathcal{X}$ pour FDL(400,32000000)

Lien avec une loi normale

Conjecture (Graham Medland)

Pour FDL(l, $\lfloor \frac{1}{2} l^3 \rfloor$) et l très grand, $\mathcal{X} \hookrightarrow \mathcal{N}(\frac{l}{2}, \sigma^2)$

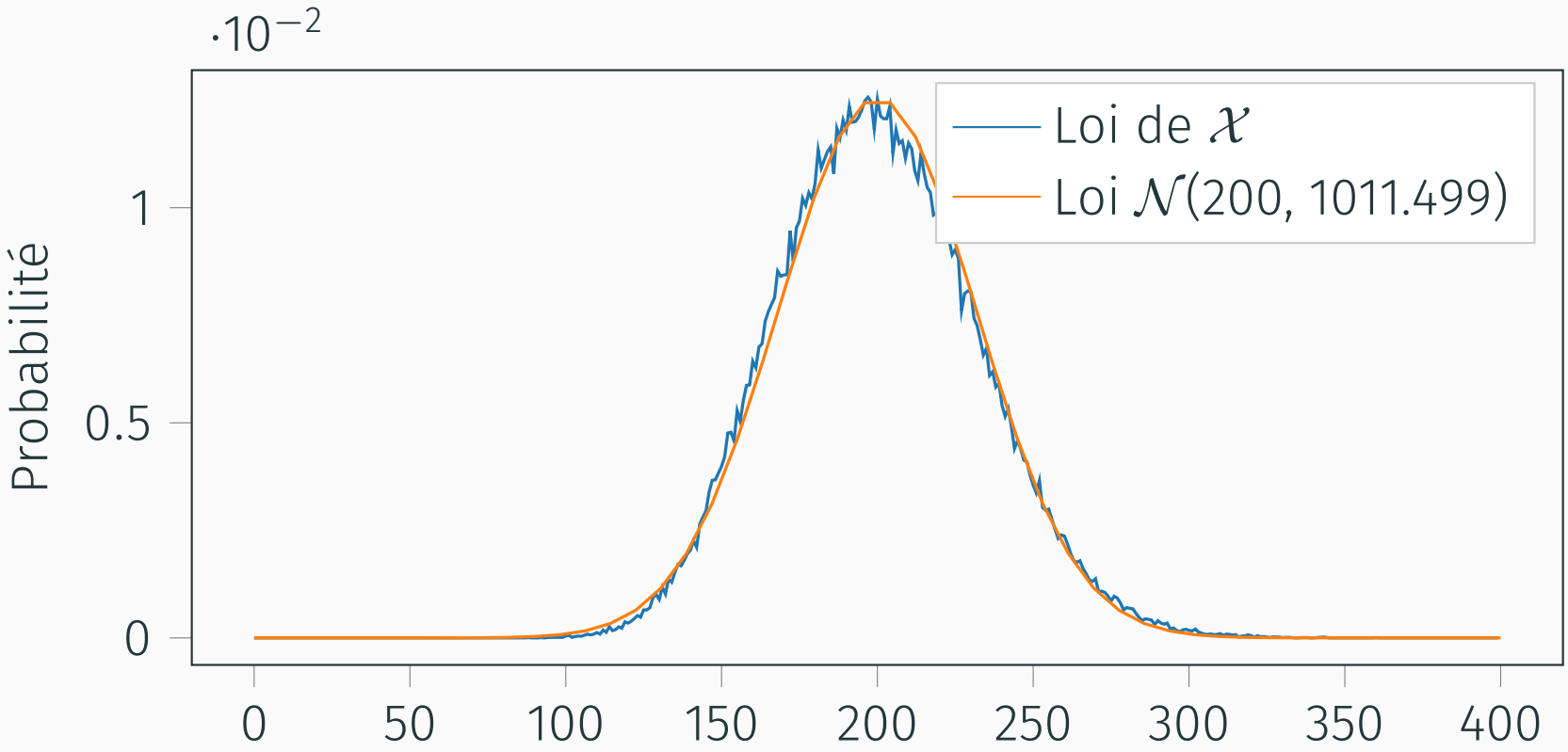


Figure 12 – Loi de $\mathcal{X}$ pour FDL(400,$\lfloor \frac{1}{2} 400^3 \rfloor$)

Tests statistiques

Premiers tests

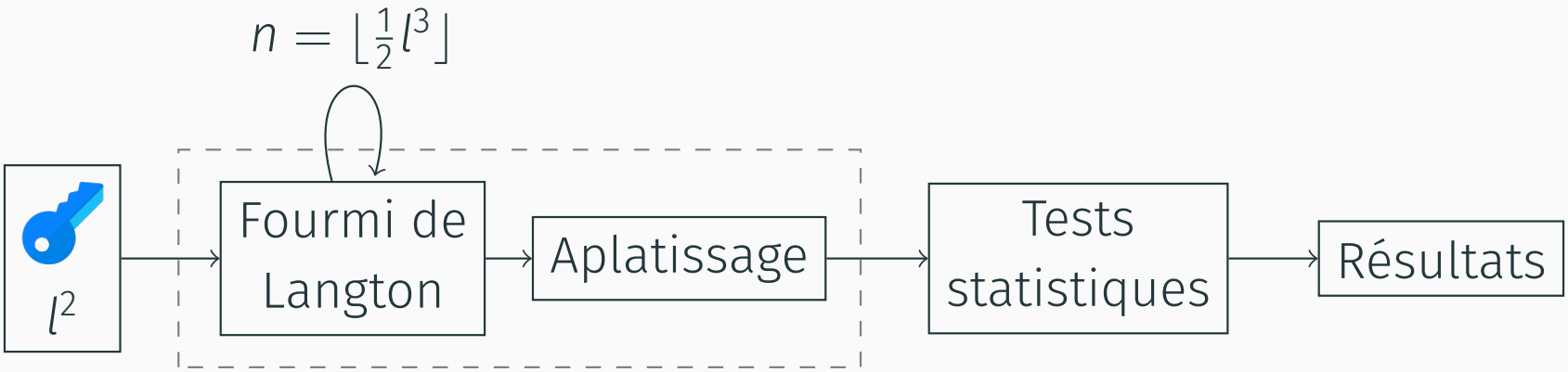


Figure 13 – Procédure de test

Test de fréquence

Fréquence de 0 et 1 proche de $\frac{1}{2}$

Contre-exemple : 1111010101110101110111111101011111011111

Test de fréquence pour une subdivision

Fréquence de 0 et 1 proche de $\frac{1}{2}$ dans un bloc de données

Contre-exemple : **01101111011111011111**0101010101010101010101

Tests avancés

Test du nombre de plateaux

Oscillations entre 0 et 1 ni trop lentes ni trop rapides

Contre-exemple : 01010101010010110101010101010110101010

Test de la longueur du plus long plateau de 1 pour une subdivision

Plus long plateau ni trop grand ni trop petit

Contre-exemple : 01010**1111111111111**01010101010110101010

Test du rang de sous-matrices

Indépendance linéaire entre blocs de données

Contre-exemple : **0100.1000.0001.0010.1111**01100011110011100011

Test de la transformée de Fourier discrète

Peu de répétitions d'un motif

Contre-exemple : 11001100110011001**1100**1100110011001100

Résultats et interprétation

Table 1 – Résultats pour $l = 32, \ldots, 512$ et 🔑 = 00...0

Test	Fréquence de succès
Fréquence	99.6%
Fréquence pour une subdivision	99.4%
Nombre de plateaux	99.2%
Plus long plateau pour une subdivision	98.5%
Rang de sous-matrices	98.5%
Transformée de Fourier discrète	98.8%

Étude pour l = 64

Complexité temporelle

$C_t(l, n) = C_t(n) = \mathcal{O}(l^3)$

Complexité spatiale

$C_s(l, n) = C_s(l) = \mathcal{O}(l^2)$

TABLE 2 – Résultats pour 1000x($l = 64$ et 🔑 : 12x8 = 96 caractères)

Test	Fréquence de succès
Fréquence	99.5%
Fréquence pour une subdivision	99.3%
Nombre de plateaux	98.5%
Plus long plateau pour une subdivision	99.4%
Rang de sous-matrices	98.1%
Transformée de Fourier discrète	98.4%

Exemple de chiffrement

Fonctionnement

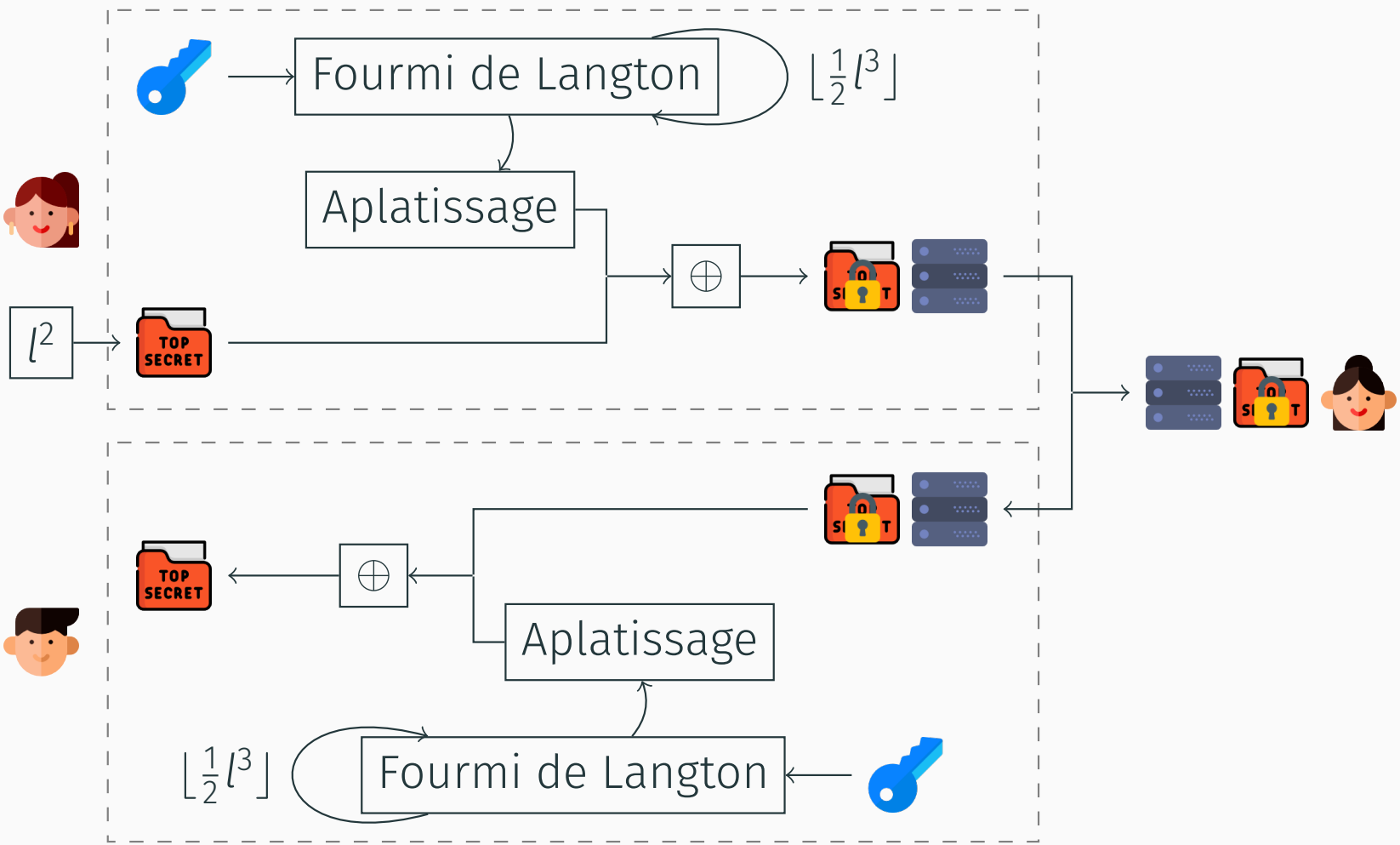


Figure 14 – Schéma de fonctionnement

Chiffrement d’un message

Message brut :
Les sanglots longs
Des violons
De l’automne
Blessent mon coeur
D’une langueur
Monotone.

Tout suffocant
Et blême, quand
Sonne l’heure,
Je me souviens
Des jours anciens
Et je pleure

Et je m’en vais
Au vent mauvais
Qui m’emporte
Deçà, delà,
Pareil à la
Feuille morte.

Paul Verlaine, Poèmes saturniens

Clé :
RadioLondresOpérationNeptune

Message chiffré :
Wu|H|||KW|!|A|||6|||||Un|=|||||”|)||||||
||||dBx||A||||||||<|||||||S|||||||||z|jn
|||||||||||||B[,||5PQ|||D||F|||5>z|||||
||z||||b|’||;|||||||!||’|||=0||||UR|wv|I
|||L||(X|||z|m7M||||;|o|87)V|I||gWc||x|0
)|||||||>|||mR|||i|?||a||||h||,||||||0||
|||||||x|d|Y|Qs|-||||kH||0|r||NZ|L)||| :0
h||”|||H|||=||||||K|||||(||||s|||O|||M||
|89||N||?|||||||1||2||||||||||||1x|||||o
I|2||?|?|||||||M||||LA]US’||qg||BTry|z||
||[|||qHY”|||l|||5||||||||||mq|||D||K|||
|Awf|:i||3b|7|||z|m7M|||||q|4|||S

Note : *L’encodage UTF-8 rend certains caractères illisibles*

Message déchiffré :
Les sanglots longs
Des violons
De l’automne
Blessent mon coeur
D’une langueur
Monotone.

Tout suffocant
Et blême, quand
Sonne l’heure,
Je me souviens
Des jours anciens
Et je pleure

Et je m’en vais
Au vent mauvais
Qui m’emporte
Deçà, delà,
Pareil à la
Feuille morte.

Paul Verlaine, Poèmes saturniens

Chiffrement d'une image

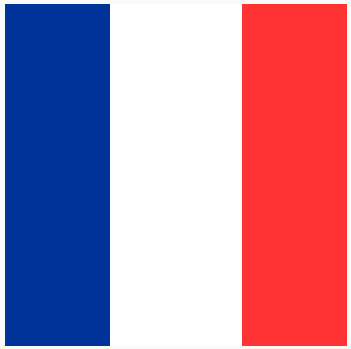


Figure 15 – Image brute

Image 13x13 pt

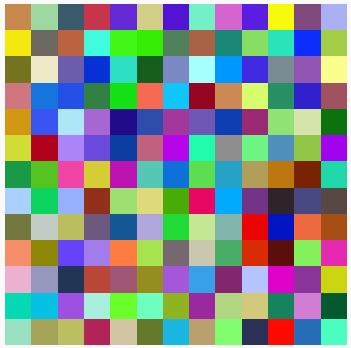


Figure 16 – Image chiffrée

Clé : 10-août-1792

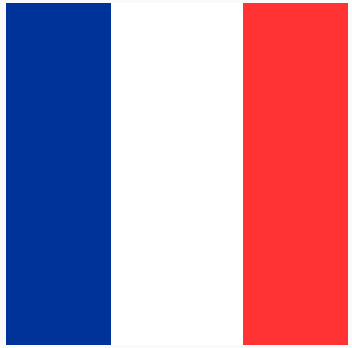


Figure 17 – Image déchiffrée

Clé : 10-août-1792

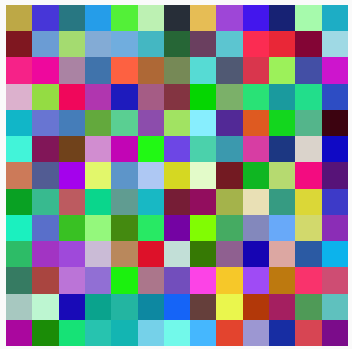


Figure 18 – Tentative de décryptage

Clé : 10-août-1791

Conclusion et limites

Conclusion

Il est possible de généraliser la méthode de chiffrement de Vernam à un automate cellulaire de dimension 2

Réutilisation de la clé

$(A \oplus Clé) \oplus (B \oplus Clé) = A \oplus B$

Solution partielle : Fonction de dérivation de clé

Complexité temporelle

- Pour l = 64 : 0.6s
- Pour l = 512 : Plusieurs minutes

Questions?

Annexes

Fourmi de Langton :

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib2tikz as tikz


def fdl(longueur, n):
    # Réseau à l'état initial
    reseau = np.zeros((longueur, longueur), dtype = np.int)
    # Coordonnées de la fourmi à l'état initial
    x = np.int(longueur / 2)
    y = np.int(longueur / 2)
    orientation = 3

    for i in range(n):
        # Rotation de la fourmi
        if (reseau[x, y] % 2) == 0:
            rotation = -1
            avancer = -1
        else:
            rotation = 1
            avancer = 1

        # Inversion de la couleur de la cellule
        reseau[x, y] = not reseau[x, y]

```

```
26          # Mouvement de la fourmi
27          if orientation == 0:
28              y += avancer
29          elif orientation == 1:
30              x -= avancer
31          elif orientation == 2:
32              y -= avancer
33          else:
34              x += avancer
35
36          # Enroulement du réseau sur le tore
37          x = x % longueur
38          y = y % longueur
39          orientation = (orientation + rotation) % 4
40
41      # Modification des couleurs pour prendre en compte la transparence
42      cdict = {"red": ((0.0, 1.0, 1.0), (1.0, 0.0, 0.0)),
43              "green": ((0.0, 1.0, 1.0), (1.0, 0.0, 0.0)),
44              "blue": ((0.0, 1.0, 1.0), (1.0, 0.0, 0.0))}
45      cdict["alpha"] = ((0.0, 0.0, 0.0), (1.0, 1.0, 1.0))
46      plt.register_cmap(name="FDL", data=cdict)
47
48      plt.matshow(reseau, origin="lower", cmap="FDL",
49          interpolation="none")
50      plt.axis("off")
51      tikz.save("../ressources/fdl-{}-{}.tex".format(longueur, n),
52          figureheight="\\figureheight", figurewidth="\\figurewidth",
53          textsize=6,
54          tex_relative_path_to_data="ressources/", strict=False)
```

```
55
56
57  def fdlo(longueur, n):
58      # Réseau à l'état initial
59      reseau = np.zeros((longueur, longueur), dtype = np.int)
60      occupation = np.zeros((longueur, longueur), dtype = np.int)
61      # Coordonnées de la fourmi à l'état initial
62      x = np.int(longueur / 2)
63      y = np.int(longueur / 2)
64      orientation = 3
65
66      for i in range(n):
67          # Rotation de la fourmi
68          if (reseau[x, y] % 2) == 0:
69              rotation = -1
70              avancer = -1
71          else:
72              rotation = 1
73              avancer = 1
74
75          # Inversion de la couleur de la cellule
76          reseau[x, y] = not reseau[x, y]
77
78          occupation[x, y] += 1
79
80          # Mouvement de la fourmi
81          if orientation == 0:
82              y += avancer
83          elif orientation == 1:
```

```
            x -= avancer
        elif orientation == 2:
            y -= avancer
        else:
            x += avancer

        # Enroulement du réseau sur le tore
        x = x % longueur
        y = y % longueur
        orientation = (orientation + rotation) % 4

    plt.matshow(occupation, origin="lower", cmap=plt.cm.jet,
        interpolation="none")
    plt.axis("off")
    cbar = plt.colorbar()
    cbar.set_label("Valeur de $\\mathcal{X}$")
    tikz.save("../ressources/fdlo-{}-{}.tex".format(longueur, n),
        figureheight="\\figureheight", figurewidth="\\figurewidth",
        textsize=6,
        tex_relative_path_to_data="ressources/", strict=False)
```

Graphique de la loi normale :

```python
import numpy as np
import scipy.stats as scps
import matplotlib.pyplot as plt
import matplotlib2tikz as tikz


def loi(longueur):
    # Nombre d'itérations à effectuer
    n = np.int(0.5 * longueur * longueur * longueur)
    # Réseau à l'état initial
    reseau = np.zeros((longueur, longueur), dtype = np.int)
    occupation = np.zeros((longueur, longueur), dtype = np.int)
    proba_occupation = np.zeros(longueur, dtype = np.int)
    proba_occupation[0] = longueur * longueur
    # Coordonnées de la fourmi à l'état initial
    x = np.int(longueur / 2)
    y = np.int(longueur / 2)
    orientation = 3

    for i in range(n):
        # Rotation de la fourmi
        if (reseau[x, y] % 2) == 0:
            rotation = -1
            avancer = -1
        else:
            rotation = 1
            avancer = 1

```

```
29            # Inversion de la couleur de la cellule
30            reseau[x, y] = not reseau[x, y]
31
32            occupation[x, y] += 1
33            if (occupation[x, y] + 1) <= len(occupation):
34                proba_occupation[occupation[x, y] - 1] -= 1
35                proba_occupation[occupation[x, y]] += 1
36
37            # Mouvement de la fourmi
38            if orientation == 0:
39                y += avancer
40            elif orientation == 1:
41                x -= avancer
42            elif orientation == 2:
43                y -= avancer
44            else:
45                x += avancer
46
47            # Enroulement du réseau sur le tore
48            x = x % longueur
49            y = y % longueur
50            orientation = (orientation + rotation) % 4
51
52        X = np.linspace(0, longueur)
53
54        proba_occupation_normale = proba_occupation / proba_occupation.sum()
55        # L'espérance de la loi normale étant connue, il suffit de calculer sa
56        # variance en résolvant une équation
57        u = longueur / 2
```

```
    y_sommet = ((proba_occupation_normale.max() +
        proba_occupation_normale[np.int(u)]) / 2)
    s = 1 / (y_sommet * np.sqrt(2 * np.pi))
    gaussian = lambda x : ((np.exp(-((x - u) ** 2) / (2 * s * s)))
        / (s * np.sqrt(2 * np.pi)))

    plt.plot(proba_occupation_normale, label = "Loi de $\\mathcal{X}$")
    plt.plot(X, gaussian(X),
            label = "Loi $\\mathcal{{N}}$({}, {})".format(np.int(u),
            np.round(s * s, 3)))
    plt.ylabel("Probabilité")
    plt.legend()

    tikz.save("../ressources/loi-{}.tex".format(longueur),
        figureheight="\\figureheight", figurewidth="\\figurewidth",
        textsize=6,
        tex_relative_path_to_data="ressources/", strict=False)
```

Tests statistiques :

```python
import numpy as np
import scipy as scp
import scipy.special as scpsp
import scipy.fftpack as sfft
import BinaryMatrix as bm


def generer(reseau):
    longueur = len(reseau)
    # Nombre d'itérations à effectuer
    n = np.int(0.5 * longueur * longueur * longueur)
    # Coordonnées de la fourmi à l'état initial
    x = np.int(longueur / 2)
    y = np.int(longueur / 2)
    orientation = 3

    for i in range(n):
        # Rotation de la fourmi
        if (reseau[x, y] % 2) == 0:
            rotation = -1
            avancer = -1
        else:
            rotation = 1
            avancer = 1

        # Inversion de la couleur de la cellule
        reseau[x, y] = not reseau[x, y]

```

```
29          # Mouvement de la fourmi
30          if orientation == 0:
31              y += avancer
32          elif orientation == 1:
33              x -= avancer
34          elif orientation == 2:
35              y -= avancer
36          else:
37              x += avancer
38
39          # Enroulement du réseau sur le tore
40          x = x % longueur
41          y = y % longueur
42          orientation = (orientation + rotation) % 4
43
44      return reseau
45
46
47  def aplatir(matrice):
48      # Aplatissage de la matrice en un vecteur en juxtaposant chaque lignes
49      # de la matrice côte à côte
50      return np.ravel(matrice)
51
52
53  # Fréquence
54  def frequency_monobit_test(vecteur):
55      n = len(vecteur)
56      # Transformation des 0 en -1 et somme des éléments
57      x = 0
```

```
    for e in vecteur:
        x += 2 * e - 1
    x = np.abs(x) / np.sqrt(2 * n)
    return scp.math.erfc(x) >= 0.01


def frequency_block_m_test(vecteur, m):
    n = len(vecteur)
    N = np.int(n / m)
    x = 0
    for i in range(N):
        y = 0
        for j in range(m):
            y += vecteur[m * i + j]
        y = (y / m - 0.5) * (y / m - 0.5)
        x += y
    x = 4 * m * x
    return scpsp.gammaincc(N / 2, x / 2) >= 0.01


# Fréquence pour une subdivision
def frequency_block_test(vecteur):
    n = len(vecteur)
    m = 128
    assert n >= m
    return frequency_block_m_test(vecteur, m)


# Nombre de plateaux
```

```python
# CONDITION : frequency_monobit_test == True
def runs_test(vecteur):
    n = len(vecteur)
    p = 0
    for e in vecteur:
        p += e
    p = p / n
    v = 0
    for i in range(n - 1):
        if vecteur[i] == vecteur[i + 1]:
            v += 0
        else:
            v += 1
    v = v + 1
    x = (np.abs(v - 2 * n * p * (1 - p))) / (2 * np.sqrt(2 * n) * p * (1 - p))
    return scp.math.erfc(x) >= 0.01


LONGEST_RUN_ONES_MIN = np.array([128, 6272, 750000], dtype = np.int)
LONGEST_RUN_ONES_M = np.array([8, 128, 10000], dtype = np.int)
LONGEST_RUN_ONES_K = np.array([3, 5, 6], dtype = np.int)
LONGEST_RUN_ONES_CATEGORY = np.array([  [1, 4, 10],
                                        [2, 5, 11],
                                        [3, 6, 12],
                                        [4, 7, 13],
                                        [-1, 8, 14],
                                        [-1, 9, 15],
                                        [-1, -1, 16]
                                    ], dtype = np.int)
```

```
LONGEST_RUN_ONES_FREQUENCY = np.array([ [0.2148, 0.1174, 0.0882],
                                        [0.3672, 0.2430, 0.2092],
                                        [0.2305, 0.2493, 0.2483],
                                        [0.1875, 0.1752, 0.1933],
                                        [-1, 0.1027, 0.1208],
                                        [-1, 0.1124, 0.0675],
                                        [-1, -1, 0.0727]
                                      ], dtype = np.float)

def longest_run_ones_m_block_test(vecteur, m, m_i):
    m_longueur = LONGEST_RUN_ONES_K[m_i] + 1
    n = len(vecteur)
    N = np.int(n / m)
    longest_run_ones_of_length = np.zeros(m_longueur)
    for i in range(N):
        v = 0
        w = 0
        for j in range(m):
            if vecteur[m * i + j] == 1:
                w += 1
            else:
                v = np.max([v, w])
                w = 0
        v = np.max([v, w])
        if v <= LONGEST_RUN_ONES_CATEGORY[0, m_i]:
            longest_run_ones_of_length[0] += 1
        elif v >= LONGEST_RUN_ONES_CATEGORY[m_longueur - 1, m_i]:
            longest_run_ones_of_length[m_longueur - 1] += 1
        else:
```

```
145                 for k in range(1, m_longueur - 1):
146                     if v == LONGEST_RUN_ONES_CATEGORY[k, m_i]:
147                         longest_run_ones_of_length[k] += 1
148         x = 0
149         for i in range(m_longueur):
150             x += (np.square(longest_run_ones_of_length[i] - (N
151                 * LONGEST_RUN_ONES_FREQUENCY[i][m_i]))
152                 / (N * LONGEST_RUN_ONES_FREQUENCY[i][m_i]))
153         return scpsp.gammaincc(LONGEST_RUN_ONES_K[m_i] / 2, x / 2) >= 0.01
154 
155 
156     # Plus long plateau pour une subdivision
157     def longest_run_ones_block_test(vecteur):
158         n = len(vecteur)
159         assert n >= LONGEST_RUN_ONES_MIN[0]
160         for i in range(len(LONGEST_RUN_ONES_MIN) - 1, -1, -1):
161             if n >= LONGEST_RUN_ONES_MIN[i]:
162                 succes = longest_run_ones_m_block_test(vecteur,
163                     LONGEST_RUN_ONES_M[i], i)
164                 return succes
165         return False # Ce cas ne devrait jamais arriver.
166 
167 
168     BINARY_MATRIX_RANK_FREQUENCY = np.array([0.2888,
169                                              0.5776,
170                                              0.1336], dtype = np.float)
171 
172     def binary_matrix_mq_rank_test(vecteur, m, q):
173         n = len(vecteur)
```

```
    N = np.int(n / (m * q))
    f = [0, 0, 0]
    for i in range(N):
        mat = np.zeros((m, q))
        for k in range(q):
            for l in range(m):
                mat[l, k] = vecteur[(m * q) * i + k + q * l]
        # La fonction de calcul du rang avec Numpy ne fonctionne pas dans
        # le cas d'une grande matrice binaire.
        # rg = np.linalg.matrix_rank(mat)
        ranker = bm.BinaryMatrix(mat, m, q)
        rg = ranker.compute_rank()
        if rg == m:
            f[0] += 1
        elif rg == (m - 1):
            f[1] += 1
        else:
            f[2] += 1
    x = 0
    for i in range(len(f)):
        x += (np.square(f[i] - (N
            * BINARY_MATRIX_RANK_FREQUENCY[i]))
            / (N * BINARY_MATRIX_RANK_FREQUENCY[i]))
    return scpsp.gammaincc(1, x / 2) >= 0.01


# Rang de sous-matrices
def binary_matrix_rank_test(vecteur):
    n = len(vecteur)
```

```
203         m = 32
204         q = 32
205         assert n >= m * q
206         return binary_matrix_mq_rank_test(vecteur, m, q)
207 
208 
209     # Transformée de Fourier discrète
210     def discrete_fourier_transform_test(vecteur):
211         n = len(vecteur)
212         vecteur_transforme = []
213         for e in vecteur:
214             vecteur_transforme.append(2 * e - 1)
215         s = sfft.fft(vecteur_transforme)
216         s = s[:np.int(n / 2)]
217         m = np.abs(s)
218         # Valeur seuil pour les amplitudes
219         t = np.sqrt(np.log(1 / 0.05) * n)
220         N0 = 0.95 * n / 2
221         # Nombre d'éléments inférieurs à la valeur seuil
222         N1 = 0
223         for hauteur in m:
224             if hauteur < t:
225                 N1 += 1
226         d = (N1 - N0) / np.sqrt(n * 0.95 * 0.05 / 4)
227         x = np.abs(d) / np.sqrt(2)
228         return scp.math.erfc(x) >= 0.01
229 
230 
231     tests = [frequency_monobit_test, frequency_block_test, runs_test,
```

```
    longest_run_ones_block_test, binary_matrix_rank_test,
    discrete_fourier_transform_test]

# Test correspondant à la table 1
def test():
    n = 0
    resultats = {}
    for i in range(len(tests)):
        resultats[tests[i].__name__] = 0
    for l in range(32, 256 + 1, 1):
        n += 1
        reseau = np.zeros((l, l), dtype = np.int)
        vecteur = aplatir(generer(reseau))
        for i in range(len(tests)):
            if tests[i](vecteur):
                resultats[tests[i].__name__] += 1
    print("Nombre d'expériences effectuées : {}".format(n))
    print("Résultats :")
    print(resultats)


# Test correspondant à la table 2
def test64():
    n = 0
    resultats = {}
    for i in range(len(tests)):
        resultats[tests[i].__name__] = 0
    l = 64
    for k in range(1000):
```

```
            n += 1
            reseau = np.zeros((l, l), dtype = np.int)
            # Création d'une clé aléatoire de départ
            for i in range(l):
                for j in range(l):
                    reseau[i, j] = np.random.randint(0, 2)
            vecteur = aplatir(generer(reseau))
            for i in range(len(tests)):
                if tests[i](vecteur):
                    resultats[tests[i].__name__] += 1
        print("Nombre d'expériences effectuées pour l = 64 : {}".format(n))
        print("Résultats :")
        print(resultats)
```

Chiffrement :

```
import numpy as np
import matplotlib.image as mpimg


LONGUEUR = 64


def generer(reseau):
    # Nombre d'itérations à effectuer
    n = np.int(0.5 * LONGUEUR * LONGUEUR * LONGUEUR)
    # Coordonnées de la fourmi à l'état initial
```

```
    x = np.int(LONGUEUR / 2)
    y = np.int(LONGUEUR / 2)
    orientation = 3

    for i in range(n):
        # Rotation de la fourmi
        if (reseau[x, y] % 2) == 0:
            rotation = -1
            avancer = -1
        else:
            rotation = 1
            avancer = 1

        # Inversion de la couleur de la cellule
        reseau[x, y] = not reseau[x, y]

        # Mouvement de la fourmi
        if orientation == 0:
            y += avancer
        elif orientation == 1:
            x -= avancer
        elif orientation == 2:
            y -= avancer
        else:
            x += avancer

        # Enroulement du réseau sur le tore
        x = x % LONGUEUR
        y = y % LONGUEUR
```

```
41          orientation = (orientation + rotation) % 4
42
43      return reseau
44
45
46  def aplatir(matrice):
47      # Aplatissage de la matrice en un vecteur en juxtaposant chaque lignes
48      # de la matrice côte à côte
49      return np.ravel(matrice)
50
51
52  def int_to_bits(n):
53      if n < 0:
54          raise("Nombre négatif !")
55      elif n < 2:
56          return [n]
57      else:
58          return int_to_bits(n // 2) + [n % 2]
59
60
61  def bits_to_int(bits):
62      l = len(bits)
63      n = 0
64      for i in range(0, l):
65          n += bits[l - i - 1] * (2 ** i)
66
67      return n
68
69
```

```
70  # Retourne la représentation ASCII binaire d'une chaîne de caractères
71  def string_to_bits(str, bits_len):
72      bits = []
73      for lettre in str:
74          char_bits = int_to_bits(ord(lettre))
75          for i in range(0, bits_len - len(char_bits)):
76              char_bits.insert(0, 0)
77          bits.extend(char_bits)
78
79      return bits
80
81
82  # Retourne une chaîne de caractères à partir de sa représentation ASCII binaire
83  def bits_to_string(bits, bits_len):
84      str = ""
85      for char_bits in ([bits[i:i + bits_len]
86          for i in range(0, len(bits), bits_len)]):
87              str = str + chr(bits_to_int(char_bits))
88
89      return str
90
91
92  def img_to_bits(fichier):
93      bits = []
94
95      img = mpimg.imread(fichier)
96      for ligne in img:
97          for cellule in ligne:
98              r = int_to_bits(np.int(cellule[0] * 255))
```

```
            for i in range(0, 8 - len(r)):
                r.insert(0, 0)
            bits.extend(r)
            v = int_to_bits(np.int(cellule[1] * 255))
            for i in range(0, 8 - len(v)):
                v.insert(0, 0)
            bits.extend(v)
            b = int_to_bits(np.int(cellule[2] * 255))
            for i in range(0, 8 - len(b)):
                b.insert(0, 0)
            bits.extend(b)
    return bits, len(img[0])


def bits_to_img(bits, c):
    img = []

    ligne = []
    for img_bits in [bits[i:i + 24] for i in range(0, len(bits), 24)]:
        cellule = []
        for c_bits in [img_bits[i:i + 8] for i in range(0, 24, 8)]:
            cellule.append(bits_to_int(c_bits) / 255)
        ligne.append(cellule)
        if len(ligne) ==  c:
            img.append(ligne)
            ligne = []

    return img

```

```python

def chiffrer_str(message, cle):
    assert 8 * len(message) <= LONGUEUR * LONGUEUR
    assert 8 * len(cle) <= LONGUEUR * LONGUEUR

    reseau = np.zeros((LONGUEUR, LONGUEUR), dtype = np.int)
    cle_binaire = string_to_bits(cle, 8)
    message_binaire = string_to_bits(message, 8)
    k = 0
    for bit in cle_binaire:
        reseau[k // LONGUEUR, k % LONGUEUR] = bit
        k += 1
    vecteur = aplatir(generer(reseau))
    for i in range(len(message_binaire), len(vecteur)):
        message_binaire.append(0)
    message_chiffre_binaire = np.bitwise_xor(vecteur, message_binaire).tolist()
    message_chiffre = bits_to_string(message_chiffre_binaire, 8)[:len(message)]

    return message_chiffre


def dechiffrer_str(message_chiffre, cle):
    return chiffrer(message_chiffre, cle)


def chiffrer_img(fichier, cle):
    assert str.split(fichier, ".")[-1] == "png"
    assert 8 * len(cle) <= LONGUEUR * LONGUEUR

```

```
    img_binaire, colonnes = img_to_bits(fichier)

    assert len(img_binaire) <= LONGUEUR * LONGUEUR

    reseau = np.zeros((LONGUEUR, LONGUEUR), dtype = np.int)
    cle_binaire = string_to_bits(cle, 8)
    k = 0
    for bit in cle_binaire:
        reseau[k // LONGUEUR, k % LONGUEUR] = bit
        k += 1
    vecteur = aplatir(generer(reseau))
    for i in range(len(img_binaire), len(vecteur)):
        img_binaire.append(0)
    img_chiffre_binaire = np.bitwise_xor(vecteur, img_binaire).tolist()
    img_chiffre = bits_to_img(img_chiffre_binaire, 13)

    return img_chiffre


def dechiffrer_img(message_chiffre, cle):
    return chiffrer_img(message_chiffre, cle)


def exemple_str():
    message = """
        Les sanglots longs
        Des violons
        De l'automne
        Blessent mon coeur
```

```
            D'une langueur
            Monotone.

            Tout suffocant
            Et blême, quand
            Sonne l'heure,
            Je me souviens
            Des jours anciens
            Et je pleure

            Et je m'en vais
            Au vent mauvais
            Qui m'emporte
            Deçà, delà,
            Pareil à la
            Feuille morte.

            Paul Verlaine, Poèmes saturniens"""
        cle = "RadioLondresOpérationNeptune"

        print("Message chiffré :")
        print(chiffrer_str(message, cle))
        print("\n\n\nMessage déchiffré :")
        print(chiffrer_str(chiffrer_str(message, cle), cle))


    def exemple_img():
        fichier = "../ressources/france.png"
        cle = "10-août-1792"
```

```
    mpimg.imsave("../ressources/france-chiffre.png",
        chiffrer_img(fichier, cle))

    fichier = "../ressources/france-chiffre.png"
    cle = "10-août-1792"
    mpimg.imsave("../ressources/france-dechiffre.png",
        dechiffrer_img(fichier, cle))

    fichier = "../ressources/france-chiffre.png"
    cle = "10-août-1791"
    mpimg.imsave("../ressources/france-dechiffre-chaos.png",
        dechiffrer_img(fichier, cle))
```

Explications supplémentaires

Définition : Automate cellulaire

$\lambda = (d, Q, V, \delta)$ où :

- $d \in \mathbb{N}^*$: **dimension** (son réseau est alors $\mathbb{Z}^d$)
- Q : **alphabet**, ensemble fini non vide
- $V \subset \mathbb{Z}^d$: **voisinage**
- $\delta : Q^a \to Q$: **règle locale de transition**, où $a = |V|$

Fonction globale d'évolution :

$$F_\delta : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$$

$$\forall c \in Q^{\mathbb{Z}^d}, F_\delta(c) : z \mapsto \delta(c(z + v_1), ..., c(z + v_a))$$

où $V = \{v_1, ..., v_a\}$

Définition : Voisinage de von Neumann de rayon r

$$V^{v}_{(x_0,y_0)} = \{M : N_1(M - M_0) \leqslant r\} = \{(x,y) : |x - x_0| + |y - y_0| \leqslant r\}$$

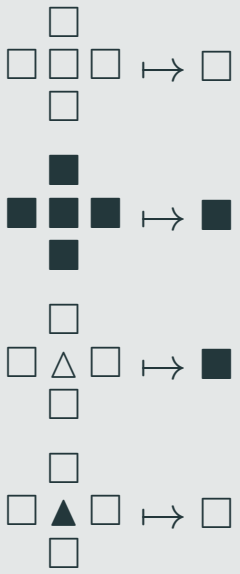
Classification de Wolfram

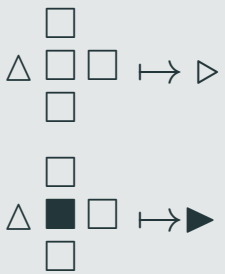
- **Classe I :** État terminal homogène sans motif stable périodique
- **Classe II :** Structures stables ou périodiques
- **Classe III :** Chaotique avec motifs apériodiques
- **Classe IV :** Émergence de structures complexes capables d'osciller ou se mouvoir

Définition : Automate cellulaire « Fourmi de Langton »

$\lambda = (d_\lambda, Q_\lambda, V_\lambda, \delta_\lambda)$ où :

- $d_\lambda = 2$
- $Q_\lambda = \{\square; \blacksquare; \triangle; \triangledown; \triangleright; \triangleleft; \blacktriangle; \blacktriangledown; \blacktriangleright; \blacktriangleleft\}$
- $V_\lambda = V^v_{(0,0)} = \{(0,0); (0,1); (1,0); (0,-1); (-1,0)\}$
- $\delta_\lambda : Q_\lambda^5 \to Q_\lambda$





$\vdots$

Définition : $\oplus$ « ou exclusif »

Table 3 – Table de vérité de $\oplus$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Propriétés : $\oplus$

- $A \oplus A = 0$
- $(A \oplus B) \oplus B = A$

Preuve du test de fréquence

Soit $n \in \mathbb{N}^*$ et $\mathcal{E} = \varepsilon_1 ... \varepsilon_n$ où $\forall i \in [\![1, n]\!], \varepsilon_i \in \{0, 1\}$.
Soit $\mathcal{X}_{th} \hookrightarrow \mathcal{B}(n, \frac{1}{2})$.

D'après le théorème de Moivre-Laplace :
$\frac{\mathcal{X}_{th} - E(\mathcal{X}_{th})}{\sigma(\mathcal{X}_{th})}$ converge en loi vers $\mathcal{Y}_{th} \hookrightarrow \mathcal{N}(0, 1)$.

Soit $\mathcal{X}_{exp} = \sum_{i=1}^{n} \varepsilon_i$. On pose $\mathcal{Y}_{exp} = \frac{\sum_{i=1}^{n} (2\varepsilon_i - 1)}{\sqrt{n}}$.
D'après le principe de valeur p, on accepte que la séquence $\mathcal{E}$ soit aléatoire à 1% près si :

$P(|\mathcal{Y}_{th}| \geqslant |\mathcal{Y}_{exp}|) \geqslant 0.01$. On pose $z = \frac{\left|\sum_{i=1}^{n} (2\varepsilon_i - 1)\right|}{\sqrt{2n}}$.

$$P(|\mathcal{Y}_{th}| \geqslant |\mathcal{Y}_{exp}|) \geqslant 0.01 \implies P\left(|\mathcal{Y}_{th}| \geqslant \frac{\left|\sum_{i=1}^{n} (2\varepsilon_i - 1)\right|}{\sqrt{n}}\right) \geqslant 0.01 \implies P(|\mathcal{Y}_{th}| \geqslant \sqrt{2}z) \geqslant 0.01$$

$$\implies 2P(\mathcal{Y}_{th} \geqslant \sqrt{2}z) \geqslant 0.01 \implies \frac{2}{\sqrt{2\pi}} \int_{\sqrt{2}z}^{+\infty} e^{-\frac{x^2}{2}} \mathrm{d}x \geqslant 0.01 \implies \sqrt{\frac{2}{\pi}} \int_{\sqrt{2}z}^{+\infty} e^{-\frac{x^2}{2}} \mathrm{d}x \geqslant 0.01$$

$$\implies \boxed{\frac{2}{\sqrt{\pi}} \int_{z}^{+\infty} e^{-u^2} \mathrm{d}u \geqslant 0.01} \quad (u = \tfrac{x}{\sqrt{2}})$$