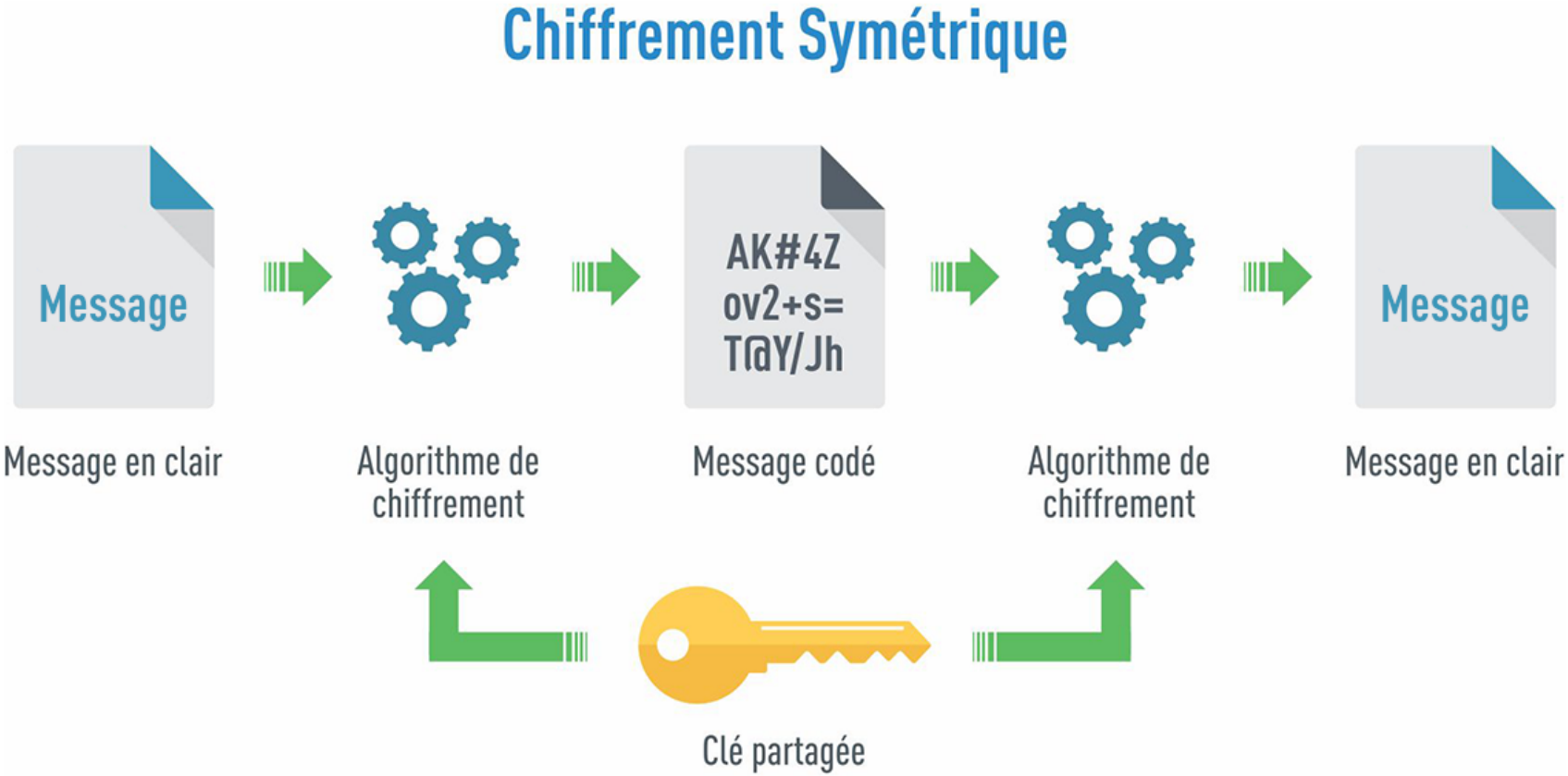
Sécurisation du transport de données

Baptiste Pasquier 30771

2019

Cryptographie symétrique



Source: `blog.emsisoft.com`

Définition 1 : Réseau de substitution-permutation

Un **réseau de substitution-permutation** est une architecture de chiffrement par bloc constituée d'une couche d'addition de clé, d'une couche de substitution et d'une couche de permutation

Définition 1 : Réseau de substitution-permutation

Un **réseau de substitution-permutation** est une architecture de chiffrement par bloc constituée d'une couche d'addition de clé, d'une couche de substitution et d'une couche de permutation

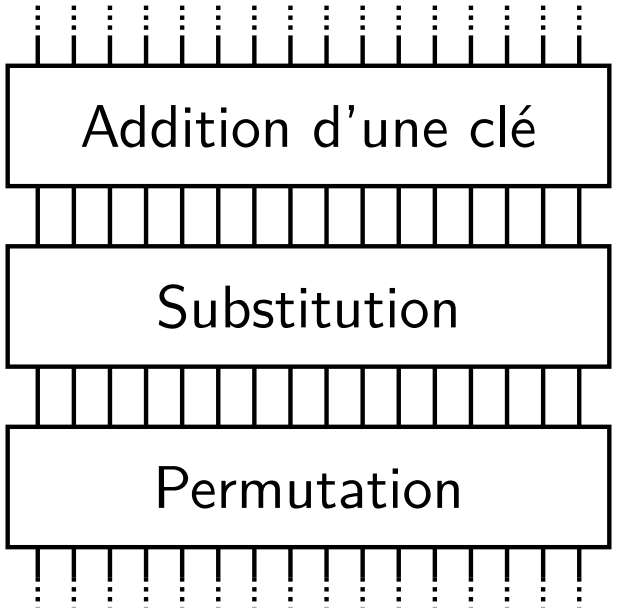


FIGURE – Tour d'un SPN sur 16 bits

Définition 2 : Fonction XOR

On définit la fonction OU exclusif (XOR, $\oplus$) par sa table de vérité :

x	0	0	1	1
y	0	1	0	1
$x \oplus y$	0	1	1	0

Définition 2 : Fonction XOR

On définit la fonction OU exclusif (XOR, $\oplus$) par sa table de vérité :

x	0	0	1	1
y	0	1	0	1
$x \oplus y$	0	1	1	0

On note $\mathbb{F}_2$ le corps $\mathbb{Z}/2\mathbb{Z}$.

Définition 3 : Extension à $\mathbb{F}_2^n$

Soit $n \in \mathbb{N}^*$. On étend la définition de la fonction $\oplus$ pour définir la fonction $\oplus : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ par :

$$\oplus : \begin{cases} \mathbb{F}_2^n \times \mathbb{F}_2^n & \longrightarrow \mathbb{F}_2^n \\ (x_1, \ldots, x_n), (y_1, \ldots, y_n) & \longmapsto x_1 \oplus y_1, \ldots, x_n \oplus y_n \end{cases}$$

Définition 4 : Fonction booléenne

Une fonction booléenne de n variables est une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2$.

Définition 4 : Fonction booléenne

Une fonction booléenne de n variables est une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2$.

x_1	0	1	0	1
x_2	0	0	1	1
$f(x_1, x_2)$	0	1	0	0

TABLE – Table de vérité d'une fonction booléenne de 2 variables

Définition 5 : Sbox

Une table de substitution (Sbox) est la table de vérité d'une fonction de $\mathbb{F}_2^m$ dans $\mathbb{F}_2^n$. Elle est donc composée de n fonctions booléennes de m variables.

x_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$S_1(x_1, x_2, x_3, x_4)$	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	1
$S_2(x_1, x_2, x_3, x_4)$	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	1
$S_3(x_1, x_2, x_3, x_4)$	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0	1
$S_4(x_1, x_2, x_3, x_4)$	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0	0

TABLE – Sbox de $\mathbb{F}_2^4$ dans $\mathbb{F}_2^4$

x_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$S_1(x_1, x_2, x_3, x_4)$	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	1
$S_2(x_1, x_2, x_3, x_4)$	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	1
$S_3(x_1, x_2, x_3, x_4)$	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0	1
$S_4(x_1, x_2, x_3, x_4)$	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0	0

TABLE – Sbox de $\mathbb{F}_2^4$ dans $\mathbb{F}_2^4$

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	e	4	d	1	2	f	b	8	3	a	6	c	5	9	0	7

TABLE – Même Sbox en écriture hexadécimale

Définition 6 : Pbox

Une table de permutation (Pbox) est le tableau de valeurs d'une bijection de $\{0, \ldots, n-1\}$.

Définition 6 : Pbox

Une table de permutation (Pbox) est le tableau de valeurs d'une bijection de $\{0, \ldots, n-1\}$.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$P(x)$	0	4	8	c	1	5	9	d	2	6	a	e	3	7	b	f

TABLE – Pbox

Définition 6 : Pbox

Une table de permutation (Pbox) est le tableau de valeurs d'une bijection de $\{0, \ldots, n-1\}$.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$P(x)$	0	4	8	c	1	5	9	d	2	6	a	e	3	7	b	f

TABLE – Pbox

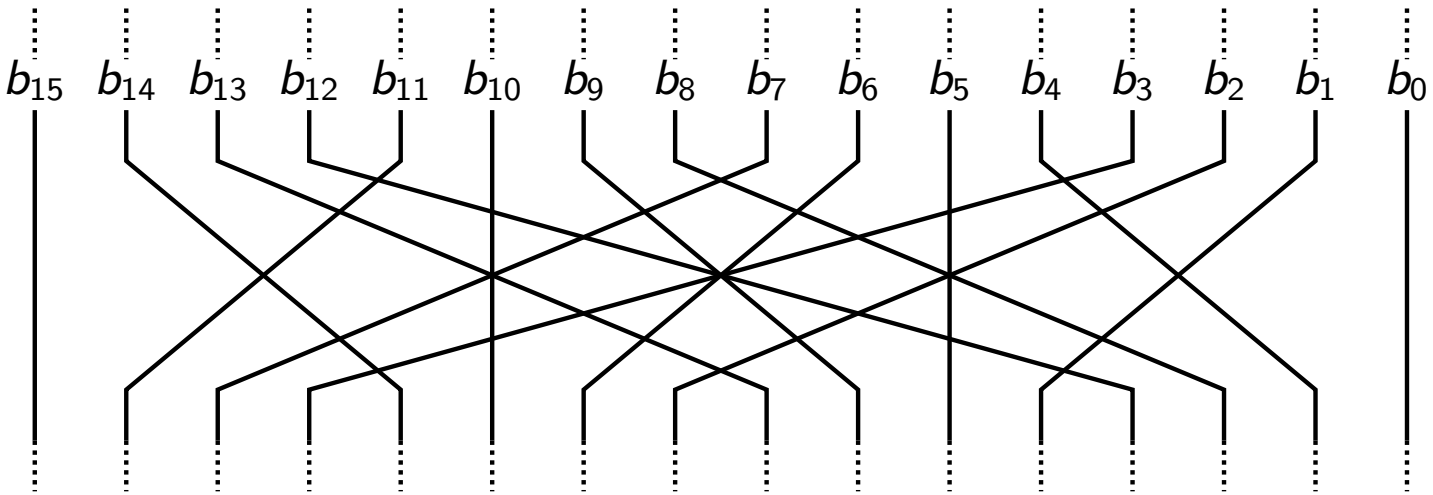
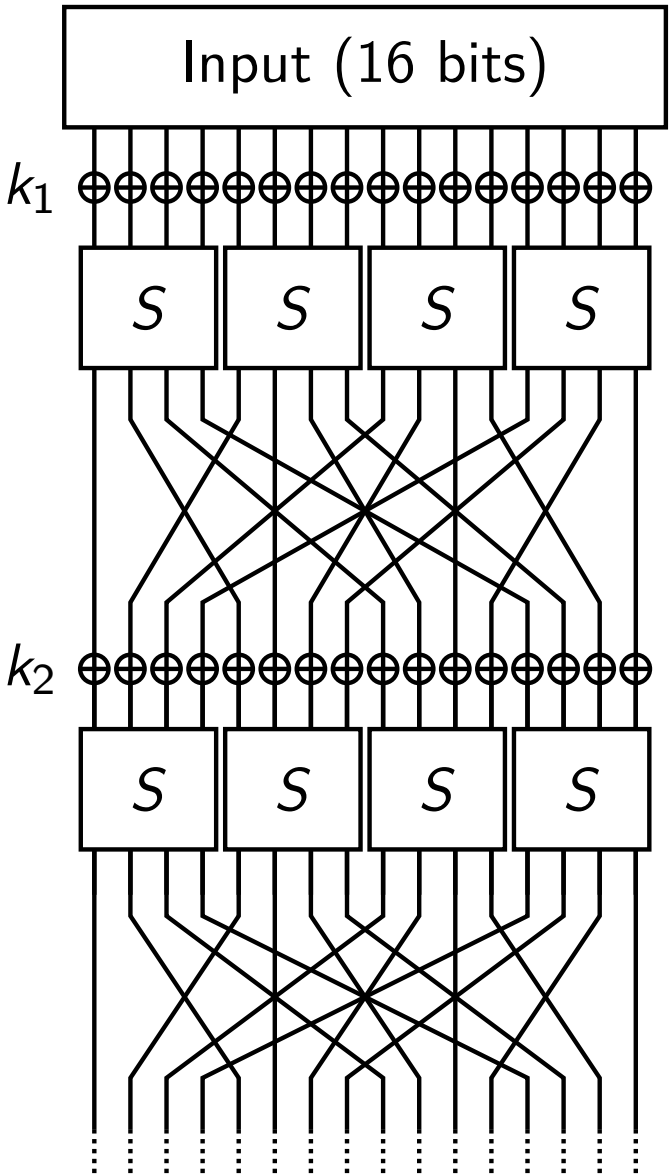
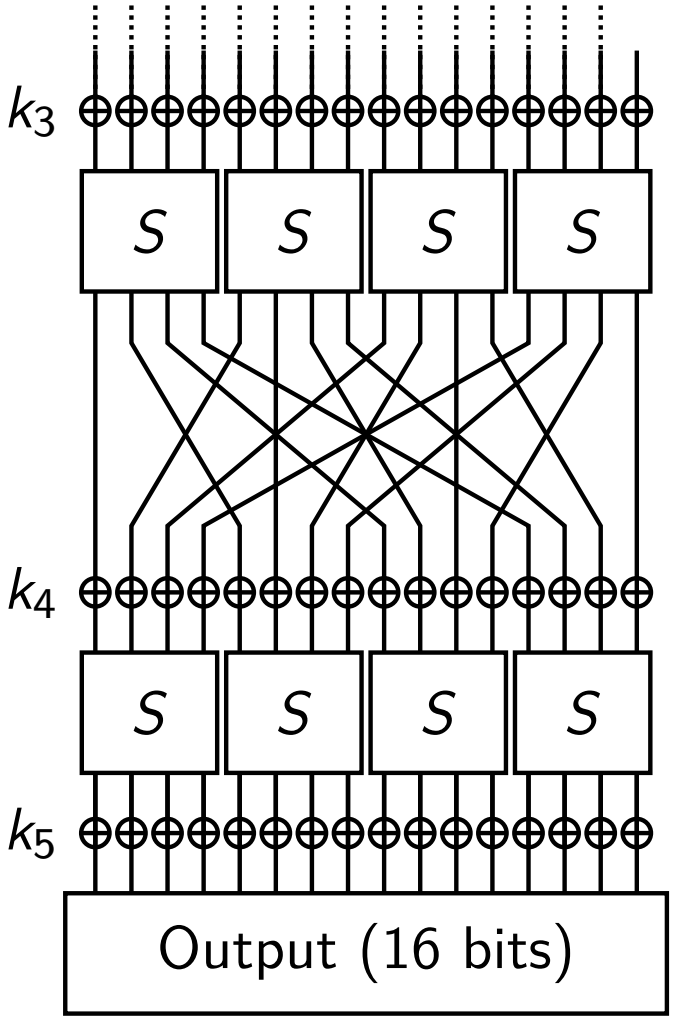


FIGURE – Schéma de la Pbox


Input (16 bits)
k_1
S S S S
k_2
S S S S


k_3
S S S S
k_4
S S S S
k_5
Output (16 bits)

Problème

Comment retrouver les clés k_i en supposant pouvoir appliquer l'algorithme de chiffrement à n'importe quelle chaine ?

Soit un algorithme de chiffrement.

Définition 7 : Différentielle

Une **différentielle** est un couple $(\Delta X, \Delta Y) \in (\mathbb{F}_2^n)^2$ de différence en entrée et de différence en sortie de l'algorithme de chiffrement.

Définition 8 : Chemin différentiel

Un **chemin différentiel** est un (r)-uplet de $(F_2^n)^r$ correspondant à des différences à chaque étape de l'algorithme de chiffrement.

Soit $X, X' \in (\mathbb{F}_2^n)^2$. On note $\Delta X = X \oplus X'$.

Proposition 1 :

Pour toute clé K, on a :

$$(X \oplus K) \oplus (X' \oplus K) = X \oplus X' = \Delta X$$

Proposition 2 :

Pour toute Pbox P, on a :

$$P(X) \oplus P(X') = P(X \oplus X') = P(\Delta X)$$

Exemple :

$$\left.\begin{array}{l} X = 0011 \\ X' = 0101 \end{array}\right\} \Delta X = 0110$$

Exemple :

$$\left.\begin{array}{l} X = 0011 \\ X' = 0101 \end{array}\right\} \Delta X = 0110$$

$$k = 0100$$

Exemple :

$$\left.\begin{array}{l} X = 0011 \\ X' = 0101 \end{array}\right\} \Delta X = 0110$$

$$k = 0100$$

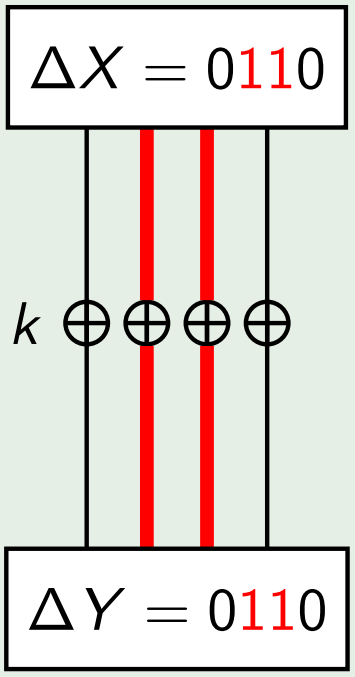
$$\left.\begin{array}{l} Y = 0111 \\ Y' = 0001 \end{array}\right\} \Delta Y = 0110$$

Exemple :

$$\left.\begin{array}{l} X = 0011 \\ X' = 0101 \end{array}\right\} \Delta X = 0110$$

$$k = 0100$$

$$\left.\begin{array}{l} Y = 0111 \\ Y' = 0001 \end{array}\right\} \Delta Y = 0110$$



Exemple :

$$\left.\begin{array}{l} X = 0011 \\ X' = 0101 \end{array}\right\} \Delta X = 0110$$

x	0	1	2	3
$P(x)$	2	0	1	3

TABLE – Pbox

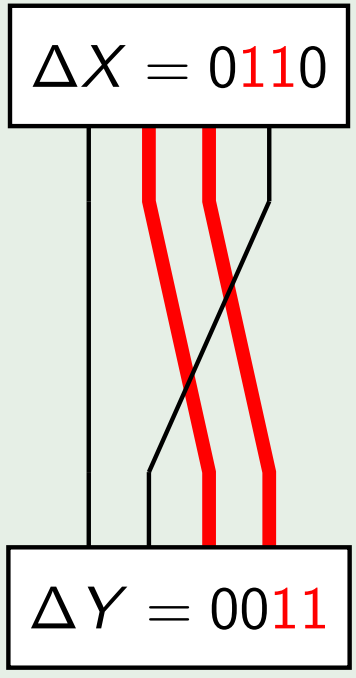
$$\left.\begin{array}{l} Y = 0101 \\ Y' = 0110 \end{array}\right\} \Delta Y = 0011$$

Exemple :

$$\left.\begin{array}{l} X = 0011 \\ X' = 0101 \end{array}\right\} \Delta X = 0110$$

x	0	1	2	3
$P(x)$	2	0	1	3

TABLE – Pbox



$$\left.\begin{array}{l} Y = 0101 \\ Y' = 0110 \end{array}\right\} \Delta Y = 0011$$

Tableau de distribution des différences d'une Sbox

Soit S une Sbox.

Notations :

Soit $(\Delta X, \Delta Y) \in (\mathbb{F}_2^n)^2$. On définit :

$$\delta(\Delta X, \Delta Y) = \text{card}\{X \in \mathbb{F}_2^n \mid S(X) \oplus S(X \oplus \Delta X) = \Delta Y\}$$

Tableau de distribution des différences d'une Sbox

Soit S une Sbox.

Notations :

Soit $(\Delta X, \Delta Y) \in (\mathbb{F}_2^n)^2$. On définit :

$$\delta(\Delta X, \Delta Y) = \text{card}\{X \in \mathbb{F}_2^n \mid S(X) \oplus S(X \oplus \Delta X) = \Delta Y\}$$

Définition 9 : Difference distribution table (DDT)

Le **tableau de distribution des différences** de la Sbox S donne les valeurs de $\delta(\Delta X, \Delta Y)$ pour tout $(\Delta X, \Delta Y) \in (\mathbb{F}_2^n)^2$.

Exemple

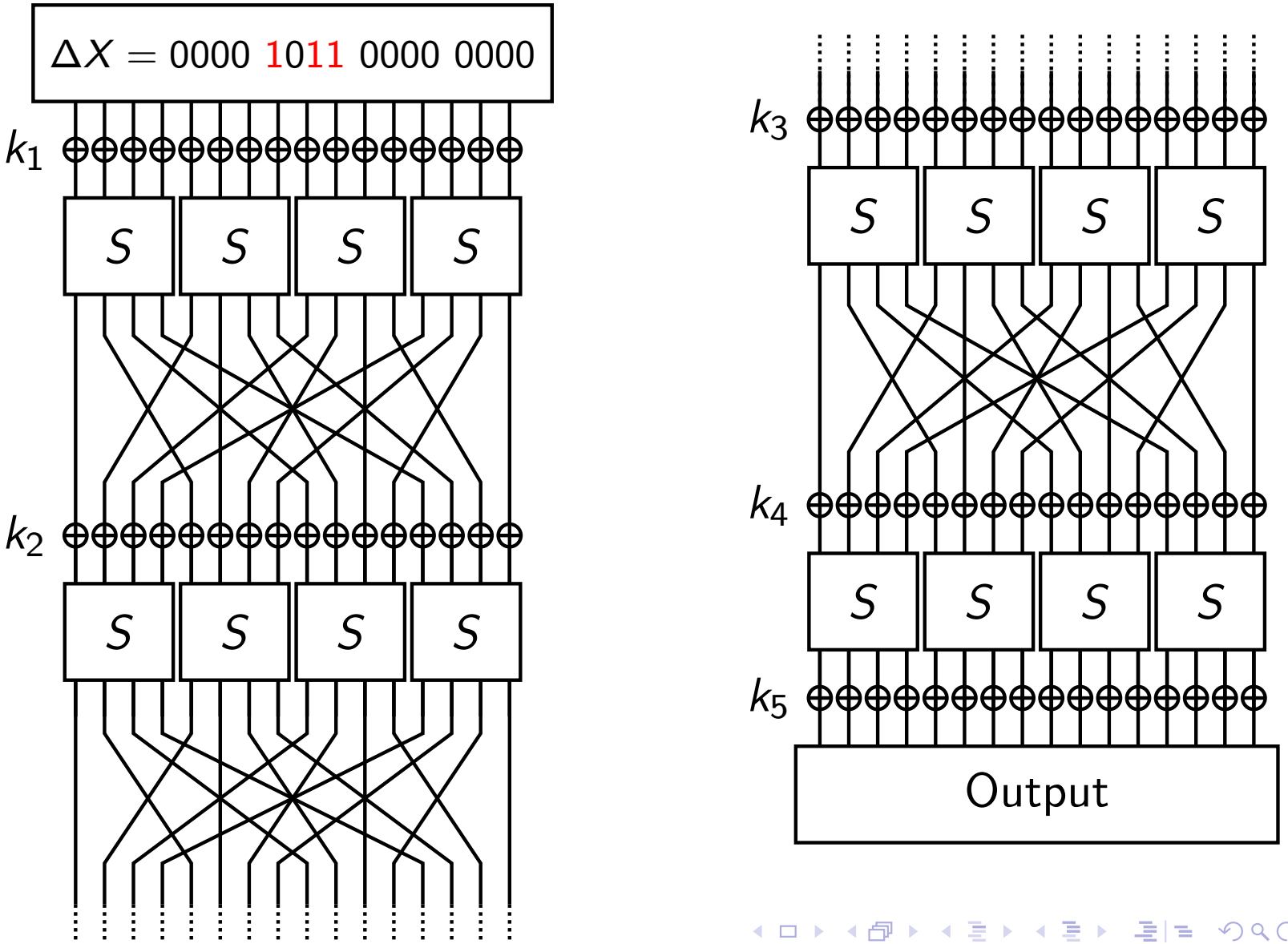
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	e	4	d	1	2	f	b	8	3	a	6	c	5	9	0	7

TABLE – Sbox

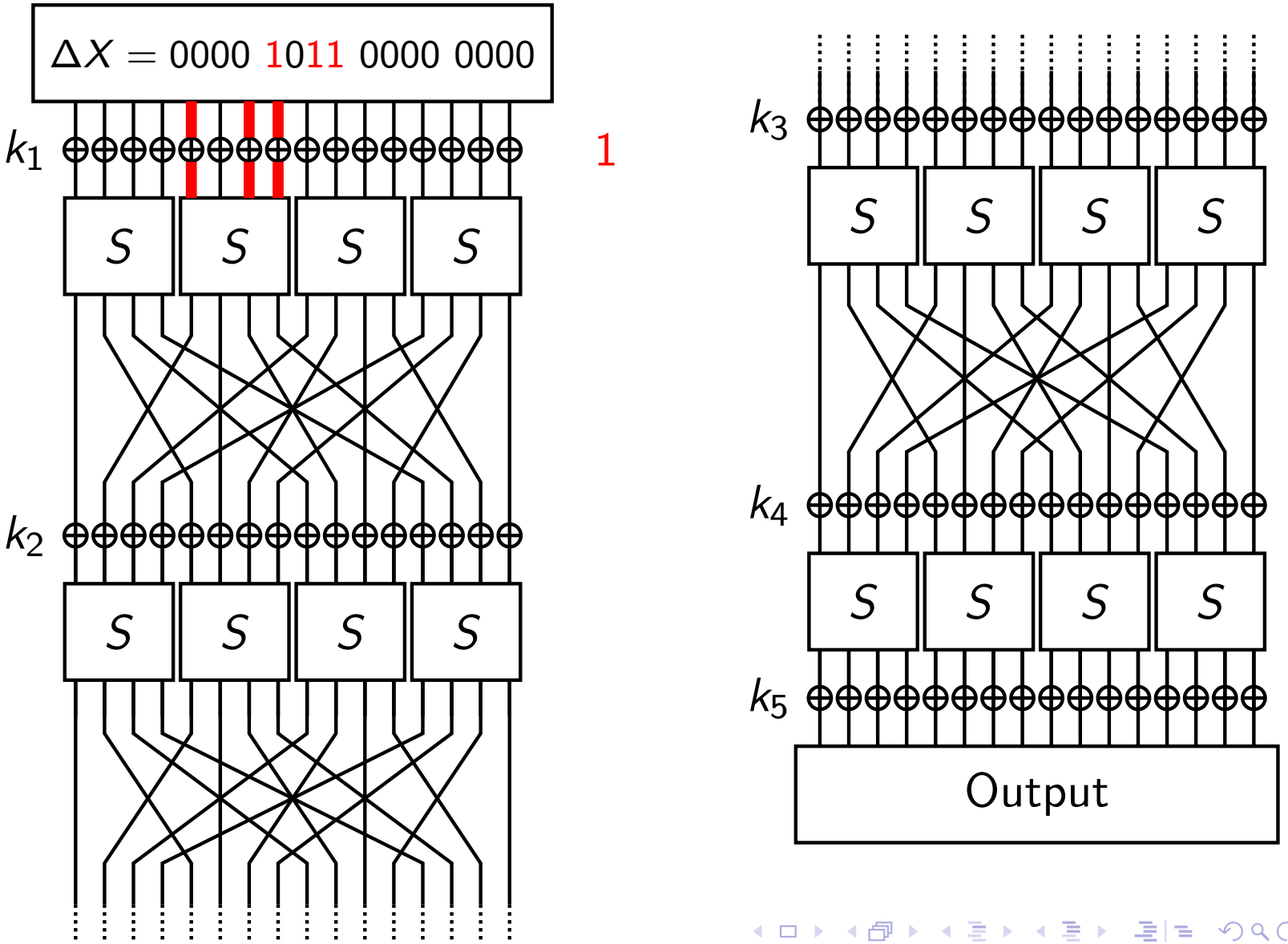
Input difference \ Output difference	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	2	-	-	-	2	-	2	4	-	4	2	-	-
2	-	-	-	2	-	6	2	2	-	2	-	-	-	-	2	-
3	-	-	2	-	2	-	-	-	-	4	2	-	2	-	-	4
4	-	-	-	2	-	-	6	-	-	2	-	4	2	-	-	-
5	-	4	-	-	-	2	2	-	-	-	4	-	2	-	-	2
6	-	-	-	4	-	4	-	-	-	-	-	-	2	2	2	2
7	-	-	2	2	2	-	2	-	-	2	2	-	-	-	-	4
8	-	-	-	-	-	-	2	2	-	-	-	4	-	4	2	2
9	-	2	-	-	2	-	-	4	2	-	2	2	2	-	-	-
10	-	2	2	-	-	-	-	-	6	-	-	2	-	-	4	0
11	-	-	8	-	-	2	-	2	-	-	-	-	-	2	-	2
12	-	2	-	-	2	2	2	-	-	-	-	2	-	6	-	-
13	-	4	-	-	-	-	-	4	2	-	2	-	2	-	2	-
14	-	-	2	4	2	-	-	-	6	-	-	-	-	-	2	-
15	-	2	-	-	6	-	-	-	-	4	-	2	-	-	2	-

TABLE – DDT de la Sbox

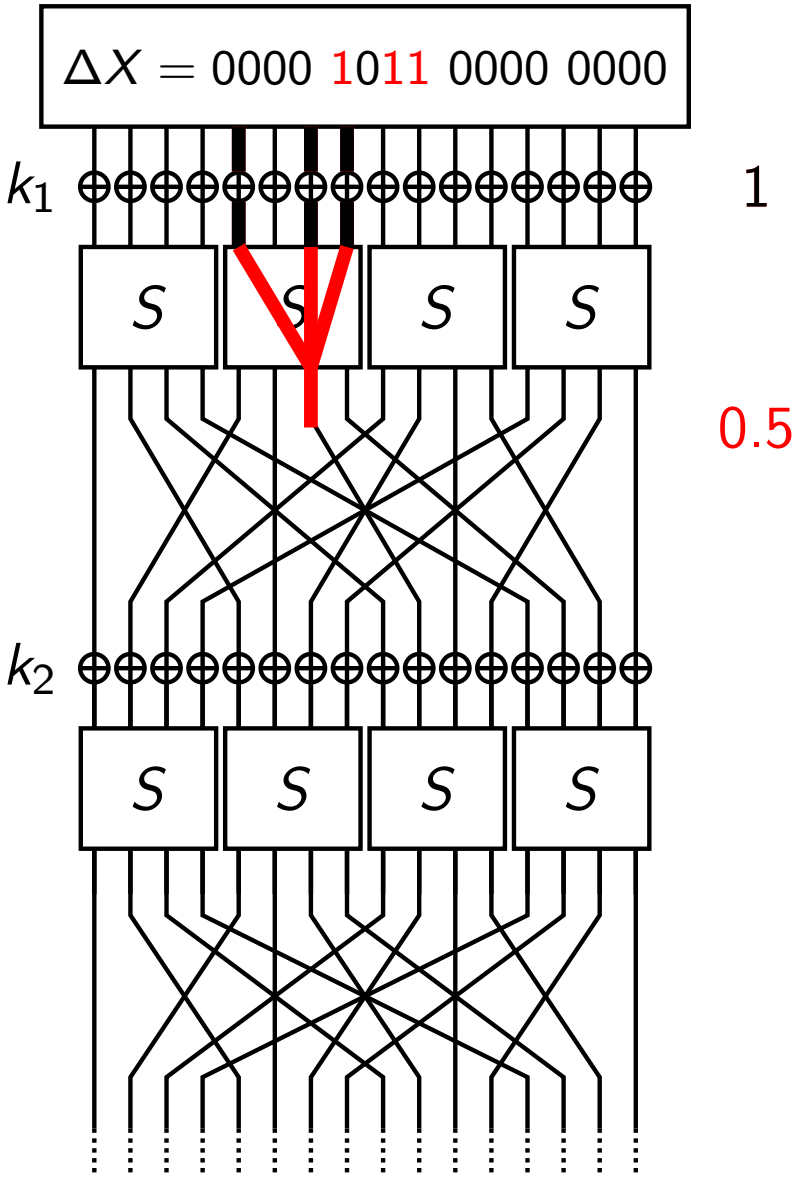
Exemple

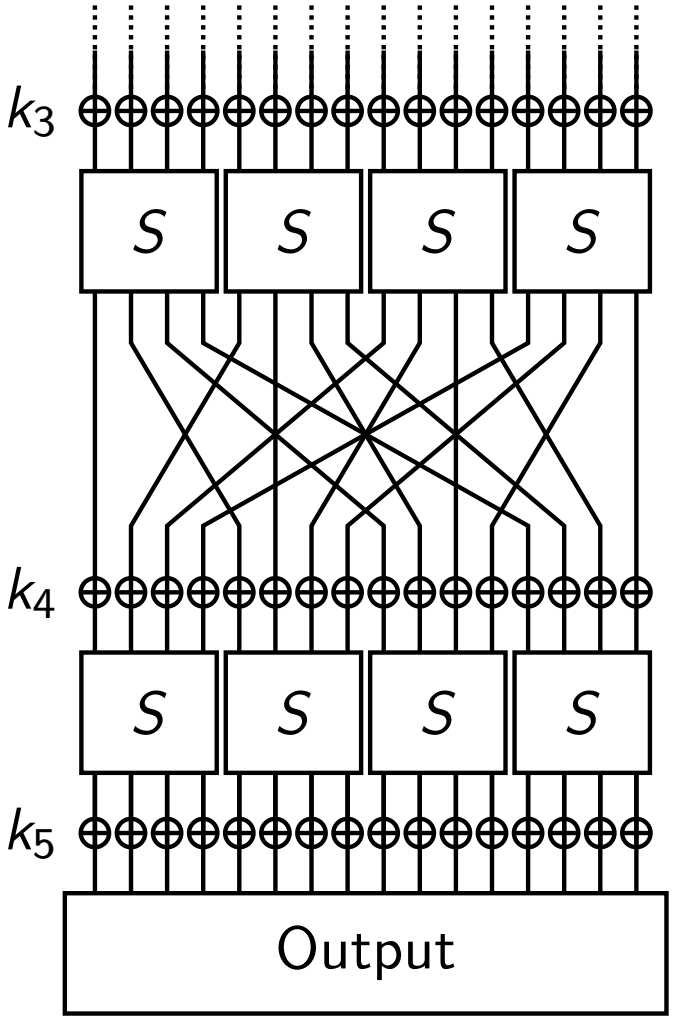

ΔX = 0000 1011 0000 0000
k1
S S S S
k2
S S S S
k3
S S S S
k4
S S S S
k5
Output

Exemple

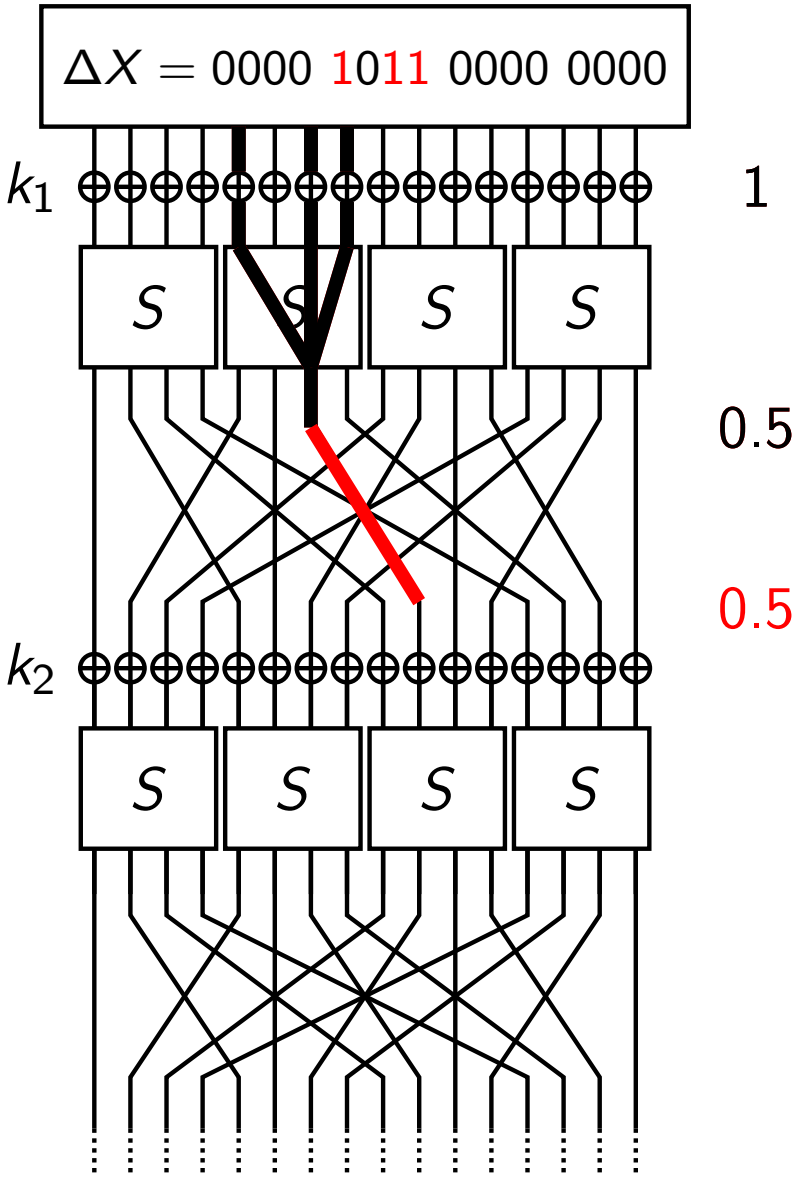


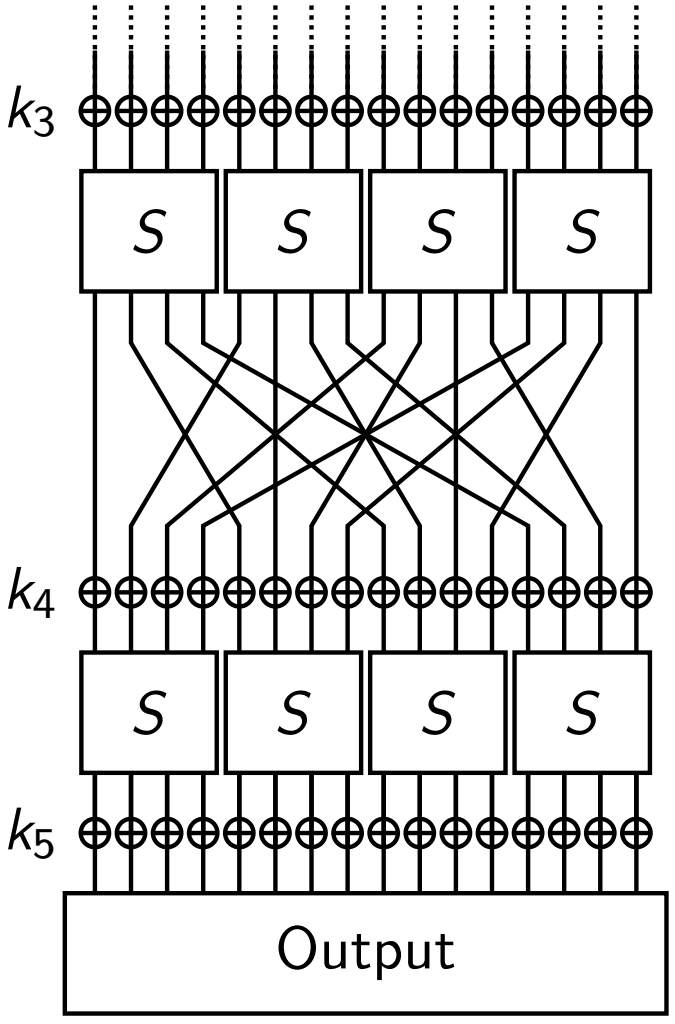
Exemple


ΔX = 0000 1011 0000 0000
k1
S
S
S
S
1
0.5
k2
S
S
S
S

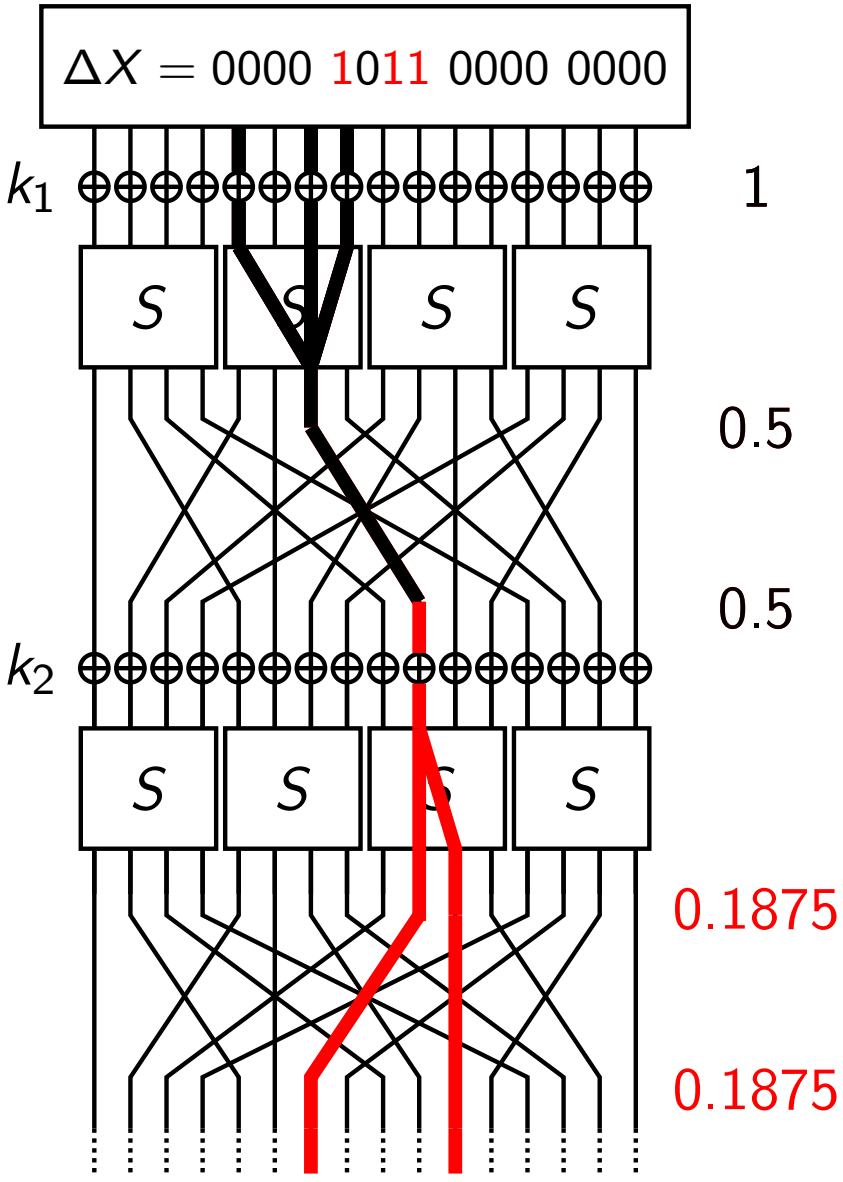

k3
S
S
S
S
k4
S
S
S
S
k5
Output

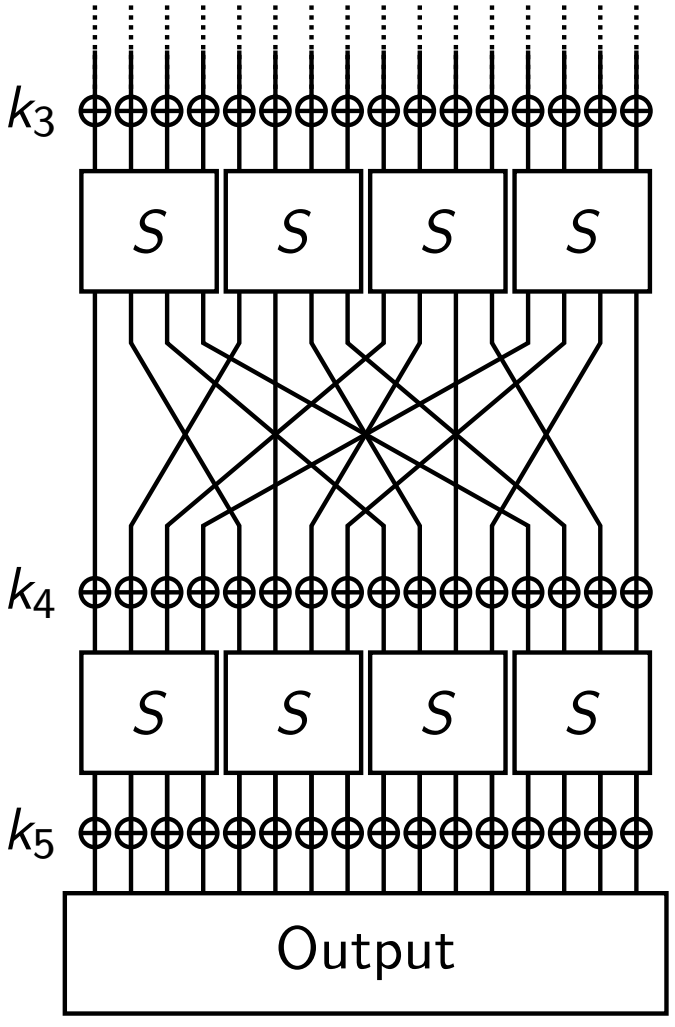
Exemple


ΔX = 0000 1011 0000 0000
k1
S
S
S
S
1
0.5
0.5
k2
S
S
S
S

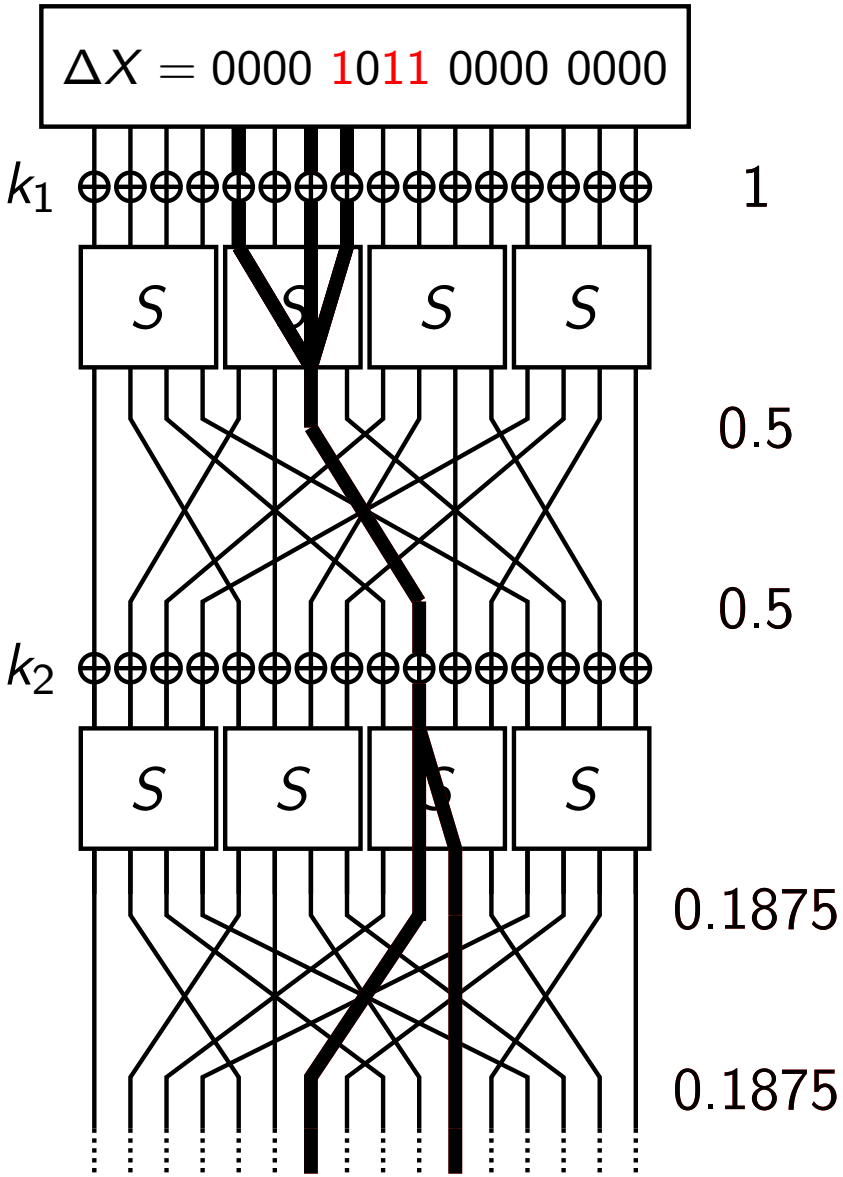

k3
S
S
S
S
k4
S
S
S
S
k5
Output

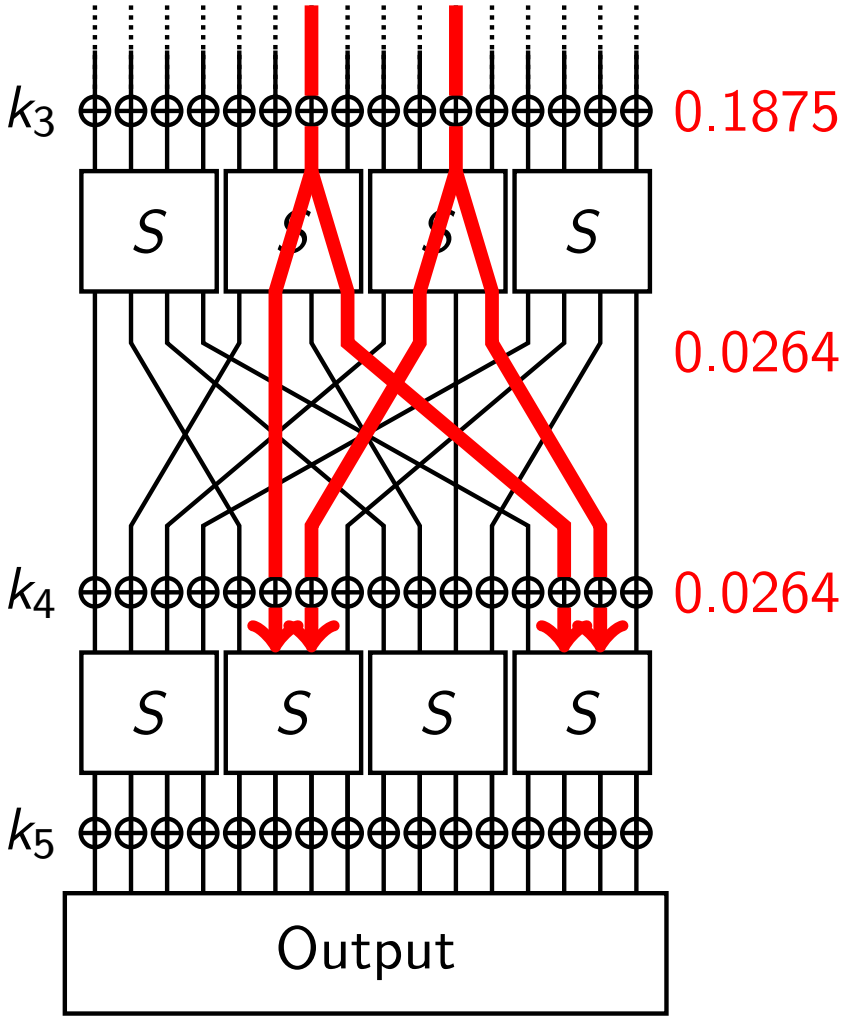
Exemple


ΔX = 0000 1011 0000 0000
k1
S
S
S
S
1
0.5
0.5
k2
S
S
S
S
0.1875
0.1875


k3
S
S
S
S
k4
S
S
S
S
k5
Output

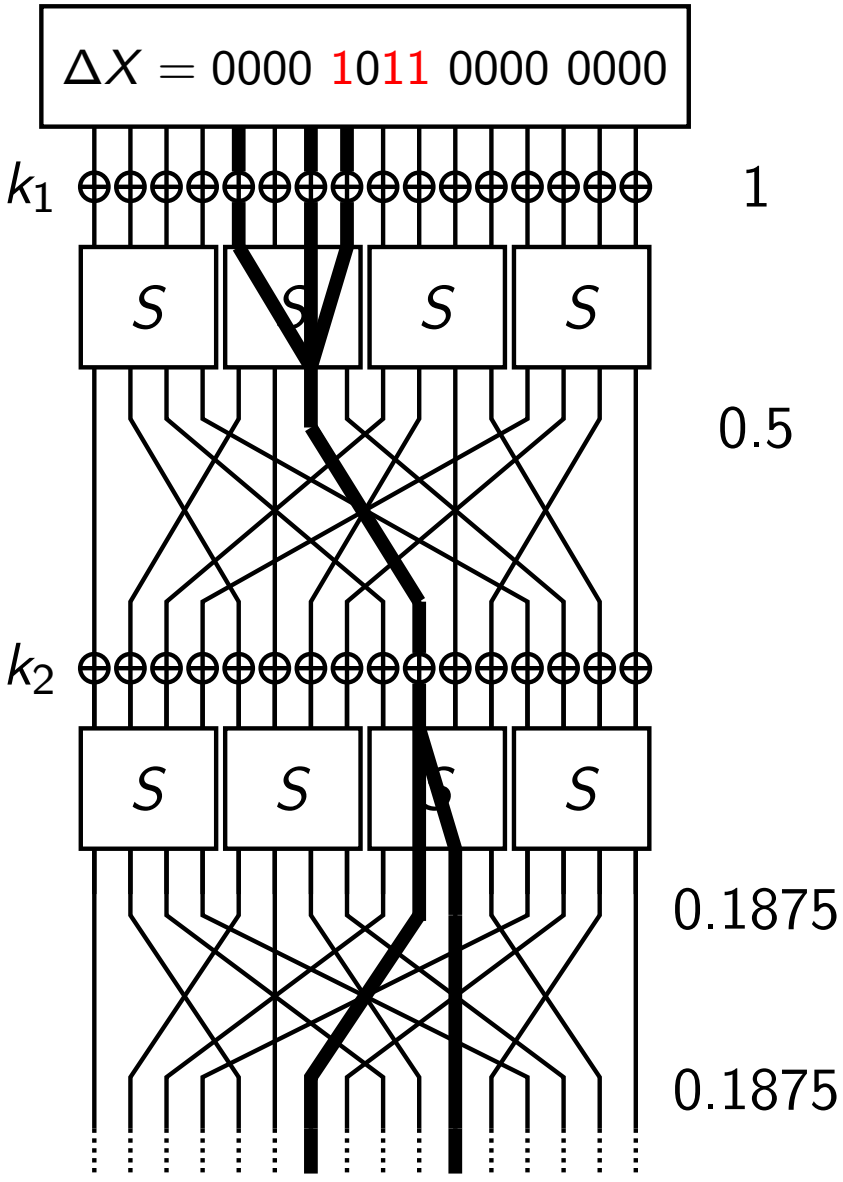
Exemple

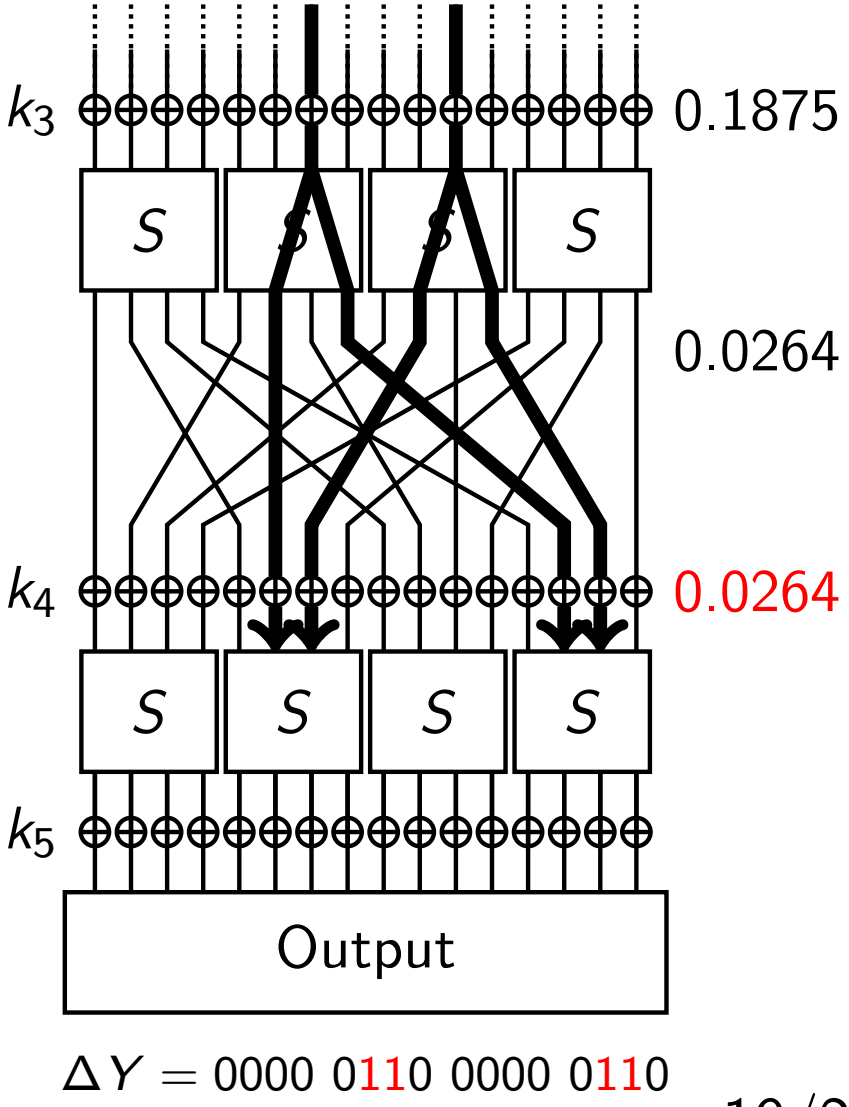




$\Delta Y = 0000\ 0110\ 0000\ 0110$

Comment déterminer k_5 ?





Définition 10 : Uniformité différentielle

On définit l'**uniformité différentielle** $\mu(S)$ d'une Sbox S par le **maximum** de son tableau de distribution des différences.

$$\mu(S) = \max_{\Delta X, \Delta Y \in (F_2^n)^2 \, , \, \Delta X \neq 0} \delta(\Delta X, \Delta Y)$$

Définition 10 : Uniformité différentielle

On définit l'**uniformité différentielle** $\mu(S)$ d'une Sbox S par le **maximum** de son tableau de distribution des différences.

$$\mu(S) = \max_{\Delta X, \Delta Y \in (F_2^n)^2 \, , \, \Delta X \neq 0} \delta(\Delta X, \Delta Y)$$

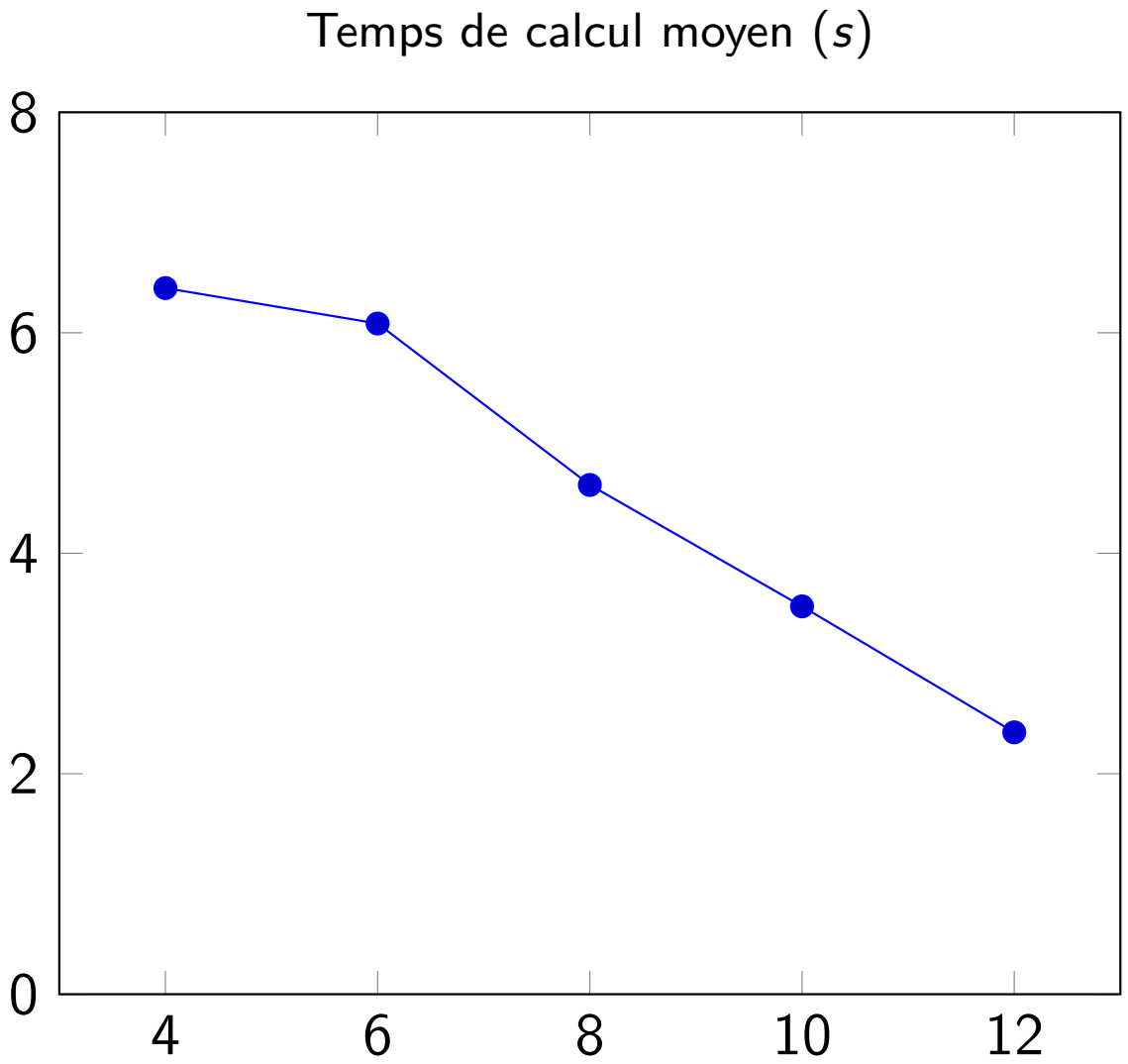
Input difference \ Output difference	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	2	-	-	-	2	-	2	4	-	4	2	-	-
2	-	-	-	2	-	6	2	2	-	2	-	-	-	-	2	-
3	-	-	2	-	2	-	-	-	-	4	2	-	2	-	-	4
4	-	-	-	2	-	-	6	-	-	2	-	4	2	-	-	-
5	-	4	-	-	-	2	2	-	-	-	4	-	2	-	-	2
6	-	-	-	4	-	4	-	-	-	-	-	-	2	2	2	2
7	-	-	2	2	2	-	2	-	-	2	2	-	-	-	-	4
8	-	-	-	-	-	-	2	2	-	-	-	4	-	4	2	2
9	-	2	-	-	2	-	-	4	2	-	2	2	2	-	-	-
10	-	2	2	-	-	-	-	-	6	-	-	2	-	-	4	0
11	-	-	8	-	-	2	-	2	-	-	-	-	-	2	-	2
12	-	2	-	-	2	2	2	-	-	-	-	2	-	6	-	-
13	-	4	-	-	-	-	-	4	2	-	2	-	2	-	2	-
14	-	-	2	4	2	-	-	-	6	-	-	-	-	-	2	-
15	-	2	-	-	6	-	-	-	-	4	-	2	-	-	2	-

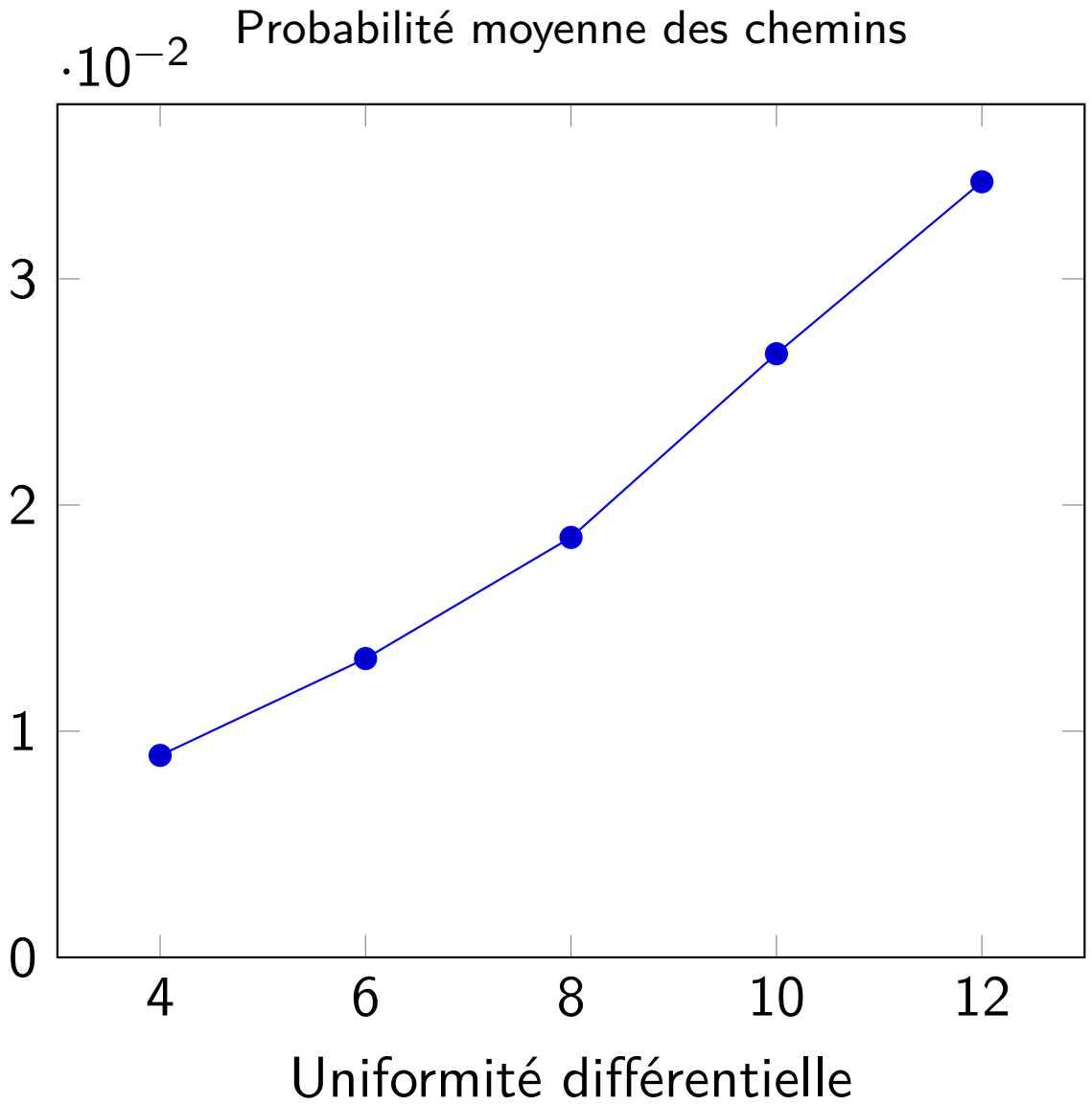
TABLE – DDT de la Sbox

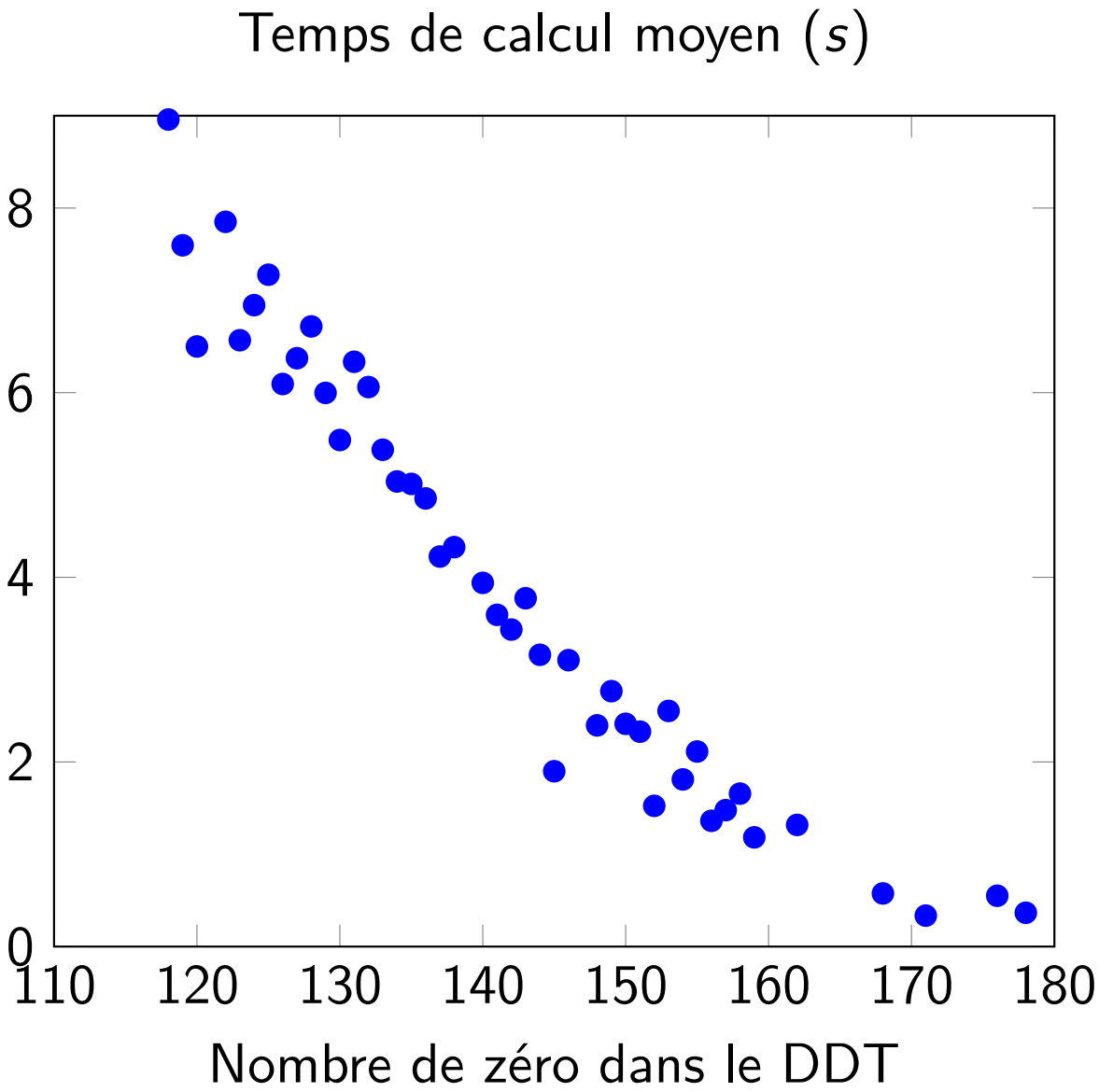
Hypothèse :

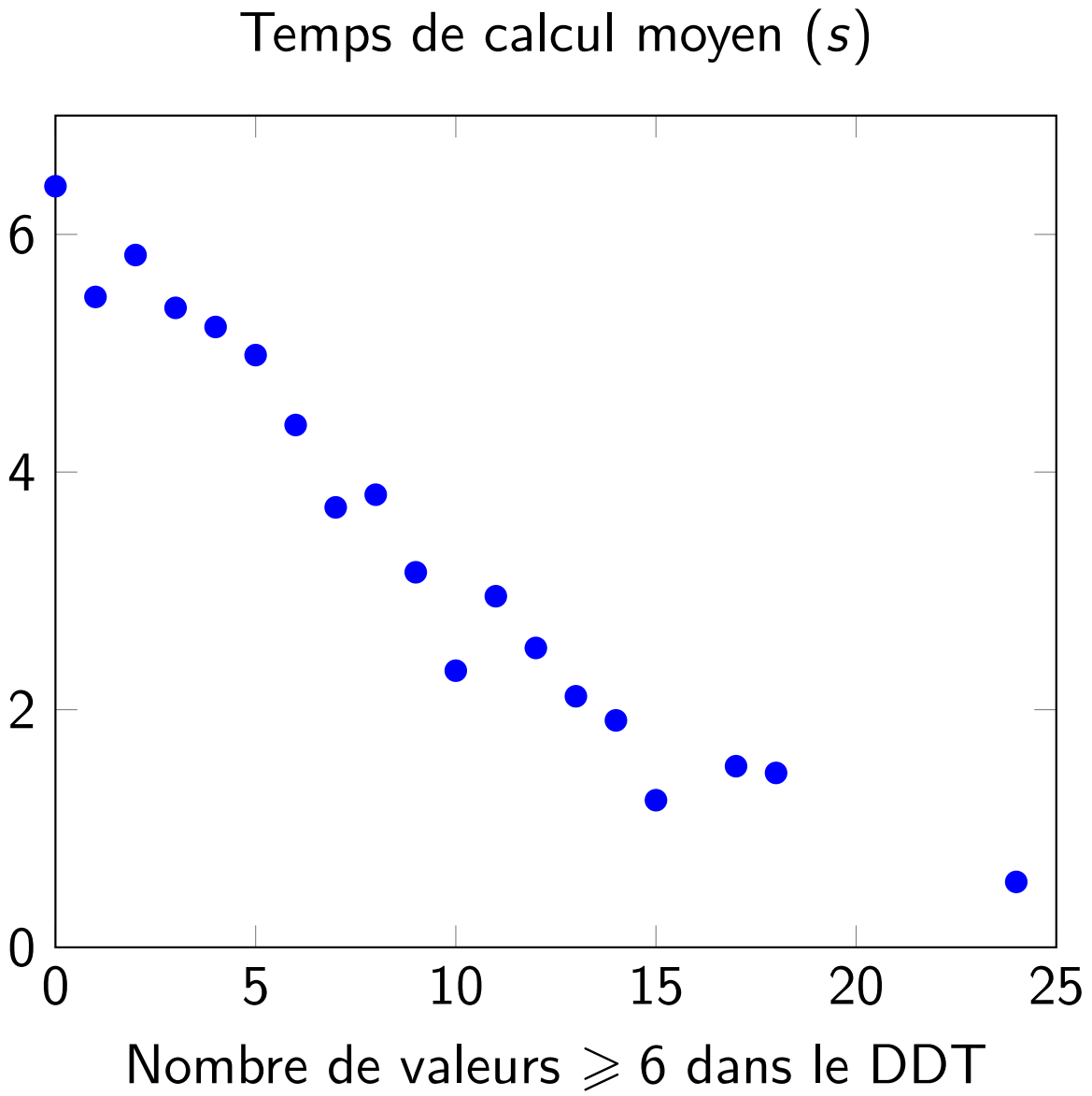
Un bon algorithme de chiffrement nécessite une Sbox avec la plus **faible uniformité différentielle**.

Vérification




Probabilité moyenne des chemins
$\cdot 10^{-2}$
3
2
1
0
4
6
8
10
12
Uniformité différentielle


Temps de calcul moyen (s)
8
6
4
2
0
110
120
130
140
150
160
170
180
Nombre de zéro dans le DDT


Temps de calcul moyen (s)
6
4
2
0
0
5
10
15
20
25
Nombre de valeurs ⩾ 6 dans le DDT

Conclusion

- Importance de l'uniformité différentielle

Conclusion

- Importance de l'uniformité différentielle
- Permutations APN

Conclusion

- Importance de l'uniformité différentielle
- Permutations APN
- Couche de diffusion/permutation

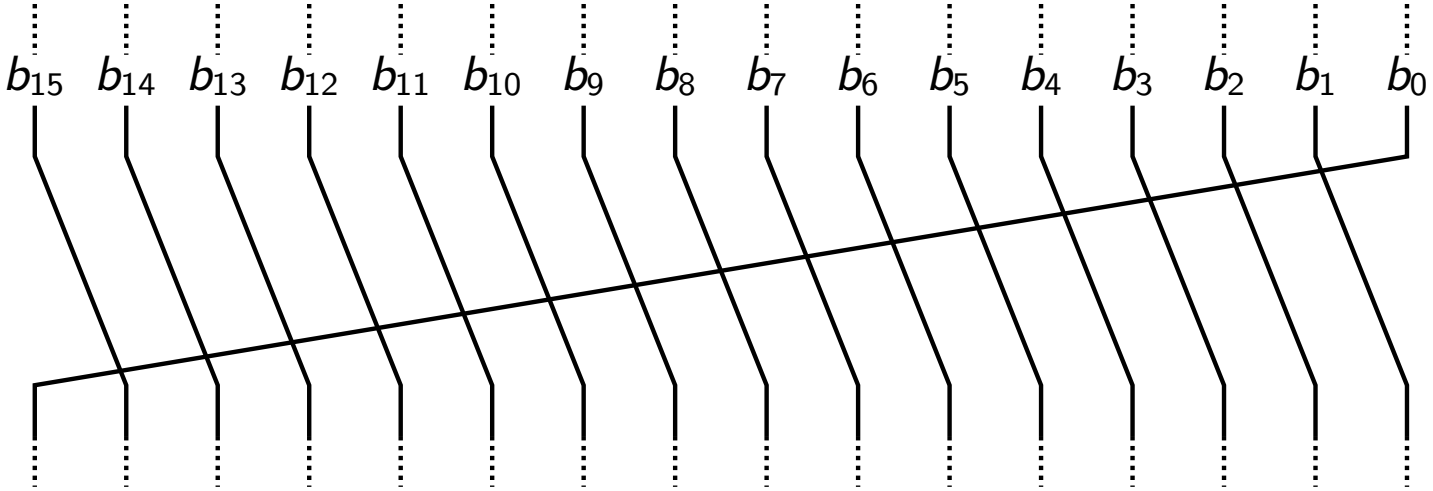


FIGURE – Pbox A

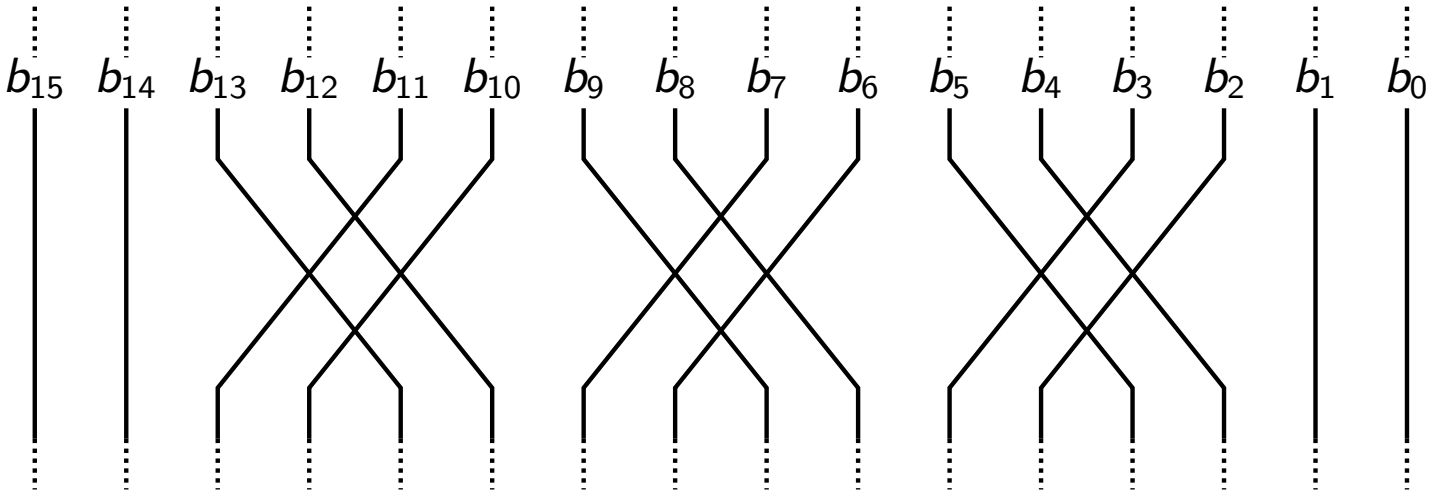


FIGURE – Pbox B

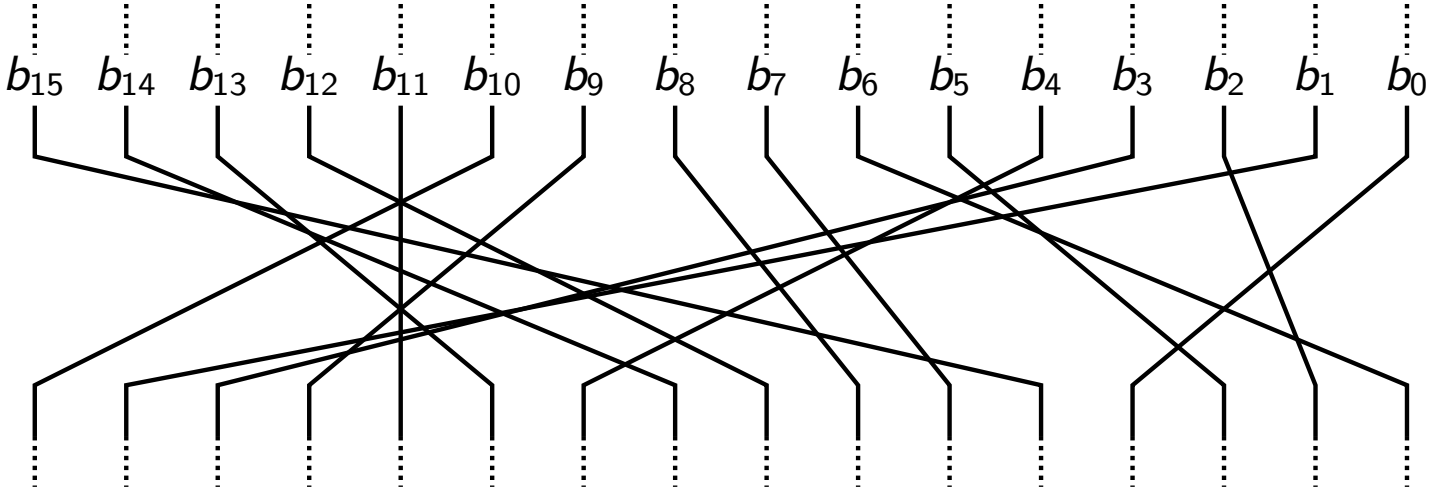


FIGURE – Pbox C

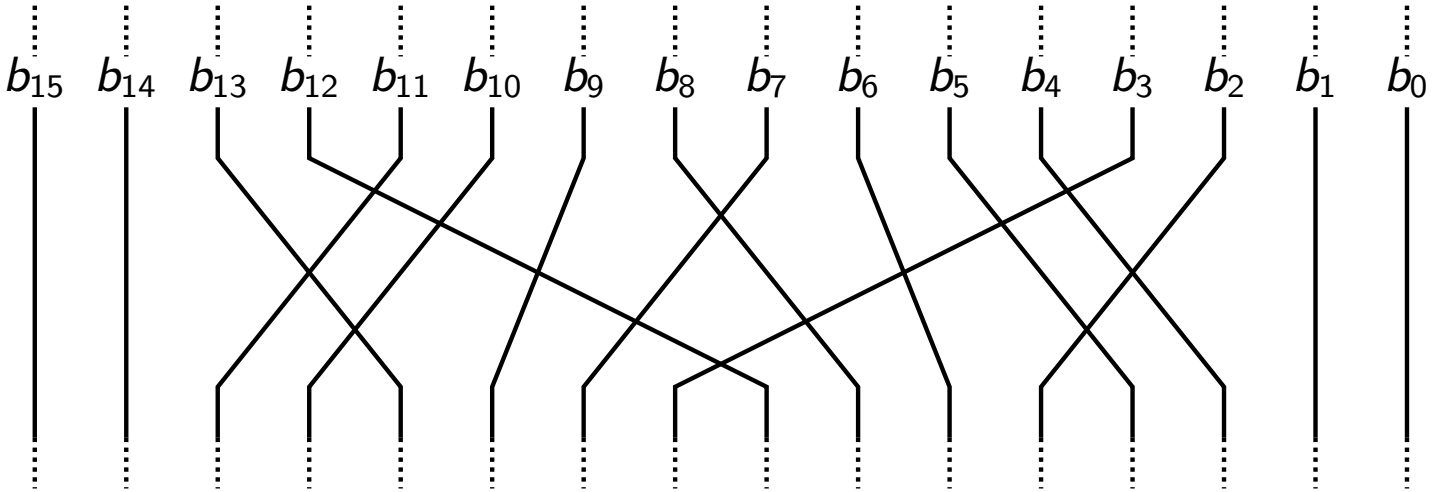
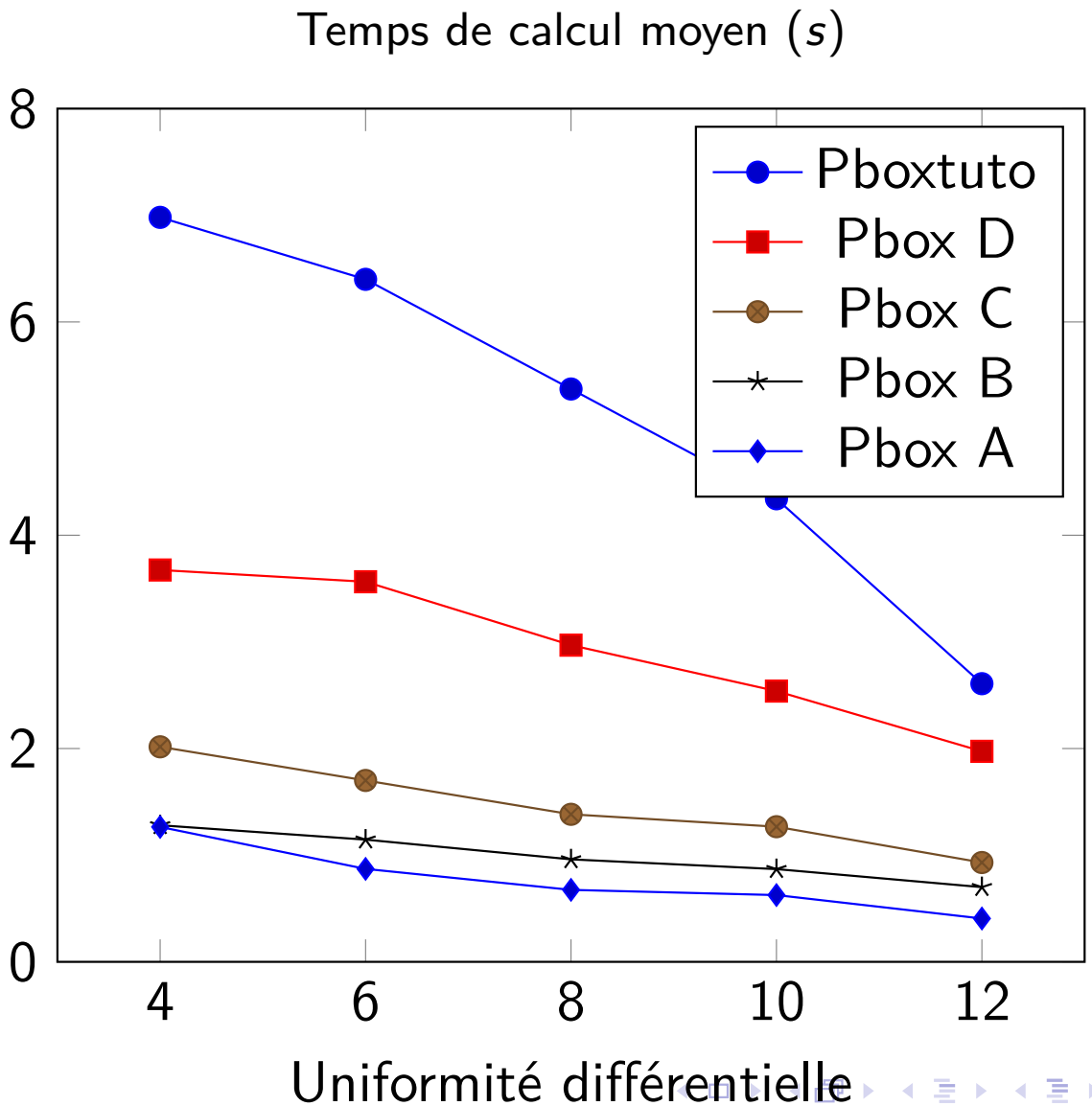


FIGURE – Pbox D

Vérification



Modules

```
1  import numpy as np
2  import random
3  import time
4  import cProfile
5  import re
6  import functools  # Mémoïsation
7  import itertools
8  import pickle  # Binary protocols
9  import pandas as pd
10 import logging
```

Sbox

```
1
2   ### Février ###
3
4
5   def SboxLayer(state, Sbox):
6       """Sbox function (4 Sbox)
7       Input:  16-bit integer, Sbox list
8       Output: 16-bit integer
9       """
```

Pbox

```
#         output += ((state >> i) & 0x1) << Pbox[i]
#     return output


@functools.lru_cache(maxsize=None)   # Mémoïsation, taille du cache infinie
def PboxLayer(state, Pbox):
    """Pbox function (4 Pbox)
    Input: 16-bit integer, Pbox list
    Output: 16-bit integer
    """
```

Chiffrement

```
1
2   # deltaP = 0b0000101100000000
3
4   # deltaU4 = 0b0000011000000110
5
6
7   def Ciphertuto(message, K, Sbox, Pbox):
8       """CipherFour function
9       Input: 16-bit integer
10             K = [k1, k2, k3, k4, k5] (16-bit integer)
11             Sbox, Pbox
12      Output : 16-bit integer
13      """
14      state = message
15      for i in range(3):
16          state = state ^ K[i]
```

Tableau de distribution des différences

```
1
2
3
4
5   def diffdistrib(Sbox):
6       """
7       Input : Sbox list
8       Output : difference distribution table
9       """
10      InLength = 16
11      OutLength = 16
12      tab = np.zeros((InLength, OutLength), int)
13      for m0 in range(0, InLength):
14          for m1 in range(0, OutLength):
```

Calcul des sorties d'une Sbox I

```
1  #                OUT16 = (OUT1[0] << 12) + (OUT2[0] << 8) + (OUT3[0] <<
   ↪  4) + OUT4[0]
2  #                proba = OUT1[1] * OUT2[1] * OUT3[1] * OUT4[1]
3  #                proba = proba / (16**4)
4
5  #                resul.append((OUT16, proba))
6  #            return resul
7
8  #        return fusion(OUTresul)
9
```

Calcul des sorties d'une Sbox II

```
1   @functools.lru_cache(maxsize=None)  # Mémoisation, taille du cache infini
2   def sortiepossible(deltaIN, difftabtuple):
3       """
4       Input : deltaIN (16 bit), difftabtuple : difference distribution table sous forme de tuple (pour mémosiation hashable)
5       Output : liste des différences (deltaOUT) et de leur probabilité, en sortie d'une couche de Sboxs (correspondant au difftab) avec une différence d'entrée deltaIN SUR UNE ETAPE
6       """
7       deltaINlist = []    # Division de deltaIN en 4 entiers de 4 bits
8       for i in range(0, 4):
9           deltaINlist.append(deltaIN >> ((3-i)*4) & 0xF)
10
11      def OUTprobable(IN):
12          """
13          Input : différence d'entrée (entier de 4bit)
14          Output : liste des différences possibles (4bit) et de leur probabilité, en sortie de la Sbox (correspondant au difftab) avec une différence d'entrée IN et tel que la probabilité soit >= nbMIN/16
15          """
```

Calcul des sorties d'une Sbox III

```
1                proba = difftabtuple[IN][OUT]
2                if proba >= nbMIN:
3                    resul.append((OUT, proba))
4            return resul
5
6        OUTresul = []    # Liste de la liste des différences de sorties
         ↪  (4bit)
7                         # et proba pour chaque Sbox
8        for i in range(0, 4):
9            OUTresul.append(OUTprobable(deltaINlist[i]))
10
11       def fusion(OUTresul):
12           """
13           Input : Liste de la liste des différences de sorties (4bit)
     ↪   et proba pour chaque Sbox
14           Output : Liste des différences de sortie (16bit) (toutes les
     ↪   combinaisons des sorties 4bits) et probas
```

Filtres I

```
1         return fusion(OUTresul)
2
3     # Exemple page 24 "A tutorial on..."
4     #
5     # >>> sortiepossible(0b0000101100000000,difftabtuto)
6     # [(512, 0.5), (1280, 0.125), (1792, 0.125), (3328, 0.125), (3840,
      ↪  0.125)]
7
8     # sortiepossible = memoize(sortiepossiblebefore)
9
10
```

Filtres II

```
1        Output : boolean
2        """
3        if b <= probaMIN:
4            return False
5        if nombreboiteactive(a) > nbboiteMAX:
6            return False
7        return True
8
9
10   ###  08/03/19  ###
```

Filtres III

```
1        Ouput : Liste des boites activées par cette différence
2        """
3        resul = [False, False, False, False]
4        for i in range(0, 4):
5            if ((a >> (i*4)) & 0xF) > 0:
6                resul[3-i] = True
7        return resul
8
9
10   ###  08/03/19  ###
11   def filtreboiteactive(a, listeboitesactives):
12       """  Filtrage sur les boites actives
```

Recherche de chemins I

```
1   #          if a != 0:
2   #              resul.append(a)
3   #      trichemin(resul)
4   #      return maxprobachemin(resul), resul
5
6
7   ### 08/03/19 ###
8   # Autre méthode dans la fonction chemin, à chaque fois chemin,
    ↪  rechercher
9   # chemin dont la proba est plus élevée que celle du précédent
10
11  def cheminV2(difftabtuple, Pbox, U1, probaMIN, nbboiteMAX,
    ↪  listeboitesactives):
12      """
13      Input : difference distribution table de la Sbox, Pbox
```

Recherche de chemins II

```
15      Probabilité minimum du chemin à la dernière étape, pour le premier chemin calculé
16      Nombre de boites S activées maximale en dernière étape
17      Liste des boites S qui doivent être au moins activées en dernière étape
18      Output : Chemin commencant par U1, ayant en dernière étape une probabilité maximale, un
        ↪ nombre de boites S activée <= nbboiteMAX, et tel que toutes les boites S demandées
        ↪ (listeboitesactives) soient activées
19      """
20      resul = 0
21      cheminencours = [0, 0, 0, 0, 0, 0, 0]
22      cheminencours[0] = (U1, 1)     # 1ère étape
23      S1 = sortiepossible(U1, difftabtuple)     # Ensemble des sorties possibles après la
        ↪ premiere couche de boites S
24
25      for V1 in S1:
26          cheminencours[1] = (V1[0], V1[1]*cheminencours[0][1])
27          U2 = PboxLayer(V1[0], Pbox), cheminencours[1][1]
28          if filtreP(U2[0], U2[1], probaMIN, 4):    # Filtrage en sortie des Pbox
29              cheminencours[2] = (U2[0], U2[1])
30              S2 = sortiepossible(U2[0], difftabtuple)
31              for V2 in S2:
32                  cheminencours[3] = ((V2[0], V2[1]*cheminencours[2][1]))
33                  U3 = PboxLayer(V2[0], Pbox), cheminencours[3][1]
34                  if filtreP(U3[0], U3[1], probaMIN, 4):
35                      cheminencours[4] = (U3[0], U3[1])
```

Recherche de chemins III

```
1                                   cheminencours[5] = (V3[0],
                                    ↪  V3[1]*cheminencours[4][1])
2                                   U4 = PboxLayer(V3[0], Pbox),
                                    ↪  cheminencours[5][1]
3                                   if filtreP(U4[0], U4[1], probaMIN,
                                    ↪  nbboiteMAX) and \
4                                           filtreboiteactive(U4[0],
                                            ↪  listeboitesactives):
5                                       cheminencours[6] = (U4[0], U4[1])
6                                       resul = cheminencours.copy()  # /!\
                                        ↪  Référence /!\
7                                       probaMIN = U4[1]  # Mise à jour de
                                        ↪  probaMIN
8           return resul
9
10
11  class RechercheCheminTropLong(Exception):
12      pass
```

Recherche de chemins IV

```
14
15      def recherchecheminV2(difftabtuple, Pbox, listeIN, probaMINdébut, nbboiteMAX,
        ↪  listeboitesactives, tempsmaxrecherchechemin):
16          """
17          Input : listeIN: liste des U1
18          Output : chemin ayant "la" probabilité maximale
19          """
20          probaMIN = probaMINdébut
21
22          debut = time.time()
```

Décryptage I

```
1
2   def inverse(liste):
3       """
4       Input : 16-bit permutation
5       Output : Inverse de la permutation
6       """
7       result = list()
8       for i in range(16):
9           result.append(liste.index(i))
```

Décryptage II

```
1
2
3   # def trichemin(listechemin):   ###  08/03/19  ###
4   #     """
5   #     Input : liste de chemins
6   #     Output : chemin trié EN PLACE par probabilité finale
    ↪  décroissante
7   #     """
8   #     listechemin.sort(key=lambda x: x[-1][1], reverse=True)
9
10
```

Décryptage III

```
#                 compteur += 1
#                 resul.append(liste[i])
#     return compteur, resul


# nombreboiteactivefinliste(a, 2)
# b = (9, [[(2816, 1), (512, 0.5), (64, 0.5), (48, 0.0625), (34,
↪  0.0625), (85, 0.0087890625), (771, 0.0087890625)], [(2816, 1),
↪  (512, 0.5), (64, 0.5), (96, 0.1875), (544, 0.1875), (816,
↪  0.0029296875), (102, 0.0029296875)], [(2816, 1), (512, 0.5), (64,
↪  0.5), (96, 0.1875), (544, 0.1875), (1360, 0.0263671875), (1542,
↪  0.0263671875)], [(2816, 1), (512, 0.5), (64, 0.5), (96, 0.1875),
↪  (544, 0.1875), (1632, 0.0029296875), (1632, 0.0029296875)],
↪  [(2816, 1), (512, 0.5), (64, 0.5), (96, 0.1875), (544, 0.1875),
↪  (2448, 0.0029296875), (24582, 0.0029296875)], [(2816, 1), (512,
↪  0.5), (64, 0.5), (144, 0.0625), (8194, 0.0625), (20485,
↪  0.0087890625), (2313, 0.0087890625)], [(2816, 1), (512, 0.5),
↪  (64, 0.5), (176, 0.125), (8226, 0.125), (20565, 0.006591796875),
↪  (2827, 0.006591796875)], [(2816, 1), (512, 0.5), (64, 0.5), (192,
↪  0.0625), (8704, 0.0625), (21760, 0.0087890625), (3084,
↪  0.0087890625)], [(2816, 1), (1280, 0.125), (1028, 0.125), (1542,
↪  0.017578125), (1360, 0.017578125), (2720, 0.0010986328125),
```

Décryptage IV

```
20      """
21      difftab = diffdistrib(Sbox)
22      difftabtuple = tuple(map(tuple, difftab))
23      invSbox = inverse(Sbox)
24      probaMIN = 1/(2**16)
25      listresul = [-1, -1, -1, -1]
26      resul = 0
27      compteurMAXLISTE = 0  # Nombre de fois ou la fonction Maxliste à
        ↪  renvoyer plus qu'une valeur
28      tabproba = [0, 0, 0, 0]
29
30      for partial in range(0, 4):   # Portion de la clé K5 à décrypter
31          listeboitesactives = [False, False, False, False]
32          listeboitesactives[partial] = True
33          chemin = recherchecheminV2(difftabtuple, Pbox, listeIN,
            ↪  probaMIN, 1, listeboitesactives, tempsmaxrecherchechemin)
34          tabk5partial = [0]*16
```

Décryptage V

```
36          tabproba[partial] = chemin[-1][1]
37
38          deltaP = chemin[0][0]  # U1
39          deltaU4 = chemin[-1][0]
40
41          ensemblefiltreCouple = set()
42          sorties = sortiepossible(deltaU4, difftabtuple)
43          for k in sorties:
44              ensemblefiltreCouple.add(k[0])
45
46          for i in range(nbcouples):
47              X = random.randint(0, 65535)
48              Xprim = X ^ deltaP
49              Y = Ciphertuto(X, K, Sbox, Pbox)
```

Décryptage VI

```
51                  iteration = 0
52                  while Y ^ Yprim not in ensemblefiltreCouple:
53                      if iteration > 65535:
54                          raise RechercheCoupleImpossible
55                      X = random.randint(0, 65535)
56                      Xprim = X ^ deltaP
57                      Y = Ciphertuto(X, K, Sbox, Pbox)
58                      Yprim = Ciphertuto(Xprim, K, Sbox, Pbox)
59                      iteration += 1
60
61
62                  for k5partial in range(16):  # 2**4
63                      V4_X_partial = (Y >> (3-partial)*4) & 15 ^ k5partial
64                      V4_Xprim_partial = (Yprim >> (3-partial)*4) & 15 ^
                        ↪  k5partial
65                      U4_X_partial = invSbox[V4_X_partial]
66                      U4_Xprim_partial = invSbox[V4_Xprim_partial]
67                      U4_Xpartial = U4_X_partial << (3-partial)*4
68                      U4_Xprimpartial = U4_Xprim_partial << (3-partial)*4
```

Analyse I

```
1   # compteur = 17 * [0]
2
3   # for i in range(10**5):
4   #     Sbox = np.random.permutation(16)
5   #     #print(maxtab(diffdistrib(Sbox))[2])
6   #     compteur[maxtab(diffdistrib(Sbox))[2]] += 1
7   # for i in range(0, 17):
8   #     compteur[i] = compteur[i] / 10**3
9
10  # print(compteur)
11
12  def permutdiffmax(diffmaxdemande):
13      """
14      Entrée : entier
15      Sortie : liste d'une permutation telle que le max du tableau de
    ↪   distribution des différences soit l'entier diffmaxdemande, et
    ↪   tableau
```

Analyse II

```
1  Pbox13 = (4, 5, 6, 7, 12, 13, 14, 15, 0, 1, 2, 3, 8, 9, 10, 11)
2  Pbox14 = (12, 13, 14, 15, 4, 5, 6, 8, 7, 9, 10, 11, 0, 1, 2, 3)
   ↪  # n6
3  Pbox15 = (6, 4, 0xc, 5, 0, 7, 2, 0xe, 1, 0xf, 3, 0xd, 8, 0xa, 9, 0xb)
   ↪  # n7
4  Pbox16 = (12, 7, 13, 15,  0,  1,  4,  9, 11,  5,  8,  2,  6, 10,  3,
   ↪  14)   # n8
5  Pbox17 = (6, 15, 14, 13,  7,  8,  1,  3, 10, 11,  5,  4, 12,  2,  0,
   ↪  9)   # n9
6  Pbox18 = (15,  2,  5,  1,  4, 12,  9, 14, 10, 11,  8,  0,  6, 13,  3,
   ↪  7)   # n10
7  Pbox19 = (11, 7,  5,  8,  4,  0,  3,  9, 10, 15, 13,  6,  2, 14,  1,
   ↪  12)   # n11
8  Pbox20 = (2,  3, 14, 13,  5, 10,  8,  7,  1,  6,  9,  0,  4, 11, 15,
   ↪  12)   # n12
9
10 Pboxtuto = (0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15)  #
   ↪  Pboxtuto # n13
11
12 Pboxliste = [Pbox0, Pbox1, Pbox2, Pbox3, Pbox4, Pbox5, Pbox6, Pbox7,
   ↪  Pbox8, Pbox9, Pbox10, Pbox11, Pbox12, Pbox13, Pbox14, Pbox15,
   ↪  Pbox16, Pbox17, Pbox18, Pbox19, Pbox20, Pboxtuto]
```

Analyse III

```
17      Input : Nombre de Sbox par uniffdiff (au total : nb*5 Sbox)
18      Name : pour identifier les fichiers
19      Nombre de couples pour DECRYPT
20      Tempsmaxrecherchechemin
21      """
22      K = Ktuto
23      K5 = K[4]
24      Pbox = Pboxtuto
25      nbcouples = nbc
26      tempsmaxrecherchechemin = temps
27      columns = ['Sbox', 'Unifdiff', 'Resultat', 'Time',
        ↪  'CompteurMaxListe', 'RechercheCheminTropLong',
        ↪  'RechercheCoupleImpossible', 'Proba1', 'Proba2', 'Proba3',
        ↪  'Proba4', '0', '2', '4', '6', '8', '10', '12', '14', '16']
28      index = range(nb * 5)
```

Analyse IV

```
30
31      # fichierlog = '/content/gdrive/My Drive/Colab
        ↪  Notebooks/Resul/'+name+'log.txt'
32      # fichierresul = '/content/gdrive/My Drive/Colab
        ↪  Notebooks/Resul/'+name+'resultat.xlsx'
33      # fichierLISTSBOX = '/content/gdrive/My Drive/Colab
        ↪  Notebooks/Resul/'+name+'LISTSBOX'
34      # fichierdata = '/content/gdrive/My Drive/Colab
        ↪  Notebooks/Resul/'+name+'data'
35
36      fichierlog = name+'log.txt'
37      fichierresul = name+'resultat.xlsx'
38      fichierLISTSBOX = name+'LISTSBOX'
39      fichierdata = name+'data'
40
41      with open(fichierlog, 'w'):      # clear log
42          pass
43
44      logging.basicConfig(filename=fichierlog, level=logging.INFO,
        ↪  format='%(asctime)s - %(message)s')
45      logging.info('nb = '+str(nb))
46      logging.info('name = '+name)
```

Analyse V

```
49      with open(fichierlog, 'r'):
50          pass  # pour voir en temps réeel
51
52      nblog = 25     # Intervalle log
53      logging.info('Début calcul LISTSBOX')
54      with open(fichierlog, 'r'):
55          pass
56
57      LISTSBOX = []
58      for i in range(nb):
59          for unifdiff in [4, 6, 8, 10, 12]:
60              Sbox, unifdiff, compteurtab = permutdiffmax(unifdiff)
61              LISTSBOX.append([Sbox, unifdiff, compteurtab])
62
63      logging.info('Fin calcul LISTSBOX')
64      compteur = 0
65      tempsdebut = time.time()
66
67      for elem in LISTSBOX:
68          if compteur % nblog == 0:    # LOG
```

Analyse VI

```
70                  logging.info('count '+str(compteur))
71              else:
72                  logging.info('count '+str(compteur)+'  timeavg
                    ↪  '+str((time.time()-tempsdebut)/compteur))
73              with open(fichierlog, 'r'):
74                  pass  # pour voir en temps réeel
75          if compteur % 100 == 0:
76              print(compteur)
77
78          Sbox = elem[0]
79          K[4] = np.random.randint(0, 2**16)  # Varier K5
80          K5 = K[4]
81          unifdiff = elem[1]
82          compteurtab = elem[2]
83
84          sortiepossible.cache_clear()  # Vider cache pour ne pas
            ↪  favorier la suite
85          PboxLayer.cache_clear()
86          boolCheminTropLong = False
87          boolCoupleImpossible = False
```

Analyse VII

```
89          a = time.time()
90          try:
91              K5cal, compteurMAXLISTE, tabproba = decryptK5(Sbox, Pbox,
                ↪  K, nbcouples, tempsmaxrecherchechemin)
92          except RechercheCheminTropLong:
93              boolCheminTropLong = True
94              K5cal = None
95              compteurMAXLISTE = 0
96              tabproba = [0, 0, 0, 0]
97          except RechercheCoupleImpossible:
98              boolCoupleImpossible = True
99              K5cal = None
100             compteurMAXLISTE = 0
101             tabproba = [0, 0, 0, 0]
```

Analyse VIII

```
103             Temps = time.time() - a
104             Resul = (K5cal == K5)
105             Sboxstr = '_'.join([str(elem) for elem in Sbox])
106             data.append([Sboxstr, unifdiff, Resul, Temps,
                ↪  compteurMAXLISTE, boolCheminTropLong,
                ↪  boolCoupleImpossible, *tabproba, *compteurtab])
107             compteur += 1
108
109         logging.info('Fin calculs')
110
111         compteurResultat = 0
112         compteurTime = 0
113         compteurMaxListe = 0
114         compteurchemin = 0
115         compteurcouple = 0
116
117         for i in range(0, len(data)):
118             if data[i][2]:              # Resultat
119                 compteurResultat += 1
120             compteurTime += data[i][3]  # Temps
121             if data[i][4] > 0:          # CompteurMaxListe
```

Analyse IX

```
123             if data[i][5]:                  # Recherchechemintroplong
124                 compteurchemin += 1
125             if data[i][6]:                  # Recherchecoupleimpossible
126                 compteurcouple += 1
127
128         ResultatAVG = (compteurResultat / (len(data)))*100
129         logging.info('ResultatAVG = '+str(ResultatAVG))
130         TIMEAVG = compteurTime / (len(data))
131         logging.info('TIMEAVG = '+str(TIMEAVG))
132         logging.info('CompteurMaxListe = '+str(compteurMaxListe))
```