Exercices de cryptographie

M1 informatique

1 Cryptographie classique

1.1 Divers

1. Donnez le texte en clair correspondant au texte crypté suivant :

   ```
   irwimkriqirx hi wigyvmxi qewxiv mrjsvqexmuyi
   ```

2. On considère le chiffre de Hill avec $m = 2$, $a = 3$, $b = 5$, $c = 2$ et $d = 7$. Codez le message *Hello* et décodez le message *PBAQT BRD*.
3. Relevez les challenges de cryptographie proposés par le site NewbieContest.

1.2 Programmation

Implémentez les chiffres suivants, par exemple en Python ou pour une plate-forme mobile. L'utilisateur doit pouvoir : choisir sa clé s'il y a lieu, crypter et décrypter.

— **(Substitutions)** Atbash, César, Vigenère, homophone avec carré de Polybe, Playfair, Hill (cas où $m = 2$ uniquement).
— **(Transpositions)** Transposition rectangulaire.
— **(Surchiffrements)** Delastelle.

Dans un premier temps, seules les lettres du message à traiter seront cryptées/décryptées (les autres caractères seront ignorés). Dans un second temps, vous étendrez les programmes n'utilisant pas de carré de Polybe à la portion de la table ASCII allant du caractère 32 (espace) au caractère 126 (tilde).

2 DES simplifié

On considère une simplification de l'algorithme DES dans laquelle les clés et les blocs ont une taille de 16 bits (au lieu de 64). Les tables utilisées dans cette simplification sont en annexe, à la fin de ce document (section A). On considère les éléments suivants donnés en notation hexadécimale :

$$\begin{array}{lcll} K & = & \text{0x1a2b}, & \text{la clé} \\ M & = & \text{0x3c4d}, & \text{un bloc du texte en clair.} \end{array}$$

Les conversions entre écritures hexadécimales et binaires se font simplement en utilisant le tableau de correspondances suivant :

hexa	0	1	2	3	4	5	6	7
bin	0000	0001	0010	0011	0100	0101	0110	0111
hexa	8	9	a	b	c	d	e	f
bin	1000	1001	1010	1011	1100	1101	1110	1111

Par exemple, 23 en hexadécimal correspond à $\underbrace{0010}_{2}\underbrace{0011}_{3}$ en binaire.

1. Calculer les deux sous-blocs de 8 bits L_0 et R_0 (figure 1).

2. Appliquer l'expansion E sur R_0 pour obtenir $E(R_0)$ (figure 2).
3. Dériver la première sous-clé K_1 (figure 3) ; la rotation à gauche est de 1 bit.
4. Calculer $A = E(R_0) \oplus K_1$, où $\oplus$ est le ou exclusif (xor).
5. Grouper les bits de A en 2 blocs de 6 éléments et calculer la valeur fournie par la S-Box S_1 sur le premier bloc et celle fournie par S_2 sur le second bloc (figures 2 et 4).
6. Concaténer les résultats obtenus à la question (5) ci-dessus pour obtenir la suite B de 8 bits.
7. Appliquer la permutation P à B pour obtenir $P(B)$ (figure 2).
8. Calculer $R_1 = P(B) \oplus L_0$.
9. Écrire en hexadécimal le bloc chiffré L_1R_1 obtenu (figure 1).

3 DES complet

Les tables utilisées dans l'algorithme DES complet sont en annexe, à la fin de ce document (section B). On considère les éléments suivants donnés en notation hexadécimale :

$$\begin{array}{lcll} K & = & \text{0x0123456789abcdef}, & \text{la clé} \\ M & = & \text{0x23456789abcdef01}, & \text{un bloc du texte en clair.} \end{array}$$

Les conversions entre écritures hexadécimales et binaires se font comme à l'exercice précédent.

1. Calculer les deux sous-blocs de 32 bits L_0 et R_0 (figure 1).
2. Appliquer l'expansion E sur R_0 pour obtenir $E(R_0)$ (figure 2).
3. Dériver la première sous-clé K_1 (figure 3) ; la rotation à gauche est de 1 bit.
4. Calculer $A = E(R_0) \oplus K_1$, où $\oplus$ est le ou exclusif (xor).
5. Grouper les bits de A en blocs de 6 éléments et calculer les valeurs fournies par les S-Box $S_1, \ldots, S_8$ correspondantes (figures 2 et 4).
6. Concaténer les résultats obtenus à la question (5) ci-dessus pour obtenir la suite B de 32 bits.
7. Appliquer la permutation P à B pour obtenir $P(B)$ (figure 2).
8. Calculer $R_1 = P(B) \oplus L_0$.
9. Écrire en hexadécimal le bloc chiffré L_1R_1 obtenu (figure 1).

Vérifiez vos réponses en implémentant l'algorithme du DES en Python.

4 RSA

Calculs à la main

On choisit $p = 5$ et $q = 11$.

1. Combien vaut n ?
2. Combien vaut $\phi(n)$?
3. On choisit $e = 3$: est-ce correct ? Pourquoi ?
4. Donnez une valeur correcte pour d.
5. Cryptez le message $M = 5$.

Un sujet d'examen

Résolvez l'exercice 1 du sujet d'examen de 2009.

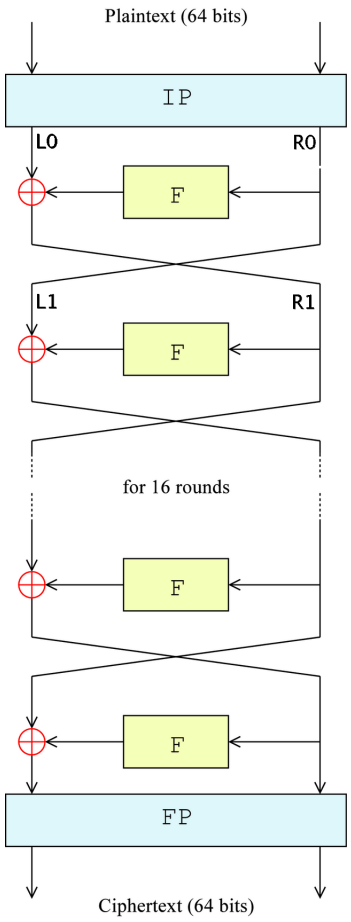

FIGURE 1 – Les 16 rondes du DES, où IP est la permutation initiale, FP la permutation finale, F la fonction de Feistel et $\oplus$ le ou exclusif (xor).

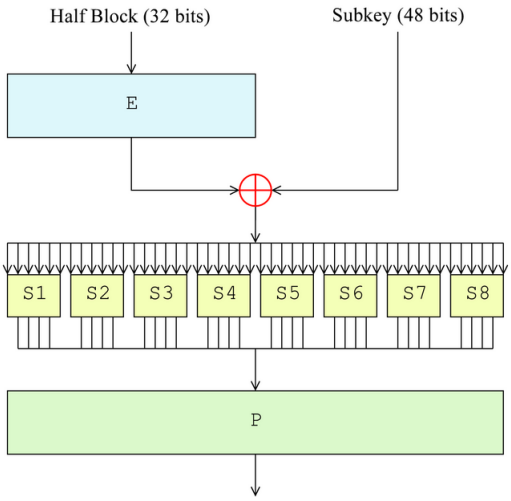

FIGURE 2 – La fonction de Feistel F, où E est l'expansion, $\oplus$ le ou exclusif (xor), les $S_1, \ldots, S_8$ sont les S-Box et P est la permutation.

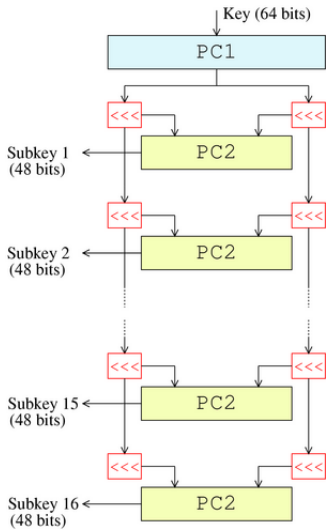

FIGURE 3 – Calcul des sous-clés du DES. La notation $\lll$ désigne des rotations à gauche.

Un petit (?) programme

Le but de cet exercice est d'écrire un programme Python permettant de créer des espions s'envoyant des messages cryptés au moyen de l'algorithme RSA. Chaque espion possède une clé publique et une clé privée qui sont fabriquées à la création de l'espion. Le programme doit proposer le menu suivant à l'utilisateur et exécuter l'action correspondant à son choix :

```
1) Creer un espion
2) Afficher la liste des espions
3) Envoyer un message
4) Quitter le programme
```

Nombres premiers

Écrivez le module `Eratosthene` permettant de gérer les nombres premiers. Vous prévoirez les éléments suivants.

- La liste `crible` qui sert à stocker les nombres premiers qui sont calculés pendant l'exécution du programme.
- La fonction `estPremier` prenant en argument un entier et renvoyant `true` si et seulement si l'entier est premier. Vous écrirez cette méthode en utilisant la technique du crible d'Eratosthène.
- La fonction `get` prenant en argument une borne (un entier) et renvoyant un nombre premier strictement plus petit que cette borne et choisi au hasard.

Modulo

Écrivez le module `Modulo` permettant de gérer des calculs dans $\mathbb{Z}/n\mathbb{Z}$. Vous prévoirez les éléments suivants.

- La fonction `inverse` prenant en argument deux entiers `a` et `n` et renvoyant l'inverse de `a` modulo `n`.
- La fonction `puissance` prenant en argument trois entiers `a`, `b` et `n` et renvoyant le nombre : (`a` à la puissance `b`) modulo `n`.

Schéma de remplissage

Écrivez le module `OAEP` permettant de gérer le schéma de remplissage précédant le cryptage. Vous prévoirez :

- les fonctions `G` et `H` calculant un résumé (*digest*) de leur argument, par exemple en utilisant SHA1 ou MD5 (module hashlib de Python) ;
- la fonction `pad` prenant en argument un message en clair `m` et renvoyant les nombres `X` et `Y` calculés par le schéma de remplissage OAEP ; vous vous arrangerez pour que ces nombres soient plus petits que l'entier `n` du cryptage RSA ;
- la fonction `unpad` prenant en argument des nombres `X` et `Y` (supposés calculés par OAEP) et renvoyant le message `m` correspondant.

Les espions

Écrivez la classe `Espion` permettant de gérer les espions créés par l'utilisateur. Vous prévoirez les éléments suivants.

- Les attributs `p`, `q`, `n`, `phi_n`, `e` et `d` (des entiers) correspondant aux nombres du même nom dans l'algorithme RSA.
- L'attribut `nom` (une chaîne de caractères) permettant de stocker le nom de l'espion.
- Un constructeur dont les arguments sont : une chaîne de caractères permettant d'initialiser le nom, un entier permettant d'initialiser `p` et un entier permettant d'initialiser `q`. Ce constructeur devra calculer la valeur de `n` et de `phi_n`, choisir au hasard une valeur convenable pour `e` puis calculer la valeur de `d`.
- Les méthodes `get_e`, `get_n` et `getNom` qui renvoient respectivement la valeur de l'attribut `e`, `n` et `nom`.
- La méthode `crypter` prenant en argument un message (une chaîne de caractères) et deux entiers `e` et `n` et qui crypte le message en utilisant la clé publique `(e,n)`.
- La méthode `decrypter` prenant en argument un message crypté, qui décrypte le message en utilisant la clé privée `(d,n)` (attributs définis ci dessus) et qui renvoie une chaîne de caractères correspondant au message en clair.
- La méthode `__str__` qui renvoie une chaîne de caractères de la forme `nom(e,n)` (où `nom`, `e` et `n` sont remplacés par leur valeur).

Le programme principal

Le programme principal devra afficher le menu, demander à l'utilisateur quel est son choix, effectuer l'opération correspondante puis recommencer, ceci tant que l'utilisateur ne demande pas de quitter le programme. Le programme devra gérer la liste des espions créés par l'utilisateur.

- **Création d'un espion** : le programme doit demander à l'utilisateur le nom de l'espion ainsi que deux entiers premiers `p` et `q`, vérifier que `p` et `q` sont bien premiers (si ce n'est pas le cas il faut redemander ces valeurs), créer un espion à partir de ces données et ajouter l'espion créé à la liste des espions.
- **Afficher la liste des espions** : le programme doit afficher la liste des espions qui ont été créés par l'utilisateur.
- **Envoyer un message** : le programme doit demander à l'utilisateur le nom de l'espion expéditeur, le nom de l'espion destinataire (si ces noms n'existent pas il faut les redemander), le contenu du message, afficher le message crypté par l'expéditeur et afficher le résultat du décryptage (par le destinataire) du message crypté par l'expéditeur (ce message décrypté doit correspondre au message en clair original lorsque tout va bien). Par exemple :

```
Nom de l'expediteur: Austin
Nom du destinataire: Felicity
Message a envoyer: Yeah baby, yeah!
Austin envoie le message crypte suivant a Felicity:  37 57 95 25 95 37 57
151 0 0 95 95 95 37 95 95 94 25 1 151 0 0 94 25 95 25 95 37 57 151 0 1
Felicity decrypte le message de Austin: Yeah baby, yeah!
```

5 Fonctions de hachage

On considère l'algorithme MD5 appliqué à des blocs de 32 bits (au lieu de 512). On part du message $M = 0x2345abc$ (notation hexadécimale). Les conversions entre écritures hexadécimales et binaires se font simplement en considérant le tableau de correspondances de l'exercice 3.

1. **Remplissage.** On modifie le schéma de remplissage de MD5 pour obtenir un message dont la longueur est un multiple de 32. Soit l la taille en bits de M. On ajoute un 1 à la fin de M, puis suffisamment de 0 pour que le message obtenu ait une longueur inférieure de 8 bits à un multiple de 32. Ensuite, on ajoute à la fin de ce message la valeur de l, codée en binaire sur 8 bits. Calculez le message M' obtenu par ce procédé et les blocs de 32 bits M'_0, M'_1, ... issus de M'.
2. **Une étape de calcul.** On s'intéresse au traitement du bloc M'_0. On utilise les éléments suivants.
   — 4 buffers de 8 bits A, B, C et D initialisés ainsi :
   $$A = 0x01 \quad B = 0x89 \quad C = 0xfe \quad D = 0x76 .$$
   — La fonction F qui prend des arguments codés sur 8 bits et renvoie une valeur sur 8 bits, les opérations se faisant bit à bit :
   $$F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D) .$$
   — Un tableau K de 64 cases initialisées ainsi : pour tout $0 \leq i \leq 63$,
   $$K[i] = \lfloor 2^8 \times |\sin(i + 1)| \rfloor .$$
   — Un tableau W de 4 cases initialisées ainsi : $W[0]$ contient les 8 premiers bits (en partant de la gauche) de M'_0, $W[1]$ contient les 8 bits suivants, et ainsi de suite.
   La partie itérative de l'algorithme MD5 consiste à effectuer 4 rondes, chacune subdivisée en 16 opérations similaires. Calculez le contenu du registre B après la première opération de la première ronde, définie de la façon suivante :
   $$B = B + ((A + F(B, C, D) + W[0] + K[0]) \lll 7)$$
   où $+$ désigne l'addition modulo 2^8 et $\lll 7$ une rotation à gauche de 7 bits.

A Tables pour l'algorithme DES simplifié

A.1 Permutation initiale *IP*

Entrée : 16 bits. Sortie : 16 bits. Le 1er bit de la sortie est obtenu en prenant le 16e bit de l'entrée, le second en prenant le 14e bit de l'entrée, ..., le dernier en prenant le 1er bit de l'entrée.

16	14	12	10	8	6	4	2
15	13	11	9	7	5	3	1

A.2 Expansion *E*

Entrée : 8 bits. Sortie : 12 bits. Le 1er bit de la sortie est obtenu en prenant le 8e bit de l'entrée, le second en prenant le 1er bit de l'entrée, ..., le dernier en prenant le 1er bit de l'entrée.

8	1	2	3	4	5
4	5	6	7	8	1

A.3 Permutation P

Entrée : 8 bits. Sortie : 8 bits. Le 1$^{\text{er}}$ bit de la sortie est obtenu en prenant le 8$^{\text{e}}$ bit de l'entrée, le second en prenant le 4$^{\text{e}}$ bit de l'entrée, ..., le dernier en prenant le 7$^{\text{e}}$ bit de l'entrée.

8	4	2	6	1	5	3	7

A.4 Permutation PC_1

Entrée : une clé de 16 bits. Sortie : 14 bits. Le 1$^{\text{er}}$ bit de la moitié gauche de la sortie est obtenu en prenant le 14$^{\text{e}}$ bit de la clé, le second en prenant le 12$^{\text{e}}$ bit de la clé, ... Le 1$^{\text{er}}$ bit de la moitié droite de la sortie est obtenu en prenant le 15$^{\text{e}}$ bit de la clé, le second en prenant le 13$^{\text{e}}$ bit de la clé, ...

Moitié gauche (7 bits)							Moitié droite (7 bits)						
14	12	10	6	4	2	1	15	13	11	9	7	5	3

A.5 Permutation PC_2

Entrée : 14 bits. Sortie : 12 bits. Le 1$^{\text{er}}$ bit de la sortie est obtenu en prenant le 10$^{\text{e}}$ bit de l'entrée, le second en prenant le 8$^{\text{e}}$ bit de l'entrée, ..., le dernier en prenant le 2$^{\text{e}}$ bit de l'entrée.

10	8	1	6	12	4
13	3	11	5	9	2

B Tables pour l'algorithme DES complet

B.1 Permutation initiale IP

Entrée : 64 bits. Sortie : 64 bits. Le 1$^{\text{er}}$ bit de la sortie est obtenu en prenant le 58$^{\text{e}}$ bit de l'entrée, le second en prenant le 50$^{\text{e}}$ bit de l'entrée, ..., le dernier en prenant le 7$^{\text{e}}$ bit de l'entrée.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

B.2 Expansion E

Entrée : 32 bits. Sortie : 48 bits. Le 1$^{\text{er}}$ bit de la sortie est obtenu en prenant le 32$^{\text{e}}$ bit de l'entrée, le second en prenant le 1$^{\text{er}}$ bit de l'entrée, ..., le dernier en prenant le 1$^{\text{er}}$ bit de l'entrée.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

B.3 Permutation P

Entrée : 32 bits. Sortie : 32 bits. Le 1$^{\text{er}}$ bit de la sortie est obtenu en prenant le 16$^{\text{e}}$ bit de l'entrée, le second en prenant le 7$^{\text{e}}$ bit de l'entrée, ..., le dernier en prenant le 25$^{\text{e}}$ bit de l'entrée.

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

B.4 Permutation PC_1

Entrée : une clé de 64 bits. Sortie : 56 bits. Le 1$^{\text{er}}$ bit de la moitié gauche de la sortie est obtenu en prenant le 57$^{\text{e}}$ bit de la clé, le second en prenant le 49$^{\text{e}}$ bit de la clé, ... Le 1$^{\text{er}}$ bit de la moitié droite de la sortie est obtenu en prenant le 63$^{\text{e}}$ bit de la clé, le second en prenant le 55$^{\text{e}}$ bit de la clé, ...

Moitié gauche (28 bits)							Moitié droite (28 bits)						
57	49	41	33	25	17	9	63	55	47	39	31	23	15
1	58	50	42	34	26	18	7	62	54	46	38	30	22
10	2	59	51	43	35	27	14	6	61	53	45	37	29
19	11	3	60	52	44	36	21	13	5	28	20	12	4

B.5 Permutation PC_2

Entrée : 56 bits. Sortie : 48 bits. Le 1$^{\text{er}}$ bit de la sortie est obtenu en prenant le 14$^{\text{e}}$ bit de l'entrée, le second en prenant le 17$^{\text{e}}$ bit de l'entrée, ..., le dernier en prenant le 32$^{\text{e}}$ bit de l'entrée.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

B.6 Rotations pour le calcul des sous-clés

ronde	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
rotations	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

S_1																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
$0yyyy1$	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
$1yyyy0$	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
$1yyyy1$	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
$0yyyy1$	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
$1yyyy0$	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
$1yyyy1$	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
$0yyyy1$	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
$1yyyy0$	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
$1yyyy1$	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
$0yyyy1$	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
$1yyyy0$	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
$1yyyy1$	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
$0yyyy1$	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
$1yyyy0$	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
$1yyyy1$	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
$0yyyy1$	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
$1yyyy0$	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
$1yyyy1$	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
$0yyyy1$	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
$1yyyy0$	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
$1yyyy1$	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8																
	$x0000x$	$x0001x$	$x0010x$	$x0011x$	$x0100x$	$x0101x$	$x0110x$	$x0111x$	$x1000x$	$x1001x$	$x1010x$	$x1011x$	$x1100x$	$x1101x$	$x1110x$	$x1111x$
$0yyyy0$	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
$0yyyy1$	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
$1yyyy0$	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
$1yyyy1$	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

FIGURE 4 – Les 8 S-Box. Entrée : 6 bits. Sortie : 4 bits. Par exemple, pour S_5 avec l'entrée **0**11011**1**, on regarde la case ligne $0yyyy1$, colonne $x1101x$, et on obtient comme sortie 9 *i.e.* 1001.