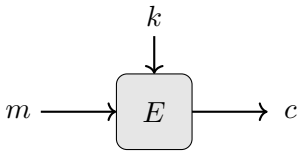
Chiffrements par bloc

Christina Boura
christina.boura@irif.fr

Les chiffrements par bloc sont les primitives symétriques les plus utilisées pour garantir la confidentialité des données. Comme leur nom l'indique, ces chiffrements opèrent sur des *blocs* de données, généralement de taille 64, 128 ou 256 bits. Un chiffrement par bloc permet de transformer un bloc de message clair m en un bloc de texte chiffré c sous l'action d'une clé secrète k :



Deux paramètres essentiels caractérisent un chiffrement par bloc :

1. la taille des blocs, notée n, et
2. la taille de la clé, notée κ.

La taille de la clé est généralement comprise entre 80 et 256 bits.

Pour une clé donnée, un chiffrement par bloc de n bits associe l'ensemble des 2^n blocs d'entrée de n bits au même ensemble de 2^n blocs de sortie :

$$\{\texttt{0}\ldots\texttt{00}, \texttt{0}\ldots\texttt{01}, \texttt{0}\ldots\texttt{10}, \texttt{0}\ldots\texttt{11}, \ldots, \texttt{1}\ldots\texttt{1}\}.$$

Cette association est réalisée de manière à ce que chaque bloc de sortie apparaisse exactement une fois. En d'autres termes, pour une clé fixée, un chiffrement par bloc peut être vu comme une permutation de $\mathbb{F}_2^n$, c'est-à-dire de l'ensemble des mots de n bits. En variant la clé secrète, on obtient différentes permutations, faisant ainsi du chiffrement par bloc un moyen de générer une famille de permutations, chaque permutation étant indexée par une clé secrète $k \in \mathbb{F}_2^\kappa$.

La définition suivante formalise cette description.

Définition 1. *Un chiffrement par bloc est une permutation E qui prend en entrée deux arguments : une clé secrète de κ bits et un bloc de n bits, et qui retourne en sortie un bloc de n bits.*

$$\begin{aligned} E : \{0,1\}^\kappa \times \{0,1\}^n &\rightarrow \{0,1\}^n \\ (k, m) &\mapsto E_k(m) \stackrel{déf}{=} E(k, m) \end{aligned}$$

La taille du bloc n détermine l'espace de toutes les permutations possibles que le chiffrement par bloc pourrait générer. La taille de la clé κ détermine le nombre de permutations effectivement générées. Pour apprécier la difficulté de la tâche du concepteur, il est utile d'examiner certains chiffres impliqués. Pour un chiffrement par bloc avec une taille de clé κ, il existe 2^κ clés possibles et chaque clé spécifie une permutation de 2^n messages. Il existe $(2^n)!$ permutations différentes sur des mots de n bits, ce qui, en utilisant l'approximation de Stirling, est proche de $2^{(n-1)2^n}$. Pour

des valeurs typiques de n et κ, un chiffrement par bloc ne fournira qu'une infime fraction de toutes les permutations disponibles. De plus, il le fera de manière très structurée. Cependant, nous attendons d'un bon chiffrement par bloc qu'il dissimule cette structure et, pour simplifier, nous espérons qu'une clé choisie aléatoirement "sélectionne" une permutation de manière qui semble aléatoire parmi les $2^{(n-1)2^n}$ possibilités. De manière encore plus exigeante, nous demandons que des clés présentant une relation quelconque produisent des permutations sans relation exploitable entre elles.

1 Les chiffrements par bloc itératifs

Aujourd'hui, la méthode la plus courante pour construire un chiffrement par bloc repose sur une approche *itérative*. Cela consiste à utiliser une même fonction interne appliquée plusieurs fois, chaque itération utilisant une clé différente. Cette fonction interne est généralement cryptographiquement faible prise isolément ; cependant, en l'itérant plusieurs fois, on obtient une fonction robuste et résistante aux diverses attaques.

De manière formelle, un chiffrement par bloc itératif consiste à appliquer r fois une fonction interne $\mathcal{R}$, appelée *fonction de tour*, comme illustré dans la figure 1. Chaque itération utilise un paramètre secret k_i, nommé *clé de tour*. Ces clés de tour sont dérivées de la clé secrète principale du chiffrement, k, aussi appelée *clé maître*, par un algorithme de *cadencement de clés* :

$$E_k = \mathcal{R}_{k_r} \circ \mathcal{R}_{k_{r-1}} \circ \cdots \circ \mathcal{R}_{k_1}.$$

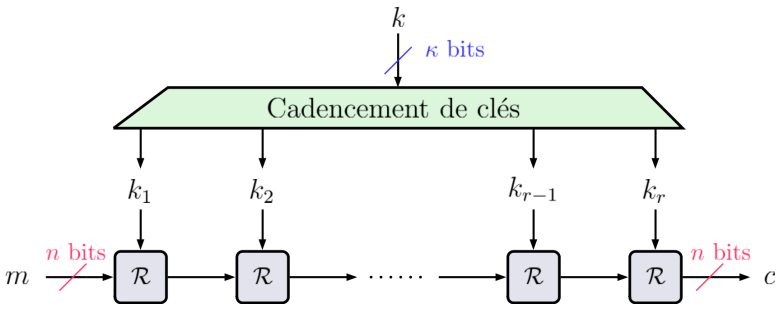


FIGURE 1 – Le chiffrement itératif par bloc

Cette construction présente plusieurs avantages : premièrement, l'usage d'une seule composante (la fonction de tour) permet la conception de fonctions qui sont bien comprises et ainsi faciles à analyser. Ensuite, ces constructions peuvent être implémentées d'une manière compacte et efficaces et offrent des performances très intéressantes. Les chiffrements par bloc sont exécutés aujourd'hui sur plusieurs plateformes différentes et dispositifs différents, ainsi ce dernier point est très important.

1.1 Structure de la fonction de tour

La structure de la fonction interne varie selon les algorithmes. Cependant, elle suit généralement les principes de conception définis par Claude Shannon [17]. Selon ces principes, un algorithme de chiffrement symétrique doit répondre aux deux besoins suivants, la *confusion* et la *diffusion* :

— **Confusion :** La confusion correspond à la volonté de rendre la relation entre la clé de chiffrement et le texte chiffré la plus complexe possible.

— **Diffusion :** Chaque bit du texte clair et de la clé secrète doit influencer un grand nombre de bits dans le texte chiffré.

La confusion est assurée par des fonctions *non-linéaires*, souvent par l'application en parallèle des petites fonctions appelées *boîtes-S*. La diffusion est quant à elle assurée principalement par des fonctions linéaires. Dans des chiffrements comme le DES ou PRESENT, la permutation agit au niveau des bits, ce qui est efficace en matériel mais moins performant en logiciel. Par contre, des chiffrements comme l'AES utilisent des opérations linéaires plus complexes au niveau des octets, optimisées pour les architectures 32 bits, ce qui améliore la performance logiciel.

Aujourd'hui il existe deux grandes familles de chiffrement itératifs par bloc : les *réseaux de Feistel* et les *réseaux de substitution-permutation.*

1.2 Réseaux de Feistel

Un réseau de Feistel (figure 2), qui port le nom du cryptologue d'IBM Hors Feistel, est une structure développée dans les années 1970. Dans la version équilibrée de cette construction, les données d'entrée sont divisées en deux parties de taille identique. Les deux parties sont ensuite traitées de façon différente :

$$\begin{aligned} F_k : \mathbb{F}_2^n \times \mathbb{F}_2^n &\rightarrow \mathbb{F}_2^n \times \mathbb{F}_2^n \\ (L, R) &\mapsto (R, L \oplus f(k, R)) \end{aligned}$$

La fonction interne f, de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ est une fonction composée de manière générale d'une couche non-linéaire suivie d'une couche linéaire. Il est intéressant de noter que la fonction $f_k(\cdot)$ elle-même n'a pas besoin d'être inversible pour permettre le déchiffrement. Par exemple, dans le cas du DES, c'est effectivement le cas. Le déchiffrement est donc réalisé en appelant la même structure, mais en utilisant les sous-clés dans l'ordre inverse. Autrement dit, si le chiffrement est effectué avec les sous-clés $k_1, k_2, \ldots, k_r$, alors le déchiffrement est réalisé avec les sous-clés $k_r, k_{r-1}, \ldots, k_1$. Cela est très intéressant, car cela permette d'utiliser le même circuit pour chiffrer et déchiffrer les données.

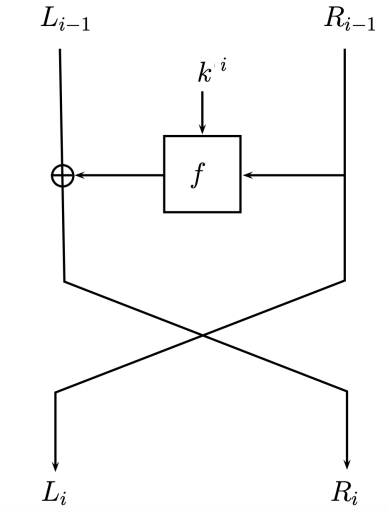


FIGURE 2 – Un tour d'un chiffrement d'un réseau de type Feistel

La fonction de chiffrement DES, standardisée en 1977, utilise cette méthode de construction. Ce standard sera décrit dans la section 2. Aujourd'hui, il existe des généralisations de cette

construction, notamment des versions *non-équilibrées* où les données sont découpées en deux parties de tailles différentes [16] ou encore des versions où les données sont découpées en plus de deux parties [20].

1.3 Réseaux de substitution-permutation

Un algorithme de substitution-permutation ou SPN (Substitution Permutation Network en anglais) est constitué d'une succession de tours de la forme $L \circ N \circ K$, où K est une opération de mélange de clé (souvent on ajoute simplement la clé avec un XOR ($\oplus$)), N une fonction non-linéaire assurant la confusion et L une fonction linéaire assurant la diffusion (figure 3).

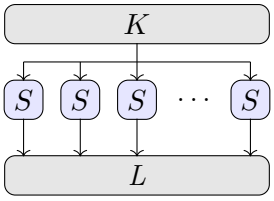


FIGURE 3 – Fonction de tour d'un réseau substitution-permutation. La couche non-linéaire est constituée de l'application en parallèle d'une boîte-S S.

Cette construction ne garantit pas en elle-même l'inversibilité du chiffrement. Pour cette raison il est nécessaire que chacune des composantes soit inversible.

Le standard actuel de chiffrement par bloc, AES, suit ce principe de conception. Il sera décrit dans la section 5.

Boîtes-S La construction d'une fonction non-linéaire ayant de bonnes propriétés cryptographiques est un problème très important. Cependant, la mise en œuvre d'une telle fonction de grande taille est très coûteuse tant en termes de temps qu'en termes de mémoire. Pour cette raison il est courant de découper l'état interne en petits morceaux et d'appliquer une petite fonction non-linéaire, appelée *boîte-S*, ou Sbox en anglais, sur chacun d'entre eux. Parmi les primitives symétriques connues, certaines utilisent des boîtes-S de 8 bits (AES) ou des boîtes-S de 4 bits, comme c'est le cas des chiffrements à bas coût PRESENT, SKINNY-64 ou GIFT. Plus rarement, des boîtes-S de taille 3 bits (chiffrement à bas coût PRINTCIPHER) ou 5 bits (KECCAK, ASCON) sont utilisées. Il est aussi possible dans d'autres constructions d'utiliser des boîtes-S non-inversibles, comme c'est le cas de DES, mais cela reste extrêmement rare.

2 Data Encryption Standard (DES)

DES a été publié en 1975. Cet algorithme, développé en interne chez IBM entre 1973 et 1974, était basé sur une conception initiale de Horst Feistel, appelée LUCIFER. Après des modifications apportées à cette première version (notamment par la NSA), DES est devenu un standard fédéral en août 1976 et a été publié sous le nom de FIPS PUB 46 en janvier 1977 [11]. DES chiffre et déchiffre des blocs de 64 bits en utilisant une clé de 56 bits. Le chiffrement suit une structure classique de Feistel et itère une fonction de tour 16 fois. Les bits sont numérotés de 1 (bit le plus à gauche) à 64 (bit le plus à droite). Même si aujourd'hui, nous avons tendance à numéroter les bits en commençant à partir de 0, nous suivrons ici, pour des raisons historiques, la spécification originale du DES, telle que donnée dans la documentation originale du FIPS [11]

2.1 Permutations initiale et finale

Avant le début du chiffrement, une permutation bit-à-bit initiale IP est appliquée au bloc d'entrée de 64 bits. Pour conserver la propriété des réseaux de Feistel permettant d'utiliser le circuit de chiffrement pour le déchiffrement, l'inverse de cette permutation, notée IP^{-1}, est appliquée à l'état juste avant de produire le texte chiffré. La permutation IP et son inverse sont données dans la table 1. Cette table indique que le premier bit (bit 1) après la permutation contiendra le contenu du bit 58 avant la permutation. De la même manière, le deuxième bit (bit 2) contiendra le bit 50 et le dernier bit (bit 64) contiendra le contenu du bit 7.

IP							
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

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

TABLE 1 – La permutation IP (à gauche) et son inverse IP^{-1} (à droite)

2.2 Fonction de tour du DES

Après l'application de la permutation initiale IP, le bloc de 64 bits est divisé en deux mots de 32 bits : une moitié gauche L_0 et une moitié droite R_0. La même fonction de tour est ensuite itérée 16 fois. Ce procédé est illustré dans la figure 5. Au tour i, $1 \leq i \leq 16$, une fonction f prend en entrée la moitié droite de l'état R_{i-1} (32 bits) avec la sous-clé K_i (48 bits) et produit une sortie de 32 bits, qui est ensuite XORée avec la moitié gauche L_{i-1}. Les branches droite et gauche sont alors échangées :

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

pour $1 \leq i \leq 16$. Le bloc pré-sortie, qui est l'entrée de la permutation IP^{-1}, est $R_{16}||L_{16}$.

2.2.1 La fonction f

La fonction f prend en entrée, au tour i, une valeur R_{i-1} de 32 bits et une sous-clé K_i de 48 bits. Les transformations suivantes (voir figure 5) sont ensuite appliquées :

1. La première opération de f est l'application à R_{i-1} d'une fonction d'expansion bit-à-bit E. L'objectif de cette expansion est de transformer la valeur R_{i-1} (32 bits) en une valeur intermédiaire de 48 bits. Cela se fait en dupliquant 16 des bits d'entrée de sorte qu'ils apparaissent deux fois en sortie. L'action de E peut être représentée par la partie gauche de la table 2. Cette table doit être lue de la même manière que celles de IP et IP^{-1}. Par exemple, les trois premiers bits du registre intermédiaire de 48 bits après application de E contiendront les bits 32, 1 et 2 avant l'application.
2. La sortie de 48 bits après l'application de E est XORée avec la sous-clé K_i de 48 bits.
3. Le résultat de 48 bits de l'opération précédente est ensuite divisé en huit mots de 6 bits, et une boîte-S différente est appliquée à chaque mot. Les huit boîtes-S, notées $S_1, S_2, \ldots, S_8$

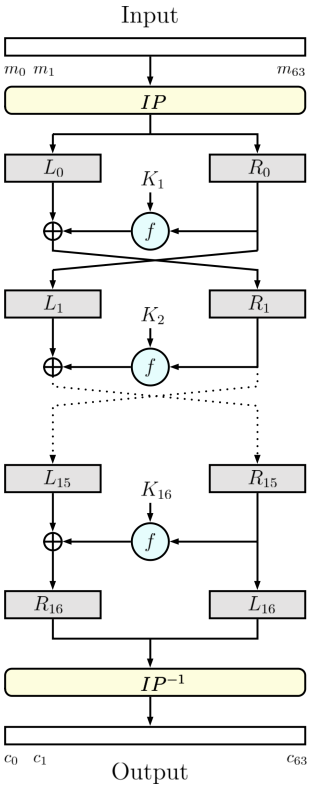


FIGURE 4 – Chiffrement avec DES

sont données dans la table 3. Chaque boîte-S prend en entrée 6 bits et produit une sortie de 4 bits. En concaténant les sorties de toutes les boîtes-S, on obtient une valeur intermédiaire de 32 bits.

4. Une permutation de bits P est finalement appliquée à la sortie de 32 bits de l'étape précédente. Cette permutation est donnée dans la table 2 et doit être interprétée comme toutes les tables précédemment fournies.

E					
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

P			
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

TABLE 2 – Table de sélection des bits pour E (à gauche) et permutation P (à droite)

Boîtes-S Les 8 boîtes-S du DES sont très importantes pour la sécurité du DES car elles sont les seuls composants non-linéaires du chiffrement. Les boîtes-S finalement utilisées dans le DES ont été choisies par la NSA et différaient de celles présentées dans la première version de l'algorithme. Aujourd'hui, nous savons que les boîtes-S ont été notamment manipulées pour que le DES résiste à l'attaque différentielle, quinze ans avant que cette attaque ne soit découverte et publiée publiquement par Biham et Shamir. Toutes les boîtes-S sont différentes mais ont des propriétés cryptographiques similaires (par exemple, toutes ont un degré algébrique de 5). Les

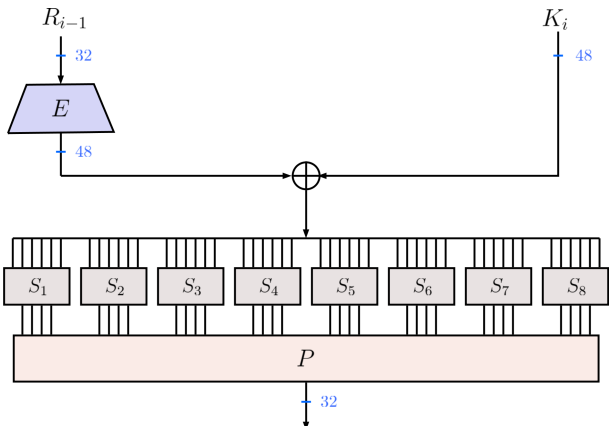


FIGURE 5 – Fonction f utilisée dans DES

huit boîtes-S sont décrites dans la table 3. Chaque table doit être interprétée de la manière suivante. L'entrée de 6 bits pour chaque S-box est divisée en deux parties. Tout d'abord, les deux bits extérieurs, concaténés en une valeur de 2 bits, sont utilisés pour choisir une ligne dans la table : (`00` pour la ligne 0, `01` pour la ligne 1, `10` pour la ligne 2 et `11` pour la ligne 3). Ensuite, les quatre bits intérieurs sont utilisés pour choisir une colonne de la table. Prenons l'exemple de la S-box S_1 et supposons que l'entrée de la S-box soit $(\texttt{110101})_2$. Les deux bits extérieurs `11` pointent vers la ligne 3, tandis que les quatre bits intérieurs `1010` forment la valeur hexadécimale `a`. Ainsi, la sortie est la valeur hexadécimale `3`, soit `0011` en binaire : $S_1\texttt{[110101]} = \texttt{0011}$. Il est à noter que, en raison de l'action de E, les bits entrant dans une boîte-S sont partagés entre deux boîtes-S voisines.

3 Génération des sous-clés du DES

Cette section décrit l'algorithme de cadencement de clés du DES. Cet algorithme prend en entrée la clé maître de 56 bits K du DES et génère seize sous-clés de 48 bits $K_1, \ldots, K_{16}$. La clé maître du DES contient 56 bits générés aléatoirement, placés dans un registre de 64 bits. Les 8 bits restants dans ce registre, aux positions 8, 16, 24, ..., 64, sont des *bits de parité*, ce qui signifie que la valeur de chacun de ces bits est choisie de manière à ce que la parité de l'octet correspondant (XOR de tous les bits de l'octet) soit impaire. Cette caractéristique pourrait être utilisée, par exemple, pour détecter des erreurs lors de la génération, la transmission ou le stockage de la clé.

La première opération dans la génération des sous-clés est l'application de la permutation de bits $PC1$, décrite dans la table 4. Nous voyons notamment dans cette table que les huit bits de parité sont ignorés. Les 56 bits restants, après permutation, sont chargés dans deux registres de 28 bits, C et D. Les bits chargés dans le registre C sont ceux correspondant aux quatre premières lignes de la table, tandis que ceux chargés dans D sont ceux des quatre lignes restantes. Ensuite, les deux registres C et D sont décalés vers la gauche de 1 ou 2 positions. Ce nombre de décalages dépend du tour i, est noté r_i, $1 \leq i \leq 16$, et est donné dans la table 4. Enfin, 48 bits sont extraits des registres C et D selon une table appelée $PC2$) qui est également décrite dans la table 4. Pour cela, les registres C et D sont concaténés pour former un registre unique de 56 bits $C||D$, et $PC2$ choisit 48 bits parmi $C||D$ pour former la sous-clé de tour. Par exemple, le bit 1 de la sous-clé de tour K_i est le bit 14 du registre C, tandis que le dernier bit de K_i est le bit 32 de $C||D$, soit le bit 4 du registre D.

Nous concluons cette section par un résumé des résultats de cryptanalyse les plus importants

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S_1																
0	e	4	d	1	2	f	b	8	3	a	6	c	5	9	0	7
1	0	f	7	4	e	2	d	1	a	6	c	b	9	5	3	8
2	4	1	e	8	d	6	2	b	f	c	9	7	3	a	5	0
3	f	c	8	2	4	9	1	7	5	b	3	e	a	0	6	d
S_2																
0	f	1	8	e	6	b	3	4	9	7	2	d	c	0	5	a
1	3	d	4	7	f	2	8	e	c	0	1	a	6	9	b	5
2	0	e	7	b	a	4	d	1	5	8	c	6	9	3	2	f
3	d	8	a	1	3	f	4	2	b	6	7	c	0	5	e	9
S_3																
0	a	0	9	e	6	3	f	5	1	d	c	7	b	4	2	8
1	d	7	0	9	3	4	6	a	2	8	5	e	c	b	f	1
2	d	6	4	9	8	f	3	0	b	1	2	c	5	a	e	7
3	1	a	d	0	6	9	8	7	4	f	e	3	b	5	2	c
S_4																
0	7	d	e	3	0	6	9	a	1	2	8	5	b	c	4	f
1	d	8	b	5	6	f	0	3	4	7	2	c	1	a	e	9
2	a	6	9	0	c	b	7	d	f	1	3	e	5	2	8	4
3	3	f	0	6	a	1	d	8	9	4	5	b	c	7	2	e
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S_5																
0	2	c	4	1	7	a	b	6	8	5	3	f	d	0	e	9
1	e	b	2	c	4	7	d	1	5	0	f	a	3	9	8	6
2	4	2	1	b	a	d	7	8	f	9	c	5	6	3	0	e
3	b	8	c	7	1	e	2	d	6	f	0	9	a	4	5	3
S_6																
0	c	1	a	f	9	2	6	8	0	d	3	4	e	7	5	b
1	a	f	4	2	7	c	9	5	6	1	d	e	0	b	3	8
2	9	e	f	5	2	8	c	3	7	0	4	a	1	d	b	6
3	4	3	2	c	9	5	f	a	b	e	1	7	6	0	8	d
S_7																
0	4	b	2	e	f	0	8	d	3	c	9	7	5	a	6	1
1	d	0	b	7	4	9	1	a	e	3	5	c	2	f	8	6
2	1	4	b	d	c	3	7	e	a	f	6	8	0	5	9	2
3	6	b	d	8	1	4	a	7	9	5	0	f	e	2	3	c
S_8																
0	d	2	8	4	6	f	b	1	a	9	3	e	5	0	c	7
1	1	f	d	8	a	3	7	4	c	5	6	b	0	e	9	2
2	7	b	4	1	9	c	e	2	0	6	a	d	f	3	5	8
3	2	1	e	7	4	a	8	d	f	c	9	0	3	5	6	b

TABLE 3 – DES Les boîtes-S du DES

contre le DES.

$PC1$						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

$PC2$					
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

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

TABLE 4 – Permutations $PC1$ et $PC2$. La dernière table contient les constantes de décalage dépendant des tours r_i.

4 Meilleures attaques contre le DES

L'invention de la cryptanalyse différentielle par Biham et Shamir [3] dans les années 1990 a permis de casser les 16 tours du DES. Cette attaque, bien qu'ayant une complexité en temps particulièrement faible, n'a pas été mise en œuvre en raison du nombre relativement élevé de paires texte clair/texte chiffré choisi nécessaires à son exécution. L'invention de la cryptanalyse linéaire par Gilbert et Tardy-Corfdir [18] et Matsui [9] a permis à Matsui de réaliser une attaque pratique de récupération de clé contre le DES et de retrouver la clé secrète après quelques jours de calcul [10]. Depuis ces résultats de cryptanalyse, le DES est considéré comme cassé, et des attaques par force brute réussies entre 1997 et 1999 ont démontré que la longueur de clé du DES devenait trop courte pour les machines actuelles. Certaines améliorations et raffinements de l'attaque linéaire initiale contre le DES sont apparus après 2000. Toutes ces attaques sont résumées dans la table 5.

Année	Cryptanalyse	Données	Temps	Scénario d'attaque	Référence
1990	Différentielle	2^{47}	2^{37}	à clair choisi	[3]
1994	Linéaire	2^{43}	2^{43}	à clair connu	[10]
2001	Linéaire	2^{43}	2^{41}	à clair connu	[7]
2017	Linéaire multiple	$2^{42.78}$	$2^{38.86}$	à clair connu	[4]
2017	Linéaire multiple	2^{41}	$2^{49.76}$	à clair connu	[4]

TABLE 5 – Tous les résultats connus de cryptanalyse contre le DES complet

Retrait À la fin des années 1990, de nombreuses démonstrations pratiques ont montré que la longueur de clé du DES était bien trop courte pour les nouvelles machines, mais le standard n'a été officiellement retiré qu'en 2004. En 2018, le NIST a également déprécié le 3DES pour toutes les nouvelles applications. Son utilisation devrait être interdite après 2023.

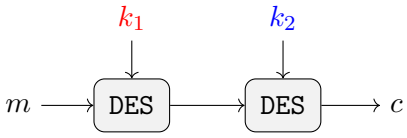
4.1 Double DES

Une idée pour continuer à utiliser le DES sans avoir les problèmes liés à sa clé de 56 bits, beaucoup trop courte, était de composer deux instances de DES avec deux clés indépendantes. Ainsi, la taille de la clé augmentait à 112 bits. Cette approche a donné lieu au DOUBLE DES

(ou 2DES).

Le 2DES repose sur une double application du chiffrement DES avec deux clés différentes. Ainsi, pour un message m et deux clés secrètes k_1 et k_2, le chiffrement 2DES produit un texte chiffré c de la manière suivante :

$$c = \text{DES}_{k_2}(\text{DES}_{k_1}(m))$$



L'idée derrière cette extension était qu'en utilisant deux clés indépendantes, la sécurité devait être renforcée, rendant plus difficile une attaque par recherche exhaustive. Théoriquement, la recherche exhaustive aurait une complexité de 2^{112}, ce qui était considéré comme impraticable à l'époque et l'est encore aujourd'hui. Cependant, une attaque appelée *attaque par le milieu* ou *Meet-in-the-Middle (MITM)* permet de réduire cette complexité de façon significative. Cette attaque, décrite dans la section suivante, est la raison pour laquelle le 2DES n'a jamais été utilisé en pratique.

4.2 L'attaque par le milieu sur le 2DES

L'attaque par le milieu (en anglais Meet-in-the-Middle attack) a été introduite par Diffie et Hellman [6]. Elle démontre que le 2DES n'atteint pas la sécurité espérée de 112 bits.

L'attaque fonctionne en utilisant un couple clair/chiffré (m, c) connu de l'attaquant, où m est le message en clair et c est le texte chiffré correspondant. Puisque $c = \text{DES}_{k_2}(\text{DES}_{k_1}(m))$, on a :

$$\text{DES}_{k_1}(m) = \text{DES}^{-1}_{k_2}(c).$$

L'attaque fonctionne de la manière suivante :

1. L'attaquant calcule $x = \text{DES}_{k_1}(m)$ pour toutes les 2^{56} valeurs possibles de k_1 et stocke chaque paire (k_1, x) dans une table.
2. Ensuite, pour chaque clé k_2, l'attaquant calcule $x' = \text{DES}^{-1}_{k_2}(c)$, où $\text{DES}^{-1}_{k_2}$ représente le déchiffrement avec la clé k_2. À chaque fois que le résultat x' obtenu dans l'étape 2 correspond à une valeur x dans la table de l'étape 1, l'attaquant a trouvé une paire de clés (k_1, k_2) candidate.

Avec une paire supplémentaire de texte clair/texte chiffré (m', c'), l'attaquant peut vérifier si chaque clé candidate est correcte en vérifiant si $\text{DES}_{k_2}(\text{DES}_{k_1}(m')) = c'$.

Complexité de l'attaque Grâce à cette attaque, la complexité pour retrouver la clé secrète est bien inférieure à 2^{112}. En effet :

— **Temps de calcul** : L'attaquant effectue 2^{56} opérations pour chaque clé k_1 et 2^{56} pour chaque clé k_2, soit un total de $2 \times 2^{56} = 2^{57}$ opérations.
— **Espace mémoire** : La table pour stocker les couples $(k_1, \text{DES}_{k_1}(m))$ nécessite 2^{56} emplacements.

Au final, la clé correcte sera retrouvée en environ 2^{56} opérations : malgré sa clé de 112 bits, le 2DES n'atteint qu'une sécurité de 56 bits. C'est pourquoi le 2DES a été abandonné au profit du Triple DES (3DES), qui applique trois fois l'algorithme DES avec deux ou trois clés différentes.

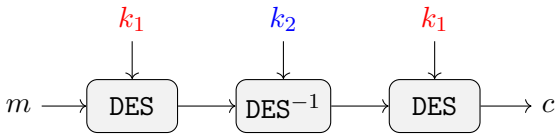
4.3 Le triple DES (3DES)

Pour augmenter le coût de la recherche exhaustive, il a été proposé d'utiliser le triple DES (3DES), qui consiste en trois applications successives du chiffrement DES, le plus souvent avec deux clés secrètes. Cette technique permet de doubler la taille de la clé secrète, ce qui protège contre les attaques par recherche exhaustive. Comme le DES, le 3DES opère sur des blocs de 64 bits mais utilise une clé secrète de 112 bits.

La raison pour laquelle on utilise uniquement deux clés secrètes (et non trois) est liée à l'attaque par le milieu décrite précédemment. Cette attaque peut également s'appliquer lorsque trois clés différentes sont utilisées dans un triple DES : dans ce cas, il est possible de réduire la complexité de la recherche exhaustive à 2^{112} essais, ce qui revient à la complexité de la recherche exhaustive sur deux clés DES.

Le 3DES à deux clés fonctionne de la manière suivante : on effectue d'abord un chiffrement DES avec la première clé k_1, suivi d'un déchiffrement DES avec la seconde clé k_2, et enfin un autre chiffrement DES avec la première clé k_1. En utilisant les fonctions de chiffrement et de déchiffrement de DES, la fonction de chiffrement du3DES est définie par :

$$c \leftarrow \mathrm{DES}_{k_1}(\mathrm{DES}^{-1}_{k_2}(\mathrm{DES}_{k_1}(m))).$$



Le fait de combiner les trois DES sous la forme *chiffrement-déchiffrement-chiffrement* (mode EDE), plutôt que par trois chiffrements successifs, permet de préserver la compatibilité avec le DES simple. En effet, si l'on utilise deux clés identiques $k_1 = k_2$, alors le 3DES est équivalent au DES standard.

Le 3DES à deux clés est défini dans le document du NIST SP-800-67 [2] et a été adopté dans la norme ANSI X9.17 [1]. Cet algorithme a été largement utilisé par le système bancaire. Cependant, avec les avancées en cryptanalyse et l'apparition de méthodes de chiffrement plus performantes, le NIST a annoncé en 2017 la fin de son utilisation progressive. Aujourd'hui, l'AES (Advanced Encryption Standard) est le standard recommandé pour le chiffrement symétrique dans la plupart des systèmes de sécurité.

5 L'AES

.

Au milieu des années 1990, après le succès de plusieurs attaques par force brute contre le DES, il est devenu clair que la clé du DES était beaucoup trop courte et qu'un nouveau chiffrement symétrique était nécessaire pour remplacer l'ancien standard. Entre 1997 et 2000, l'Institut National des Standards et de la Technologie américain (NIST) a organisé un concours public visant à remplacer le DES par un nouveau chiffrement par bloc sécurisé. Les candidats à ce concours devaient supporter des clés de 128, 192 et 256 bits, et offrir un niveau de sécurité au moins égal à celui du triple DES avec deux clés.

En tout, 15 propositions ont été acceptées pour le premier tour du concours, et 5 ont été sélectionnées en août 1999 pour la phase finale. Le 2 octobre 2000, le NIST a annoncé que

Rijndael, un chiffrement par bloc conçu par Joan Daemen et Vincent Rijmen, avait remporté le concours et devait devenir le *Advanced Encryption Standard*, ou AES.

Dans cette section, nous donnons une brève description de ce standard. Une description plus détaillée des spécifications du chiffrement et de ses critères de conception peut, par exemple, être consultée dans [8].

5.1 Arithmétique dans les corps finis

Comprendre l'arithmétique dans le corps fini $\mathbb{F}_{2^n}$ est essentiel pour calculer avec l'AES. La conception de l'AES est orientée par octets, et son état peut être vu comme une matrice de 4×4 octets. Chaque octet peut être représenté comme un vecteur binaire $(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ (le bit le plus à droite est le moins significatif) ou peut être exprimé en tant que polynôme de degré 7 dans la variable X :

$$a_7X^7 + a_6X^6 + a_5X^5 + a_4X^4 + a_3X^3 + a_2X^2 + a_1X + a_0.$$

Cette représentation polynomiale est très pratique pour réaliser des opérations mathématiques, notamment les opérations d'addition et de multiplication.

Addition

Pour additionner deux octets $a = (a_7, a_6, \ldots, a_0)$ et $b = (b_7, b_6, \ldots, b_0)$, on additionne les deux polynômes correspondants :

$$\begin{aligned}
&a_7X^7 + a_6X^6 + a_5X^5 + a_4X^4 + a_3X^3 + a_2X^2 + a_1X + a_0 \\
&+ b_7X^7 + b_6X^6 + b_5X^5 + b_4X^4 + b_3X^3 + b_2X^2 + b_1X + b_0 \\
&= (a_7 \oplus b_7)X^7 + (a_6 \oplus b_6)X^6 + (a_5 \oplus b_5)X^5 + (a_4 \oplus b_4)X^4 \\
&+ (a_3 \oplus b_3)X^3 + (a_2 \oplus b_2)X^2 + (a_1 \oplus b_1)X + (a_0 \oplus b_0).
\end{aligned}$$

Le résultat de l'opération d'addition est donc simplement le XOR entre les bits correspondants dans les deux représentations binaires :

$$(a_7 \oplus b_7, a_6 \oplus b_6, a_5 \oplus b_5, a_4 \oplus b_4, a_3 \oplus b_3, a_2 \oplus b_2, a_1 \oplus b_1, a_0 \oplus b_0).$$

Multiplication

La multiplication est une opération légèrement plus complexe. Lorsque l'on multiplie deux polynômes de degré 7, le degré du polynôme résultant sera potentiellement plus élevé que ceux des polynômes d'entrée. Pour cette raison, nous devons réduire le polynôme produit à un degré d'au plus 7 à l'aide d'un polynôme irréductible R_p. Pour l'AES, ce polynôme est :

$$R_p = X^8 + X^4 + X^3 + X + 1,$$

et est mieux connu sous le nom de *polynôme de Rijndael*.

Multiplication de deux octets. Supposons que nous voulons multiplier les polynômes $X^5 + X^3 + X + 1$ et $X^3 + X^2$, correspondant aux valeurs d'octet `0x2b` et `0x0c` respectivement. Le

calcul se fait comme suit :

$$\begin{aligned}
&(X^5 + X^3 + X + 1)(X^3 + X^2) \\
&= X^8 + X^7 + X^6 + X^5 + X^4 + X^3 + X^3 + X^2 \\
&= (X^4 + X^3 + X + 1) + X^7 + X^6 + X^5 + X^4 + X^2 \mod R_p \\
&= X^7 + X^6 + X^5 + X^3 + X^2 + X + 1,
\end{aligned}$$

et ce résultat correspond à la valeur d'octet `0xef`. Comme on peut le voir, le monôme X^8 dans la deuxième ligne est remplacé par $X^4 + X^3 + X + 1$, car $X^8 \equiv X^4 + X^3 + X + 1 \mod R_p$.

5.2 Fonction de tour de l'AES

L'AES [5] est un réseau de substitution-permutation (SPN) qui travaille sur des blocs de 128 bits et utilise des clés de 128, 192, ou 256 bits. L'état interne est traité comme une matrice d'octets $(s_{i,j})$, $0 \leq i, j < 4$, de taille 4×4, où chaque octet représente une valeur dans $\mathbb{F}_{2^8}$:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

FIGURE 6 – Représentation de l'état de l'AES

Un tour de l'AES applique les quatre opérations suivantes à cette matrice d'état : `SubBytes (SB)`, `ShiftRows (SR)`, `MixColumns (MC)` et `AddRoundKey (ARK)`.

5.2.1 `SubBytes (SB)`

Cette opération applique la même boîte-S inversible de 8 bits à 8 bits 16 fois en parallèle sur chaque octet de l'état. Cette boîte-S est basée sur l'opération d'inversion sur le corps $\mathbb{F}_{2^8}$ et est inspirée par les travaux de Knudsen et Nyberg sur la preuve de sécurité contre la cryptanalyse différentielle et linéaire [14]. Plus précisément, pour un octet d'entrée x,

$$S(x) = L \cdot x^{(-1)} + c,$$

où L est la matrice 8×8 sur $\mathbb{F}_2$

$$L = \begin{pmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{pmatrix},$$

c est la constante `01100011 = 0x63`, et $x^{(-1)}$ est l'opération d'inversion modifiée dans $\mathbb{F}_{2^8}$, correspondant à l'inversion multiplicative pour tout $x \neq 0$ et où $x^{-1}(0) \triangleq 0$. La représentation

en table de cette boîte-S est donnée dans la table 6. Par exemple, l'image de `0x1a` par la boîte-S est `0xa2`, ce qui correspond à la valeur située à l'intersection de la ligne `0x10` et de la colonne `0x0a`.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

TABLE 6 – La boîte-S de l'AES

La boîte-S de l'AES a été conçue de manière à renforcer la sécurité du chiffrement face aux attaques, notamment la cryptanalyse différentielle et linéaire. Les principaux choix faits par les concepteurs sont donnés ci-dessous :

Inversion dans le corps fini $\mathbb{F}_{2^8}$ La fonction inverse dans le corps fini $\mathbb{F}_{2^8}$ garantit la meilleure résistance connue possible pour une permutation des mots de 8 bits contre la cryptanalyse différentielle et linéaire [12]. Associée à une couche de diffusion soigneusement conçue, cette propriété fournit une excellente sécurité contre ces types d'attaques. Les fondements théoriques de l'utilisation de telles opérations algébriques dans les constructions de chiffrements par bloc ont été établis dans des travaux antérieurs, ceux notamment de Kaisa Nyberg [12, 13, 14].

Ajout d'une transformation linéaire L'inversion seule dans $\mathbb{F}_{2^8}$ est une opération simple mais est une opération avec des fortes propriétés algébriques. Par exemple, si $y = x^{-1}$ pour des éléments $x, y \in \mathbb{F}_{2^8}$, alors $y \cdot x = 1$, sauf si $x = y = 0$. Pour compliquer l'exploitation de cette propriété par un adversaire, les concepteurs de l'AES ont décidé d'ajouter L définie sur $\mathbb{F}_2$. Cette opération conserve les bonnes propriétés différentielles et linéaires de la fonction inverse mais rend plus difficile l'exploitant des propriétés mathématiques fortes comme celle donnée ci-dessus.

Ajout d'une constante L'ajout de la constante `0x63` vise à éviter que l'entrée $S(0) = 0$, ce qui créerait un *point fixe*. Bien que l'impact de cette propriété sur la sécurité globale de l'AES soit incertain, les concepteurs ont jugé préférable d'éliminer tous les points fixes.

5.2.2 `ShiftRows (SR)`

Cette opération décale cycliquement la i-ème ligne de i octets vers la gauche, où 0 est la première ligne en haut et 3 est la ligne en bas, comme illustré dans la figure 7.

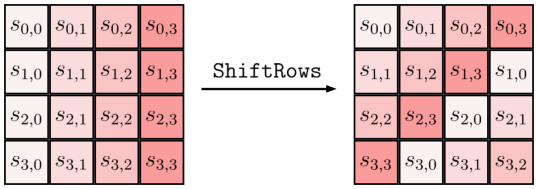


FIGURE 7 – L'opération `ShiftRows`

5.2.3 `MixColumns (MC)`

Cette opération consiste en une multiplication de chaque colonne par une matrice constante 4×4 dans le corps $\mathbb{F}_{2^8}$. Cette matrice est généralement écrite comme suit :

$$M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix},$$

où 1, 2 et 3 représentent des éléments du corps fini $\mathbb{F}_{2^8}$, exprimés en notation hexadécimale. Rappelons que, conformément à la représentation donnée en section 5.1, ces éléments correspondent respectivement aux polynômes 1, X et $X + 1$.

Exemple de calcul On souhaite calculer le produit $02 \times a_3$ avec $a_3 = X^7 + X^5 + X + 1$, qui ferait partie d'un calcul complet d'un octet en utilisant `MixColumns`.

$$\begin{aligned} 02 \times a_3 &= X \times (X^7 + X^5 + X + 1) = X^8 + X^6 + X^2 + X \\ &= (R_p + X^4 + X^3 + X + 1) + X^6 + X^2 + X \\ &= X^6 + X^4 + X^3 + X^2 + 1 \mod R_p \\ &= 5d. \end{aligned}$$

Ainsi, $02 \times a_3 = 5d$.

Matrices MDS La matrice M a été choisie pour ses bonnes propriétés de diffusion. Elle fait partie des matrices appelées *MDS*, de l'acronyme anglais "Maximum Distance Separable", qui assurent une diffusion *parfaite*. Pour une fonction de diffusion L agissant sur des octets, nous souhaitons de façon générale que même si très peu d'octets sont modifiés en entrée, le nombre d'octets affectés en sortie soit grand. Autrement dit, si $w(x)$, avec x un vecteur d'octets, symbolise le nombre d'octets non-nuls, nous voulons que la quantité suivante :

$$B_d(L) = \min_{x \neq 0} \left(w(x) + w(L(x)) \right),$$

appelée en anglais *branch number différentiel* [19, 15], soit la plus élevée possible. Les matrices représentant une fonction L pour laquelle la borne supérieure est atteinte sont appelées *matrices MDS*. En effet, le branch number de L correspond à la distance minimale du code correcteur linéaire composé des mots $(x||L(x))$. Si L est une fonction de $\mathbb{F}_q^n$ vers $\mathbb{F}_q^n$, cette distance atteint son maximum, $n + 1$, pour les codes dits MDS. Pour une fonction s'appliquant sur une colonne de 4 octets à la fois, comme c'est le cas de `MixColumns`, ce nombre peut être au plus 5, car si un seul octet en entrée est modifié, il peut influencer au plus les 4 octets de sa colonne. La matrice M de `MixColumns` est choisie de façon que cette borne maximale soit atteinte.

Si la matrice est MDS cela signifie en pratique que changer la valeur de u mots d'entrée affecte au moins $n + 1 - u$ mots de sortie.

Nous allons voir avec un exemple les conséquences de cette propriété sur la qualité de diffusion dans le cas de l'AES. Supposons que nous ayons huit octets $x_0, x_1, x_2, x_3, y_0, y_1, y_2$, et y_3 tels que

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}.$$

Alors, soit tous les x_i et y_i sont nuls ($0 \leq i \leq 3$), soit il y a au moins 5 valeurs non nulles parmi les x_i et y_i.

Cela peut ne pas sembler immédiatement important, mais nous pouvons au moins justifier l'utilisation d'une matrice MDS en termes d'« effet avalanche ». Supposons que nous ayons deux entrées pour la matrice M, données par

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} x'_0 \\ x'_1 \\ x'_2 \\ x'_3 \end{pmatrix},$$

et supposons que ces deux entrées soient très similaires, de sorte que $x_1 = x'_1$, $x_2 = x'_2$ et $x_3 = x'_3$, mais $x_0 \neq x'_0$.

Puisque la multiplication par M est une opération linéaire sur GF(2), nous pouvons écrire

$$\begin{pmatrix} y_0 + y'_0 \\ y_1 + y'_1 \\ y_2 + y'_2 \\ y_3 + y'_3 \end{pmatrix} = M \begin{pmatrix} x_0 + x'_0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Étant donné que le branch number est 5, cela signifie que toutes les valeurs $y_0 + y'_0$, $y_1 + y'_1$, $y_2 + y'_2$ et $y_3 + y'_3$ doivent être non nulles, ce qui implique que les deux sorties générées à partir d'entrées très proches doivent différer dans chaque octet. Même dans le cas légèrement plus complexe de deux octets d'entrée non nuls, nous pouvons être sûrs qu'au moins trois des octets de sortie doivent être non nuls. Ainsi, lorsque deux entrées de la couche `MixColumns` sont très similaires en termes de différence au niveau binaire, les sorties seront très différentes, ce qui aide à amplifier même une petite variation ou différence au fil des tours du chiffrement.

Ensemble, les opérations `ShiftRows` et `MixColumns` garantissent que le nombre de ce qu'on appelle des *boîtes-S actives* est soit grand dès le départ, soit devient rapidement grand. Cela constitue la base de la stratégie *wild trail*, utilisée par les concepteurs de l'AES pour garantir la sécurité de l'AES contre la cryptanalyse différentielle et linéaire.

5.2.4 `AddRoundKey (ARK)`

Lors de cette dernière opération, l'état est XORé avec une sous-clé de 128 bits produite par l'algorithme de génération de clés.

Une opération supplémentaire `AddRoundKey` est appliquée avant le premier tour (sinon, le premier tour pourrait être facilement calculée sans connaître la clé secrète). Dans le dernier tour, l'opération `MixColumns` est omise (son impact sur la sécurité est négligeable).

Le nombre de tours Nr dans l'AES dépend de la longueur de la clé : 10 tours pour les clés de 128 bits, 12 tours pour les clés de 192 bits, et 14 tours pour les clés de 256 bits. Les tours sont numérotées de 0 à $Nr - 1$.

Un exemple de calcul pour un tour de l'AES peut être visualisé dans la figure 8.

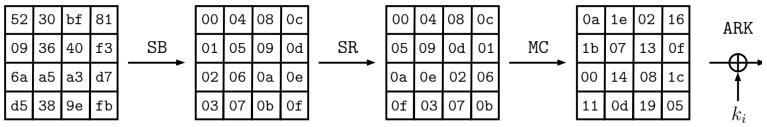


FIGURE 8 – Exemple de calcul d'un tour de l'AES

5.3 Génération de clés de l'AES

La génération de clés de l'AES transforme la clé de Nk mots [1] en $Nr + 1$ sous-clés de 128 bits. Nous désignons le tableau de sous-clés par $W[0, \ldots, 4 \cdot Nr + 3]$, où chaque mot de $W[\cdot]$ consiste en 32 bits. La longueur de la clé est de Nk mots de 32 bits, la clé maître est recopiée dans les Nk premiers mots de $W[\cdot]$, et les mots restants de $W[\cdot]$ sont mis à jour selon la règle suivante :

— Pour $i = Nk, \ldots, 4 \cdot Nr + 3$, faire
  — Si $i \equiv 0 \bmod Nk$ alors $W[i] = W[i - Nk] \oplus \texttt{SB}(W[i-1] \lll 8) \oplus RCON[i/Nk]$,
  — sinon si $Nk = 8$ et $i \equiv 4 \bmod 8$ alors $W[i] = W[i-8] \oplus \texttt{SB}(W[i-1])$,
  — Sinon $W[i] = W[i-1] \oplus W[i - Nk]$,

où $\lll$ désigne une rotation du mot de 8 bits vers la gauche, et $RCON[\cdot]$ est un tableau de constantes de 32 bits, donné sous la forme $RCON[i] = [rc_i, \texttt{00}, \texttt{00}, \texttt{00}]$, où rc_i est un octet dont la valeur en notation hexadécimale est donnée dans la table 7.

i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1b	36

TABLE 7 – Les constantes rc_i, pour $i = 1, \ldots, 10$.

La génération de clés de l'AES-128 est illustrée graphiquement dans la figure 9, tandis que celles de l'AES-192 et de l'AES-256 sont représentées respectivement dans les figures 10 et 11.

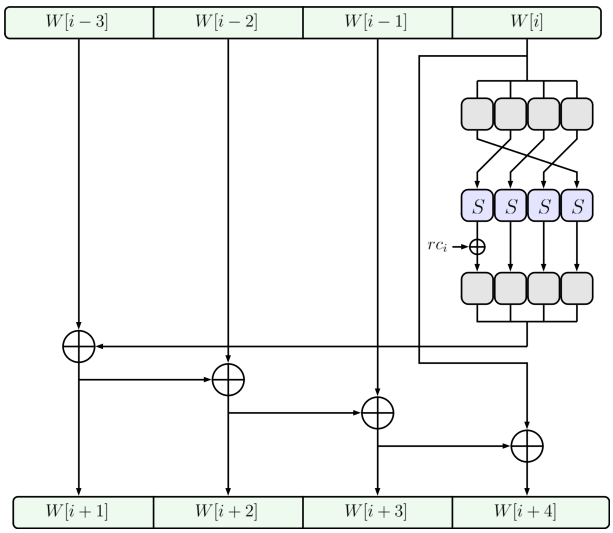


FIGURE 9 – La génération de clés de l'AES-128

1. Nous adoptons la description originale de l'AES, bien qu'elle puisse prêter à confusion. La génération de clés traite la clé comme Nk mots de 32 bits chacun, soit $Nk = 4$ pour l'AES avec des clés de 128 bits, $Nk = 6$ pour les clés de 192 bits, et $Nk = 8$ pour les clés de 256 bits.

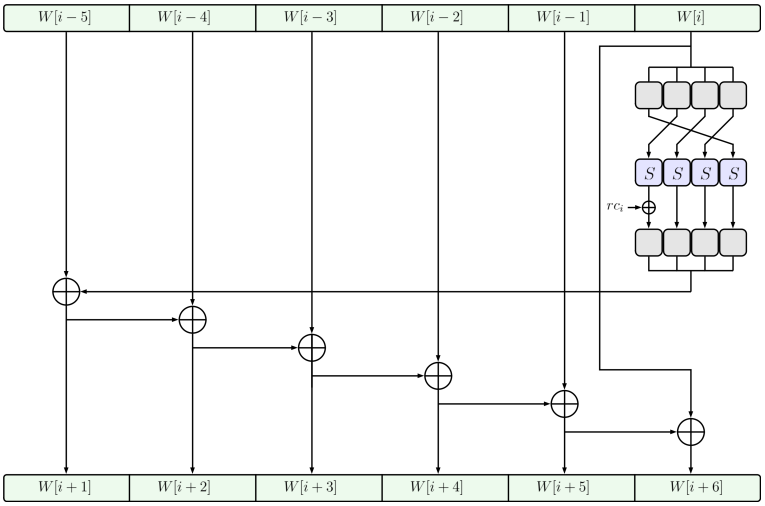


FIGURE 10 – La génération de clés de l'AES-192

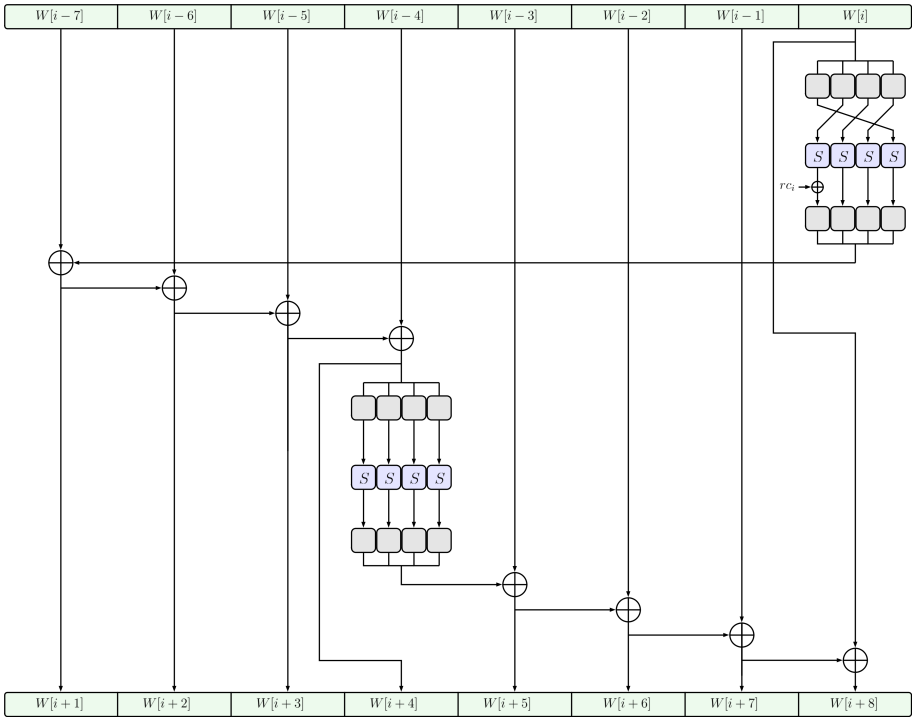


FIGURE 11 – La génération de clés de l'AES-256

6 Cryptanalyse des chiffrements par bloc

Pour utiliser un chiffrement par bloc en pratique, il est indispensable de garantir sa sécurité. Cela signifie qu'il ne doit exister aucune attaque contre ce chiffrement ayant une complexité inférieure à celle des meilleures attaques génériques. Un chiffrement par bloc E, opérant sur des blocs de taille n bits et utilisant des clés de taille κ bits, est toujours vulnérable aux deux attaques génériques suivantes :

Recherche exhaustive de la clé Si un attaquant dispose d'un couple clair-chiffré (m, c), où c est le bloc chiffré obtenu en chiffrant m à l'aide d'une clé inconnue k, il peut tester toutes les clés possibles $\hat{k}$. Pour cela, il chiffre m en utilisant E paramétré avec la clé $\hat{k}$ et vérifie si

$E_{\hat{k}}(m) = c$. Cette attaque, qui ne nécessite pas de mémoire, a une complexité temporelle de 2^κ dans le pire des cas, correspondant au nombre total de clés à tester.

Attaque par dictionnaire Un attaquant ayant un accès illimité à la machine chiffrante peut construire un dictionnaire contenant les 2^n messages possibles et leurs chiffrés correspondants pour une clé inconnue. Ce dictionnaire lui permet ensuite de déchiffrer instantanément n'importe quel message chiffré en recherchant simplement le chiffré dans la table préconstruite. Cette attaque nécessite 2^n opérations pour générer le dictionnaire et une quantité significative de mémoire pour le stocker. Toutefois, si ces contraintes sont satisfaites, la complexité temporelle de l'attaque "en ligne" devient négligeable, se limitant à une recherche rapide dans le dictionnaire.

L'existence de ces deux attaques génériques impose des bornes minimales sur la taille des blocs et des clés, afin que les capacités de calcul et de stockage des ordinateurs actuels rendent ces attaques impossibles à mettre en œuvre.

6.1 Attaques exploitant la structure interne de la primitive

L'objectif des cryptanalystes est d'étudier en détail la structure interne des chiffrements par bloc (et autres primitives symétriques) afin de détecter des comportement “anormaux”. En effet, il est possible d'obtenir de l'information sur la clé secrète si les permutations du chiffrement par bloc ne se comportent pas comme des permutations aléatoires.

Un *distingueur* pour une famille de permutations $P = (P_k)_{k\in\mathbb{F}_2^\kappa}$ est un algorithme qui, ayant accès en boîte noire à une permutation (*c.-à-d.* uniquement aux entrées et aux sorties), doit décider si celle-ci appartient à la famille $(P_k)_{k\in\mathbb{F}_2^\kappa}$ ou si elle a été choisie aléatoirement parmi toutes les permutations possibles.

Pour garantir la sécurité, il ne doit exister aucun distingueur capable, sur un grand nombre de tours consécutifs du chiffrement, d'utiliser peu de ressources de calcul tout en maintenant une faible probabilité d'erreur.

6.2 Attaques sur le dernier tour

Nous décrivons ici une approche classique pour exploiter un tel distingueur afin d'obtenir des informations sur la clé secrète $\hat{k}$ d'une instance $E_{\hat{k}}$ d'un chiffrement par bloc $E = (E_k)_{k\in\mathbb{F}_2^\kappa}$.

Pour simplifier, supposons que ce distingueur couvre l'ensemble des tours, sauf le dernier. Nous notons F_{k_i} les fonctions associées aux différents tours du chiffrement, et l'instance du chiffrement s'écrit alors :

$$E_k = F_{k_r} \circ \cdots \circ F_{k_1}, \quad \text{pour tout } k \in \mathbb{F}_2^\kappa.$$

L'attaquant peut alors considérer la famille de permutations $(G_k)_{k\in\mathbb{F}_2^\kappa}$ avec :

$$G_k = F_{k_{r-1}} \circ \cdots \circ F_{k_1},$$

et utiliser le distingueur sur cette famille pour valider ou invalider des hypothèses concernant la dernière sous-clé de tour $\hat{k}_r$. L'idée générale est de formuler une hypothèse k' pour la valeur de la sous-clé $\hat{k}_r$ et d'appliquer le distingueur sur la permutation suivante :

$$H_{k'} = F_{k'}^{-1} \circ E_{\hat{k}}.$$

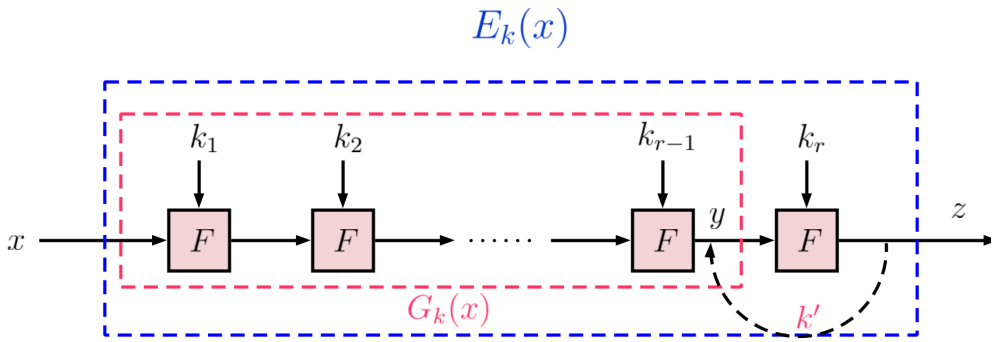


FIGURE 12 – Principes des attaques sur le dernier tour

Si l'hypothèse $k' = \hat{k}_r$ est correcte, $H_{k'}$ est la permutation $G_{\hat{k}}$ de la famille $(G_k)_{k\in\mathbb{F}_2^\kappa}$. Sinon, la fonction $H_{k'}$ est supposée avoir un comportement similaire à celui d'une permutation aléatoire. Appliquer le distingueur sur $H_{k'}$ permet donc de valider ou d'invalider (avec une certaine probabilité) l'hypothèse k'.

Tester directement toutes les valeurs possibles pour $\hat{k}_r$ reviendrait à effectuer une recherche exhaustive sur la dernière sous-clé , ce qui n'est généralement pas intéressant. Toutefois, comme une fonction de tour est souvent cryptographiquement faible, il est possible de restreindre les candidats en fixant seulement certains bits de k' et en calculant partiellement la fonction $F_{k'_r}^{-1}$ pour chaque clé candidate. Le distingueur peut alors permettre de déduire progressivement des informations sur $\hat{k}_r$, morceau par morceau. Cette méthode fournit ainsi des informations sur la clé maître $\hat{k}$.

Concrètement, pour réaliser une attaque sur le dernier tour, l'attaquant doit se procurer un ensemble de N paires entrée/sortie $\{(x_i, z_i)\}_{i\in[1,N]}$ du chiffrement $E_{\hat{k}}$. La taille de cet ensemble dépend du type de l'attaque et l'attaquant peut également imposer certaines propriétés aux éléments qu'il collecte. Ensuite, l'attaquant teste chaque candidat partiel k' : si les paires (partielles) de l'ensemble

$$\{(x_i, y_i) \mid i \in [1, N],\ y_i = F_{k'}^{-1}(z_i)\},$$

sont identifiées par le distingueur comme des paires entrée/sortie d'une permutation de la famille $(G_k)_{k\in\mathbb{F}_2^\kappa}$, alors le candidat est retenu.

L'attaque sur le dernier tour inclut $r-1$ tours durant lesquels le distingueur est appliqué, suivis d'un dernier tour appelé *recouvrement de clé*. Cette attaque peut être généralisée en intégrant des tours supplémentaires, avec un certain nombre de tours initiaux r_{init} et finaux r_{fin} dédiés au recouvrement de clé. Cette approche est fréquemment utilisée en pratique. Le choix des paramètres r_{init} et r_{fin} dépend de nombreux critères, notamment du fonctionnement du distingueur.

7 Sources

1. Christina Boura, *Analyse de Fonctions de Hachage Cryptographiques*, Thèse de Doctorat, 2012. `https://christinaboura.wordpress.com/wp-content/uploads/2019/11/boura-these-2012.pdf`
2. Christina Boura and María Naya Plasencia, *Symmetric Cryptography, Volume 1 : Design and Security Proofs*, Wiley-ISTE, 2023.
3. Christina Boura and María Naya Plasencia, *Symmetric Cryptography, Volume 2 : Cryptanalysis and Future Directions*, Wiley-ISTE, 2023.
4. Anne Canteaut, *Lecture Notes on Symmetric Cryptography*.

5. Lars Knudsen and Matthew J. Robshaw, *The Block Cipher Companion*, Springer, 2011.

Références

[1] American National Standards Institute (ANSI). Financial Institution Key Management (Wholesale), 1985. Withdrawn standard, but historically significant for DES applications in finance.

[2] Elaine Barker and William Barker. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. Technical Report SP 800-67, National Institute of Standards and Technology (NIST), January 2012. Revised Version 2.

[3] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[4] Andrey Bogdanov and Philip S. Vejre. Linear cryptanalysis of DES with asymmetries. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 187–216. Springer, 2017.

[5] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[6] Whitfield Diffie and Martin Hellman. Special feature exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6) :74–84, 1977.

[7] Pascal Junod. On the complexity of matsui's attack. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 199–211. Springer, 2001.

[8] Lars R. Knudsen and Matthew J. B. Robshaw. *The Block Cipher Companion.* Springer Publishing Company, Incorporated, 2011.

[9] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[10] Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Yvo Desmedt, editor, *CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1994.

[11] National Institute of Standards and Technology. Data encryption standard. Federal Information Processing Standard (FIPS), Publication 46. `https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf`, January 1977.

[12] Kaisa Nyberg. Perfect nonlinear s-boxes. In *EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1991.

[13] Kaisa Nyberg. S-boxes and Round Functions with Controllable Linearity and Differential Uniformity. In Bart Preneel, editor, *FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 111–130, 1995.

[14] Kaisa Nyberg and Lars R. Knudsen. Provable Security Against a Differential Attack. *J. Cryptology*, 8(1) :27–37, 1995.

[15] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher SHARK. In Dieter Gollmann, editor, *FSE 1996*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 1996.

[16] Bruce Schneier and John Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In Dieter Gollmann, editor, *FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996.

[17] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4) :656–715, 1949.

[18] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In Joan Feigenbaum, editor, *CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 172–181. Springer, 1991.

[19] Serge Vaudenay. On the need for multipermutations : Cryptanalysis of MD4 and SAFER. In Bart Preneel, editor, *FSE 1994*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1994.

[20] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 461–480. Springer, 1990.