TEST INTEGRE DE CIRCUITS CRYPTOGRAPHIQUES

Marion Doulcier

UNIVERSITE MONTPELLIER II
SCIENCES ET TECHNIQUES DU LANGUEDOC

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE MONTPELLIER II

Discipline : Génie Informatique, Automatique et Traitement du Signal
Formation Doctorale : Systèmes Automatiques et Micro-électronique
Ecole Doctorale : Information, Structures, Systèmes

Test intégré de circuits cryptographiques

Par

Marion DOULCIER

soutenue publiquement le 24 Novembre 2008 devant le jury composé de :

Mme FLOTTES	Chargé de recherche CNRS, Montpellier	Co-encadrante
M. LEVEUGLE	Professeur, Grenoble INP	Rapporteur
M. ROUZEYRE	Professeur, Université Montpellier II	Directeur de Thèse
M. SENTIEYS	Professeur, Université de Rennes I	Rapporteur
M. TORRES	Professeur, Université Montpellier II	Président du jury
Mme TRIA	Chargé de recherche CEA, ENSMSE	Examinatrice

Remerciements

Les travaux présentés dans ce manuscrit ont été réalisés au sein du Département de Microélectronique du LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier) dirigé par Monsieur Michel ROBERT.

Je remercie vivement Monsieur Régis LEVEUGLE, Professeur à Grenoble INP, et Monsieur Olivier SENTIEYS, Professeur à l'Université de Rennes I, pour l'honneur qu'ils m'ont fait et pour l'intérêt qu'ils ont porté à mes travaux en acceptant d'être les rapporteurs de ce mémoire de thèse. Je voudrais aussi remercier cordialement Madame Assia TRIA, Chargé de recherche au CEA, ainsi que Monsieur Lionel TORRES, Professeur à l'Université Montpellier II, d'avoir accepté d'être membres du jury de thèse.

Je remercie mes encadrants, Bruno ROUZEYRE et Marie Lise FLOTTES, pour leur aide, l'expérience dont ils m'ont fait profiter et la confiance qu'ils m'ont accordée tout au long du développement des recherches. Et je tiens également à remercier Giorgio DI NATALE, Chargé de recherche CNRS, pour avoir participé activement aux travaux de recherche.

Je voudrais aussi saluer mes collègues thésards et toute les personnes du laboratoire que j'ai côtoyées avec plaisir. Parmi les thésards : Alexandre, Amine, Anissa, Bettina, Boris, Elmehdi, Jean-Etienne, Julien, Lionel, Nicolas (Houarche, Pous et Saint-Jean), Olivier (Brousse et Leman), Victor, Yoann, Eric. Hors thésards, je voudrais remercier Caroline, Cécile, Céline, Elisabeth, Isabelle, Louise, Martine, Nadine, Nicole, Philippe et Stéphanie.

Ensuite mes remerciements vont à mes proches, membres de ma famille ou amis, qui m'ont toujours soutenu et encouragé tout au long de mes études. Et enfin, un immense merci à Ghislain, pour son soutien réconfortant, sa présence attentionnée et sa compréhension.

TABLE DES MATIERES

Introduction Générale

Introduction générale

Les circuits intégrés sécurisés occupent une place de plus en plus importante dans notre quotidien. Ces circuits ont pour vocation de stocker les données dites sensibles ou de les transmettre sur des réseaux non sécurisés (tels que l'Internet). Parmi leurs diverses applications, on peut citer la téléphonie, la télévision à péage, la sécurisation des accès, les cartes de paiement, la carte vitale, le passeport électronique...

Afin de garantir la sécurité des données, généralement des modules cryptographiques sont implantés dans ces circuits. Ces modules ont pour objectif de rendre les données sensibles inintelligibles sauf à leur propriétaire ou à leur destinataire.

Une des étapes essentielles de la conception d'un circuit intégré, et plus particulièrement d'un circuit sécurisé est l'étape de test. Le principe du test est d'appliquer des stimuli au circuit et d'observer les réponses obtenues. Si ces réponses sont identiques à celles attendues alors le circuit est dit « sain » autrement le circuit est défectueux. De plus, dans le contexte des circuits sécurisés, le test est primordial pour garantir la fonctionnalité du circuit, et par la même s'assurer qu'il n'y ait pas de défauts pouvant engendrer une faille sécuritaire, pouvant être exploitée par un éventuel attaquant.

Pour améliorer la qualité du test, des techniques de conception en vue du test sont utilisées. Ces techniques de conception ont pour rôle d'augmenter la testabilité du circuit intégré. Les deux principes de base de la testabilité sont la contrôlabilité (pour appliquer les stimuli de test) et l'observabilité (pour analyser les réponses aux stimuli). Ces deux principes sont opposés à ceux de la sécurité qui tendent à réduire l'accès aux données internes au minimum. Par conséquent, le test des circuits sécurisés nécessite une attention toute particulière afin de ne pas engendrer de faille sécuritaire.

La technique de « scan path » est la technique de conception en vue du test la plus utilisée dans l'industrie et son automatisation est assurée par la plupart des outils. L'association de cette technique avec une approche de test externe fournit de bon taux de couverture (c'est-à-dire que le rapport nombre de fautes testées sur nombre total de fautes dans le circuit est élevé) et facilite l'application des séquences de test. Par contre, cette technique fragilise la sécurité du circuit en permettant de retrouver les données sensibles [Yan04] [Yan05]. L'inconvénient majeur du test externe, dans le contexte des circuits sécurisés, est de fournir un accès depuis l'extérieur aux données traitées en interne. Par conséquent, les techniques de test intégré BIST (« Built-In Self Test »), ne fournissant pas d'accès depuis l'extérieur aux données traitées en interne, semblent plus appropriée à ces circuits.

A l'inverse du test externe, en test intégré (BIST) les ressources de test (générateur de vecteurs de test, analyseur de réponses, logique de contrôle) sont directement implantées dans le circuit. Par conséquent, cette approche de test induit un coût en surface non négligeable. Or, l'augmentation de la surface accentue la probabilité d'apparition d'un défaut puisque la présence de défauts augmente avec la surface de silicium utilisée. Par conséquent, une telle approche n'est viable que si la surface additionnelle est très faible par rapport au circuit sous test.

L'objectif de cette thèse est de proposer une solution de test intégré pour les circuits sécurisés permettant de réduire le coût en surface requis par l'implantation des ressources de test. Les solutions proposées dans ce manuscrit consistent à modifier un des cœurs de chiffrement déjà implantés dans le circuit de sorte à l'utiliser soit comme générateur de vecteurs de test soit comme analyseur de réponses.

Les deux premiers chapitres de ce manuscrit présentent une étude bibliographique sur les circuits sécurisés et le test des circuits intégrés. Le premier chapitre présente les circuits sécurisés et plus particulièrement les cœurs de chiffrement qui sont implantés pour garantir la sécurité des données. Les attaques matérielles auxquelles sont sujets les cœurs de chiffrement seront aussi exposées ainsi que les contre-mesures associées à ces attaques. Le deuxième chapitre présente les différentes approches de test des circuits intégrés ainsi que la difficulté de tester les circuits sécurisés sans affaiblir leur niveau de sécurité.

Le contexte de l'étude étant posé, le troisième et le quatrième chapitres exposent deux solutions permettant de réduire le coût d'intégration d'une approche de test intégré, respectivement en utilisant le cœur de chiffrement comme générateur de vecteurs de test ou comme compacteur de réponses. Pour utiliser un cœur de chiffrement comme générateur de vecteurs de test, il faut s'assurer que les données générées soient suffisamment aléatoires. Pour utiliser le cœur en tant que compacteur de réponses, il faut s'assurer que la probabilité de masquage de l'erreur soit négligeable. Le troisième chapitre présente une étude comparative entre les propriétés aléatoires d'une séquence générée à l'aide du cœur de chiffrement et d'un générateur caractéristique du test intégré pseudo-aléatoire. Les résultats de cette étude montrent que la séquence générée à l'aide du cœur de chiffrement a des propriétés aléatoires comparables à celle du générateur usuel permettant d'utiliser le cœur comme ressource de test. Le quatrième chapitre présente dans un premier temps une étude bibliographique sur les compacteurs de réponses. Dans un deuxième temps, une étude sur la probabilité de masquage d'un compacteur de réponses basé sur un cœur de chiffrement est réalisée. Cette étude montre que le cœur de chiffrement peut être utilisé comme compacteur de réponses avec la même efficacité qu'un compacteur généralement utilisé en test.

Le cinquième chapitre est divisé en deux parties. La première est consacrée à l'autotest des cœurs de chiffrement. Cette propriété d'autotest est essentielle car si le cœur peut être utilisé comme ressources de test pour les autres cœurs, il ne faut pas que son propre test induise un surcoût matériel. La deuxième partie est dédiée au coût d'intégration des différentes solutions : génération de vecteurs de test, compacteur de réponses et autotest.

Le sixième chapitre est dédié à une solution de test en ligne. Ce type de test est couramment utilisé pour s'assurer de la fiabilité d'un dispositif en vérifiant la validité des données traitées en cours de fonctionnement. Dans un premier temps, une étude bibliographique sur les solutions de test en ligne dédiées au cœur de chiffrement AES est présentée et dans un deuxième temps une solution est proposée. Cette solution est basée sur le principe de la redondance matérielle.

Chapitre 1 : Les circuits sécurisés

Chapitre 1 : LES CIRCUITS SECURISES

Les travaux menés lors de cette thèse se situent dans le contexte des circuits sécurisés. Ces circuits ont de nombreuses applications dans notre quotidien : carte bancaire, carte téléphonique… De façon générale pour garantir la sécurité des données, ces circuits intègrent un ou plusieurs blocs dédiés aux cryptages des informations appelés crypto-processeurs.

Ce chapitre met en exergue les contraintes de conception liées aux circuits sécurisés.

Dans une première partie, nous présenterons l'architecture type de ces circuits et le principe de la cryptologie afin de mieux comprendre le rôle des crypto-processeurs. Les deux principaux algorithmes de chiffrement symétrique, le DES (Data Encryption Standard) et l'AES (Advanced Encryption Standard), seront détaillés.

Les domaines d'utilisations de ces circuits, impliquent qu'ils doivent être conçus de sorte à résister à toutes sortes d'agressions et d'attaques. Dans la deuxième partie, les attaques les plus connues et les contre-mesures qui leurs sont associées seront présentées. Dans la troisième partie nous présenterons les niveaux permettant de qualifier la sécurité des circuits.

1.1 CIRCUIT ET ALGORITHME

1.1.1 LES CARTES A PUCE

Les cartes à puce peuvent être classées en deux grandes familles. La première, qui est aussi la plus ancienne, est la famille des cartes à mémoire. Les cartes à mémoire sont principalement composées de mémoire non volatile. L'utilisation qui fît connaître la carte à mémoire au grand public est la carte téléphonique France Telecom au début de l'année 1983. Cette première carte consistait simplement à décrémenter le nombre d'unités restantes, chaque unité représentant un certain temps de communication.

La seconde famille est celle des cartes à microprocesseur. Ces cartes sont désignées aujourd'hui sous le vocable unique de cartes à puce, ou smart cards, alors que les précédentes se voient appelées cartes à mémoire ou memory cards.

La carte à puce est omniprésente dans notre quotidien, ces applications sont multiples : cartes Vitale, cartes SIM des téléphones portables, carte de gestion de la télévision à péage, cartes de paiement, de sécurisation des accès, passeports électronique… Ces diverses applications imposent des niveaux de sécurité plus ou moins élevés. Par conséquent, lors de la conception de ces circuits de nombreux modules dédiés à la sécurité seront intégrés.

Les principaux modules d'une carte à puce sont :

- Un microprocesseur de 8, 16 ou 32 bits suivant l'application ciblée,
- La ROM (Read Only Memory). Les données contenues dans cette mémoire sont conservées même lorsque la carte n'est pas alimentée. Cette mémoire sert généralement à stocker le système d'exploitation ainsi que diverses applications.
- L'EEPROM (Electrical Erasable Programable Read Only Memory) ou la Flash. Elle sert à contenir, entre autre, les données spécifiques au propriétaire de la carte. Comme pour la ROM, les données stockées en EEPROM ou en Flash sont conservées quand la carte n'est pas alimentée.
- La RAM (Random Access Memory). Elle est considérée comme l'espace de travail utilisé par les applications pour manipuler leurs données. La RAM est une mémoire volatile, c'est-à-dire que les données qui y sont stockées sont effacées dès que la carte n'est plus alimentée.
- Les crypto-processeurs. Ils sont dédiés aux opérations cryptographiques symétriques et asymétriques (RSA et AES par exemple) (cf. partie 1.1.2).
- Un générateur de nombre aléatoire. Les nombres aléatoires peuvent être utilisés, par exemple, en cryptographie pour la génération de clefs de chiffrement.
- Différents détecteurs (intrusion, lumière, etc...) visant à signaler une attaque.

Les données entre ces différents blocs transitent par un bus. La figure 1-1 illustre l'architecture générale d'une carte à microprocesseur.

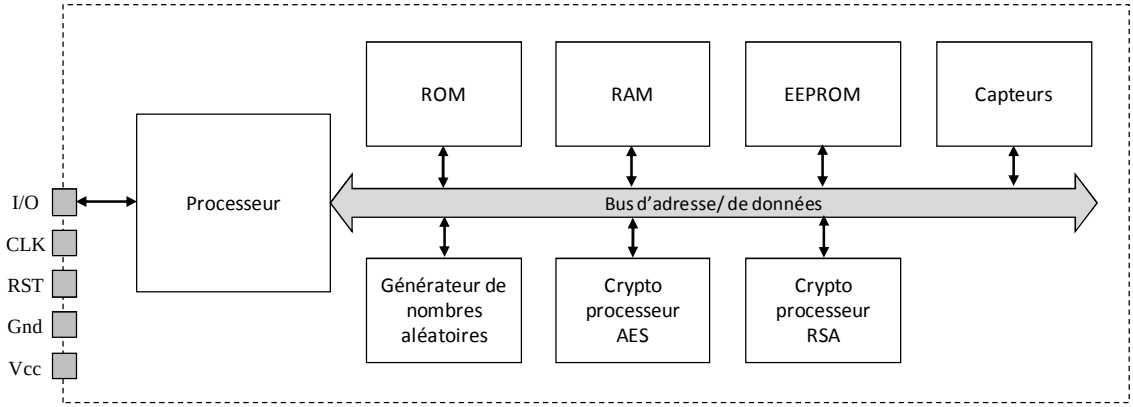


FIGURE 1-1 : ARCHITECTURE D'UNE CARTE A MICROPROCESSEUR

1.1.2 PRINCIPE DE LA CRYPTOLOGIE

La cryptologie appelée aussi la science du secret est une des branches des mathématiques. Elle se scinde à son tour en 2 branches :

- la cryptographie qui consiste à préserver le secret du message
- la cryptanalyse qui consiste à « casser » le secret.

La cryptographie regroupe les techniques permettant de chiffrer les données afin de pouvoir garantir leur confidentialité, leur intégrité, leur authenticité et leur non répudiation.

- La confidentialité consiste à rendre l'information inintelligible à d'autres personnes que les acteurs de la transaction.
- L'intégrité des données permet de déterminer si les données n'ont pas été altérées durant la communication (de manière fortuite ou intentionnelle).
- L'authentification assure l'identité d'un utilisateur, c'est-à-dire garantit à chacun des correspondants que son partenaire est bien celui qu'il croit être.
- La non-répudiation de l'information empêche chacun des correspondants de pouvoir nier la transaction. Elle est complémentaire de l'accusé de réception.

La figure 1-2 illustre le principe de la cryptologie : chiffrement-déchiffrement et cryptanalyse.

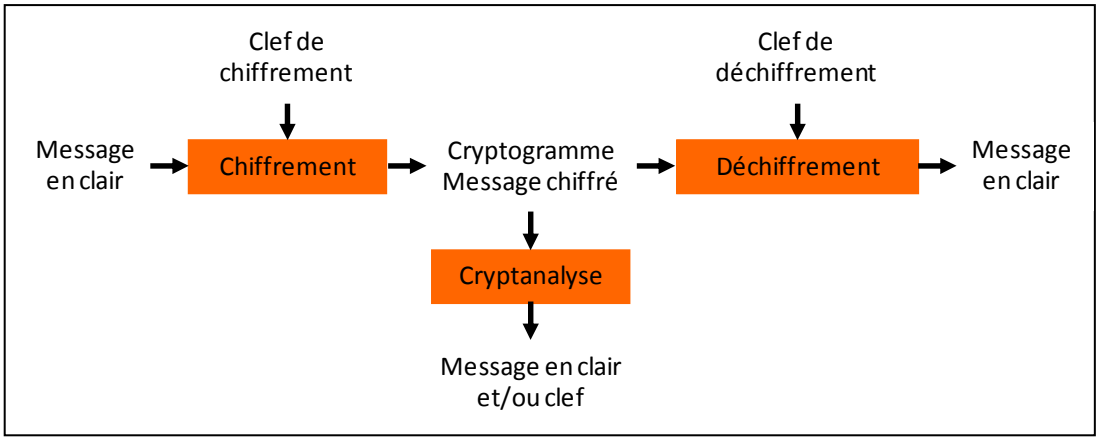


FIGURE 1-2 : PRINCIPE DE LA CRYPTOLOGIE

Les algorithmes de chiffrement sont des fonctions mathématiques, permettant de chiffrer et de déchiffrer le message. Le chiffrement transforme le message en clair (ou texte en clair) en un message chiffré (ou texte chiffré) ou cryptogramme. Inversement le déchiffrement reconstruit à partir du message chiffré le message en clair.

En 1883, Auguste Kerckhoffs posa les principes de la cryptographie moderne [Ker83]. Ces principes restent actuels et en particulier le principe qui stipule que la sécurité résultant de l'utilisation d'un crypto-système ne doit pas reposer sur le secret de ses principes de conception ou du paramétrage du dispositif, suggérant qu'elle doit uniquement reposer sur une clef secrète.

Il existe deux grandes familles d'algorithmes de chiffrement : les algorithmes asymétriques et les algorithmes symétriques.

Les algorithmes asymétriques, ou « à clef publique », utilisent deux clefs différentes pour le chiffrement et le déchiffrement. La clef de chiffrement est dite *publique* donc accessible à tous, tandis que la clef de déchiffrement est dite *privée*. Par conséquent, tout le monde peut envoyer des messages chiffrés à l'aide de la clef publique mais seul le propriétaire de la clef privée (qui a émis la clef publique) pourra les déchiffrer. Ce type d'algorithme a été introduit par W. Diffie et M. Hellman en 1976 [Dif76] afin de résoudre le problème lié aux transferts de clefs secrètes. Le RSA (du nom de ces auteurs Rivest, Shamir et Adleman) [Riv78], le ElGamal [ElG85], les courbes elliptiques [Kob84] sont les algorithmes asymétriques les plus répandus.

Les algorithmes symétriques, ou « à clef privée », utilisent la même clef pour chiffrer et déchiffrer les données. Ceci impose un accord entre les interlocuteurs qui doivent s'échanger la clef de chiffrement-déchiffrement. Usuellement, on distingue deux catégories de chiffrement symétrique :

- Le chiffrement par flots. Ces systèmes de chiffrement opèrent sur le message en clair bit par bit (ou quelquefois par petits groupements de bits).
- Le chiffrement par blocs. Ces systèmes opèrent eux sur le message en clair par « grands » groupes de bits (au moins 64 bits).

Avec un algorithme de chiffrement par blocs, le même bloc de texte en clair sera toujours chiffré en un même bloc de texte chiffré, en utilisant la même clef. Par contre avec un algorithme de chiffrement par flots, le même bit ou octet de texte en clair sera chiffré en un bit ou un octet différent à chaque chiffrement. Les modes de chiffrement ECB (carnet de codage électronique « Electronic CodeBook ») et CBC (chiffrement avec chaînage de blocs « Cipher Block Chaining ») sont liés à du chiffrement par blocs, tandis que les modes CFB (chiffrement à rétroaction « Cipher FeedBack ») et OFB (rétroaction de sortie « Output-FeedBack ») sont liés à du chiffrement par flots [Sch97]. Deux de ces modes seront présentés plus en détails dans la suite du manuscrit.

Pour construire des systèmes de chiffrement sûrs, Shannon [Sha49] a énoncé deux principes fondamentaux à respecter qui sont la confusion et la diffusion.

> La *confusion* rend la relation entre la clef de chiffrement et le texte chiffré la plus complexe possible. La confusion idéale rendrait chaque bit du texte chiffré dépendant de tous les bits de la clef.
>
> La *diffusion* doit permettre à chaque bit de texte clair d'avoir une influence sur une grande partie du texte chiffré. Ce qui signifie que chaque bit du texte chiffré doit dépendre de tous les bits du texte en clair.

Shannon introduit pour cela la notion de réseau de substitution-permutation. Le principe de substitution est lié à la confusion et celui de la permutation est lié à la diffusion. En combinant ces deux transformations, le chiffrement est robuste. Les algorithmes modernes de chiffrement par blocs tels que l'AES [FIPS197], Serpent [And98], SAFER [Mas94] (Secure And Fast Encryption Routine) sont basés sur le principe de réseau de substitution-permutation.

En pratique, dans une carte à puce, les algorithmes asymétriques sont souvent implantés sous forme logicielle (car ils sont trop coûteux en matériel) tandis que les algorithmes symétriques sont implantés sous forme matérielle pour des questions de performance. Dans les cartes à puce, l'algorithme symétrique implanté est soit un DES [FIPS46a] (ou dérivé Triple-DES [FIPS46b]) soit un AES.

Dans la suite de cette partie, les deux algorithmes de chiffrement symétriques (le DES et l'AES) sont décrits.

1.1.3 L'ALGORITHME DE CHIFFREMENT DES

Le chiffrement DES a été adopté comme standard par le NSB (National Bureau of Standard) le 23 novembre 1976. Depuis cet algorithme a été régulièrement réévalué et renouvelé comme standard jusqu'en 1999. A cette date, la taille de la clef du DES a été jugée trop faible, et pour pallier ce problème le Triple-DES a été adopté comme nouveau standard. Le Triple-DES est une variante du DES, où l'algorithme DES est effectué trois fois, généralement l'approche est : chiffrement avec une clef K1-déchiffrement avec une clef K2-chiffrement avec une clef K3. Cet algorithme utilise donc trois clefs secrètes (de 64 bits), éventuellement uniquement 2 si K1=K3, d'où une longueur minimale de 128 bits pour la clef secrète d'un tel algorithme.

Le DES utilise une clef secrète de 64 bits (dont 56 bits sont réellement utilisés, 1 bit par octet étant un bit de parité non utilisé pour le chiffrement) pour chiffrer un bloc de texte en clair de 64 bits afin d'obtenir un bloc de texte chiffré de 64 bits. Cet algorithme est basé sur le réseau de Feistel [Sch97] dont les principes sont : des permutations des substitutions, des échanges de blocs de données et une fonction prenant comment entrée une information secrète (clef). Un réseau de Feistel est divisé en plusieurs étages (rondes). Les données sont traitées en 2 parties (gauche et droite). A chaque ronde les 2 parties sont échangées, une partie est ensuite combinée avec une version transformée de l'autre partie. Pour simplifier, une partie (la moitié des données) est chiffrée avec la clef et le résultat est ajouté à la partie non chiffrée. A chaque ronde on intervertit le rôle de chaque partie droite et gauche.

L'algorithme DES se compose de 16 rondes identiques comme illustré sur la figure 1-3. Au début et à la fin du DES, il est appliqué respectivement une permutation IP sur 64 bits et son inverse IP^{-1}.

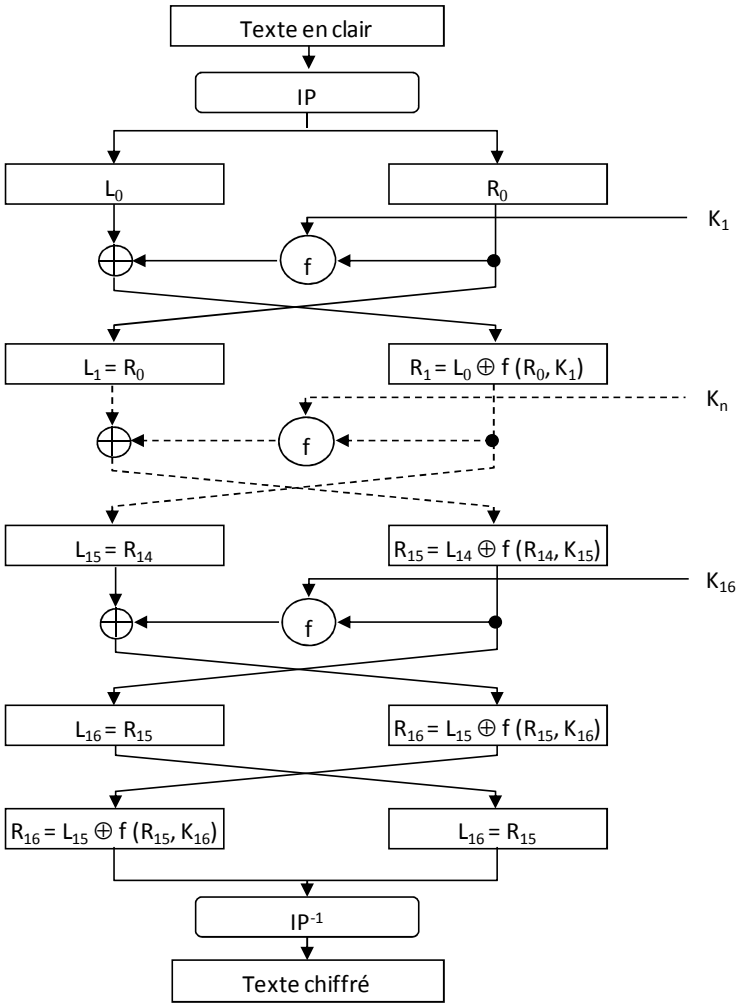


FIGURE 1-3 : ALGORITHME DE CHIFFREMENT DES

A chaque cycle de ronde, la fonction f est appliquée à la partie droite (noté R) des données. Cette fonction f se compose de 4 transformations, voir figure 1-4.

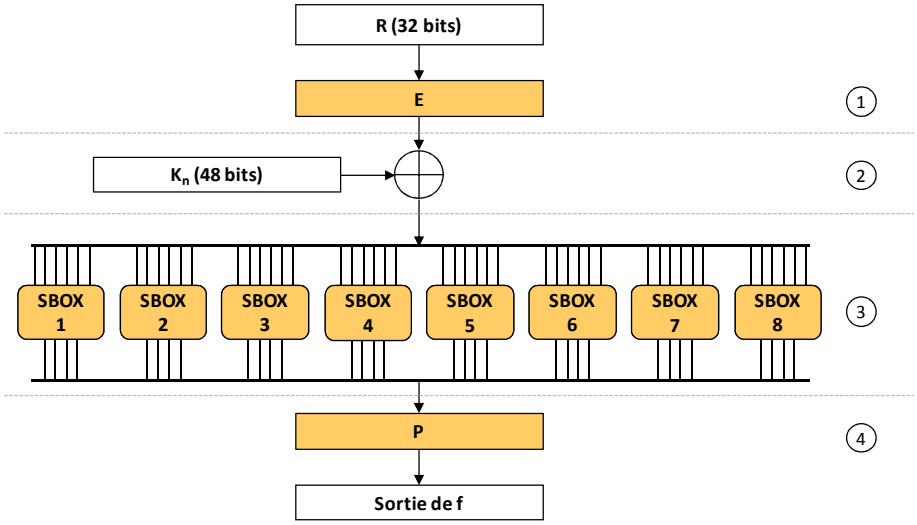


FIGURE 1-4 : LA FONCTION f DU DES

Ces 4 transformations sont :

- une permutation expansive E qui reçoit en entrée un bloc de 32 bits et retourne en sortie un bloc de 48 bits,
- un « ou-exclusif » avec la clef de ronde K_n,
- une transformation non-linéaire réalisée à l'aide 8 S-Box (6 bits vers 4 bits)
- une permutation P sur 32 bits.

Les 16 clefs de rondes K_n utilisées durant le chiffrement dérivent de la clef secrète initiale K (voir figure 1-5). La clef secrète du DES est composée de 64 bits dont 56 vont être utilisés lors du chiffrement. Les 8 autres bits sont utilisés pour vérifier l'intégrité des 56 autres bits.

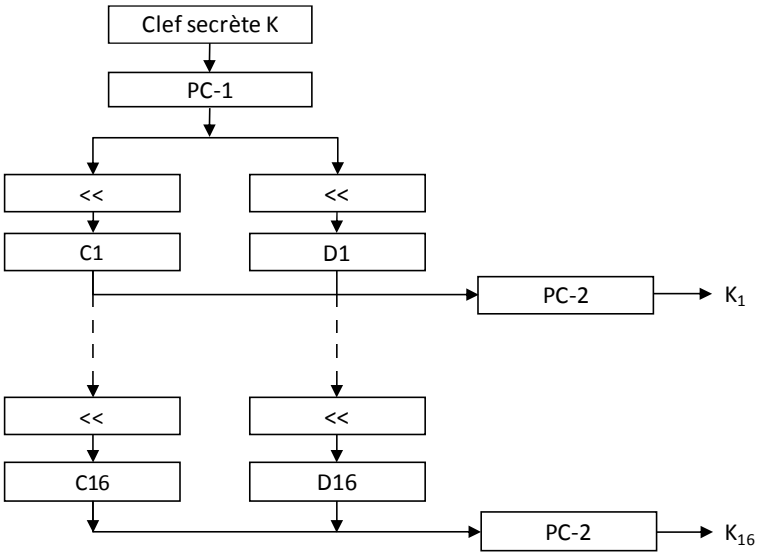


FIGURE 1-5 : GENERATEUR DES 16 CLEFS DE RONDES DU DES

Les trois opérations réalisées pour la génération des clefs de rondes sont :

- PC-1 (permutation compressive), les bits de parité sont supprimés et les 56 bits restants sont permutés. Cette opération reçoit 64 bits et en retourne 56.
- Des opérations de rotations vers la gauche (<<) d'un ou deux bits appliquées à des blocs de 28 bits.
- PC-2 (permutation compressive), cette opération reçoit en entrée une valeur sur 56 bits et retourne une valeur codée sur 48 bits qui sera utilisée comme clef de ronde. Par conséquent, les 16 clefs de rondes K_n sont codées sur 48 bits.

L'algorithme de déchiffrement du DES est identique au chiffrement, seules les clefs de rondes sont utilisées dans l'ordre inverse de K_{16} à K_1.

1.1.4 L'ALGORITHME DE CHIFFREMENT AES

Suite à la faiblesse du DES vis-à-vis de la cryptanalyse, principalement due à la relative faible longueur de la clef secrète et à l'augmentation des puissances de calcul permettant de lancer des recherches exhaustives, le NIST (National Institute of Standard and Technology) a lancé, en janvier 1997, un appel d'offre pour l'élaboration d'un nouveau système cryptographique. En août 1999, cinq finalistes ont été retenus (voir tableau 1-1).

Finalistes	Mise en œuvre de l'algorithme	Vitesse d'exécution	Sécurité
MARS (*IBM*)	-	+	+ +
RC6 (*RSA Laboratories*)	+ +	+ +	-
Rijndael (*J. Daemen, V. Rijmen*)	+	+	+
Serpent (*R. Anderson, E. Biham, L. Knudsen*)	+	-	+ + +
Twofish (*B. Schneier, J. Kelsey, D. Whiting, D. Wagner, N. Ferguson, C. Hall*)	-	+ +	+ +

TABLEAU 1-1 : LES CINQ FINALISTES DE L'AES

Finalement le 2 octobre 2000, le NIST annonce que l'algorithme « Rijndael », développé par Joan Daemen et Vincent Rijmen, a été sélectionné. Rijndael sera officiellement approuvé comme standard par le FIPS en décembre 2001 et deviendra l'AES.

L'AES traite des blocs de données de 128 bits et la taille de la clef généralement utilisée est de 128 bits avec des variantes de 192 et 256 bits. L'algorithme AES est basé sur une succession de rondes. Le nombre de rondes varie en fonction de la taille de la clef : si celle-ci est de 128 bits alors 10 rondes sont effectuées. Pour une taille de 192 bits et 256 bits, le nombre de rondes à effectuer est respectivement de 12 et 14. Dans le reste du manuscrit nous emploierons le terme AES pour décrire un AES à 128 bits de clef.

L'algorithme AES en mode chiffrement ou déchiffrement (voir figure 1-6) est composé de 3 étapes.

La première est une opération d'« ou-exclusif » entre le texte en clair (si chiffrement) ou le texte chiffré (si déchiffrement) et une clef (clef secrète K ou clef de ronde RK10).

La deuxième étape est composée d'un ensemble de 9 rondes chacune réalisant 4 opérations. Dans le cas du chiffrement les opérations sont SubBytes, ShiftRows, MixColumns et AddRoundKey et dans le cas du déchiffrement, l'inverse de l'ensemble des 4 opérations précédentes interviennent $MixColumns^{-1}$, $ShiftRows^{-1}$, $SubBytes^{-1}$ et AddRoundKey.

La dernière étape est une ronde dans laquelle l'opération MixColumns ou $MixColumns^{-1}$ n'est pas réalisée.

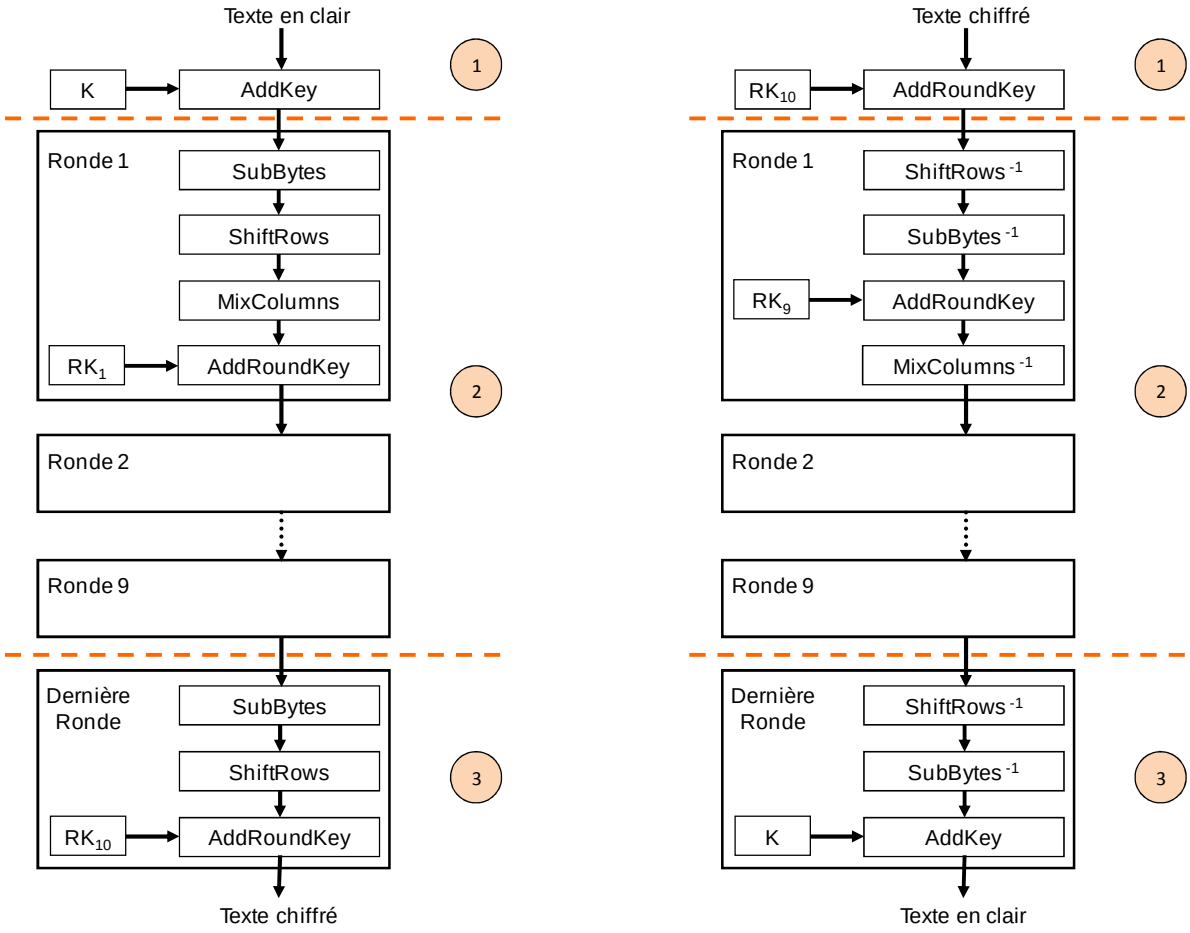


FIGURE 1-6 : CHIFFREMENT ET DECHIFFREMENT AES

Les différentes opérations agissent sur des blocs de données de 128 bits organisés sous forme de matrice d'octet 4×4.

a) L'opération SubBytes :

C'est une opération non linéaire qui s'applique à chacun des octets de la matrice. Cette opération consiste à remplacer chaque octet par une autre valeur (figure 1-7). La substitution se fait en utilisant une table appelée S-Box (tableau 1-2). Chaque octet S_{lc} (lc pour ligne-colonne) est représenté par deux nombres de 4 bits x et y. Les octets (notés sous forme

hexadécimale) contenus dans la table sont les octets de remplacement S'_{lc}. L'intersection de la ligne x avec la colonne y représente la valeur de remplacement de l'octet xy (exemple : si l'octet de départ vaut B8, x = B et y = 8, l'octet de remplacement vaut 6C). Ils sont définis sur la base mathématique des champs finis (ou champs de Galois). Les valeurs de la S-Box sont construites par deux fonctions : la première consiste à prendre l'inverse de l'octet dans GF(2^8) sachant que l'octet xy = 00 est son propre inverse, et la seconde consiste à lui appliquer une transformation affine dans GF(2).

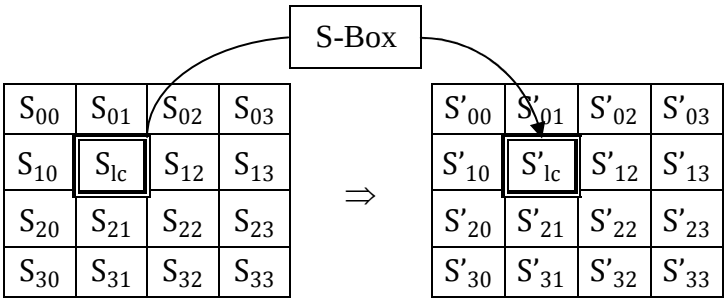


FIGURE 1-7 : PRINCIPE DE L'OPERATION SUBBYTES

hex		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

TABLEAU 1-2 : S-BOX : TABLE DE SUBSTITUTION

b) L'opération ShiftRows :

Elle agit sur les trois dernières lignes de la matrice résultant de l'opération SubBytes. En effet, au cours de l'opération ShiftRows la première ligne de la matrice ne subit aucune modification, par contre la deuxième ligne est décalée de 1 octet vers la gauche, la troisième

ligne est décalée de 2 octets vers la gauche et la quatrième ligne est décalée de 3 octets vers la gauche (figure 1-8).

S_{00}	S_{01}	S_{02}	S_{03}			S_{00}	S_{01}	S_{02}	S_{03}
S_{10}	S_{11}	S_{12}	S_{13}	$\leftarrow$		S_{11}	S_{12}	S_{13}	S_{10}
S_{20}	S_{21}	S_{22}	S_{23}	$\leftarrow \leftarrow$	$\Rightarrow$	S_{22}	S_{23}	S_{20}	S_{21}
S_{30}	S_{31}	S_{32}	S_{33}	$\leftarrow \leftarrow \leftarrow$		S_{33}	S_{30}	S_{31}	S_{32}

FIGURE 1-8 : PRINCIPE DE L'OPERATION SHIFT ROW

c) L’opération MixColumns :

C’est une multiplication de deux matrices : une matrice constante et la matrice obtenue à la fin de l’opération ShiftRows. Chaque colonne est considérée comme un polynôme de GF(2^8) et est multiplié par $a(x)= (03)_8 \cdot x^3+(01)_8 \cdot x^2+(01)_8 \cdot x+(02)_8$ modulo x^4+1 (figure 1-9).

02	03	01	01		S_{00}	S_{01}	S_{02}	S_{03}		S'_{00}	S'_{01}	S'_{02}	S'_{03}
01	02	03	01	$\times$	S_{10}	S_{11}	S_{12}	S_{13}	=	S'_{10}	S'_{11}	S'_{12}	S'_{13}
01	01	02	03		S_{20}	S_{21}	S_{22}	S_{23}		S'_{20}	S'_{21}	S'_{22}	S'_{23}
03	01	01	02		S_{30}	S_{31}	S_{32}	S_{33}		S'_{30}	S'_{31}	S'_{32}	S'_{33}

FIGURE 1-9 : PRINCIPE DE L'OPERATION MIX COLUMNS

d) L’opération AddRoundKey :

C’est une simple opération d’« ou-exclusif » bit à bit entre la matrice de données et la matrice de clef de rondes RKi (RKi est la $i^{ème}$ clef de ronde). Il y a 10 clefs de rondes, chacune dérivant de la clef secrète.

Pour obtenir les 10 clefs de rondes, une opération de génération ou d’expansion de clef est réalisée. Ainsi une clef de ronde différente est utilisée à chaque ronde.

Considérons les deux matrices suivantes, Clef Secrète et Rcon :

K_{00}	K_{01}	K_{02}	K_{03}
K_{10}	K_{11}	K_{12}	K_{13}
K_{20}	K_{21}	K_{22}	K_{23}
K_{30}	K_{31}	K_{32}	K_{33}

Clef secrète

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon

La première clef de ronde dérive de la clef secrète, puis la seconde de la première clef et ainsi de suite. En prenant une représentation matricielle de la clef, chaque clef est engendrée colonne par colonne. Le terme vecteur est employé pour parler d'une colonne.

V_{i-4}			V_{i-1}	V_i
K_{00}	K_{01}	K_{02}	K_{03}	?
K_{10}	K_{11}	K_{12}	K_{13}	?
K_{20}	K_{21}	K_{22}	K_{23}	?
K_{30}	K_{31}	K_{32}	K_{33}	?

Clef secrète

Le vecteur V_{i-1} subit une opération dite de rotation qui consiste en un simple décalage des quatre octets du vecteur vers le haut. Ensuite, on applique à ce résultat l'opération SubBytes :

V_{i-1}		$Rot(V_{i-1})$		$SubBytes\{Rot(V_{i-1})\}$
K_{03}	$\Rightarrow$	K_{13}	$\Rightarrow$	K'_{13}
K_{13}		K_{23}		K'_{23}
K_{23}		K_{33}		K'_{33}
K_{33}		K_{03}		K'_{03}

Le vecteur obtenu ($SubBytes\{Rot(V_{i-1})\}$) doit encore être additionné modulo 2 avec le vecteur V_{i-4}, ainsi qu'avec le premier vecteur de la matrice Rcon :

V_{i-4}		Rcon		$SubBytes\{Rot(V_{i-1})\}$		V_i
K_{00}	$\oplus$	01	$\oplus$	K'_{13}	=	$RK1_{00}$
K_{10}		00		K'_{23}		$RK1_{10}$
K_{20}		00		K'_{33}		$RK1_{20}$
K_{30}		00		K'_{03}		$RK1_{30}$

La clef de ronde s'écrit donc sous la forme suivante :

V_{i-4}				V_{i-1}	V_i		
K_{00}	K_{01}	K_{02}	K_{03}	$RK1_{00}$	?		
K_{10}	K_{11}	K_{12}	K_{13}	$RK1_{10}$	?		
K_{20}	K_{21}	K_{22}	K_{23}	$RK1_{20}$	?		
K_{30}	K_{31}	K_{32}	K_{33}	$RK1_{30}$	?		

Clef secrète Clef de ronde RK1

Le deuxième vecteur de la clef de ronde s'obtient par l'opération $V_i = V_{i-4} \oplus V_{i-1}$.

Sur le même principe le troisième et le quatrième vecteur de la clef de ronde s'obtiennent en faisant $V_i = V_{i-4} \oplus V_{i-1}$ (avec V_i désignant le vecteur à déterminer).

			V_{i-4}			V_{i-1}	V_i
K_{00}	K_{01}	K_{02}	K_{03}	$RK1_{00}$	$RK1_{01}$	$RK1_{02}$	$RK1_{03}$
K_{10}	K_{11}	K_{12}	K_{13}	$RK1_{10}$	$RK1_{11}$	$RK1_{12}$	$RK1_{13}$
K_{20}	K_{21}	K_{22}	K_{23}	$RK1_{20}$	$RK1_{21}$	$RK1_{22}$	$RK1_{23}$
K_{30}	K_{31}	K_{32}	K_{33}	$RK1_{30}$	$RK1_{31}$	$RK1_{32}$	$RK1_{33}$

Clef secrète Clef de ronde RK1

Pour déterminer la clef de ronde RK2, on utilise la même procédure en remplaçant la clef secrète par RK1 et en utilisant les données de la deuxième colonne de Rcon et ainsi de suite pour les autres clefs de rondes. De fait cette procédure est effectuée au total 10 fois pour générer les 10 clefs de rondes requises pour le chiffrement.

Toutes les opérations de chiffrement sont inversibles. Ainsi pour déchiffrer un texte, les opérations inverses sont appliquées. La procédure de génération des clefs de rondes est identique en chiffrement et déchiffrement. La différence majeure au niveau des clefs de rondes par rapport au chiffrement est que celles-ci sont utilisées dans l'ordre inverse. Ainsi la première clef utilisée dans le processus de déchiffrement est la clef de ronde numéro dix (RK_{10}) (voir figure 1-6).

Dans le tableau 1-3, nous résumons les principales caractéristiques des algorithmes de chiffrement les plus courants : DES, Triple-DES, AES et RSA. Pour chaque algorithme nous donnons la taille de la clef secrète, la taille du bloc traité, le nombre de rondes à exécuter, et enfin le niveau de sécurité. Les trois premières lignes correspondent à des algorithmes symétriques (DES, Triple-DES et AES) et la dernière à un algorithme asymétrique (RSA).

	Longueur de la clef secrète	Longueur du bloc de données	Nombre de rondes	Cassé ?	Sécurité
DES (1976) [FIPS46a]	64 (56 effectifs)	64	16	en 1997	faible
Triple-DES (1999) [FIPS 46b]	192 (168 effectifs)	64	3×16	non	moyenne
AES (2000) [FIPS197]	128, 192, 256	128	10, 12, 14	non	haute
RSA (1983) [Riv78]	Choisir p et q (n = p×q) deux grands nombres premiers tenus secrets *Clef publique « **e** »* **e** et **(p-1)×(q-1)** sont premiers entre eux. *Clef privée « d »* $\mathbf{d = e^{-1} \bmod (p-1)\times(q-1)}$	≤ n		oui clef < 640 bits	haute clef > 640 bits

TABLEAU 1-3 : ALGORITHMES CRYPTOGRAPHIE

1.2 LES ATTAQUES ET CONTRE-MESURES

Tous ces algorithmes de chiffrement font l'objet de nombreuses tentatives de cryptanalyse. La cryptanalyse étudie et utilise les faiblesses des algorithmes cryptographiques afin de déchiffrer illégalement les textes chiffrés. Une tentative de cryptanalyse est communément appelée « attaque ». Si elle est réussie, elle permet de récupérer soit le texte en clair soit la clef secrète.

Il existe quatre méthodes de cryptanalyse, qui reposent toutes sur l'hypothèse que l'attaquant connaît le détail de l'algorithme de chiffrement :

1**. L'attaque à texte chiffré seulement.** L'attaquant a connaissance d'un ou plusieurs textes chiffrés avec la même méthode.

2. **L'attaque à texte clair connu.** Dans ce cas (beaucoup plus rare), l'attaquant a accès à plusieurs textes chiffrés ainsi qu'aux textes clairs correspondants.

3. **L'attaque à texte clair choisi.** L'attaquant a accès à des textes chiffrés et aux textes clairs correspondants, mais il peut en plus chiffrer les textes qu'il veut (Il possède la machine à chiffrer mais il lui manque la clef).

4. **L'attaque à texte chiffré choisi.** Dans ce cas l'attaquant peut déchiffrer des textes, mais évidemment, il ne connaît pas la clef.

En plus de ces attaques de cryptanalyse, la carte à puce, en tant que composant électronique, est également sujette à de nombreuses attaques dites matérielles. Ces attaques matérielles peuvent être rangées en trois sous-catégories :

- Les attaques invasives qui nécessitent de manipuler physiquement la puce et qui peuvent conduire à sa destruction.
- Les attaques semi-invasives qui agissent physiquement sur la puce sans pour autant aller jusqu'à sa détérioration.
- Les attaques non-invasives qui consistent à étudier les signaux compromettant émis par la puce sans la modifier.

Dans la partie suivante nous présentons ces différentes attaques matérielles et les contre-mesures qui leurs sont associées.

1.2.1 ATTAQUES INVASIVES ET CONTRE-MESURES

– **Les attaques invasives :**

Les attaques invasives nécessitent un accès direct aux composants internes ce qui implique que la puce doive être retirée de son boîtier plastique (depackaging) et que la couche de passivation doit être retirée. Une attaque invasive peut nécessiter des heures de travail dans des laboratoires spécialisés. Les attaques invasives sont donc réalisées par des attaquants ayant de solides connaissances dans le domaine et un matériel approprié.

Les attaques invasives les plus connues sont la reconstruction de « layout » (reverse engineering) et l'attaque par sonde (microprobing).

Par exemple, l'attaque par reconstruction de « layout » consiste à étudier la structure interne de la puce pour en déduire son fonctionnement. En effet cette attaque doit permettre de définir la localisation précise de tous les transistors et interconnexions de la puce. Concrètement toutes les couches technologiques formées durant la fabrication sont enlevées

une par une (dans l'ordre inverse du processus de fabrication) et sont cartographiées. Finalement, toutes les informations collectées sont rassemblées et permettent de reconstruire entièrement le « layout » ou de créer des fichiers de « netlist » de simulation.

- **Les contre-mesures :**

Pour se prémunir des attaques invasives, les contre-mesures doivent détecter l'attaque au plus tôt pour bloquer l'activité de la puce et détruire les données sensibles avant qu'elles ne soient récupérées par l'attaquant. Les contre-mesures les plus répandues pour ce genre d'attaque sont :

a) La contre-mesure appelée : « niveau ou couche de protection (Shielding [Kom99]) », consiste à empêcher l'analyse de l'information circulant dans la puce. La couche de protection est une grille active portant un signal de protection. L'interruption de ce signal induira l'effacement des mémoires et l'arrêt de fonctionnement de la puce.

b) La contre-mesure appelée : « brouillage de conception [Sam02] », dite aussi « placement routage aléatoire » consiste à placer et router de façon aléatoire les connexions d'un bus de données entre les divers modules (par exemple les processeurs et les mémoires). Ainsi un attaquant ne pourra pas facilement interpréter la valeur portée par un bus.

c) La contre mesure appelée : « réseaux de capteurs [Kom99] » consiste à placer la puce sous des couches de métal sur lesquelles sont placés des détecteurs d'activité. Ceux-ci sont sensibles à la température, aux UV, aux rayons X et communiquent à la puce toute opération suspecte. En cas d'une tentative d'intrusion, la puce se bloquera voire provoquera la destruction des données sensibles.

1.2.2 ATTAQUES SEMI-INVASIVES ET CONTRE-MESURES

- **Les attaques semi-invasives :**

Les attaques semi-invasives sont une nouvelle classe d'attaques publiées pour la première fois en 2002 quand fut introduit par Seigei Skorobogatov et Ross Anderson [Sko02] les attaques par perturbation lumineuse. L'attaque proposée dans [Sko02] utilise un simple flash d'appareil photo pour modifier la valeur d'une cellule mémoire.

Les attaques semi-invasives comme les attaques invasives requièrent, dans la majorité des cas, d'enlever la puce de son boitier plastique. Cependant la couche de passivation reste

intacte puisque ces attaques ne nécessitent pas de contacts physiques directs sur la puce. L'attaque semi-invasive la plus connue est : « l'attaque par injection de fautes ». Cette attaque consiste à générer de façon intentionnelle des fautes dans le circuit en cours de fonctionnement et à comparer le résultat normal avec le résultat fauté.

Ces fautes peuvent être générées par :

a) flash lumineux. Ils permettent de modifier une partie du système. Ces attaques ciblent les technologies EPROM, EEPROM, Flash.

b) ionisation. L'attaquant peut faire usage d'un faisceau laser pour ioniser le dispositif afin de changer quelques bits de mémoire (SRAM), et par conséquent, modifier la sortie.

Dans [Gir04] par exemple, l'auteur décrit comment le fonctionnement fautif du circuit peut être utilisé pour retrouver les données secrètes. En effet, l'auteur décrit une attaque DFA (Differential fault analysis [Bih97]) ciblant l'algorithme de chiffrement AES. Si l'injection d'une faute rend un octet fautif alors en analysant moins de 250 textes chiffrés, la clef secrète est retrouvée.

- **Les contre-mesures :**

Comme pour les attaques invasives, la contre-mesure appelée « niveau ou couche de protection » peut être utilisée pour contrer les attaques semi-invasives.

Une méthode communément utilisée pour protéger un système vis-à-vis des attaques par injection de faute consiste à doubler l'exécution de la fonction critique puis à comparer les deux résultats correspondants. Cette méthode a deux principaux inconvénients : elle est coûteuse en temps d'exécution et elle ne décèle pas les fautes permanentes.

Une autre contre-mesure [Kar01] utilisée pour les attaques induisant des fautes consiste à modifier le circuit pour exécuter, en parallèle d'une opération de chiffrement, l'opération inverse (avec différents niveau de granularité : ronde, opération de ronde…) et de comparer le résultat obtenu avec le texte entrant. Si le résultat obtenu en sortie de l'opération inverse est différent de celui entrant dans l'opération de chiffrement alors une erreur s'est produite. Il faut donc implanter sur la puce à la fois le chiffrement et une partie du déchiffrement de sorte qu'ils puissent être exécutés en parallèle. Une telle contre-mesure est efficace mais énormément coûteuse en surface.

1.2.3 ATTAQUES NON-INVASIVES ET CONTRE-MESURES

- **Les attaques non-invasives :**

Les attaques non-invasives (ou dites également par canaux cachés ou par canaux auxiliaires) reposent sur l'étude des informations physiques (temps d'exécution, courant consommé, champ magnétique émis…) qui émanent de la puce lors de son fonctionnement. Ces attaques ne nécessitent pas de modifier le dispositif attaqué. La mise en œuvre de ces attaques est moins coûteuse en matériel que les deux précédentes, et plus rapide. Une autre propriété inquiétante du point de vue de la sécurité est que ces attaques sont indétectables. Toutes ces propriétés font que ces attaques sont considérées comme les menaces les plus sérieuses visant les circuits sécurisés.

Les attaques non-invasives les plus connues sont : l'analyse du temps de calcul (timing attaque) [Koc96], l'analyse de la consommation (SPA et DPA Single Power Analysis et Differential Power Analysis) [Koc99], l'analyse de l'émission électromagnétique [Qui00]...

Par exemple, « l'attaque par analyse de consommation » consiste à étudier l'activité du courant de la puce. La consommation du dispositif dépendant en CMOS des changements d'états des composants, il est possible de retrouver une partie des données manipulées (y compris la clef secrète) en analysant les fluctuations de courant. Les deux techniques les plus connues d'attaques sont la SPA (Simple Power Analysis) et la DPA (Differential Power Analysis). La SPA est une analyse directe des profils en courant de l'opérateur ou du module cryptographique. La SPA révèle des informations sur les opérations exécutées par le système et sur la clef secrète de l'algorithme de chiffrement. Toutefois pour mener une telle attaque, l'attaquant doit connaître parfaitement l'algorithme de chiffrement. La DPA est une attaque plus puissante que la SPA, car l'attaquant n'a pas besoin de connaître les détails de l'implantation de l'algorithme de chiffrement. Une analyse statistique est utilisée pour établir une corrélation entre la consommation et les données traitées. A partir de cette consommation, il est alors possible d'extraire des informations sur la clef secrète.

- **Les contre-mesures :**

Les contre-mesures vont consister soit à brouiller l'information (la consommation par exemple), soit à atténuer ou éliminer le signal émis. En majorité, ces contre-mesures sont propres à l'attaque ciblée.

Il existe deux solutions majeures pour brouiller l'information :

- le chiffrement interne des bus de données et des mémoires,
- la génération d'activité aléatoire.

Chiffrer les bus de données et les mémoires signifie que les données ne transitent jamais en clair même à l'intérieur de la puce. Par conséquent, toute attaque par analyse d'activité sera vaine dans la mesure où l'attaquant ne récupèrera que des données chiffrées.

Pour générer de l'activité aléatoire, des opérations leurres et aléatoires seront effectuées par la puce en parallèle des opérations nécessaires au calcul. Ces opérations supplémentaires induiront l'attaquant en erreur.

Par exemple, les contre-mesures proposées pour une attaque DPA sont :

- Augmenter le niveau de bruit en exécutant des processus aléatoires concourants.
- Introduire du bruit de synchronisation avec des interruptions de processus, des fréquences d'horloge variables, des retards aléatoires ou des chemins alternatifs.
- Réduire les signaux « critiques » par un ordonnancement aléatoire du processus. Les substitutions parallèles (comme les S-Box dans le DES et dans l'AES) peuvent par exemple être exécutées dans un ordre aléatoire.
- Masquer les valeurs intermédiaires par des valeurs aléatoires. Si des données aléatoires sont ajoutées aux données à traiter et soustraites plus tard, l'analyse de la consommation par exemple montre des profils différents même lors d'exécutions répétées de données à traiter identiques.
- Limiter la contrôlabilité et l'observabilité des entrées et des sorties du cœur de chiffrement. En effet, une attaque ne peut être réalisée si l'information entrante et sortante n'est pas complète.

1.3 NIVEAU DE SECURITE

Afin d'évaluer les risques de sécurité, il est important d'avoir une idée des différents types d'agresseurs. Une classification a été proposée par IBM [Abr91]. Cette classification prend en compte le niveau d'expertise et de ressources des attaquants. Les attaquants sont regroupés en trois catégories :

- *Classe 1* : « Étrangers intelligents » (Clever Outsiders). Il s'agit de personnes ayant un niveau de connaissance du domaine important n'ayant qu'une connaissance insuffisante du système et peu de matériel à disposition. Souvent ces personnes essaient d'utiliser les failles existantes du système plutôt que d'en créer d'autres.
- *Classe 2* : « Initiés cultivés » (Knowledgeable Insiders). Il s'agit de spécialistes expérimentés possédant une bonne connaissance du système visé et ayant accès à des outils très sophistiqués et à des instruments pour l'analyse.
- *Classe 3* : « Organismes financés » (Funded Organizations). Il s'agit d'organisations capables de rassembler des équipes de spécialistes avec des compétences complémentaires et soutenus par de grandes ressources financières. Ces groupes de spécialistes sont capables d'analyser en profondeur le système visé et de concevoir de nouvelles attaques. D'une manière générale, ils ont accès à des équipements ultrasophistiqués et ont des moyens illimités.

Suivant le domaine d'application (télévision à péage, carte bancaire…) de la carte à puce, les attaquants et par la même le niveau de sécurité requis, ne sont pas les mêmes. La norme FIPS 140-2 [FIPS140] spécifie les exigences en matière de sécurité que les modules cryptographiques devront respecter. Cette norme définit quatre niveaux de sécurité :

Niveau de sécurité 1 : c'est le niveau le plus bas de sécurité. Aucun mécanisme physique spécifique de sécurité n'est exigé dans un module cryptographique de niveau 1. Seule l'utilisation d'un algorithme de chiffrement standard est requise (e.g. AES, DES...).

Niveau de sécurité 2 : il augmente la sécurité physique du niveau 1 en exigeant l'identification de l'utilisateur, et en mettant en évidence d'éventuelles tentatives d'intrusion.

Niveau de sécurité 3 : il impose que les mécanismes de sécurité détectent l'attaque mais aussi contrecarrent les tentatives d'intrusion, d'utilisation, ou de modification du dispositif cryptographique.

Niveau de sécurité 4 : c'est le niveau de sécurité le plus élevé. Le module cryptographique doit être parfaitement protégé contre les tentatives d'intrusion. Ce niveau nécessite la mise en place des mesures prises pour les 3 premiers niveaux, et inclut la possibilité de détecter des variations anormales des paramètres environnementaux, tels que la température ou la puissance.

1.4 CONCLUSION

L'objectif de ce chapitre était de présenter les circuits sécurisés, leurs fonctionnements et leurs contraintes de conception. En effet, la sécurité qui est primordiale pour ces circuits impose des contraintes sur le processus de conception de la carte à puce. A la fin du processus de fabrication, le système est évalué et son niveau de sécurité est certifié de sorte qu'il puisse être utilisé pour un certain domaine d'application.

Néanmoins, même si le système a été correctement conçu et que tous les principes de sécurité ont été approuvés et certifiés, le niveau de sécurité assuré par conception peut être remis en cause après fabrication. Un défaut de fabrication pouvant altérer l'efficacité d'une contre-mesure, par conséquent un test de production de très bonne qualité est requis. Ce test doit permettre de livrer aux utilisateurs uniquement des circuits sains.

Afin de proposer des solutions de test appropriées à ce domaine et notamment au test des crypto-processeurs digitaux implantant des algorithmes de chiffrement standards, nous avons d'abord analysé les méthodes de test disponibles et leur impact sur la sécurité du dispositif. Cette analyse est présentée dans le chapitre suivant.

Chapitre 2 : Test des circuits sécurisés & Implications

Chapitre 2 : TEST DES CIRCUITS SECURISES & IMPLICATIONS

Ce chapitre présente le test des circuits numériques, et en premier lieu son rôle et les diverses approches de test possibles : test externe, test intégré et techniques de conception en vue du test.

Dans une deuxième partie, le conflit existant entre les techniques de conception en vue du test et la sécurité est présenté. Pour illustrer ce problème, l'attaque du cœur de chiffrement AES, implanté avec une chaîne de scan, est décrite. Plusieurs contre-mesures à ces attaques sont exposées.

La dernière partie de ce chapitre justifie le choix d'une approche de test intégrée pour des modules cryptographiques.

2.1 LES TECHNIQUES DE TEST

2.1.1 PRINCIPES ET BUT DU TEST

Le rôle du test est de vérifier le bon fonctionnement de la puce après fabrication et donc l'absence de défaillance physique. Deux types de tests sont utilisés pour mettre en évidence les défaillances éventuelles d'un circuit numérique : un test de caractérisation qui permet de vérifier les marges de fonctionnement et un test dit « de production » qui vise à traquer les défauts localisés.

Le test de caractérisation consiste à effectuer des mesures paramétriques (courant, tension, temps de montée et de descente...) sur le circuit. Ce test paramétrique fait intervenir des techniques de métrologie particulières et ne sera pas abordé dans le cadre de ce manuscrit.

Le test de production est systématiquement réalisé sur l'ensemble de tous les circuits issus des chaînes de fonderie. Ce test sert à trier les circuits « sains », pouvant êtres livrés au client, des circuits défectueux, à éliminer. Il n'a pas pour rôle de définir la cause et la localisation de la défaillance : erreur de conception résiduelle ou défaut de fabrication.

Le test de production se déroule de la façon suivante : à l'aide d'équipements dédiés au test, un ensemble de stimuli est appliqué en entrée du circuit et les réponses obtenues en sortie sont collectées. Si les réponses sont égales à celles attendues, le circuit est déclaré sain et peut être délivré à l'utilisateur. Par contre, si le circuit présente des erreurs, il est marqué comme fautif et sera rejeté.

Les termes défauts, erreurs et fautes sont utilisés pour qualifier l'apparition d'un comportement inexact. Leurs définitions sont les suivantes [Bus00] :

> *Définition* : Un défaut (aussi appelé défaillance) est la différence, non voulue, entre l'implantation réelle et l'implantation désirée du système.

Dans les circuits VLSI, les principaux défauts sont les suivants [How81] :

- Défauts du procédé de fabrication : rupture d'oxyde, transistor parasite, …
- Défauts du matériau : impureté, fissure du matériau, …
- Défauts du boîtier après encapsulation : dégradation des contacts, étanchéité du boitier, ...

Afin de simplifier l'analyse de ces défauts, des modèles de fautes logiques ont été proposés pour les représenter. Cette modélisation permet de simplifier l'analyse en passant de grandeurs électriques à des niveaux logiques par exemple, et d'autre part de diminuer la complexité algorithmique car de nombreux défauts peuvent être modélisés par une même faute logique.

> *Définition* : Une faute ou panne correspond à la représentation d'un défaut suivant un modèle donné. La modélisation permet l'utilisation d'outils de simulation de fautes et de génération automatique de vecteurs de test (ATPG).

> *Définition* : Une erreur est la manifestation logique d'un défaut observable sur une sortie du circuit. Elle peut se produire en présence de certains stimuli.

Les modèles de fautes les plus courants sont :

a) le collage :

Le modèle de collage logique considère que chaque ligne, chaque équipotentielle de la description du circuit au niveau porte peut être collée de façon permanente à la valeur logique 0 ou 1 quelles que soient les valeurs d'entrée du circuit. Physiquement, il peut correspondre à

une connexion permanente de l'équipotentielle soit à la masse, soit à l'alimentation. Les vecteurs de test générés avec ce modèle de collage permettent néanmoins de détecter d'autres défauts.

b) le court-circuit :

Le modèle « court-circuit » représente un contact entre deux ou plusieurs lignes adjacentes du circuit logique. En présence d'une telle faute, les lignes en court-circuit vont avoir le même comportement durant tout le temps de fonctionnement. Les vecteurs générés à l'aide du modèle de collage fournissent de bon résultat sur les fautes de court-circuit.

c) les fautes de délais :

Ces fautes permettent de représenter un disfonctionnement dynamique, le circuit fonctionne correctement à faible fréquence mais présente des erreurs de fonctionnement à haute fréquence c'est-à-dire lors d'un fonctionnement à vitesse nominale ou au delà. Les modèles de fautes de délais prennent de plus en plus importance du fait des vitesses de fonctionnement de plus en plus élevées et des dérives technologiques plus fréquentes dans les technologies profondément submicroniques.

Des outils logiciels permettent d'extraire la liste de fautes pouvant affecter un circuit en fonction du modèle choisi. C'est à partir de cette liste que l'on va chercher la séquence de stimuli de test la plus adéquate pour détecter ces fautes, séquence qui sera ensuite appliquée à l'ensemble des circuits fabriqués.

La nature des fautes considérées a une grande influence sur la génération de la séquence de test. Plus le modèle de fautes est lié à la nature physique du circuit plus l'élaboration de la séquence de test est complexe (plus de fautes dans le modèle et/ou recherche des stimuli de test plus complexe).

Vis-à-vis d'un modèle de faute donné, la qualité de la séquence de stimuli permettant de détecter les fautes du modèle est exprimée en fonction de sa longueur (influant sur le temps de test après fabrication), de la difficulté avec laquelle elle a été générée (influant sur le temps de conception), et de sa capacité à détecter plus ou moins de fautes. Ce dernier critère s'exprime en termes de taux de couverture de fautes, qui représente le rapport entre le nombre de fautes détectées et le nombre total de fautes dans le modèle considéré.

2.1.2 METHODES DE TEST

Lors de la conception d'un circuit, et donc avant fabrication, les méthodes de test mises en jeux regroupent « les méthodes d'obtention de la séquence de test » et « les méthodes de mise en œuvre du test ». Concernant l'élaboration de la séquence de test, plusieurs approches sont envisageables : génération de séquences déterministes, aléatoires ou mixtes selon les critères qualitatifs visés. Les méthodes de mise en œuvre du test regroupent les solutions permettant d'appliquer la séquence de test au circuit.

1. Elaboration de la séquence de test

La séquence de test peut être obtenue suivant différentes approches. Chacune présente des avantages et des inconvénients.

a) *L'Approche déterministe* : le but est de déterminer de façon explicite un stimulus (ou vecteur) de test pour chacune des fautes de la liste considérée. La détermination des vecteurs de test utilise des algorithmes de sensibilisation de chemins disponibles dans la littérature (D-algorithme, PODEM,…) [Rot66], [Goe81]. En raison de la complexité grandissante des circuits, l'élaboration de la séquence devient de plus en plus difficile, l'approche « diviser pour régner » qui consiste à traiter le circuit par sous ensemble de portes permet de gérer cette complexité. Reste la difficulté à construire un vecteur de test pour une faute difficile c'est-à-dire ne pouvant être détectée que par un nombre très restreint de stimuli.

b) *L'Approche exhaustive* (ou pseudo-exhaustive) : l'approche consiste à appliquer en entrée du circuit toutes les combinaisons possibles. Si la génération d'une telle séquence est immédiate et garantit un taux de couverture optimal, la séquence de vecteurs de test obtenue est très longue (pour un circuit combinatoire à n entrées, la séquence est de longueur 2^n). Cette approche est donc inapplicable lorsque le nombre d'entrée est trop important.

c) *L'Approche aléatoire* (ou pseudo-aléatoire) : contrairement à l'approche déterministe où les vecteurs de test sont sélectionnés en fonction des fautes ciblées, l'approche aléatoire délivre au circuit un grand nombre de vecteurs choisis au hasard, en espérant que parmi eux il y en est au moins un pour tester chacune des fautes. Les questions qui vont de pair avec une telle approche sont : « Combien de vecteurs aléatoires faut-il envoyer au circuit sous test pour avoir un bon taux de couverture ? » « Quelle est la durée de test résultante ? ». En pratique, cette longueur est calculée par simulation de faute pour une séquence aléatoire donnée ou estimée en fonction du nombre de vecteurs détectant la faute ou les fautes les plus difficiles

[Dav98]. En effet, Il faut trouver une séquence suffisamment longue pour tester la ou les fautes les plus difficiles à tester. Pour un circuit combinatoire à n entrées, chaque faute détectable du circuit est testée par un ensemble de vecteurs d'entrées parmi les 2^n possibles. La faute la plus difficile à tester est celle détectée par le plus petit nombre m de vecteurs (il peut en exister plusieurs). La relation [Bard84] :

$$L = \left\lceil \frac{\ln(1-C) - \ln(k)}{\ln(1-d_{min})} \right\rceil,$$

permet de calculer la longueur nécessaire L de la séquence aléatoire de test en fonction de :

- C (niveau de confiance ou la qualité de test), c'est-à-dire la probabilité que la séquence détecte toutes les fautes détectables. Cette valeur est couramment fixée par l'utilisateur à 99%.
- d_{min}, la plus petite probabilité de détection d'une faute dans le circuit $\left(d_{min} = \frac{m}{2^n}\right)$.
- k, le nombre de fautes ayant une probabilité de détection comprise entre [d_{min}, $2d_{min}$].

La simulation de fautes, permettant d'estimer la qualité d'une séquence aléatoire de vecteurs de test en termes de taux de couverture, peut engendrer des temps de calcul importants. En effet, il n'est pas possible de garantir un taux de couverture à priori car les séquences générées sont longues (voir très longues) pour atteindre un taux de couverture acceptable (e.g. > 95%). Néanmoins la génération de la séquence reste très peu complexe puisqu'il s'agit ici de générer des nombres aléatoires.

d) *L'Approche Mixte* : c'est un mélange de vecteurs déterministes générés pour détecter les fautes dites difficiles, c'est-à-dire les fautes ayant un faible nombre de vecteurs détectant, et de vecteurs aléatoires pour tester les autres fautes « faciles ».

Notons enfin que si ces trois approches restent envisageable pour des circuits combinatoire, elles sont difficiles voire impossibles à mettre en œuvre en pratique sur des circuits séquentiels. Rappelons que la détection d'une faute de collage dans un circuit séquentiel demande l'application non pas d'un stimulus unique mais d'un ensemble ordonné de stimuli (séquence) permettant de tenter de positionner l'équipotentielle à la valeur désirée puis de propager la valeur portée par cette équipotentielle vers une sortie observable. Il existe des ATPG pour circuits séquentiels mais la complexité de recherche d'une séquence augmente de façon exponentielle avec la longueur des cycles entre éléments de mémorisation

(bascules). L'application exhaustive de toutes les séquences ordonnées de vecteurs n'est pas possible en pratique pour des questions de temps d'application, enfin les probabilités d'obtenir des séquences ordonnées de vecteurs efficaces (séquences "détectantes") en générant des valeurs aléatoires est très faible.

Suite à la définition de ces différentes approches pour l'élaboration des séquences de test, il est nécessaire de décrire les méthodes de mise en œuvre du test puisqu'elles constituent l'autre paramètre des méthodes de test.

2. Mise en œuvre du test

Il existe deux méthodes principales de mise en œuvre du test : le test externe et le test intégré.

a) *Le test externe* :

Le test externe est basé sur l'utilisation d'équipements de test automatique (ATE, Automatic Test Equipments ou testeurs) chargé d'appliquer les vecteurs de test aux entrées du circuit, de collecter les réponses et de les comparer avec celles attendues. Les vecteurs de test et les réponses attendues pré-calculées par simulation sont stockés au sein du testeur (d'où l'appellation de test externe).

Le test externe a comme avantage de ne nécessiter l'implantation d'aucunes ressources de test (générateur de vecteurs et analyseur de réponses) sur le circuit et de pouvoir facilement modifié la séquence de test ou le système d'analyse des réponses si besoin est.

L'inconvénient majeur de cette technique est le coût d'équipement. En effet, de nos jours les testeurs sont obligatoirement des systèmes complexes ayant une capacité de stockage élevée (vecteurs de test + réponse(s) attendue(s)) et de nombreuses entrées-sorties capables de hauts débits, car les fréquences de fonctionnement des circuits et leurs nombres d'entrées et de sorties ne cessent d'augmenter. De plus, le testeur ne permet pas toujours de tester le circuit à sa fréquence nominale de fonctionnement.

Un autre inconvénient de cette méthode est que le testeur ne peut pas être utilisé pour des opérations de maintenance sur site.

b) *Le test intégré* :

Il existe deux catégories de test intégré (ou BIST (Built-In Self Test)) : le test intégré en ligne et le test intégré hors-ligne (voir figure 2-1) [Abra94].

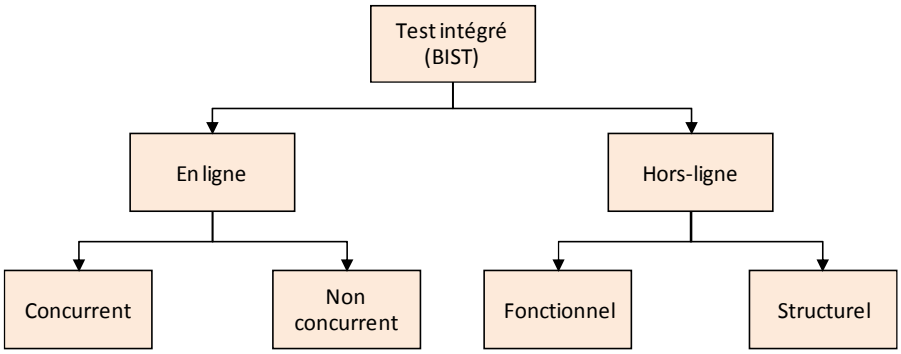


FIGURE 2-1 : DEFINITION DU TEST INTEGRE

Le test intégré *en ligne* consiste à tester le circuit durant son fonctionnement normal. Il n'y a pas besoin d'un mode spécifique de test. Le test en ligne peut être :

- Concurrent : c'est-à-dire qu'il s'effectue en parallèle de l'application. Il s'agit alors de test en ligne continu.
- Non concurrent : c'est-à-dire qu'il s'effectue pendant les instants de « dormance ». Il s'agit alors de test en ligne périodique.

Le test en ligne est couramment utilisé pour s'assurer de la fiabilité d'un dispositif en vérifiant la validité des données traitées en cours de fonctionnement. Par contre une telle approche ne permet pas d'assurer un test de production puisqu'elle ne vise pas à déceler la présence d'une faute avant de délivrer le circuit au client. Cette approche sera plus longuement discutée au chapitre 6.

Par opposition au test intégré en ligne, le test intégré *hors-ligne* est une approche de test utilisable en fin de production pour trier les circuits sains des circuits fautifs. Il peut aussi être utilisé comme solution de test durant la vie du circuit. Une telle approche ne peut toutefois être appliquée en parallèle du fonctionnement normal puisqu'elle nécessite l'application de données de test spécifiques et non de données quelconques de fonctionnement. Le test intégré hors ligne implique donc la création d'un nouveau mode de fonctionnement dit : « mode test ». Par abus de langage, dans le reste du manuscrit sauf mention contraire, le terme test intégré sera employé pour parler d'un test intégré hors-ligne.

Le test intégré ou BIST (Built In Self Test) [Agr93, Agr93bis] consiste à intégrer sur la puce de la logique supplémentaire dédiée au test (voir figure 2-2). Cette logique comprend un générateur de vecteurs de test, un analyseur de réponses, la connectique permettant le lien entre les divers modules et un contrôleur pour gérer le mode test. L'intégration des fonctions de test dans le circuit permet d'exécuter le test de façon autonome.

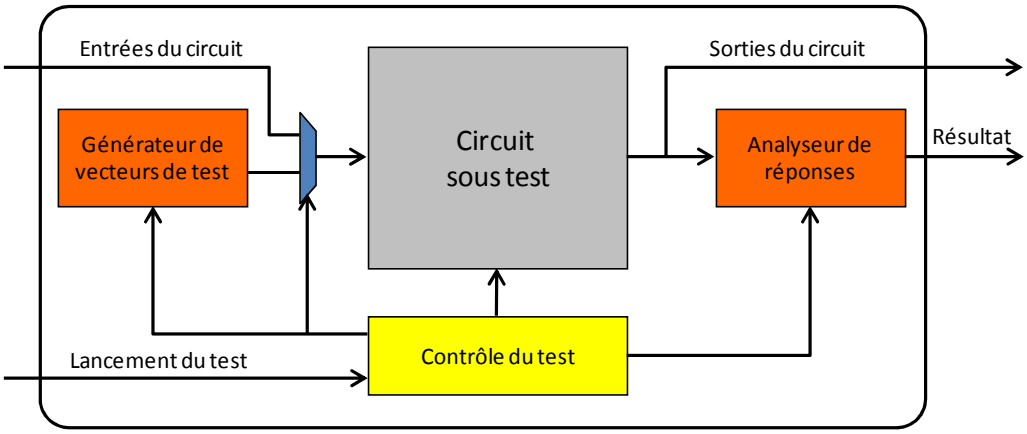


FIGURE 2-2 : ARCHITECTURE DU TEST INTEGRE

Le circuit possède alors deux modes de fonctionnement : le mode normal et le mode test. En mode test, les vecteurs sont générés et appliqués sur les entrées du circuit et les réponses obtenues en sortie sont compactées avant comparaison du résultat de la compaction avec la « signature » attendue (la signature pouvant être stockée ou pas au sein du circuit).

Les générateurs de vecteurs de test implantés dépendent de l'approche de test envisagée :

- Dans le cas d'une *Approche déterministe*, la solution serait de stocker les vecteurs dans une mémoire type ROM, ou d'utiliser une machine d'états finis MEF capable de fournir la séquence attendue. La complexité des séquences (vecteurs spécifiques en grand nombre) rend l'approche souvent inapplicable en pratique.

- Dans le cas d'une *Approche exhaustive*, la méthode la plus simple est d'utiliser les valeurs sortant d'un compteur ou d'un LFSR [McC86] parcourant l'ensemble complet de ces états. Lorsque des séquences exhaustives sont appliquées uniquement sur des sous-ensembles d'entrée on parle alors de test pseudo-exhaustif.

- Dans le cas d'une *Approche aléatoire*, les générateurs employés sont en fait des générateurs pseudo-aléatoires. Une séquence pseudo-aléatoire est une séquence contenant des données vérifiant des propriétés de données aléatoire (e.g nombre de 1 et de 0 quasi-identique) mais reproductible.

La séquence générée étant toujours la même, il est possible de déterminer à l'avance la séquence de sorties correspondant à un circuit exempt de fautes et de calculer un taux de couverture.

Pour la génération d'une séquence pseudo-aléatoire, les méthodes employées sont basées sur l'utilisation de Registres à Décalage à Rétroaction Linéaire (LFSR [Bard87]) ou des automates cellulaires [Hort89, Hort90]. Par abus de langage, on parle de séquences aléatoires même si elles ne sont que pseudo-aléatoires (c'est-à-dire reproductibles à partir d'un état initial donné).

La faible complexité des LFSRs en termes de porte logique et de mise en œuvre rend cette approche particulièrement attractive pour le test intégré.

- Dans le cas d'une *Approche Mixte*, parmi les solutions qui existent nous pouvons citer celles dites de « Reseeding ». Elles utilisent différentes « graines » pour initialiser le générateur ceci afin d'atteindre des vecteurs ciblant les fautes difficiles [Hell95, Kris01, Kris02, Alya03]. Il existe aussi celles dites de « Bit-Flipping » qui consistent à ajouter de la logique entre le générateur pseudo-aléatoire et le circuit afin de permuter certains bits toujours dans le but d'atteindre les vecteurs détectant les fautes difficiles [Wun96, Kie00, Tou01, Fag03].

L'utilisation d'un test intégré présente les avantages suivant :

- le matériel de test est limité,
- il offre la possibilité de tester le circuit à sa vitesse nominale de fonctionnement,
- il offre aussi la possibilité de tester le circuit sur site non seulement en phase de maintenance mais aussi en fonctionnement en utilisant les temps de « dormance » du système (cas des systèmes temps réel en particulier).

La qualité du test, qu'il soit externe ou intégré, dépend principalement de la séquence de test appliquée au circuit. Lorsque la qualité du test n'est pas satisfaisante (faible taux de couverture), deux alternatives s'offrent au concepteur : soit il consacre plus de temps à la génération de la séquence de test, soit il modifie le circuit pour faciliter l'élaboration d'une séquence de bonne qualité. Avec la complexité des circuits actuels (nombre de portes, profondeur séquentielle,…), la deuxième alternative est la plus souvent employée. Les

solutions engendrant une modification du circuit sont regroupées sous le vocable de « techniques de conception en vue du test » (CVT ou DFT).

2.1.3 TECHNIQUES DE CONCEPTION EN VUE DU TEST (CVT OU DFT DESIGN FOR TEST)

Les techniques de conception en vue du test ont pour rôle d'améliorer la « testabilité » du circuit sous test en augmentant sa contrôlabilité et son observabilité.

> La *contrôlabilité* caractérise la facilité de positionner un nœud à une valeur logique prédéfinie à partir des entrées primaires du circuit.
>
> L'*observabilité* représente la facilité de vérifier sur les sorties primaires du circuit la présence d'une valeur donnée sur un nœud.

Evaluer la testabilité du circuit pendant sa phase de conception permet par exemple d'intervenir sur son architecture en ajoutant des entrées ou des sorties pour améliorer le contrôle ou l'observation de certaines parties du circuit.

La conception en vue du test s'appuie sur deux types d'approches : les approches ad-hoc et les approches structurées. Les approches ad-hoc (e.g. ajout de points de contrôle et d'observation) sont adaptées à un circuit donné. Les approches structurées relèvent de règles de conception plus générales. De nombreuses techniques appartenant à l'une ou l'autre de ces approches sont présentées dans la littérature. Dans cette partie, seule l'approche structurée du « scan-path » (chaîne de scan) est présentée, car cette technique est largement utilisée dans le monde industriel et son automatisation est assurée par la plupart des outils.

Cette technique a été développée pour améliorer la testabilité des circuits séquentiels. Comme mentionné plus haut, pour tester une faute sur un nœud donné, ce nœud doit être stimulé avec la bonne valeur et le résultat obtenu doit être propagé vers des nœuds observables. Or générer et appliquer un stimulus ne pose aucun problème dans le cas d'un circuit purement combinatoire mais devient problématique dans le cas d'un circuit séquentiel. En effet, afin d'obtenir le bon stimulus de test sur le nœud donné, il est souvent nécessaire d'appliquer un nombre important de vecteurs de test [Che89] pour amener le circuit dans l'état désiré.

La technique du scan-path transforme les circuits séquentiels en circuits combinatoires pendant la phase de test. Par conséquent, un circuit intégrant la technique de scan-path combine deux modes de fonctionnement : un mode de fonctionnement normal et un mode de

test. En mode test, les éléments internes de mémorisation (bascules) sont interconnectés afin de former un (voire plusieurs) registre(s) à décalage (chaîne de scan). Ainsi dans ce mode test, les vecteurs de test sont chargés en série dans la chaîne avant d'être appliqués à la partie combinatoire.

La mise en œuvre de cette technique se fait dès la phase de conception, en remplaçant les bascules originelles par des bascules scan (voir figure 2-3) et en les reliant entre elles afin de former la chaîne de scan (ou registre à décalage).

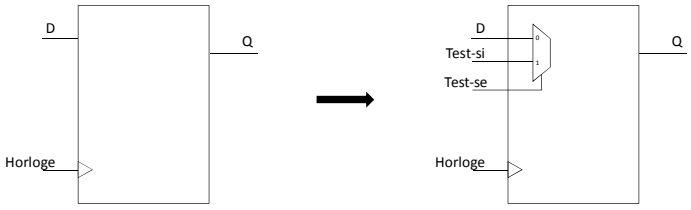


FIGURE 2-3: BASCULE VERS BASCULE SCAN

La différence conceptuelle avec les bascules d'origine repose sur la présence d'un multiplexeur sur l'entrée de donnée D. Un signal de sélection (Test-se) permet de commuter entre le mode de fonctionnement normal (mémorisation de la valeur de l'entrée D) et le mode « scan » (mémorisation de l'état de la bascule précédente dans la chaîne).

La différence conceptuelle avec les bascules d'origine repose sur la présence d'un multiplexeur sur l'entrée de donnée D. Un signal de sélection (Test-se) permet de commuter entre en mode test afin de créer la chaîne de scan.

L'ordre des bascules intervenant dans la chaîne de scan est généralement déterminé pendant la phase de routage. Par conséquent, celle-ci ne prend pas en compte la fonctionnalité du circuit mais uniquement les contraintes de conception.

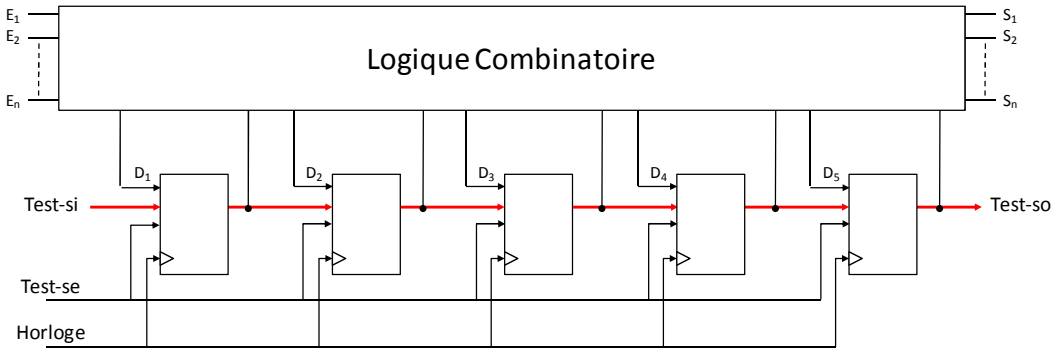


FIGURE 2-4 : ARCHITECTURE GENERALE D'UN CIRCUIT INTEGRANT LA TECHNIQUE SCAN-PATH

La figure 2-4 représente l'architecture d'un circuit intégrant une chaîne de scan. Le test du circuit s'effectue alors en suivant la procédure suivante :

- Positionner le circuit en mode test, Test-se égal 1.
- Entrer par décalage, via le port d'entrée nommé Test-si, le vecteur de test à l'intérieur des bascules et positionner les valeurs de test correspondantes sur les entrées primaires (E_1 … E_n).
- Positionner le circuit en mode normal, Test-se égal 0, et appliquer une impulsion d'horloge système. Les résultats de la partie combinatoire sont alors présents sur les sorties primaires (S_1…S_n) et capturés au sein des bascules.
- Repositionner le circuit en mode test et récupérer par décalage le contenu des bascules sur le port Test-so. Et enfin comparer le résultat avec celui attendu.

  Il est à noter que le chargement et le déchargement de la chaîne de scan peuvent être simultanés.

Cette technique de conception en vue du test est utilisée en test externe pour simplifier la génération automatique des vecteurs (les bascules sont alors considérées comme des pseudo-entrées/sortie, le circuit à tester est combinatoire), et en test intégré afin de fournir un test de qualité.

Dans la partie suivante, nous présenterons le conflit entre le test et la sécurité, ainsi que l'importance du choix de la technique de test.

2.2 LES ATTAQUES SPECIFIQUES

2.2.1 TEST ET SECURITE : LE CONFLIT

Dans le contexte des circuits sécurisés, une erreur de fonctionnement peut engendrer une faille sécuritaire mettant en danger la sécurité de l'application réalisée par le circuit. De récentes techniques d'attaques ont montré que des résultats de chiffrement erronés peuvent être exploités pour retrouver la clef secrète avec un coût de mise en œuvre négligeable [Mor06]. Le test de production des circuits sécurisés s'avère donc primordial pour s'assurer de fournir au client un circuit sans défaillance. Le test de production de ces circuits peut donc être vu comme un élément contribuant à assurer un bon niveau de sécurité.

Les principes fondamentaux de la conception en vue du test sont d'augmenter la contrôlabilité et l'observabilité du circuit sous test. Or ces principes sont contradictoires avec les impératifs de la sécurité qui doivent réduire au maximum l'accès aux données internes du circuit. En effet, accroître l'observabilité d'un circuit, par exemple en ajoutant des points de test, peut créer de nouvelles possibilités d'accès pour récupérer les données internes, qui devraient être inaccessibles en mode fonctionnel. De même, l'augmentation de la contrôlabilité peut offrir des opportunités aux attaquants de mettre la puce dans une configuration permettant de simplifier la réalisation de l'attaque.

Ce conflit entre la conception en vue du test et la sécurité pose plus particulièrement problème dans le cas d'une approche de test externe. En effet, pour une telle approche les données de test sont échangées avec le testeur donc avec l'extérieur, tandis qu'avec une approche de test intégré les données de test restent au sein du circuit, seules les commandes de lancement du mode test sont accessibles. Dans ce dernier cas, l'utilisateur, éventuellement l'attaquant, n'a pas le choix des stimuli appliqués.

En résumé, dans le contexte des circuits sécurisés, le test (la conception en vue du test) peut être vu comme une « porte ouverte » sur le circuit pour les attaquants et ainsi leur permettre de créer de nouvelles attaques ou bien de faciliter des attaques déjà connues.

Afin de mieux appréhender les attaques développées sur les cœurs de chiffrement, nous présentons d'abord les risques induits par la technique de scan-path en termes de contrôlabilité et d'observabilité. En activant le mode test, le contenu des bascules de la chaîne de scan peut être chargé et/ou déchargé. Un attaquant activant illégalement ce mode peut donc observer par exemple l'état du circuit puisqu'il a accès aux données mémorisées au sein des bascules et ces données peuvent fournir des informations sur les données secrètes traitées par le cœur de chiffrement. Il a été démontré dans [Yan04] et [Yan05] qu'un attaquant peut recalculer la clef secrète seulement avec ces données récupérées au sein de la chaîne de scan. Une telle attaque sur le cœur de chiffrement AES sera présentée dans le sous-chapitre suivant.

D'un point de vue de la contrôlabilité, la technique du scan augmente les risques. En effet, celle-ci permet de contrôler l'état des nœuds internes du circuit, ce qui donne la possibilité, pour un attaquant, de positionner le circuit dans un état désiré. Cet état peut être un état atteignable en fonctionnement normal ou un état qui ne doit jamais être atteignable.

2.2.2 ATTAQUES PAR SCAN (DES, AES)

Ces attaques ont été présentées dans [Yan04] et [Yan05] respectivement pour des cœurs de chiffrement DES et AES. L'attaque sur un AES est décrite ci-dessous.

- *Principe et connaissance de l'attaquant*

Le but d'une telle attaque est de retrouver la clef secrète en utilisant la chaîne de scan pour observer à différents moments les données traitées par le circuit. Il est supposé que le registre mémorisant la clef secrète n'est pas inclus dans la chaîne de scan.

Cette attaque se déroule en deux étapes. La première consiste à analyser la structure de la chaîne de scan afin d'identifier les bascules « sensibles » c'est-à-dire porteuses d'information secrète. La seconde vise à déterminer la clef de chiffrement en appliquant des textes en clair et en observant les textes chiffrés correspondants.

Un certain nombre d'hypothèses de départ sont requises :

- L'attaquant connaît l'algorithme AES, car celui-ci est un algorithme public.
- L'attaquant n'a pas accès ni aux clefs de rondes, ni à la clef secrète ; mais le registre de sortie est accessible par la chaîne de scan.
- L'attaquant a un accès direct aux ports liés à la chaîne de scan : test-se, afin de pouvoir commuter du mode test au mode normal quand il le souhaite, et test-so pour observer les données sortant de la chaîne de scan (figure 2-4).
- L'attaquant ne connaît pas la structure de la chaîne de scan (c'est-à-dire l'ordre des bascules dans la chaîne de scan).
- La complexité de l'AES est réduite à une seule ronde (plus précisément à la première ronde de chiffrement).

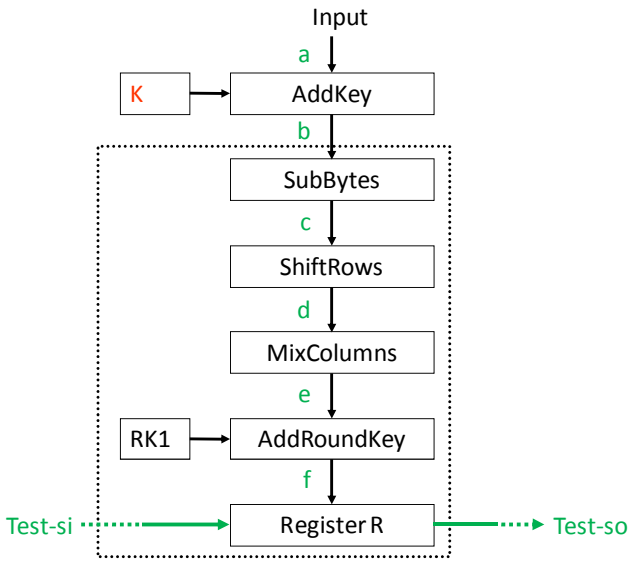


FIGURE 2-5 : RONDE DE CHIFFREMENT DE L'AES

- *Implantation de l'AES visé par l'attaquant figure 2-5*

Dans la suite, le texte en clair (ou l'entrée) sera noté *a*. Le résultat de l'opération d'« ou-exclusif » entre la clef secrète K et *a* sera noté *b*. Le résultat de l'opération SubBytes sera noté *c*. Pour rappel, cette opération est une substitution non-linéaire d'octets à l'aide d'une table appelée S-Box. Le résultat de l'opération ShiftRows sera noté *d*. Cette opération est un décalage circulaire des octets. Le résultat de l'opération MixColumns sera noté *e*. C'est une opération de multiplication entre *d* et une matrice d'éléments connus. L'opération AddRoundKey est une opération d'« ou-exclusif » entre la matrice de données *e* et la clef de ronde. Le résultat de cette opération est noté *f*. Les bascules constituant le registre de sortie R appartiennent à la chaîne de scan.

- *Première étape de l'attaque : Déterminer la structure de la chaîne de scan*

L'entrée *a* est partitionnée en 16 octets a_{00}, …, a_{33} (voir tableau 2-1).

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

TABLEAU 2-1 : REPRESENTATION MATRICIELLE DE L'ENTREE a

Il s'agit d'observer les modifications dans le registre de sortie R, à fortiori dans la chaîne de scan, engendrées par le changement d'un octet en entrée. Supposons que le changement se produise sur l'octet a_{11}, le tableau 2-2 représente l'impact de cette modification au travers des opérations d'une ronde.

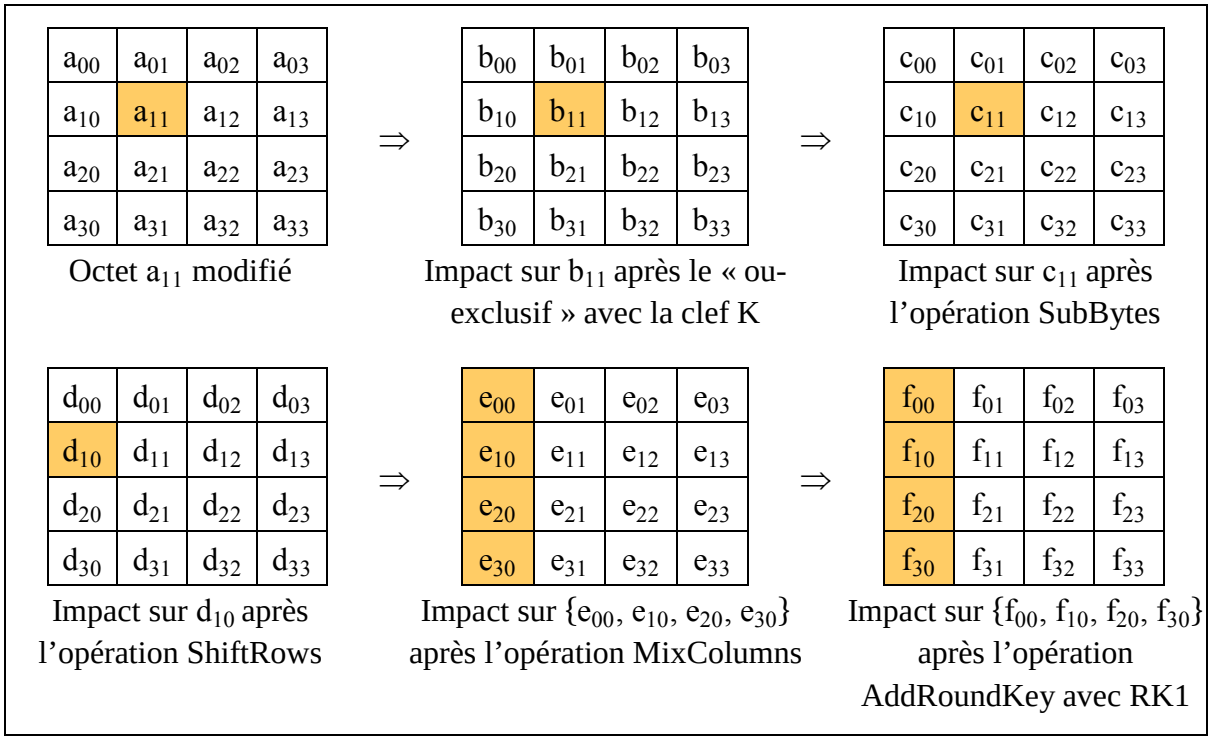


TABLEAU 2-2 : IMPACT DE LA MODIFICATION DE L'OCTET a_{11}

En résumé, la modification d'un octet dans le mot d'entrée (a_{11}) induit une modification de 4 octets (f_{00}, f_{10}, f_{20}, f_{30}) après l'exécution d'une ronde. En analysant le flot de bits sortants, la première partie de l'attaque localise dans la trame de la chaîne de scan, les 32 bascules (du registre R) correspondantes au stockage de (f_{00}, f_{10}, f_{20}, f_{30}).

La procédure permettant de localiser ces 32 bascules est la suivante :

- Initialiser la puce (faire un reset de celle-ci), puis la faire fonctionner en mode normal pendant un cycle d'horloge. Ainsi, le texte en clair (l'entrée) est traité jusqu'à la fin de la première ronde et le résultat (vecteur 1) est stocké dans le registre R. Commuter en mode test et sortir par décalage le résultat contenu dans la chaîne de scan.

- Répéter l'étape précédente en envoyant un texte en clair différent sur un et un seul octet par rapport au précédent et sortir le vecteur 2. Répéter cette étape et sauvegarder les vecteurs 3, 4, …, 256.

Les bits, dont les valeurs diffèrent dans les vecteurs 1 et 2, correspondent à des bascules du registre R. De même, en comparant le vecteur 1 avec le vecteur 3, d'autres bascules appartenant au registre R peuvent être identifiées. Ce processus est répété tant que les 32 bascules du registre R correspondantes au stockage de (f_{00}, f_{10}, f_{20}, f_{30}) n'ont pas été identifiées. Dans le pire des cas, 256 vecteurs devront être appliqués (un octet => 2^8 combinaisons) et 255 comparaisons devront être effectuées.

Cependant, dans le cas des algorithmes de chiffrement, la modification d'un bit en entrée de ronde se traduit par la modification de plusieurs bits en sortie de ronde (principe de la diffusion) [Bih97]. Ainsi en pratique, avec peu de vecteurs, les 32 bascules de R peuvent être localisées dans le flot de bits sortant. Par simulation d'une description en langage C de l'algorithme AES, il a été montré que l'application de 15 vecteurs dans le plus mauvais des cas et de 6 vecteurs en moyenne, permet de localiser les 32 bascules ciblées de R correspondant au stockage de (f_{00}, f_{10}, f_{20}, f_{30}) dans le flot des bits de sortie.

A la fin de l'étape un, l'attaquant peut identifier dans le flot de bits sortants, les bits correspondants aux 32 bascules ciblées du registre R (voir illustration sur la figure 2-6).

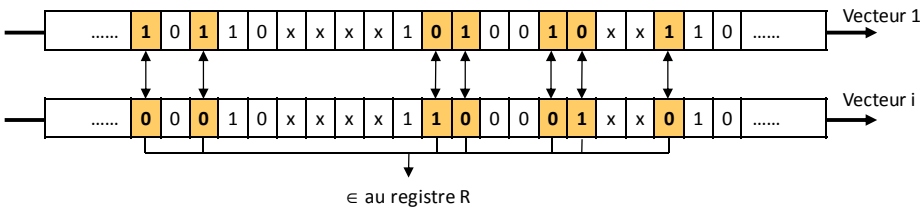


FIGURE 2-6 : PREMIERE ETAPE DE L'ATTAQUE DE LA CHAINE DE SCAN

Néanmoins, l'attaquant ne connaît pas l'ordre de ces bascules, c'est-à-dire qu'il ne peut pas retrouver la valeur de (f_{00}, f_{10}, f_{20}, f_{30}).

- *Deuxième étape : Retrouver la clef secrète K*

Pour rappel, la première opération d'« ou-exclusif » et les opérations de rondes sont exécutées en un cycle d'horloge dans notre exemple. Les deux clefs K (clef secrète) et RK1 (première clef de ronde) interviennent dans le résultat observé via la chaîne de scan. La deuxième étape de l'attaque vise à déterminer la valeur de b_{11} et puis à calculer la valeur de l'octet K_{11} de la clef secrète comme étant le résultat de $b_{11} \oplus a_{11}$ puisque $b_{11} = K_{11} \oplus a_{11}$. Tous les octets de la clef secrète K pourront être obtenus par répétition de cette étape.

Déduire la valeur de b_{11} à partir de (f_{00}, f_{10}, f_{20}, f_{30}) pose deux problèmes. Premièrement la valeur au point f est déterminée par b_{11} et la clef de ronde RK1 est inconnue jusque là. Deuxièmement la valeur exacte de (f_{00}, f_{10}, f_{20}, f_{30}) est inconnue puisque l'attaquant connaît l'ensemble des bits de ces 4 octets mais pas l'ordre. L'approche consiste à annuler l'effet de RK1. Comme le montre la figure 2-5, RK1 n'intervient que pour l'opération d'« ou-exclusif » avec le résultat de l'opération de MixColumns *e*. L'idée est de trouver une relation entre *e* et *f* indépendante de RK1.

En appliquant deux valeurs différentes b_{11}^1 et b_{11}^2, les mots de 4 octets $e^1=(e_{00}^1, e_{10}^1, e_{20}^1, e_{30}^1)$ et $e^2=(e_{00}^2, e_{10}^2, e_{20}^2, e_{30}^2)$ sont obtenus au point *e* et les mots de 4 octets $f^1=(f_{00}^1, f_{10}^1, f_{20}^1, f_{30}^1)$ et $f^2=(f_{00}^2, f_{10}^2, f_{20}^2, f_{30}^2)$ au point *f*. Il est alors possible d'annuler l'effet de RK1.

Soient $f_{ij}=e_{ij} \oplus RK1_{ij}$ avec j = 0 et i valant 0, 1, 2 ou 3. Nous pouvons alors noter que :

$$f_{i0}^1 = e_{i0}^1 \oplus RK1_{i0} \qquad \text{et} \qquad f_{i0}^2 = e_{i0}^2 \oplus RK1_{i0} \qquad (0 \le i \le 3).$$

En faisant une opération d'« ou-exclusif » entre les deux termes obtenus en f, et sachant que seuls les octets d'indice 0i (i=0, 1, 2, ou 3) peuvent différés ente f^1 et f^2, nous obtenons :

$$\begin{aligned} f^1 \oplus f^2 &= f_{i0}^1 \oplus f_{i0}^2 \\ &= (e_{i0}^1 \oplus RK1_{i0}) \oplus (e_{i0}^2 \oplus RK1_{i0}) \\ &= (e_{i0}^1 \oplus e_{i0}^2) \oplus (RK1_{i0} \oplus RK1_{i0}) = e_{i0}^1 \oplus e_{i0}^2 \\ &= e^1 \oplus e^2. \end{aligned}$$

Ainsi, le nombre de 1 dans $f^1 \oplus f^2$ est identique au nombre de 1 dans $e^1 \oplus e^2$. De plus, le nombre de 1 dans $f^1 \oplus f^2$ est indépendant de la valeur de l'octet $RK1_{i0}$ de la clef de ronde. Il est aussi uniquement déterminé par la valeur de b_{11}^1 et de b_{11}^2. Un attaquant peut donc déduire b_{11}^1 et b_{11}^2 par analyse du nombre de 1 dans $f^1 \oplus f^2$.

Les auteurs proposent d'utiliser comme entrée le couple $(a_{11}^1=2t, a_{11}^2=2t+1)$ ou t est compris entre 0 et 127. Ce choix permet d'avoir des valeurs d'entrée qui sont différentes uniquement sur le bit de poids faible (LSB : Least Significant Bit).

Or $b_{11}^1=2t \oplus K_{11}$ et $b_{11}^2=2t+1 \oplus K_{11}$ seront aussi différents uniquement sur le bit de poids faible. Nous pouvons donc noter que le couple (b_{11}^1, b_{11}^2) sera de la forme (2m, 2m+1) ou

$(2m+1, 2m)$ avec $0 \leq m \leq 127$. Si différents couples (b_{11}^1, b_{11}^2) peuvent générer le même nombre de 1 dans $f^1 \oplus f^2$, alors le nombre de 1 ne représente pas un couple unique d'entrée de (b_{11}^1, b_{11}^2).

Dans le cas de l'algorithme AES, quand l'entrée b_{11} prend pour valeur le couple $(2m, 2m+1)$ ou $(2m+1, 2m)$, le nombre de 1 dans $f^1 \oplus f^2$ varie entre 7 et 25.

De plus les auteurs ont montrés qu'il y avait 4 cas où le nombre de 1 dans $f^1 \oplus f^2$ conduisait à un couple unique d'entrée (b_{11}^1, b_{11}^2). Ces 4 cas sont présentés dans le tableau 2-3.

Nombre de 1	9	12	23	24
Couple appliqué sur b_{11}	226, 227	242, 243	122, 123	130, 131

TABLEAU 2-3 : ASSOCIATION NOMBRE DE 1 DANS $f^1 \oplus f^2$ ET COUPLE SUR b_{11}

Ainsi, la procédure permettant de déterminer la valeur d'un octet de la clef secrète K est la suivante :

- Appliquer la valeur $2t$ $(0 \leq t \leq 127)$ sur a_{11} et faire fonctionner l'AES en mode normal pendant un cycle d'horloge. Repasser en mode test et sortir le flot de bits (le vecteur) contenu dans R et déterminer f^1.
- Réinitialiser la puce. Appliquer $2t+1$ $(0 \leq t \leq 127)$ sur a_{11} et faire fonctionner la puce en mode normal pendant un cycle d'horloge. Repasser en mode test et déterminer f^2.
- Si le nombre de 1 dans $f^1 \oplus f^2$ est égal à 9, 12, 23 ou 24, déterminer b_{11}^1 et b_{11}^2 à l'aide du tableau 2-3. Autrement, repartir au premier point et appliquer une nouvelle valeur $2t$ sur a_{11}.
- Calculer l'octet de la clef secrète K, tel que $K_{11} = b_{11} \oplus a_{11}$.
- Si tous les bits de la clef secrète K sont déterminés, arrêter. Sinon repartir à l'étape 1 de l'attaque en ciblant un autre octet a_{ij}.

Par simulation, il a été montré que pour obtenir un des 4 couples sur b_{11} permettant de recalculer K_{11} parmi les 128 couples possibles $(0 \leq t \leq 127)$, il faut appliquer en moyenne 32 couples sur a_{11}, et dans le pire des cas il faut en appliquer 124.

- *Efficacité de l'attaque*

En résumé, l'étape 1 de l'attaque requiert en moyenne 6 vecteurs pour localiser la position d'un groupe de 32 bascules dans le flot de bits sortant de la chaîne de scan. Cela revient à appliquer 24 vecteurs $((128 \div 32) \times 6)$ pour localiser les 128 bascules du registre de ronde R.

Concernant l'étape 2, en moyenne 32 paires de vecteurs doivent être appliquées afin de déterminer la valeur d'un octet de la clef secrète K. Pour déterminer l'ensemble des octets de la clef secrète, 512 paires de vecteurs devront ainsi être appliquées $((128 \div 8) \times 32)$. Donc au total, pour déterminer la clef secrète d'un AES, 1048 vecteurs $(24+(512 \times 2))$ devront être appliqués en moyenne.

De même, nous pouvons évaluer la durée de l'attaque en termes de cycles d'horloge. L'étape 1 de l'attaque requiert 774 cycles d'horloge (1 cycle pour l'opération normale et 128 pour décharger la chaîne de scan en supposant que celle-ci ne contienne que les bascules du registre R et ceci multiplier par 6) pour retrouver le contenu de 32 bascules dans le flot de la chaîne de scan. Ainsi pour retrouver l'ensemble des 128 bascules du registre, il faut 3096 cycles d'horloges. Pour l'étape 2 de l'attaque il faut 8256 cycles d'horloges (32 vecteurs multipliés par 258 cycles) pour déterminer la valeur d'un octet de la clef secrète. Ainsi pour retrouver l'ensemble des octets de la clef, 132 096 cycles d'horloges sont requis.

Au final, une telle attaque va imposer l'application de 1048 vecteurs en moyenne et le fonctionnement du circuit pendant au moins 135 192 cycles d'horloges ce qui est tout à fait acceptable pour retrouver une clef secrète parmi les 2^{128} valeurs possibles.

L'attaque sur le cœur de chiffrement DES se déroule également en deux étapes : dans la première, la structure de la chaîne de scan est déterminée et dans la seconde ce sont les clefs de rondes qui le sont. La clef secrète est ensuite obtenue à partir des clefs de rondes. Pour réussir l'attaque il est nécessaire d'appliquer 41 775 cycles d'horloges afin de découvrir la clef utilisateur, mais seulement trois textes en clair doivent être appliqués.

2.2.3 CONTRE-MESURES

Le test d'un circuit, intégrant une ou plusieurs chaînes de scan, est basé sur une bonne gestion des deux modes de fonctionnement : le mode normal et le mode de test. Pour gérer le test, un contrôleur de test est implanté. Ce contrôleur, implanté sous forme de machine d'états finis, gère ainsi la totalité de la séquence de test. D'un point de vue sécuritaire, le contrôleur

de test agit comme un filtre entre les ports du circuit et la chaîne de scan. Lorsque, par exemple, il est désactivé, il s'avère impossible d'utiliser la chaîne de scan. Or nous avons vu précédemment que l'attaquant a besoin de commuter du mode test au mode normal et vice-versa.

Par conséquent, l'attaquant a deux possibilités pour activer la chaîne de scan :

- utiliser le contrôleur de test (comme le font les ingénieurs de test en production)
- utiliser des micro-sondes permettant de court-circuiter le contrôleur de test.

Ainsi, les contre-mesures proposées vont consister d'une part à sécuriser le contrôleur de test pour qu'une utilisation frauduleuse de celui-ci ne permette pas d'accéder aux données confidentielles et d'autre part à empêcher l'exploitation de la chaîne de scan par une attaque visant à court-circuiter le contrôleur de test.

Les contre-mesures sont classées en deux groupes. Nous les présenterons rapidement et en détaillerons une en particulier pour chacun des groupes :

- *1er groupe de contre-mesures* : Protection du mode Test

  - Scan protocol [Hel05]

    Le contrôleur de test est modifié afin que le circuit soit initialisé quand il y a commutation vers ou depuis le mode test. L'initialisation est vérifiée avant d'accéder à un mode, et toute commutation entre les modes qui essaie de contourner la phase d'initialisation doit être détectée.

  - Test Patterns watermarking [Hel06bis, Lee06]

    Les deux solutions reposent sur le même principe d'insertion des bits d'identification dans les vecteurs de test. La solution de [Hel06bis] place les bits d'identification en début de vecteur de test. En revanche dans la solution de [Lee06] les bits d'identification sont répartis dans tout le vecteur de test. En mode test, ces bits sont comparés à ceux attendus. S'ils sont identiques, la procédure de test se poursuit, sinon dans [Hel06bis] la procédure de test s'arrête, et dans [Lee06], la procédure se poursuit mais délivre des données aléatoires en sortie.

- Mirror Key Register [Yan05]

Cela correspond à la création de deux sous-modes, un dit sécurisé et un dit non-sécurisé. En mode non sécurisé, la puce peut commuter entre le mode test et le mode de fonctionnement normal. Une ou plusieurs clefs de chiffrement sont utilisées pour tester le circuit, il ne s'agit pas de la clef utilisée en mode de fonctionnement normal. En mode sécurisé seul le fonctionnement en mode de fonctionnement normal est accessible. Un « mirror key register » (MKR) est implanté pour mémoriser temporairement les clefs utilisées en mode test et de fonctionnement normal (voir illustration figure 2-7). En mode sécurisé (mode normal) la clef est stockée dans ce registre tandis qu'en mode non sécurisé (mode test), la clef est isolée du reste du circuit par ce registre « miroir ».

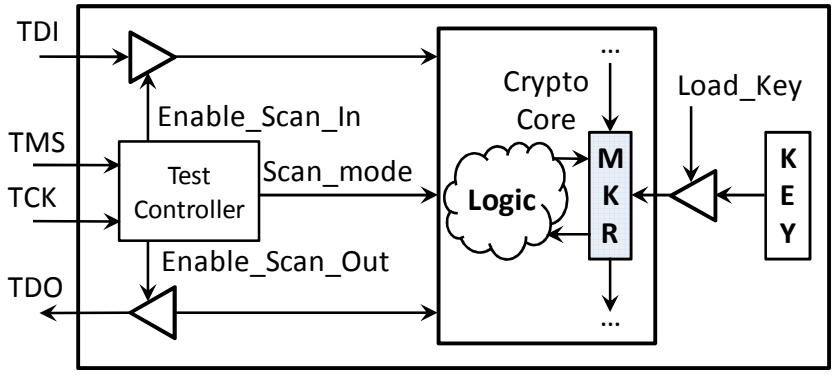


FIGURE 2-7 : ARCHITECTURE DE SCAN SECURISE BASE SUR LE MKR

Une telle sécurité ne protège pas contre les attaques de type micro-sondes, car si un attaquant fait commuter le signal de sélection scan-enable (Test-se) alors que le système est en mode normal (sans passer par le mode test), la clef secrète sera stockée dans le MKR et donc observable. Par exemple, en déchargeant plusieurs fois la chaîne de scan, il sera possible de constater que 128 bits ne changent jamais de valeur et donc en conclure qu'ils appartiennent à la clef secrète.

Les contre-mesures ci-dessus visent à protéger le circuit contre un usage illégal du mode test.

- *2ième groupe de contre-mesures* : Protection du mode normal

- Scan chain scrambling [Hel04]

Contrairement aux solutions présentées précédemment, cette technique n'a pas pour rôle d'empêcher l'activation de la chaîne de scan mais uniquement de rendre

les données inexploitables quand la chaîne n'a pas été activée en suivant le protocole de test attendu. Rappelons ici que la base des attaques de scan est d'observer plusieurs déchargements de la chaîne de scan et de comparer les données lues.

La solution du « scan chain scrambling » consiste à rendre la comparaison bit à bit entre deux déchargements inexploitable par l'attaquant. Le principe repose sur la modification continuelle de l'ordre de lecture des bascules de la chaîne de scan durant le mode normal. Ainsi une comparaison bit à bit n'a plus de sens.

Sur la figure 2-8 est illustré le principe de cette contre-mesure. Un segment représente un ensemble de bascules. En mode de test, l'ordre des segments est fixé afin de pouvoir appliquer les vecteurs de test et lire les réponses. En mode de fonctionnement normal, l'ordre des segments change aléatoirement à chaque cycle d'horloge si la chaîne est déchargée (lecture sur le port de sortie scan out).

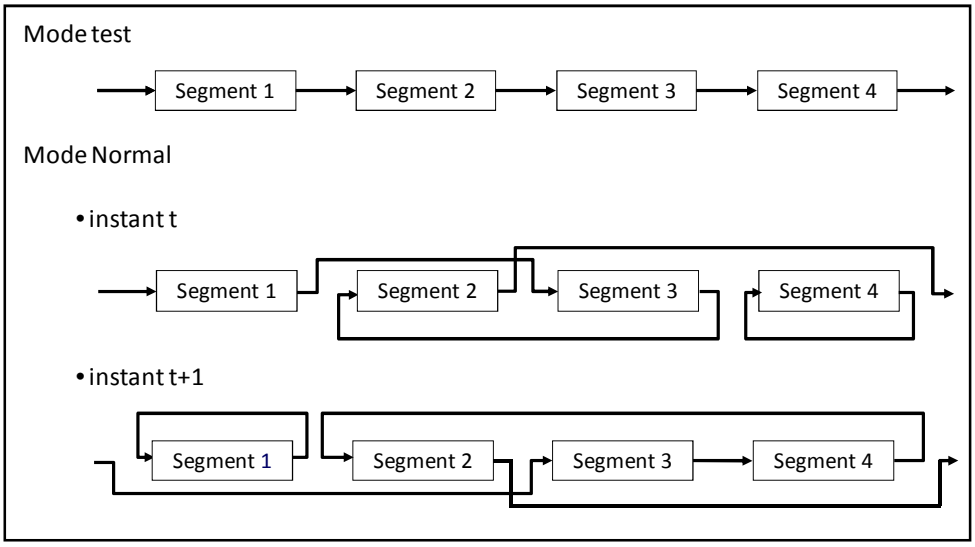


FIGURE 2-8 : SCRAMBLING DE LA CHAINE DE SCAN

Nous pouvons voir sur la figure 2-8 que, par exemple à l'instant t, le segment 1 est connecté au segment 3 puis au segment 2, le segment 4 n'est connecté qu'à lui-même. Et à l'instant t+1, le segment 3 est connecté au segment 4 puis au segment 2, le segment 1 n'est connecté qu'à lui-même. Ainsi, si le résultat obtenu à l'instant t est comparé à celui obtenu à l'instant t+1, l'attaquant va essayer de comparer les valeurs du segment 1 avec celle du segment 3, celle du segment 3 avec celle du segment 4 etc…

En résumé, avec cette contre-mesure, l'attaquant ne pourra pas exploiter les données sortantes de la chaîne de scan.

- Scan-enable tree protection [Hel05]

  La technique de « scan-enable tree protection » consiste à détecter une commutation illégale de la valeur du signal de sélection Test-se (Test-scan-enable). Afin de détecter toutes les attaques possibles, la valeur du signal scan-enable doit donc être contrôler à plusieurs niveaux de l'arbre de scan-enable (racine et branches). Si l'ensemble des valeurs observées (racine et branches) sont égales, alors il n'y a pas de problème. Par contre dans l'autre cas nous pouvons en déduire que le circuit est attaqué. Si une attaque est détectée alors le circuit est réinitialisé.

- Spy FFs [Hel06]

  La solution dite de « spy FFs» consiste à détecter une activation illégale de la chaîne de scan en observant le contenu de bascules « espionnes ». Ces bascules sont insérées dans la chaîne de scan. Leur entrée fonctionnelle D est figée à 0 ou à 1. A la sortie de la chaîne de scan, un test est fait sur la sortie de ces bascules « espionnes ». Si la valeur de la sortie ne correspond pas à la valeur de l'entrée D, le signal scan-enable a été activé. Si cela arrive alors que le système est en mode de fonctionnement normal, le système est immédiatement réinitialisé car une utilisation non autorisée de la chaîne de scan a été détectée.

Ces contre-mesures visent à protéger le circuit contre une attaque utilisant des micro-sondes pour activer le mode de décalage de la chaîne de scan.

Cinq critères peuvent être définis pour comparer ces différentes solutions. Ces critères de comparaisons sont : l'insertion dans le flot de conception, le temps de test, la surface additionnelle, la puissance consommée et le niveau de sécurité. Les résultats de la comparaison sont fournis dans le tableau 2-4. La solution de protection « Scan protocol » n'est pas présentée de façon individuelle dans le tableau, car pour assurer un bon niveau de sécurité l'auteur de cette contre-mesure conseille de l'utiliser avec une autre contre-mesure ciblant les attaques court-circuitant le contrôleur.

	Test Patterns watermarking		Mirror Key Register	Scan chain scrambling	Scan-enable tree protection	Spy FFs
	[Hel06bis]	[Lee06]	[Yan05]	[Hel04]	[Hel05]	[Hel06]
Circuit étudié	DES	ISCAS'89	AES	DES	DES	DES
Modification majeure	4Bits / vecteurs de test	10Bits/ vecteurs de test	128 bascules sont ajoutées	Chaîne de scan divisé en 6 segments	8 points de l'arbre de scan_enable sont contrôlés	6 bascules « espion » sont insérées
Insertion dans le flot de conception	RTL	RTL	RTL	RTL	RTL + place&route	RTL
Temps de test	0.4%	1.6%	≈ 0%	0%	1%	5%
Surface additionnelle	≈ 0%	1.5%	1.32%	0.2%	0.3%	1.8%
Puissance consommée	0%	0%	0%	7%	0%	0%
Niveau de sécurité	+	+	+	+++	++	++

TABLEAU 2-4 : COMPARATIF DES DIFFERENTES CONTRE-MESURES

Le niveau de sécurité, tel qu'on l'entend ici, représente la difficulté qu'a un attaquant à court-circuiter la contre-mesure.

La solution « Mirror Key Register » ne visant pas à contrer une attaque court-circuitant le contrôleur de test a par conséquent un niveau de sécurité faible. Concernant les approches de « test patterns watermarking », le niveau de sécurité est dit faible car en observant les données traitées un attaquant peut s'apercevoir que certains bits ne changent jamais de valeur et qu'en les intégrant à ces données il pourra alors mener à terme son attaque. Pour les contre-mesures « Scan enable tree protection » et « Spy FFs », l'attaquant peut les court-circuiter en bloquant le signal permettant de réinitialiser le circuit. Bien sûr cela demande une bonne connaissance du circuit et ne peut être réalisé sans avoir accès à l'ensemble des données du circuit. La solution de « Scan chain scrambling » est la plus complexe à court-circuiter pour un attaquant puisque elle lui demande de bloquer le module contrôlant les connexions des différents segments. De plus il n'est pas évident que l'attaquant s'aperçoive rapidement de l'existence de l'attaque.

En résumé, tester un circuit contenant des chaînes de scan avec un test externe nécessite l'implantation de contre-mesures, afin de ne pas nuire à la sécurité du circuit.

2.3 CHOIX DE L'APPROCHE DE TEST : LE TEST INTEGRE BIST

Dans le contexte des circuits sécurisés, utiliser une approche de test externe combiné avec la technique de « scan path » engendre une faille sécuritaire exploitable par un attaquant pour retrouver la clef secrète de chiffrement. Cette faille est due à l'échange des données internes au circuit avec l'extérieur.

Par conséquent, les techniques de test intégré semblent plus appropriées au test de ces circuits car elles n'impliquent aucun échange de données avec l'extérieur. Par contre, ces techniques engendrent un surcoût en surface (voir figure 2-9).

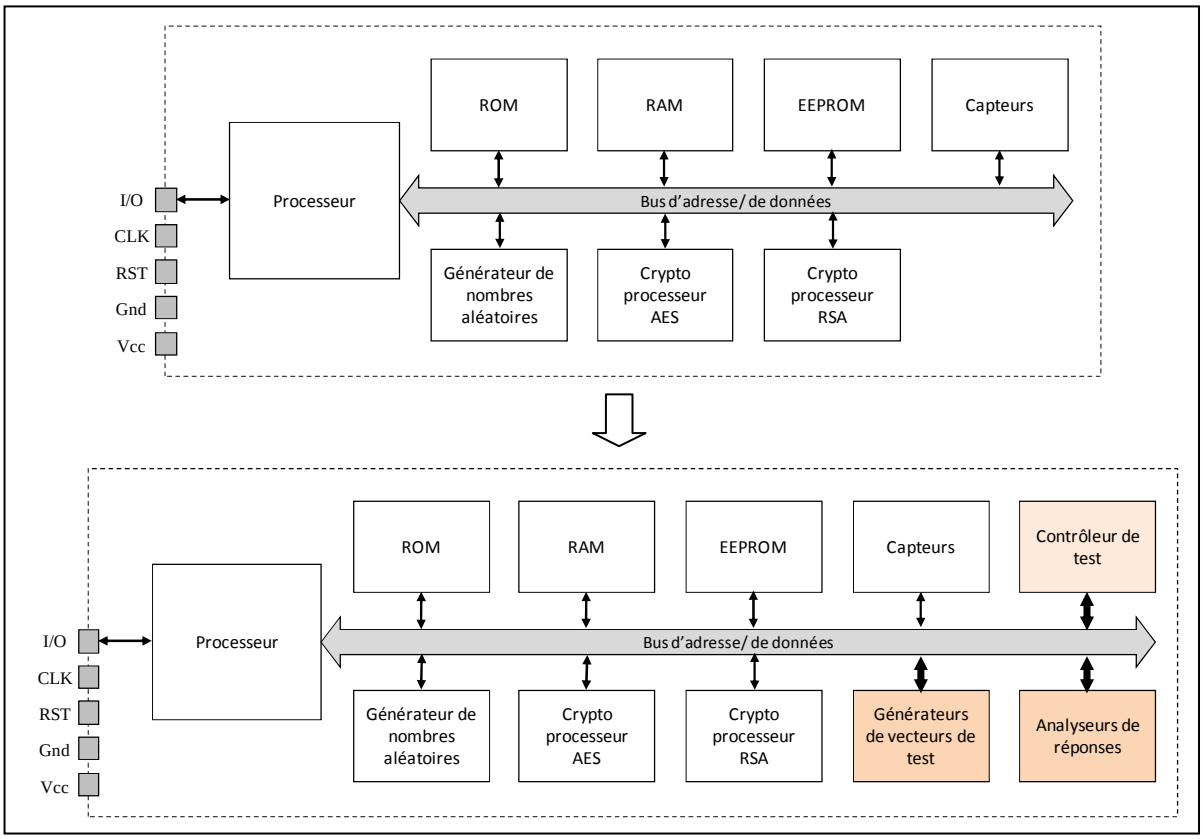
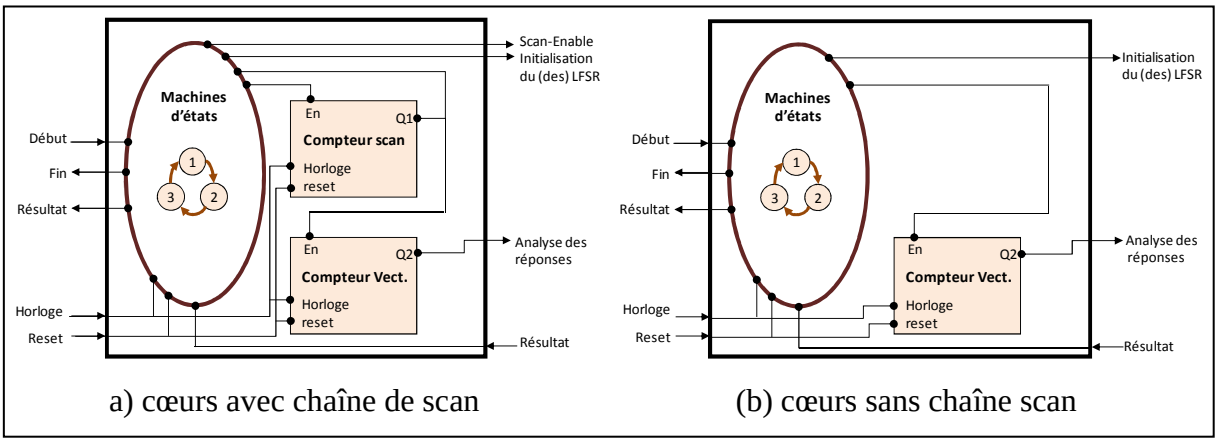


FIGURE 2-9 : IMPLANTATION DU BIST

En effet, cette technique implique l'insertion de ressources de test, pour rappel ces ressources sont :

- un générateur de vecteurs de test (typiquement un LFSR),
- un analyseur de réponses (typiquement un MISR [Kone80])
- un contrôleur de test, c'est une machine d'états gérant les signaux nécessaires à l'activation du test (figure 2-10).

Ce contrôleur est notamment responsable de la gestion des cycles d'horloge nécessaires à la phase de test (nombre de cycles nécessaires au décalage des données dans une chaîne de scan le cas échéant, nombre de cycles nécessaires pour appliquer l'ensemble des vecteurs de test dans tous les cas).



a) cœurs avec chaîne de scan (b) cœurs sans chaîne scan

FIGURE 2-10 : CONTROLEUR DE TEST

- ainsi que la connectique (fil, multiplexeurs) permettant d'intégré ces diverses ressources dans le circuit.

L'augmentation de surface liée à cette approche a l'effet néfaste d'augmenter les probabilités d'apparition d'un défaut, puisque la présence de défauts augmente avec la surface de silicium utilisée. Donc une telle approche n'est viable que si la surface supplémentaire requise est très faible par rapport au circuit sous test.

2.4 CONCLUSION

Dans ce chapitre, nous avons présenté la technique de test externe basée sur le « scan-path ». Cette technique est couramment employée dans l'industrie car elle permet d'augmenter la contrôlabilité et l'observabilité des nœuds internes d'un circuit séquentiel. Cependant dans le cas des circuits sécurisés, elle peut engendrer une faille sécuritaire. Le déchargement de la chaîne de scan se faisant via un port externe du circuit, les états internes du circuit peuvent être observés par toute personne capable d'activer le déchargement de la chaîne de scan. Ainsi, l'utilisation d'une technique de test externe requiert l'implantation de contre-mesures coûteuses pour pallier les attaques potentielles.

La technique de test intégré apparait comme la meilleure solution de test pour les circuits sécurisés. Cependant, l'inconvénient de cette technique est de nécessiter l'implantation de

matériel supplémentaire. Les travaux réalisés durant cette thèse ont eu pour objectif de réduire le surcoût en surface additionnelle lié à l'utilisation de cette technique. Ces solutions sont basées sur l'utilisation détournée de cœurs de chiffrement déjà implantés au sein du système sous test.

Chapitre 3 : La génération pseudo-aléatoire

Chapitre 3 : LA GENERATION PSEUDO-ALEATOIRE

Dans le chapitre précédent, nous justifions le développement d'une solution de test intégré pour des dispositifs de cryptographie. Ce chapitre est consacré à l'étude d'une solution permettant de réduire le coût d'implantation d'un générateur de vecteurs de test dans ce contexte. Cette solution consiste à utiliser le cœur de chiffrement déjà implanté dans le circuit comme générateur de vecteurs de test.

Après une brève introduction nous présenterons l'architecture d'un cœur de chiffrement utilisable comme générateur de vecteurs de test pseudo-aléatoire. Nous étudierons le caractère aléatoire des séquences ainsi générées. Enfin nous validerons ces séquences en tant que séquence de vecteurs de test.

3.1 INTRODUCTION

Le test intégré nécessite d'implanter un générateur de vecteurs de test dans le circuit. Afin de réduire le coût, les générateurs implantés sont généralement des générateurs pseudo-aléatoires qui requièrent peu de surface additionnelle, par exemple : des Automates Cellulaires (CA pour « Cellular Automata ») [Hort89, Hort90] ou des Registres à Décalage à Rétroaction Linéaire (LFSR) [Bard87]. Par défaut, on considère que ces générateurs délivrent une succession de 0 et de 1 avec, à chaque instant, une probabilité de ½ d'obtenir l'un ou l'autre. Ainsi, la séquence obtenue est une suite de réalisations de loi de Bernoulli indépendantes de paramètres p=1/2 (voir [Sap90]).

Contrairement aux séquences de test déterministes, ces séquences (pseudo-)aléatoires ne sont pas spécifiques à un circuit sous test donné.

Une génération pseudo-aléatoire est un procédé qui, à partir d'un état d'initialisation appelé *graine* ou *germe*, engendre de manière déterministe une suite de valeurs (des vecteurs de test dans notre cas) qui ne peut être distinguée d'une suite aléatoire quand la graine n'est pas connue.

Pour mettre en évidence le caractère aléatoire d'une séquence, nous nous attarderons plus spécifiquement sur :

- La distribution des éléments, ici des bits (0 et 1). Il faut que cette distribution soit uniforme sur les vecteurs produits.
- La corrélation des vecteurs. Les vecteurs ne doivent pas être corrélés entre eux, c'est-à-dire qu'il ne doit exister aucune forme de ressemblance mathématique ou statistique entre deux vecteurs successifs d'une même suite. Cela définit l'imprédictibilité, c'est-à-dire que l'observation d'une partie même très grande de la suite ne peut pas conduire à déterminer les parties passées ou futures.
- L'entropie de Shannon ou la densité d'information contenue ou délivrée par une source d'information [Sha48]. Par définition, plus une source est redondante, moins elle contient d'information. L'entropie est maximale pour une source dont tous les symboles sont équiprobables.

Ces notions seront reprisent lors de l'étude statistique de l'« aléatoirité » des données générées par les cœurs de chiffrement utilisés comme source (pseudo-)aléatoires de test.

3.2 UTILISATION DU CŒUR DE CHIFFREMENT COMME GENERATEUR ALEATOIRE

Dans cette partie, deux approches utilisant le cœur de chiffrement comme générateur de vecteur de test sont présentées, la première consiste à utiliser des textes chiffrés comme données de test, la seconde considère chaque résultat intermédiaire de chiffrement comme une nouvelle donnée de test.

1. Première Approche

Cette approche consiste à réaliser plusieurs cycles de chiffrements successifs et à utiliser ces résultats comme une suite (pseudo-)aléatoire de vecteurs de test. Les principes de confusion et diffusion, énoncés au chapitre 1, associés aux algorithmes de chiffrement par blocs, ont permis de démontrer qu'un algorithme de chiffrement par bloc implanté avec une rétroaction de sa sortie sur son entrée peut être considéré comme un générateur de données aléatoires [Sch97, Helle03]. Nous discutons ici de la réalisation matérielle d'un tel générateur.

L'architecture du cœur de chiffrement utilisé en générateur de vecteur de test est illustrée sur la figure 3-1, les blocs notés « opérations » ne sont pas obligatoirement présents pour tous les algorithmes, ils nous servent seulement à donner une architecture générale.

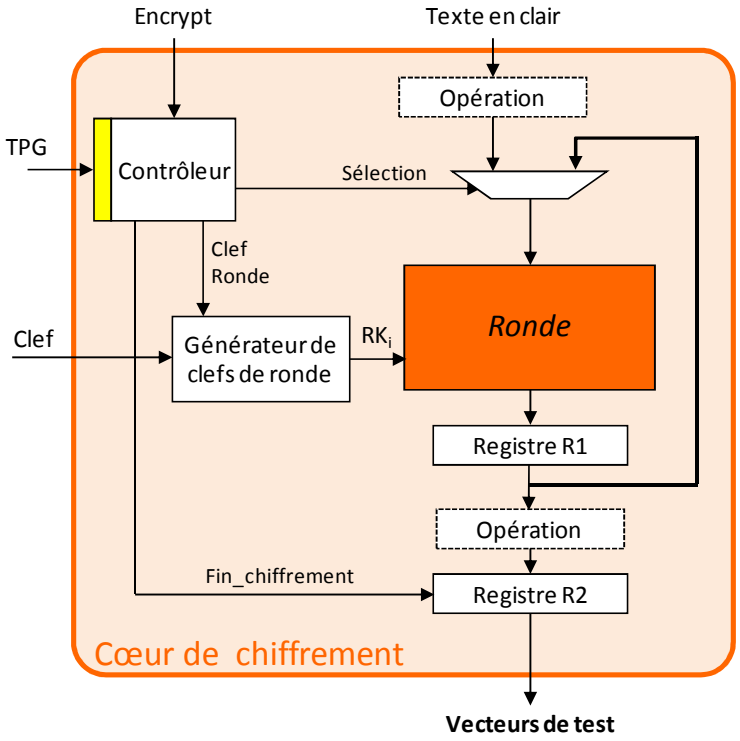


FIGURE 3-1 : COEUR DE CHIFFREMENT EN MODE DE GENERATION DE VECTEURS DE TEST

Le cœur de chiffrement, implanté comme générateur de vecteurs de test, est initialisé par un bloc de texte en clair (de taille m) et une clef choisis de façon aléatoire. Une fois l'opération de chiffrement accomplie, le bloc de texte chiffré est utilisé comme vecteur de test. Le dernier résultat de ronde obtenu (R1) est réinjecté en entrée de la ronde. Il subit les x cycles de ronde nécessaires à un chiffrement (par exemple 16 dans le cas du DES ou 10 dans le cas de l'AES) et le texte chiffré obtenu représente le prochain vecteur de test. Cette opération « chiffrement-rétroaction » est répétée jusqu'à ce que le circuit sous test ait reçu tous les vecteurs nécessaires à son test (obtention du taux de couverture souhaité).

En résumé, pour générer la séquence de test il faut appliquer en entrée du cœur de chiffrement, une graine d'initialisation (texte en clair + clef), puis exécuter une succession de cycles de rondes. Toutes les x rondes, où x représente le nombre de rondes effectué durant un cycle de chiffrement, le résultat obtenu en sortie est utilisé comme vecteurs de test.

L'utilisation du cœur de chiffrement en tant que générateur de vecteurs de test nécessite de modifier le contrôleur initial de la machine. En effet, le contrôleur doit autoriser l'exécution

d'un nombre de cycles de rondes supérieur au nombre normalement requis pour un chiffrement.

Cette solution est très proche du mode de fonctionnement dit à rétroaction de sortie (OFB) utilisé pour du chiffrement par flot (voir figure 3-2), hormis qu'aucune rétroaction de la sortie sur l'entrée n'est rajoutée. En effet, la solution proposée utilise la rétroaction de ronde déjà implantée pour le chiffrement.

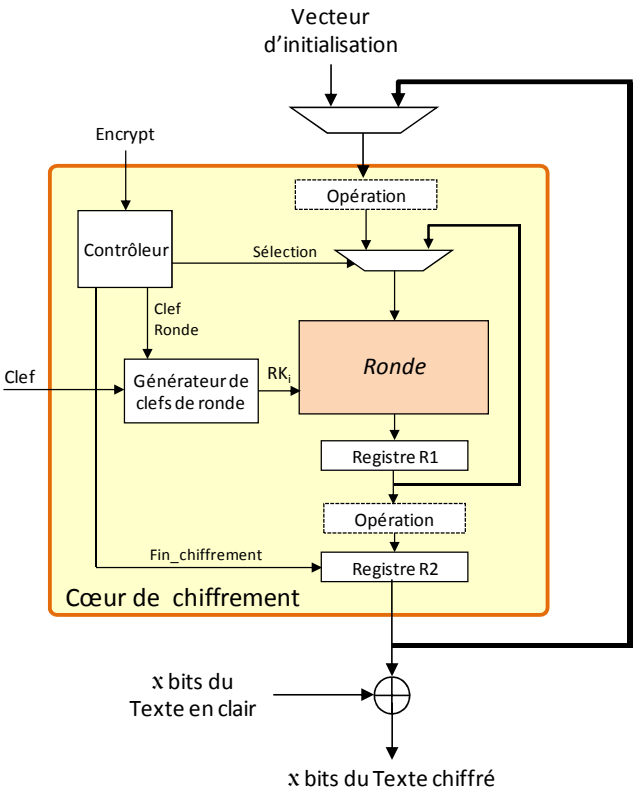


FIGURE 3-2 : MODE DE FONCTIONNEMENT OFB

Notons que le mode OFB est aussi employé pour effectuer du chiffrement à flot synchrone. Par définition, le principe du chiffrement à flot synchrone est de produire une suite de bits aléatoires (pseudo-aléatoires), qui sera combinée par ou exclusif bit à bit avec le flot de bits du texte en clair pour produire le flot de bits du texte chiffré.

Le mode OFB utilise le cœur de chiffrement comme un générateur de flot de bits (clef) aléatoires. Mais étant donné qu'il ne s'agit pas d'un générateur purement aléatoire, la séquence où série de bits générée est toujours identique pour une graine donnée et périodique. Ces deux dernières propriétés sont bien sûr un inconvénient pour ce qui est du chiffrement par flot puisqu'elles peuvent permettre d'identifier le flot de bits générés. Il est donc primordial que ce flot de bits est une période plus longue que le flot de texte en clair à chiffrer. Il a été

montré que la longueur de la période est liée à la taille de la rétroaction ou en d'autres termes au nombre de bits réinjectés en entrée du cœur de chiffrement. Si tous les bits sont renvoyés de la sortie vers l'entrée alors la longueur moyenne du cycle est de 2^{m-1} [Sch97] où m est la taille des mots (ou blocs) manipulés par le cœur de chiffrement (m vaut 128 pour un AES et 64 pour n DES). Si seulement n bits (n < m) sont renvoyés vers l'entrée alors la longueur moyenne du cycle tombe à environ $2^{n/2}$. Il est donc préférable, d'un point de vue sécuritaire d'utiliser une rétroaction sur l'ensemble des m bits disponibles. La même conclusion peut être faite pour ce qui est de la génération de vecteurs de test puisqu'il est souhaitable d'appliquer un grand nombre de vecteurs différents afin d'obtenir un taux de couverture suffisant avec une séquence (pseudo-)aléatoire. En effet, les vecteurs doivent être suffisamment nombreux pour détecter un nombre de fautes important et il est inutile de réappliquer plusieurs fois les mêmes vecteurs (cas des cycles courts).

2. Deuxième Approche

Dans la première approche, le cœur de chiffrement génère un vecteur de test (bloc de m bits), tous les x cycles de ronde, où x représente le nombre de rondes effectuées durant un cycle de chiffrement. En faisant l'hypothèse qu'une ronde est exécutée en un cycle d'horloge, cela revient à dire qu'un vecteur de test sera produit tous les x cycles d'horloge (10 pour un AES et 16 pour un DES).

L'approche proposée ici a pour but de réduire le temps de génération d'une séquence de vecteurs de test en utilisant les résultats de rondes intermédiaires comme vecteurs de test :

Vecteur 1	valeur sortante de la ronde 1
Vecteur 2	valeur sortante de la ronde 2
…	
Vecteur x	valeur sortante de la ronde x fin d'une opération de chiffrement
Vecteur x+1	valeur sortante de la ronde 1
Vecteur x+2	valeur sortante de la ronde 2
…	

Mais il a été démontré dans [Soto00] que les données ont de meilleures propriétés d'« aléatoirité » quand au moins 3 rondes consécutives sont exécutées. Ce point sera discuté ci-après tant d'un point de vue du caractère purement aléatoire des séquences que de leur capacité à constituer des séquences de test.

Enfin il est nécessaire d'envisager toute faille sécuritaire pouvant être engendrée par une modification structurelle ou une utilisation particulière d'un cœur de chiffrement puisqu'il est en premier lieu garant de la sécurité numérique du dispositif hôte. En proposant d'utiliser chaque résultat de ronde comme donnée de test il est nécessaire de modifier le cœur de chiffrement pour accéder à ses résultats intermédiaires (stockés dans R1 si l'on se refaire à la figure 3-2). Or l'accès à ces résultats de ronde, donc à des chiffrements partiels, ne garantit pas la confidentialité des données traitées. La faille sécuritaire engendrée par un nombre réduit de rondes (à fortiori une ronde unique) a été démontrée dans [Andl82, Cha86, Bir04]. Dans le cas qui nous intéresse, les données manipulées ne correspondent pas à une information confidentielle pour ce qui est du texte en clair utilisé comme état initial de la génération de vecteurs de test. Par contre la clef pourra être retrouvée par une analyse des résultats intermédiaires du chiffrement. Par conséquent, il conviendra d'utiliser une clef particulière pour la génération de vecteurs de test, différente de la clef utilisée par le cœur de chiffrement en fonctionnement normal.

Le tableau 3-1 ci-dessous résume les caractéristiques des deux approches proposées, x étant le nombre de cycles d'horloge utilisé pour réaliser un chiffrement complet.

Critères / Approches	Coût en surface	« Durée » de génération d'un vecteur	Sécurité
1 vecteur par chiffrement	Modification du contrôleur (plus de rondes)	x cycles d'horloge nécessaire à un chiffrement	+ +
1 vecteur par ronde de chiffrement	Modification du contrôleur (plus de rondes et sortie à chaque cycle)	1 cycle d'horloge	+ (+)

TABLEAU 3-1 : COMPARAISON DES DEUX APPROCHES

3.3 EVALUATION DU CARACTERE ALEATOIRE D'UNE SEQUENCE

Plusieurs outils (logiciels) peuvent être employés pour évaluer les propriétés aléatoires d'une séquence de vecteurs. Les résultats obtenus avec ces outils ne permettent pas d'affirmer avec certitude qu'une séquence est aléatoire mais par contre ils permettent de la disqualifier si certaines propriétés ne sont pas satisfaites.

Les outils les plus connus sont : l'**ENT** [ENT], la série de **DIEHARD** [DIEH] développée par G. Marsaglia de l'Université de Floride et la série **STS** (Statistical Test Suite) du **NIST** (National Institute of Standards and Technology) [NIST 800-22]. Le programme ENT calcule l'entropie, le χ^2 de Pearson, la moyenne arithmétique, la distribution des bits (le test de Monte Carlo) et enfin le coefficient de corrélation de la séquence. Les deux autres outils DIEHARD et STS se composent de plusieurs tests statistiques, 12 pour la série de DIEHARD et 15 pour la série STS du NIST.

La majorité des tests de l'ENT et de la série de DIEHARD étant compris dans la série STS du NIST, les propriétés aléatoires des séquences générées à l'aide des cœurs de chiffrement seront étudiées avec cette dernière suite.

3.3.1 TEST STATISTIQUE DU NIST

Avant de présenter la série de tests du NIST, le principe d'un test statistique est rappelé. Un test statistique consiste à énoncer une hypothèse concernant un ensemble de données puis à vérifier, si les observations constatées sont vraisemblables dans le cadre de cette hypothèse. L'hypothèse à tester est appelée hypothèse nulle (H_0). Elle s'accompagne impérativement de son hypothèse alternative appelée H_a. L'hypothèse H_0 est celle que nous cherchons à réfuter, celle qui est « vraie » tant que nous n'avons pas démontré le contraire. L'hypothèse Ha contraire à H_0 est celle que nous cherchons à démontrer.

Pour chaque test, le résultat amène à une décision : accepter ou rejeter H_0. Afin d'atteindre une décision objective, la procédure à respecter est la suivante :

1- Etablir l'hypothèse nulle H_0 (considérer l'hypothèse alternative H_a).
2- Choisir le test statistique approprié pour tester H_0,
3- Spécifier un niveau de signification (α),
4- Trouver la distribution d'échantillonnage du test statistique sous H_0,
5- Sur la base de 2, 3, 4, définir la région de rejet,
6- Calculer la valeur du test statistique à l'aide des données de l'échantillon.

Dans notre cas, l'hypothèse H_0 est : ***la séquence étudiée ou « testée » est aléatoire***, ce qui induit H_a : ***la séquence testée n'est pas aléatoire***. Le choix des tests statistiques approprié pour tester H_0 est la série de tests du NIST.

Le niveau de signification du test α représente la probabilité que l'hypothèse H_0 soit rejetée alors qu'elle aurait du être acceptée. La terminologie spécifique aux erreurs de décision dans les tests statistiques, est rappelée dans le tableau 3-2:

		Conclusion (Décision du test)	
		H_0 acceptée	H_0 rejetée (H_a acceptée)
Réalité	H_0 est vraie	Pas d'erreur	Erreur type I α
	H_0 est fausse (H_a est vraie)	Erreur type II β	Pas d'erreur

TABLEAU 3-2 : ERREUR DE DECISION D'UN TEST STATISTIQUE

Dans deux situations, la conclusion du test est incorrecte :

- le rejet de l'hypothèse nulle alors qu'elle est vraie,
  (Erreur de première espèce : Erreur type I, α),

- l'acceptation de l'hypothèse nulle alors qu'elle est fausse,
  (Erreur de seconde espèce : Erreur type II,β),

En pratique, nous nous donnons une limite supérieure du risque de première espèce, le plus souvent 5% (significatif), 1% (très significatif) ou l‰ (hautement significatif). Cette limite permet de définir la condition de rejet de l'hypothèse nulle. En ce qui concerne le calcul de l'erreur de seconde espèce β, c'est un problème complexe qui ne possède que rarement une solution simple. Par conséquent, la puissance d'un test noté P et définie comme la probabilité d'accepter H_0 quand H_0 est vraie ($P = 1- \beta$), est rarement connue. En résumé, étant donné qu'il est difficile voire impossible d'évaluer la valeur de β, le critère de décision repose uniquement sur le choix de l'erreur de première espèce α (il sera alors peu probable de rejeter H_0 quand H_0 est vraie).

Dans mes expériences, la valeur du niveau de signification α est fixée à 1%. Le calcul de l'erreur de seconde espèce β n'a pas de solution car pour un flot de bits il y a un nombre infini de raisons pour lesquelles celui-ci n'est pas aléatoire, et à chaque raison correspond un β différent.

Considérons des échantillons de même taille tirés au hasard parmi une population donnée. Pour chaque échantillon, une statistique (moyenne, variance, etc.) peut être calculée. Celle-ci varie avec l'échantillon. Par définition, la distribution d'échantillonnage est la distribution sous H_0, de toutes les valeurs possibles de la statistique. Concernant les tests de la suite du NIST, la forme de la distribution est soit une distribution suivant la loi normale, soit une distribution suivant la loi du χ^2. La distribution normale est utilisée pour comparer la valeur de la statistique de test obtenue à partir du générateur étudié avec celle attendue pour un générateur purement aléatoire. La distribution du χ^2 est utilisée pour comparer la qualité d'ajustement entre la distribution des fréquences dans l'échantillon observé avec celle attendue.

Afin de valider un test, il faut choisir la région de rejet. Par définition, la région de rejet est constituée par le sous-ensemble des valeurs de la distribution d'échantillonnage qui sont si extrêmes que lorsque H_0 est vraie, la probabilité que l'échantillon observé ait une valeur appartenant à cette région est très faible (la probabilité est α). Il existe deux types de test, les tests dits unilatéraux et les tests dits bilatéraux. Dans un test unilatéral, la région de rejet est entièrement située à une des extrémités de la distribution d'échantillonnage, alors que dans un test bilatéral, cette région est située aux deux extrémités de la distribution voir figure 3-3.

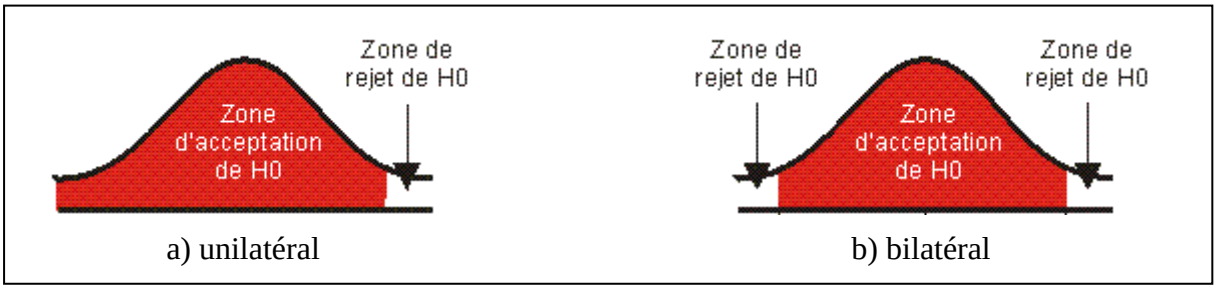


FIGURE 3-3 : REGION DE REJET DU TEST

La taille de la région de rejet est définie par α. Quand α vaut 0,01 la taille de la région de rejet correspond à 1% de l'espace inclus sous la courbe de la distribution d'échantillonnage.

Pour fixer les idées, nous rappelons le principe du test du χ^2, qui est un test unilatéral :

> *Définition* : Une "bonne" partition de l'ensemble Ω des événements possibles est une suite finie de **v+1** sous-ensembles $\boldsymbol{I_0, I_1, \ldots I_v}$ deux à deux disjoints, recouvrant Ω i.e. $\Omega = \cup I_j$ et tels que les probabilités $P_j \doteq \Pr(I_j)$ soient toutes non nulles.

Définition : Soient une variable aléatoire X, une bonne partition (I_j) de l'ensemble de ses valeurs et **n** un nombre d'essais. Nous procédons à **n** tirages de la variable X et nous appelons $\mathbf{n_j}$ le nombre de fois où la variable X est tombée dans le sous-ensemble I_j. Le test χ^2 de Pearson consiste à calculer

$$\chi^2 Pearson = \sum_{j=0}^{v} \frac{\left(n_j - n.P_j\right)^2}{n.P_j}.$$

La statistique χ^2 de Pearson donne une mesure de l'écart existant entre les effectifs théoriques attendus $\mathbf{n.P_j}$ et ceux observés dans l'échantillon $\mathbf{n_j}$, on parle alors de distance du χ^2. Si l'échantillon vérifie l'hypothèse initiale (par exemple, la séquence est aléatoire), alors cette distance aura tendance à être proche de 0. Par contre, plus le χ^2 de Pearson sera grand, plus le désaccord sera important. La valeur χ^2 de Pearson obtenue est comparée à une valeur $\chi^2_{v;1-\alpha}$ qui définie la région de rejet du test, issue de la table des fractiles de loi du χ^2 [Sap90], où v est le nombre de degré de liberté et α est la probabilité d'erreur de type 1.

Si $\chi^2 \text{Pearson} > \chi^2_{v;1-\alpha}$, et si n est suffisamment grand, alors l'hypothèse d'avoir effectivement la répartition théorique voulue est à rejeter avec une probabilité d'erreur d'au plus α. Sinon, l'hypothèse est acceptée.

Ceci peut être illustré par l'exemple suivant :

Nous considérons deux dés, et nous voulons tester si oui, ou non, ils sont équilibrés. Chaque dé est lancé 90 fois. Dans les tableaux 3-3 et 3-5, les résultats des 90 lancés sont présentés respectivement pour le premier et le deuxième dé.

Valeur obtenue	1	2	3	4	5	6
Nombre de fois rencontrée	20	23	10	8	19	10

TABLEAU 3-3 : EXEMPLE N°1 LOI DU X^2

Soit l'hypothèse H_0 : « le dé est équilibré » autrement dit $P(X=j) = \frac{1}{6}$ avec j=1,…,6. Calculons le χ^2 d'une telle répartition. A noter n = 90, n_j= {20, 23, 10, 8, 19 ou 10}, et $P_j = \frac{1}{6}$ d'où $n.P_j$=15.

$$\chi^2\text{Pearson} = \frac{(20\text{-}15)^2}{15} + \frac{(23\text{-}15)^2}{15} + \frac{(10\text{-}15)^2}{15} + \frac{(8\text{-}15)^2}{15} + \frac{(19\text{-}15)^2}{15} + \frac{(10\text{-}15)^2}{15}$$
$$= \frac{204}{15} \simeq 13{,}6$$

Maintenant, comparons notre résultat avec $\chi^2_{v;\,1\text{-}\alpha}$ où v vaut ici 5 (**v+1 = 6**) et fixons α = 5%. D'après l'extrait de la table des fractiles fourni tableau 3-4, nous pouvons conclure étant donné que $\chi^2\text{Pearson} > \chi^2_{5;0,95}$ (13,6 > 11,070) que l'hypothèse de départ est à rejeter. La probabilité de faire une erreur en rejetant cette hypothèse est inférieure à 5%.

v \ 1-α	0,001	0,005	0,010	0,050	0,1000	0,5000	0,9000	0,9500	0,9750	0,9900	0.9990
5	0.210	0,412	0,554	1,145	1,610	4,351	9,236	11,070	12,833	15,086	20,515

TABLEAU 3-4 : EXTRAIT TABLE DES FRACTILES DE LA LOI DU X^2

Les résultats pour le deuxième dé sont les suivants (tableau 3-5) :

Valeur obtenue	1	2	3	4	5	6
Nombre de fois rencontrée	17	13	11	14	16	19

TABLEAU 3-5 : EXEMPLE N°2 LOI DU X^2

La valeur du $\chi^2\text{Pearson}$ est de 2,8. Cette valeur est inférieure à $\chi^2_{5;\,0,95}$, l'hypothèse de départ est donc acceptée. Le dé est équilibré.

Pour la série de tests du NIST, une hypothèse est faite sur la longueur n de la séquence à étudié. Il faut que n soit grand (de l'ordre de 10^3 à 10^7), car les lois de distributions choisies sont représentatives pour des séquences longues. Pour des séquences plus petites il faudra choisir d'autres distributions mieux représentatives. Pour ces tests, la zone de rejet est définie avec, à chaque fois, α = 1%.

Nous utilisons le programme de calculs STS version 1.8 fourni par le NIST. Ce programme de calcul ne donne pas une valeur de statistique de test à comparer avec des bornes de rejet, mais calcule directement la *P-value* du test. La *P-value* est en fait la probabilité d'obtenir un résultat au moins aussi extrême que celui qui a été obtenu. Cette valeur représente donc la probabilité d'erreur de 1ère espèce. Pour l'exemple des dés équilibrés, la statistique de test vaut pour le premier exemple 13,6. Donc la *P-value* correspondante vaut approximativement

0,02 (puisque dans le tableau 3-4, 13,6 correspond à une valeur de 1-α comprise entre 0,975 et 0,99).

Si le résultat d'un test, autrement dit la *P-value*, est inférieur à 0,01 alors la conclusion est que la séquence ne vérifie pas le critère d'aléa correspondant. Autrement, nous pouvons considérer que la séquence est aléatoire.

La suite de tests STS du NIST :

Dans cette partie, seulement deux tests parmi les 15 de la série sont présentés, les autres sont décrits dans l'annexe A. L'objectif ici n'est pas de rentrer dans les détails des tests, mais d'en présenter le but et les grands principes. Les tests présentés sont : le test dit « test de fréquence » et le test dit « Recherche d'un motif apériodique ».

- **Test de fréquence** « Monobit Test »

Ce test s'intéresse à la proportion de « zéros » et de « uns » dans la séquence entière. Le but de ce test est de déterminer si le nombre de « zéros » et de « uns » dans la séquence est approximativement égal à celui prévu pour une vraie séquence aléatoire. Le test évalue donc la déviation de la proportion des « uns » par rapport à 1/2.

En pratique, ce test consiste à remplacer dans la séquence étudiée les 1 par (+1) et 0 par (-1) et à faire la somme (S) bit à bit de la séquence. Si S est très grand (trop de 1) ou très petit (trop de 0) alors la séquence est considérée comme non aléatoire.

Soit par exemple les séquences :

ε = 11001010110011 $\quad$ S = 1+1-1-1+1-1+1-1+1+1-1-1+1+1 = 2

ε_1 = 10111011011111 $\quad$ S_1 = 1-1+1+1+1-1+1+1-1+1+1+1+1+1+1 = 8

ε_2 = 11010001000000 $\quad$ S_2 = 1+1-1+1-1-1-1+1-1-1-1-1-1-1 = -5

Les séquences ε_1 et ε_2 contrairement à ε ne sont pas considérés comme aléatoire car les sommes obtenues sont trop éloignées de 0.

- **Recherche d'un motif apériodique** « Aperiodic Templates Test »

Ce test porte sur le nombre d'occurrence d'une sous-suite donnée à l'intérieur de la séquence. Le but de ce test est de rejeter les séquences présentant un trop grand nombre d'occurrences d'un motif apériodique. La recherche se fait en utilisant une fenêtre de m bits

glissant sur la séquence à la recherche d'un motif de taille m. Lorsque ce motif a été trouvé, la fenêtre de recherche est replacée sur le premier bit suivant le motif découvert.

En pratique, la première étape du test consiste à diviser la séquence en N blocs de taille T bits. Ensuite, la deuxième étape consiste à choisir le motif à recherché, ce motif est de taille m. Et enfin, la dernière étape consiste à parcourir les N blocs avec la fenêtre contenant le motif recherché et à compter le nombre d'occurrence du motif.

Soit par exemple la séquence ε = 01011010011101101110100010 10, N = 2 et T = 14. D'où les deux blocs sont : B_1 = 01011010011101 et B_2 = 10111010001010. Le motif à recherché est de taille m = 2 et vaut '11'. La fenêtre contenant le motif '11' parcourt d'abord le bloc B_1 (voir figure 3-4). Les $4^{ième}$ et $5^{ième}$ bits correspondent au motif recherché d'où le nombre d'occurrence est incrémenté de 1. Puis la fenêtre reprend son parcourt du bloc B_1 au $6^{ième}$ bit. Le motif '11' est rencontré 2 fois dans le bloc B_1, le bloc B_2 est traité de la même façon. Au final, le test rejettera les séquences qui ont un trop grand nombre d'occurrence d'un motif.

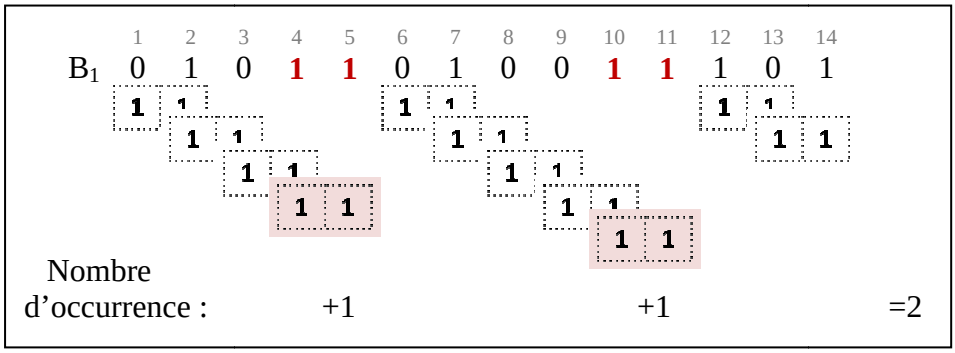


FIGURE 3-4 : TEST DU NIST RECHERCHE D'UN MOTIF APERIODIQUE

Remarque : Les tests STS du NIST n'évaluent les propriétés aléatoires que d'un flot de bit, la notion de *vecteur* n'intervient pas. Soit par exemple la séquence suivante ε = 01011010101001010101010101. Cette séquence a de bonne propriété aléatoire, si elle est considérée en tant que flot de bits Par contre, si cette séquence est étudiée comme une succession de vecteurs de 4 bits ({0101}, {1010}, {1010}, {0101}, {0101}, {0101}), seulement deux vecteurs différents sont rencontrés {0101} et {1010} alors qu'il y a 16 (2^4) vecteurs possibles. Par conséquent, cette séquence ne serait pas intéressante pour tester un circuit qui ne serait stimulé alors que par 2 vecteurs différents.

Ainsi la séquence générée à l'aide du cœur de chiffrement sera étudiée uniquement sous forme de flots de bits.

3.3.2 COMPARAISON AVEC L'EXISTANT DU CARACTERE ALEATOIRE DES SEQUENCES PROPOSEES

Avant d'étudier le caractère aléatoire des données de test il est important de comprendre comment les données produites par le générateur intégré sont appliquées au circuit sous test ou, en d'autres termes, comment sont formés les vecteurs de test puisque c'est bien le caractère aléatoire de ces vecteurs qui nous intéresse ici.

L'architecture de test intégré envisagée pour le test du circuit est illustrée sur la figure 3-5. Il s'agit d'une architecture STUMPS (Self Test Using MISR / Parallel SRSG) [Bard87].

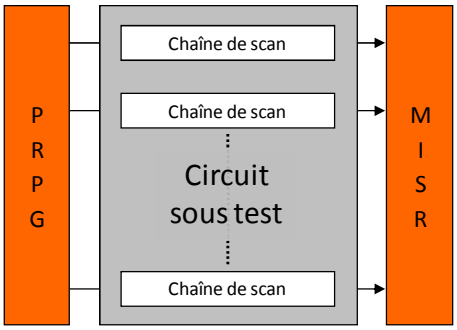


FIGURE 3-5 : ARCHITECTURE STUMPS

Les chaînes de scan du circuit sous test sont alimentées en parallèle par le générateur PRPG (Pseudo Random Pattern Generator ou SRSG Shift Register Sequence Generator), et la signature résultant de la compaction de l'ensemble des réponses de test est générée à l'aide d'un compacteur de réponses (MISR présenté dans le chapitre 4).

Utiliser une architecture à chaîne de scan multiple permet de réduire de façon significative le temps de test. Quand les chaînes de scan sont de longueur différente, le PRPG exécute K cycles pour charger les chaînes, où K est la longueur de la chaîne de scan la plus longue. Pour les chaînes les plus courtes les données générées en « trop » sortent directement des chaînes et alimentent le MISR.

Il est à noter que les performances de cette solution de test peuvent être affectées par des problèmes liées à la corrélation linaires des données de test appliquées sur les différentes chaînes de scan ou à leur périodicité. Trois configurations de génération ont été envisagées (voir figure 3-6). Dans la première, le générateur alimente une seule entrée du circuit (cas d'une chaîne de scan unique, figure 3-6 a). Ainsi pour cette configuration seront étudiées uniquement les propriétés aléatoires d'un flot unique sortant du bit de poids faible du

générateur. Dans la deuxième configuration (figure 3-6 b), le générateur alimente plusieurs entrées (e.g. plusieurs chaînes de scan), mais ce nombre d'entrées reste inférieur au nombre de sorties du générateur. Le nombre de flots étudiés a été fixé de façon arbitraire à 16. La troisième configuration (figure 3-6 c) correspond à l'utilisation de l'ensemble des flots de bits accessibles en sortie du cœur de chiffrement (e.g. 128 flots dans le cas de l'AES).

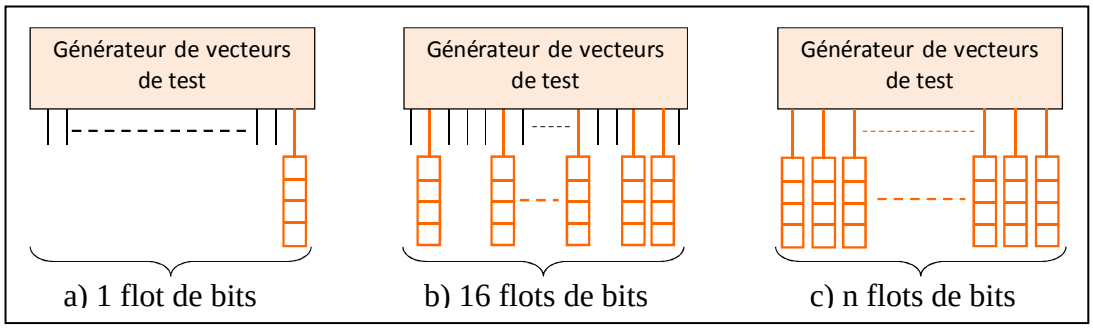


FIGURE 3-6 : LES TROIS CONFIGURATIONS ENVISAGEES POUR LE TEST

Pour chacune des deux dernières configurations, non seulement les propriétés aléatoires des n flots de bits (figure 3-7 b) seront étudiées, mais aussi celle des « vecteurs » (mots de n bits) tels qu'appliqués au circuit sous test. Ne possédant d'outil d'évaluation que pour des flots de bits et non de mots, les données issues des générateurs sont concaténées comme indiqué sur la figure 3-7 (a) pour former un flot de bit unique avant d'être analysé.

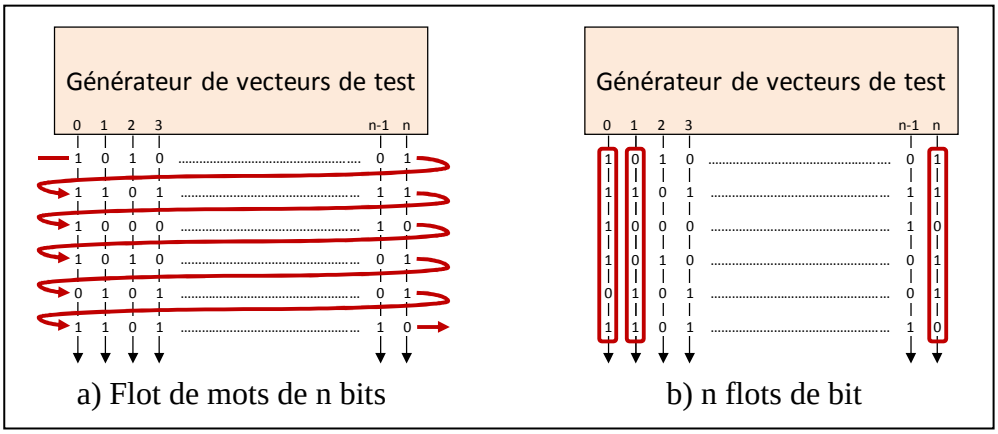


FIGURE 3-7 : FLOTS ETUDIES AVEC LES TESTS DU NIST

La longueur des flots étudiés est fixé à $1{,}5.10^6$ bits ce qui correspond à l'intervalle (entre 10^3 et 10^7) recommandé pour appliquer les tests du NIST.

La section suivante à pour but de présenter le générateur de vecteurs de test servant de référence à notre étude, notamment le LFSR. La section 2 décrit l'ensemble des générateurs comparés. Les séquences issues de ces générateurs sont analysées dans la section suivante.

1. Le LFSR

Les générateurs pseudo-aléatoires de type LFSR, sont des registres à décalages complétés par des boucles de rétroaction à base de portes « ou-exclusif ».

La forme générale des LFSRs est représentée sur la figure 3-8. Deux architectures de LFSR sont couramment utilisés : une dite de type 1 à portes « ou-exclusif » externes (figure 3-8 a) et une dite de type 2 à portes « ou-exclusif » internes (figure 3-8 b).

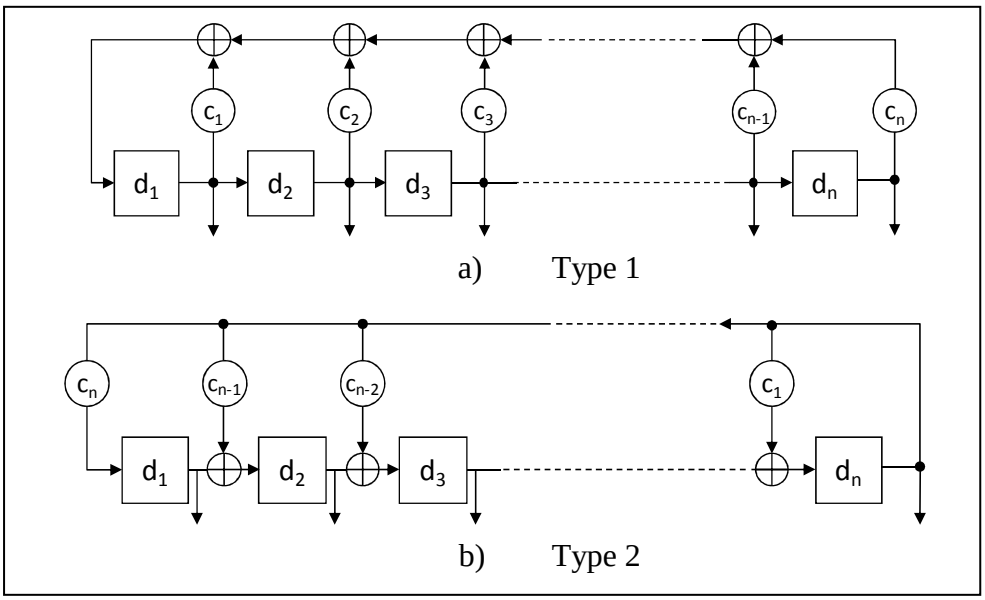


FIGURE 3-8 : ARCHITECTURE DE LFSR

La fréquence maximale de fonctionnement d'un LFSR de type 1 est limitée par le chemin de rétroaction externe pouvant comporter un grand nombre de portes. Il s'intègre par contre facilement autour d'un registre à décalage préexistant. La fréquence de fonctionnement du LFSR de type 2 est uniquement limitée par la durée de propagation des données au travers d'une porte « ou-exclusif » et d'une bascule mais il s'intégrera difficilement sur un registre à décalage du système.

Un LFSR est décrit par son polynôme caractéristique. Les bascules constituent les étages du registre à décalage, leurs nombres correspondent au degré du polynôme caractéristique du LFSR. Par exemple pour un registre à décalage constitué de 4 étages (4 bascules) le polynôme caractéristique sera de degré 4. Le nombre et la position des portes ou exclusif dépendent du choix du polynôme caractéristique. Le polynôme caractéristique décrivant un LFSR est de la forme :

$$p(x)=1+\sum_{i=1}^{n} c_i x^i$$

où n représente le nombre de bascule du registre à décalage et c_i est une composante binaire qui représente, quand elle vaut 1, la connexion d'un étage du registre avec la rétroaction, ou quand elle vaut 0, l'absence de connexion.

Par définition [Bard87], la longueur maximale d'une séquence générée par un LFSR à n étages, est de 2^n-1. Le polynôme caractéristique associé à une séquence de longueur maximale est appelé polynôme primitif.

Dans [Bard87], des exemples de polynômes primitifs sont donnés pour des valeurs de n comprises entre 1 et 300. Par exemple, pour un LFSR avec 4 étages (n = 4), le polynôme primitif peut s'écrire $P(x)= x^4+x+ 1$, l'implantation est illustré figure 3-9.

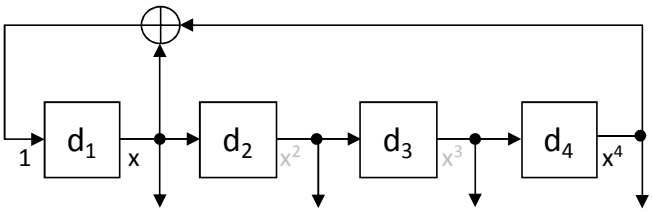


FIGURE 3-9 : EXEMPLE D'UN LFSR DE TYPE 1 A 4 ETAGES

Si les bascules du registre ne sont pas toutes initialisées à 0, la séquence aura une longueur de 2^n-1 (ex : n = 4, séquence de 15), et l'état sortant tout à 0 ne sera jamais atteint. A l'inverse si toutes les bascules sont initialisées à 0, l'état sortant sera toujours le même est égal à 0.

Une des propriétés associées aux LFSRs implantés suivant leurs polynômes primitifs est que chacun des bits générés a la même probabilité de valoir 0 ou 1. Les séquences ainsi générées sont dites séquences équiprobables. Les séquences générées par un LFSR sont reproductibles (du moment où la graine (le germe) est connue) et périodiques.

Dans le cas d'une architecture STUMPS, utiliser un LFSR de type 1 peut éventuellement limiter la qualité du test car les données générées par chacun des étages sont très corrélées. En effet, $Q_i(t) = Q_{i-1}(t-1)$, en d'autre termes les données appliquées à une chaîne de scan alimentée par l'étage i sont identiques à celles appliquées à la chaîne alimentée par l'étage précédent au cycle d'horloge précédent. Pour pallier ce problème un LFSR de type 2 peut être utilisé, la présence des portes « ou-exclusif » entre les bascules permettant de réduire la corrélation linéaire entre les étages séparés par une porte « ou-exclusif ». Le problème persiste

néanmoins pour les étages en connexion directe (pas de « ou-exclusif »). Cette situation est généralement résolue en intégrant un "décaleur" de phase ("Phase Shifter") entre le générateur et le circuit sous test au prix d'une surface de silicium supplémentaire.

2. Générateurs étudiées

Tous les générateurs présentés seront initialisés avec des données choisies de façon aléatoire. Ces générateurs sont :

- un ***LFSR***

Ce LFSR est de 128 étages et est implanté suivant le polynôme primitif $p(x) = x^{128}+x^{29}+x^{27}+x^{2}+1$. La taille du LFSR est choisie de sorte à être identique à la taille d'un bloc de données traitées par un AES. Pour les raisons évoquées précédemment, les séquences étudiées seront générées à l'aide d'un LFSR de type 2.

- un ***DES_entier***

Il s'agit d'un algorithme de chiffrement DES utilisé suivant la *première approche* (cf. partie 3.2), c'est-à-dire qu'un vecteur de test est produit tous les 16 cycles de rondes.

- un ***DES_ronde***

Il s'agit d'un algorithme de chiffrement DES utilisé suivant la *deuxième approche* (cf. partie 3.2), c'est-à-dire qu'un vecteur est produit à chaque cycle de ronde.

Les propriétés aléatoires ne seront pas étudiées sur la totalité des flots sortants (64), mais seulement sur les 32 flots de droite. Puisque les 32 bits de droite résultant d'une ronde sont copiés vers les 32 bits de gauche après l'exécution de la ronde suivante, le flot de droite et le flot de gauche sont identiques, mais décalé dans le temps d'un cycle (voir la description du DES chapitre 1). En effet, dans le cas d'une architecture STUMPS, utiliser les 64 flots du générateur ***DES_ronde*** poserait le même problème de corrélation linéaire des données de test que lors de l'utilisation d'une séquence générée à l'aide d'un LFSR de type 1.

Par conséquent, dans le cas du générateur ***DES_ronde***, les trois configurations de test étudiés sont : le flot de bits le plus à droite, les 16 flots de bits suivants ({2, 3, 4, 6, 8, 9, 11, 12, 18, 19, 20, 22, 25, 27, 30, 31}) parmi les 32 de droites et enfin les 32 flots de droites.

- un ***AES_entier***

Il s'agit d'un algorithme de chiffrement AES utilisé suivant la *première approche* c'est-à-dire qu'un vecteur de test est produit tous les 10 cycles de rondes.

- un ***AES_ronde_meme_clef***

Il s'agit d'un algorithme de chiffrement AES utilisé suivant la *deuxième approche* c'est-à-dire qu'un vecteur est produit à chaque cycle de ronde.

- un ***AES_ronde***

Il s'agit d'un algorithme de chiffrement AES utilisé suivant la *deuxième approche*, mais pour lequel le module de génération de clef de ronde est rebouclé sur lui-même. C'est-à-dire qu'une fois un chiffrement réalisé la dernière clef de ronde devient la nouvelle clef à partir de laquelle les 10 nouvelles clefs de rondes vont être déterminées. Cette modification ne change pas le principe de la génération des clefs de rondes décrit au chapitre 1.

3. Résultats obtenus en appliquant les tests du NIST

Les résultats des tests statistiques du NIST obtenus pour la première configuration (1 flot de bit) sont présentés dans le tableau 3-6. La valeur présentée dans le tableau est la *P-value*. Pour rappel celle-ci doit être inférieure à 0,01 pour considérer que la séquence ne satisfait pas aux critères d'une séquence aléatoire.

	LFSR	***DES_entier***	***DES_ronde***	***AES_entier***	***AES_ronde_meme_clef***	***AES_ronde***
1. Fréquence	***0,00256***	0,703584	0,458464	0,110981	0,208017	0,712086
2. Bloc-freq	0,441504	0,010487	0,870648	0,267765	0,725004	0,475563
3. Runs	0,143622	0,020947	0,183373	0,999049	0,247087	0,641563
4. Long Runs	0,965931	0,686393	0,158288	0,079787	0,510618	0,285461
5. Rang	0,526598	0,839072	0,244112	0,820208	0,321308	0,357221
6. DFT	0,810512	0,500094	0,6104	0,642256	0,25475	0,03397
7. Apériodique	0,499631	0,462948	0,505411	0,453813	0,533758	0,507041
8. Périodique	0,393849	0,170431	0,900547	0,336229	0,379708	0,083453
9. Maurer	0,244026	0,232251	0,866246	0,845498	0,157055	0,446354
10. Lincomp	***0***	0,146985	0,889957	0,543506	0,981514	0,867606
11. Série	0,713832	0,595071	0,427347	0,313853	0,2180015	0,623499
12. Apen	0,637473	0,233685	0,413584	0,189886	0,983707	0,441726
13. Cusum	***0,003262***	0,715753	0,557514	0,085975	0,257467	0,735655
14. Random	-	0,563382	0,367899	0,452848	0,338463	0,412837
15. Variante-R	-	0,496957	0,241771	0,546886	0,673736	0,498466

TABLEAU 3-6 : RESULTAT TEST DU NIST SUR 1 FLOT

Les résultats obtenus, pour les algorithmes de chiffrement DES et AES, quelle que soit l'approche envisagée, valident l'ensemble des 15 tests du NIST. Seule la séquence générée à

l'aide du ***LFSR*** ne valide pas 5 tests. Une des explications peut être que la séquence étudiée est trop courte ($1{,}5.10^6$) par rapport à la longueur du cycle du ***LFSR*** (2^{128}-1). Dans le cas du ***LFSR***, les tests Random (Test d'excursions aléatoires) et Variante-R (Variante du test d'excursions aléatoires) ne peuvent être menés à termes car le nombre de cycles (excursions aléatoire (définition en annexe A)) rencontrés dans la séquence n'est pas suffisant. Par la suite, les tests ne pouvant être menés à termes seront considérés comme échoués.

En conclusion, le flot sortant (le plus à droite) d'un algorithme de chiffrement a des propriétés aléatoires équivalentes ou supérieures à celle d'un ***LFSR***, et donc ces séquences peuvent être utilisées dans le cadre d'un test aléatoire.

Pour la deuxième configuration de test où 16 sorties du générateur alimentent le circuit sous test. Les 16 flots ont été pris au hasard parmi les flots sortants. Si le générateur possède 128 sorties alors les flots étudiés sont les flots {2, 18, 25, 38, 40, 41, 59, 71, 75, 80, 98, 100, 101, 110, 111 et 125}, dans le cas du DES (64 sorties) alors les flots étudiés sont {2, 7, 11, 12, 20, 24, 27, 30, 31, 40, 43, 47, 51, 58, 59 et 63} (sauf pour ***DES_ronde*** pour les raison invoquées ci-dessus). Les résultats sur le flot des mots de 16 bits mit bout à bout (figure 3-7 a) sont donnés dans le tableau 3-7.

	LFSR	***DES_entier***	***DES_ronde***	***AES_entier***	***AES_ronde_meme_clef***	***AES_ronde***
1. Fréquence	***0,000767***	0,977853	0,521025	0,282594	0,236445	0,486645
2. Bloc-freq	***0***	0,416303	0,806778	0,824684	0,057259	0,920575
3. Runs	0,041326	0,095783	0,880314	0,430817	0,651376	0,670856
4. Long Runs	0,077589	0,99031	0,628429	0,311132	***0,002281***	0,692152
5. Rang	***0***	0,716469	0,645072	0,689073	0,228172	0,189179
6. DFT	***0***	0,184772	0,34513	0,401368	0,440264	0,970115
7. Apériodique	0,414542	0,539359	0,468438	0,551260	0,523350	0,475298
8. Périodique	***0,003066***	0,998148	0,042176	0,019739	0,10249	0,077538
9. Maurer	0,037205	0,718866	0,984132	0,594385	0,339567	0,393453
10. Lincomp	0,020047	0,885161	0,14972	0,370274	0,718803	0,550562
11. Série	0,2673495	0,328926	0,21461	0,502698	0,767925	0,928997
12. Apen	0,631282	0,291474	0,03674	0,562052	0,401035	0,987071
13. Cusum	***0,000013***	0,911889	0,638184	0,470209	0,310188	0,573269
14. Random	-	0,542519	0,198786	-	0,658382	0,510973
15. Variante-R	-	0,5790711	0,463122	-	0,659468	0,716035

TABLEAU 3-7 : TEST DU NIST SUR FLOT DE VECTEURS DE 16 BITS

Les algorithmes de chiffrements passent généralement tous les tests à l'exception de l'***AES_entier*** et l'***AES_ronde_meme_clef*** qui ne valident respectivement que 13 et 14 tests. Le LFSR quant à lui ne valide que 7 tests sur les 15.

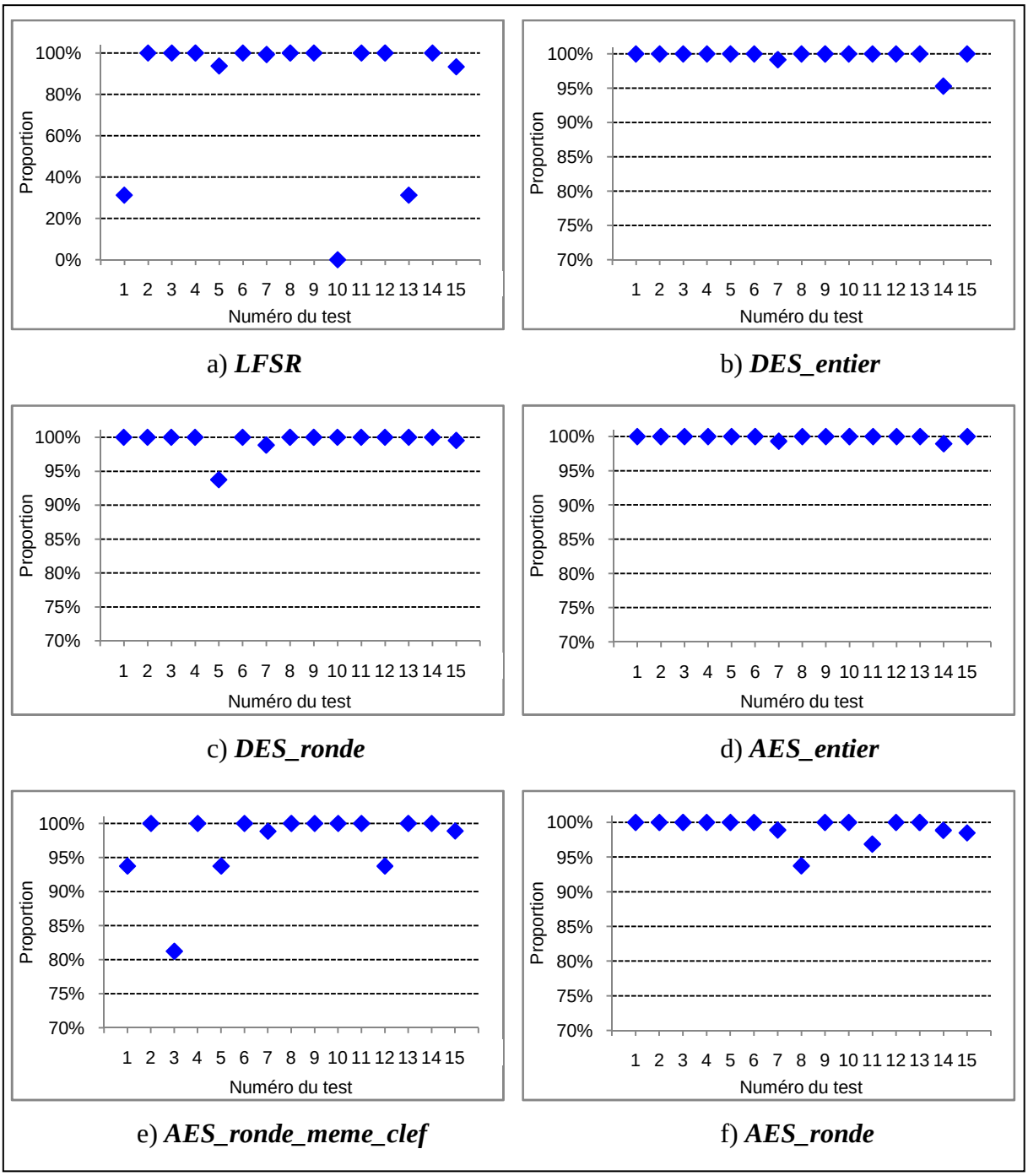


FIGURE 3-10 : POURCENTAGE DE FLOTS QUI VALIDENT CHAQUE TEST

Les résultats obtenus sur chacun des 16 flots (figure 3-7 b) sont regroupés dans la figure 3-10. Pour chacun des tests, la proportion des flots validant le test sur le nombre total de flot étudié est fournie. Les tests sont numérotés de 1 à 15, la correspondance numéro-test est donnée dans le tableau 3-7.

En résumé, pour le ***LFSR***, 3 tests ont des proportions inférieures à 40% dont le test 10 ou cette proportion vaut 0 ce qui signifie qu'aucun flot étudié parmi les 16 ne valide le test. Pour les autres générateurs la proportion la plus basse est : ***DES_entier*** 95,31%, ***DES_ronde*** 93,75%, ***AES_entier*** 98,44%, ***AES_ronde_meme_clef*** 81,25% et enfin ***AES_ronde*** 93,75%.

En conclusion, les 16 sous-séquences générées à l'aide d'un algorithme de chiffrement utilisé suivant la *première approche* (cf. partie 3.2) ont de bonnes propriétés aléatoires, elles sont même meilleures que celles d'un ***LFSR***. Concernant les sous-séquences composées des résultats de ronde, les résultats obtenus aux tests du NIST sont également meilleurs que ceux du ***LFSR***. Il est à noter que les résultats des sous-séquences de l'***AES_ronde*** sont légèrement meilleurs que ceux de l'***AES_ronde_meme_clef***. Une des explications est que la rétroaction du module de génération de clefs rajoute de l'aléa au niveau des bits de données.

Et enfin, concernant la dernière configuration de test où les n sorties du générateur alimentent le circuit sous test. Les résultats sur le flot des mots sont donnés dans le tableau 3-8. Pour rappels, n vaut 128 pour l'AES quelque soit l'approche, 128 également pour le ***LFSR***, 64 pour le ***DES_entier*** et enfin 32 pour le ***DES_ronde***.

	LFSR	***DES_entier***	***DES_ronde***	***AES_entier***	***AES_ronde_meme_clef***	***AES_ronde***
1. Fréquence	***0***	0,299759	0,919356	0,639306	0,209694	0,252322
2. Bloc-freq	***0,685858***	0,468581	0,151692	0,679431	0,471094	0,058096
3. Runs	***0***	0,775724	0,482568	0,688025	0,613435	0,321052
4. Long Runs	***0***	0,765538	0,838433	0,622062	0,600609	0,300064
5. Rang	***0,110155***	0,845479	0,696475	0,981998	0,556941	0,762689
6. DFT	***0***	0,03212	0,093276	0,71911	0,417304	0,103968
7. Apériodique	***0***	0,520102	0,480587	0,582339	0,565934	0,444649
8. Périodique	***0***	0,663874	0,315323	0,489994	0,099107	0,504435
9. Maurer	***0***	0,353189	0,964552	0,067874	0,475411	0,025904
10. Lincomp	***0***	0,51853	0,901956	0,025943	0,146473	0,352677
11. Série	***0***	0,018493	0,776016	0,7354875	0,491941	0,814178
12. Apen	***0***	0,427365	0,191059	0,671642	0,805931	0,535168
13. Cusum	***0***	0,378709	0,882564	0,52733	0,228124	0,415864
14. Random	***0***	0,415319	0,504599	0,606076	0,606076	-
15. Variante-R	***0***	0,611516	0,745599	0,618386	0,618386	-

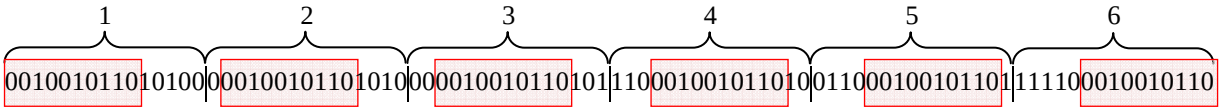
TABLEAU 3-8 : TEST DU NIST SUR FLOT DE VECTEURS DE N BITS

Le flot constitué des mots de n bits mis bout à bout pour un DES ou un AES, quelque soit, l'approche envisagé (cf. partie 3.2), est aléatoire. Seuls les tests Random et Variante-R ne sont pas validés pour le générateur ***AES_ronde*** car le nombre de cycles d'excursions est insuffisant.

Seul le flot du ***LFSR*** ne valide que 2 tests, une des explications provient du choix du polynôme primitif. En effet, le polynôme employé a peu de coefficients ($p(x) = x^{128}+x^{29}+x^{27}+x^{2}+1$) d'où peu de portes « ou-exclusif » et donc beaucoup de corrélations. Prenons par exemple, un LFSR à 15 étages, dont le polynôme primitif est le suivant :

$$p(x) = x^{15}+x+1.$$

Supposons une graine de départ choisie de façon aléatoire et regardons le flot composé des mots de 15 bits mis bout à bout, ici un flot de 6 mots :



Il est à noter que dans le flot très court étudié (6 mots), le même motif de 10 bits se répète dans chaque mot. Sur le même principe, le ***LFSR*** de 128 bits a un grand nombre d'occurrence d'un même motif. La solution qui consisterait à choisir un polynôme primitif comportant d'avantage de termes (plus de « ou-exclusif » dans l'architecture résultante) permettrait de casser d'avantage de corrélation. Cette solution est à nouveau au détriment de la surface additionnelle nécessaire à l'implantation d'un générateur.

Les tests du NIST ont ensuite été appliqués aux n flots indépendamment et pour chacun des générateurs les proportions des flots validant un test sur le nombre total de flots ont été calculés. Les résultats sont présentés sur la figure 3-11.

Les résultats obtenus ont la même allure que ceux obtenus avec les sous-séquences (16 flots tirés au hasard). Comme précédemment le ***LFSR*** à trois proportions inférieures à 40% dont une nulle. Pour chacun des autres générateurs, la proportion la plus basse est : ***DES_entier*** 95,31%, ***DES_ronde*** 96,88%, ***AES_entier*** 96,88%, ***AES_ronde_meme_clef*** 80,47% et enfin ***AES_ronde*** 97,66%.

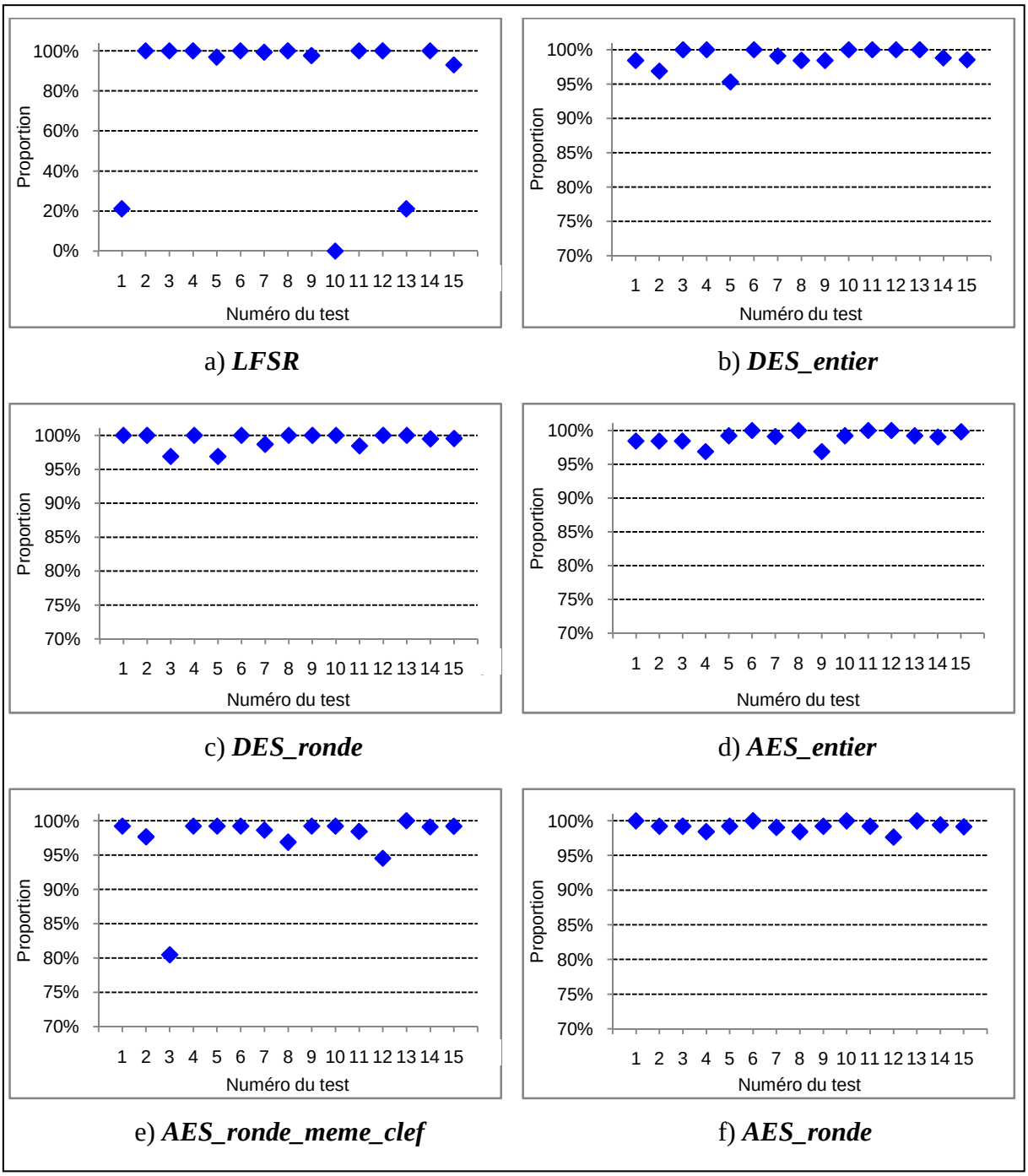


FIGURE 3-11 : POURCENTAGE DE FLOTS SUR n (64 OU 128) QUI VALIDENT CHAQUE TEST

L'allure des graphiques permet de conclure sur les propriétés aléatoires des séquences générées. En effet, pour le ***DES_entier***, le ***DES_ronde***, l'***AES_entier*** et l'***AES_ronde*** toutes les proportions obtenues sont supérieure à 95%, par conséquent ces séquences sont considérées comme aléatoire. Pour le générateur ***AES_round_meme_clef***, un test est passé par 80% des flots, mais malgré ce résultat, on peut conclure que la séquence a de bonnes propriétés aléatoires. Concernant le ***LFSR*** trois tests ont des proportions faibles voir très faibles, la conclusion ne peut être que la séquence est aléatoire.

En conclusion, quelque soit l'approche envisagée les algorithmes de chiffrement rebouclés sont de bons générateurs de nombres (flots) aléatoires. Au vu des résultats obtenus sur les tests du NIST, ils ont même des propriétés aléatoires supérieures à celles des générateurs couramment utilisées pour la génération de vecteurs de test aléatoires. Etant donné que ces générateurs ont été initialisés avec des données aléatoires au cours des expérimentations, nous pouvons supposer que quelque soit la valeur de ces données d'initialisation, les séquences générées seront toujours aléatoire.

3.4 TEST A L'AIDE D'ALGORITHME DE CHIFFREMENT

Cette partie sera consacrée à la simulation de fautes de circuits séquentiels intégrant des chaînes de scan. Le but étant de s'assurer que les séquences générées avec les cœurs de chiffrement fournissent des taux de couverture au moins équivalent à ceux obtenus avec un LFSR.

3.4.1 EXPERIENCE SUR CIRCUITS ISCAS'89

Les circuits séquentiels étudiés sont trois circuits typiques de la gamme des ISCAS'89 [Brgl89]. Les trois circuits ont été choisis de façon à être représentatifs de l'ensemble des benchmarks. Ils ont été choisis en fonction du nombre de leurs bascules. Les circuits sont : le circuit s9234 composé de 247 bascules, le circuit s13207 composé de 699 bascules et le s38584 composé de 1464 bascules.

Deux configurations ont été considérées : le cas où le circuit contient une seule chaîne de scan et le cas où il en contient 64. Les chaînes de scan ont été intégrées à l'aide de l'outil Design Compiler de Synopsys [Dcomp]. L'architecture de test utilisée est l'architecture STUMPS présentée précédemment (cf. figure 3-5). Les simulations de fautes ont été réalisées à l'aide de l'outil de Synopsys Tetramax [Tmax].

La procédure utilisée pour réaliser ces simulations de fautes est présentée sur la figure 3-12. Après avoir généré la séquence à l'aide du cœur de chiffrement, cette séquence est mise sous la forme d'un fichier de vecteurs (format STIL : Standard Test Interface Language) lisible par le logiciel Tetramax. Le circuit à tester est simulé (simulation logique) avec les vecteurs de la séquence de test afin d'obtenir les réponses du circuit sain aux vecteurs de test. Vecteurs de test et réponses attendues sont utilisés pour réaliser la simulation de fautes. A la fin de la simulation de fautes, le taux de couverture obtenu est relevé.

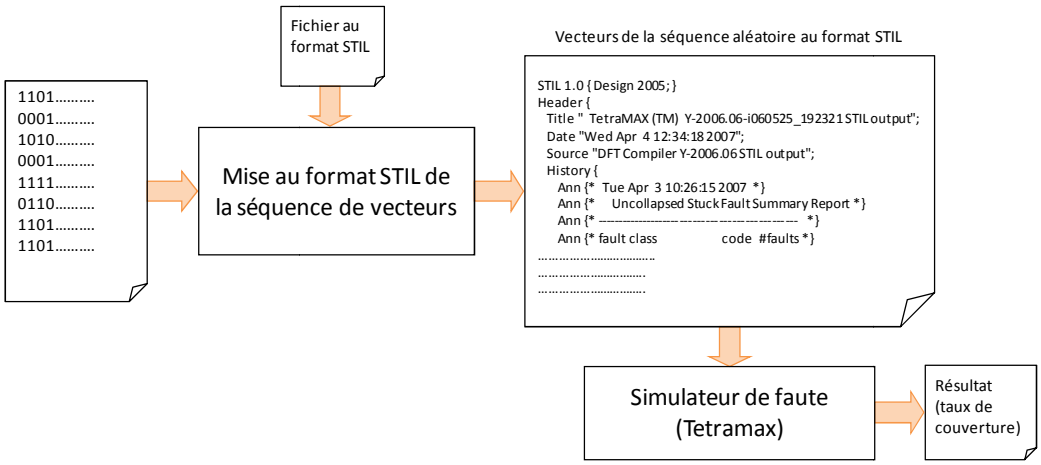


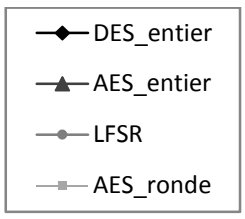
FIGURE 3-12 : PROTOCOLE UTILISER POUR LA SIMULATION DE FAUTES

Ces simulations ont été réalisées avec des séquences générées à l'aide d'un LFSR 128 bits et des cœurs de chiffrement AES et DES.

Deux configurations ont été traitées une où le générateur alimente une seule chaîne de scan et une où il alimente plusieurs chaînes de scan. L'avantage de chaînes de scan multiple est de réduire le temps de test. Pour nos simulations, une approche où le circuit sous test contient un nombre élevé de chaîne de scan a été privilégiée. Par conséquent, le cas où un générateur ***DES_ronde*** alimente le circuit sous test n'a pas été traité car le nombre de chaînes de scan serait limité à 32. Cependant, les résultats obtenus aux tests du NIST, laisse penser que le ***DES_ronde*** se comporte de façon similaire aux autres générateurs (***AES_ronde***, ***LFSR***, …).

Les taux de couvertures obtenus en appliquant ces séquences aux 3 circuits ISCAS'89 sont présentés sur la figure 3-13. Nous avons choisi de représenter ces résultats sous forme de courbe pour en faciliter leur lecture.

La légende utilisée est la suivante :



L'axe des abscisses représente le nombre de vecteurs de test appliqués au circuit sous test et l'axe des ordonnées représente le taux de couverture obtenu.

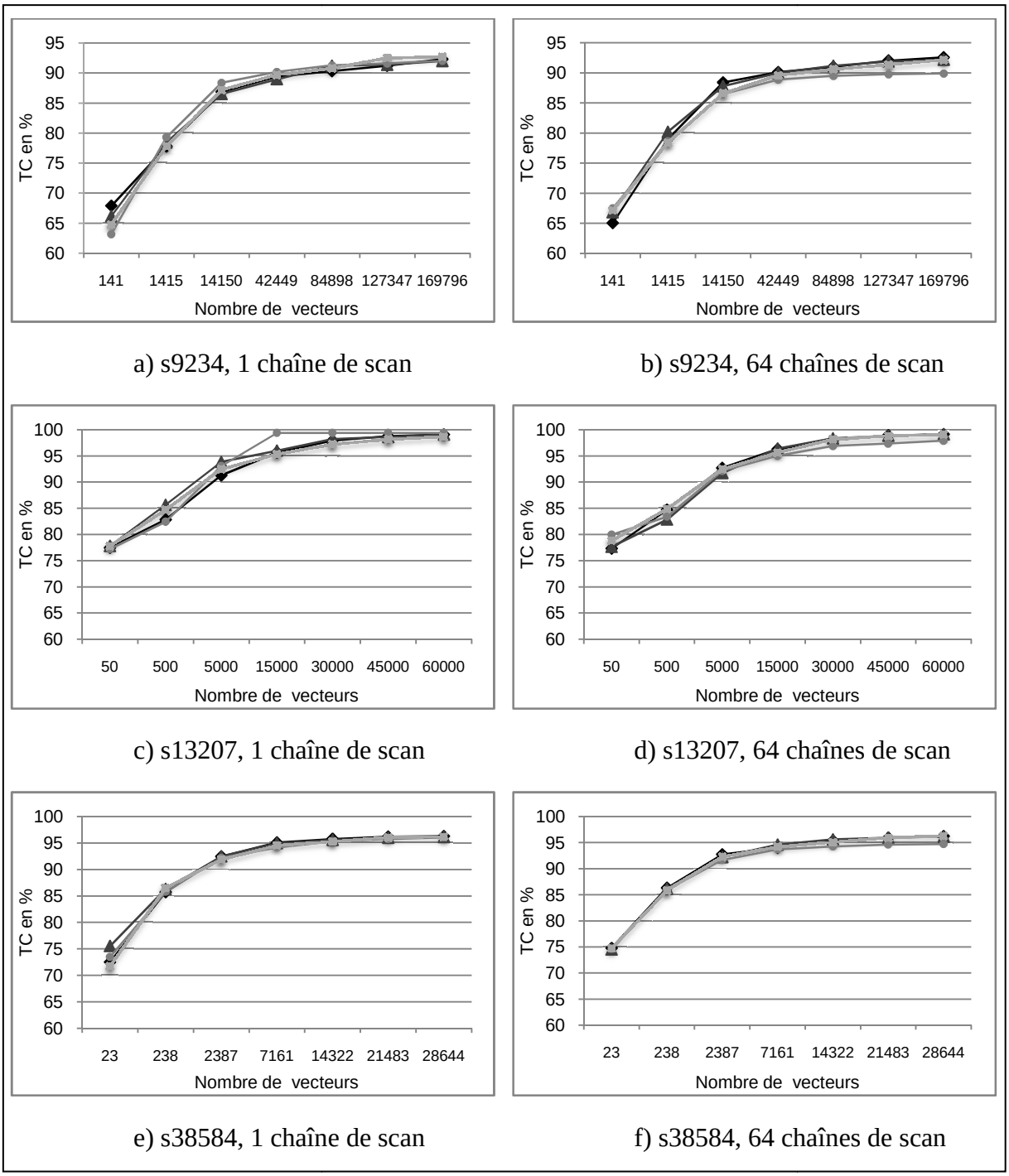


FIGURE 3-13 : TC OBTENU PAR SIMULATION DE FAUTES

Comme l'illustre la figure 3-13, quelque soit le circuit et quelque soit la configuration de test 1 chaîne de scan ou 64 chaînes de scan, les taux de couverture (TC) obtenus avec les divers générateurs (LFSR et cœurs de chiffrement) sont du même ordre de grandeur.

Sur la figure 3-13, la notion de temps de test est représentée par le nombre de vecteurs de test appliqués. Il est à noter que cette notion ne tient pas compte du temps de génération de la séquence de test. Prenons comme exemple le cas où le circuit s13207 reçoit 60000 vecteurs.

Pour rappel, dans la configuration une chaîne de scan la longueur de la chaîne est de 699 bascules et dans la configuration 64 chaînes il y a 59 chaînes de longueur 11 et 5 de longueur 10. La durée de génération est égale au produit nombre de vecteurs de test par longueur de la chaîne de scan (la plus longue). Dans le tableau 3-9, les résultats en termes de taux de couverture de fautes sont fournis ainsi que le nombre de cycles de rondes ou d'horloge qui ont été nécessaires à la génération de la séquence de test.

		LFSR	***DES_entier***	***AES_entier***	***AES_ronde***
1 chaîne de scan	Taux de couverture	99,38	99,09	99,06	98,66
	# cycles de rondes ou d'horloge	41 940 000	671 040 000	419 400 000	41 940 000
64 chaînes de scan	Taux de couverture	97,9	99, 14	99, 13	99, 08
	# cycles de rondes ou d'horloge	660 000	10 560 000	6 600 000	660 000

TABLEAU 3-9 : DUREE DE GENERATION DE LA SEQUENCE DE TEST

L'utilisation de chaînes de scan multiples permet bien sûr de réduire le temps de test. De plus, les générateurs basés sur les cœurs de chiffrement fournissent les meilleurs taux de couverture pour un nombre de vecteurs équivalent quand le circuit contient plusieurs chaînes de scan. Le problème de la corrélation entre flot de bits en sortie du LFSR évoqué plus haut explique ce dernier résultat.

3.5 CONCLUSION

Dans ce chapitre, nous avons mené une étude sur l'utilisation de cœur de chiffrement comme générateur de vecteurs de test et plus particulièrement sur l'utilisation des cœurs AES et DES. Cette étude a consisté à s'assurer que les séquences générées à l'aide des cœurs de chiffrement peuvent être considérées comme aléatoires. Pour ce faire, nous avons appliqué la série de tests statistiques fournie par le NIST les résultats obtenus ont montré que les séquences générées à l'aide des cœurs de chiffrement sont au moins aussi aléatoires que celles générées à l'aide d'un générateur LFSR.

Ensuite, les séquences générées par les cœurs de chiffrement ont été appliqué à trois circuits ISCAS'89. Les taux de couverture obtenus sur ces trois circuits sont du même ordre

de grandeur que ceux obtenus avec une séquence générée par un LFSR. De plus quand le circuit contient plusieurs chaînes de scan, les meilleurs taux de couverture sont obtenus avec les séquences générées avec les cœurs de chiffrement.

En conclusion, dans le contexte des circuits sécurisés, il n'est pas nécessaire d'implanter un générateur de vecteurs aléatoires de test car le cœur de chiffrement déjà implanté peut le remplacer. Cette solution permet un gain en surface par rapport à une solution traditionnelle de test qui consiste à implanter un générateur type LFSR et la connectique et les multiplexeurs nécessaires à son implantation.

Chapitre 4 : La compaction de réponses

Chapitre 4 : LA COMPACTION DE REPONSES

Ce chapitre est consacré à l'étude d'une solution intégré pour la compaction de réponses de test. Plus particulièrement, nous étudierons la possibilité de réutiliser le cœur de chiffrement comme compacteur de réponses.

Dans un premier temps, les différentes techniques de compaction de réponses de test seront présentées, y compris la solution de compaction proposée. Dans un deuxième temps, le phénomène de masquage de fautes sera détaillé et le calcul de la probabilité de masquage pour notre structure de compaction de réponses sera présenté.

4.1 PRINCIPE DE L'ANALYSE DES REPONSES DE TEST

La détection d'une faute nécessite de comparer, après application de chaque stimulus de test, les sorties du circuit sous test aux réponses du circuit exempt de faute. Contrairement au test externe, où les réponses du circuit aux vecteurs de test sont renvoyées vers le testeur pour être comparées, en test intégré la comparaison s'effectue sur le circuit lui-même. Cela signifie qu'il faut mémoriser les réponses attendues.

En test externe, ces réponses sont stockées dans la mémoire du testeur et comparées au fur et à mesure avec les réponses réelles. En test intégré le stockage de ces réponses, dans une mémoire interne au circuit n'est généralement pas la solution retenue. En effet le surcoût en surface lié à cette mémorisation (ROM, générateur d'adresses) rend cette solution trop onéreuse.

Pour résoudre ce problème, des techniques de compaction (de compression) de réponses de test sont généralement utilisées. Le compacteur permet de réduire l'ensemble des réponses du circuit sous test en une donnée représentative couramment appelée « *signature* ». Après application de chaque vecteur de test, la valeur de la signature est modifiée en fonction de la valeur obtenue sur les sorties du circuit et de sa valeur précédente. A la fin de la phase de test, seule la dernière valeur de la signature est comparée avec celle attendue pour le circuit exempt

de faute. La comparaison de la signature avec la signature attendue peut être faite soit sur le circuit, soit de façon externe.

4.1.1 LES TECHNIQUES DE COMPACTION DE REPONSES DE TEST

De nombreuses techniques de compaction de données ont été proposées : des techniques par vérification de parité [Cart82], des techniques de comptage [Savi85] et des techniques utilisant des LFSRs [Kone80, Hass83, Dav86] et [Bard87], que nous détaillerons ci-après :

a) Vérification de la parité :

La figure 4-1 illustre le principe de cette technique de compaction pour un circuit à une seule sortie. La signature est codée sur un seul bit. Ce bit représente la parité paire ou impaire.

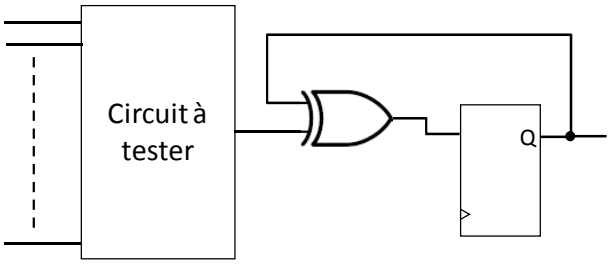


FIGURE 4-1 : COMPRERSSION PAR VERIFICATION DE PARITE

La parité est représentée par la valeur du bit sortant. Après chaque application d'un vecteur de test, le résultat obtenu en sortie du circuit est additionné modulo 2 (XOR) avec la parité préalablement stockée dans une bascule. A la fin de la phase de test, la dernière valeur sortant de la bascule représente la signature. Cette technique de compaction par vérification de parité permet de détecter toutes les erreurs simples sur un flot de bits et toutes les erreurs multiples qui entraînent un nombre d'erreurs impair dans le flot.

Dans le cas d'un circuit ayant plusieurs sorties, deux approches sont envisageables. La première consiste à associer un vérificateur de parité à chaque sortie. La deuxième moins coûteuse consiste à regrouper les sorties avant la compaction (voir figure 4-2).

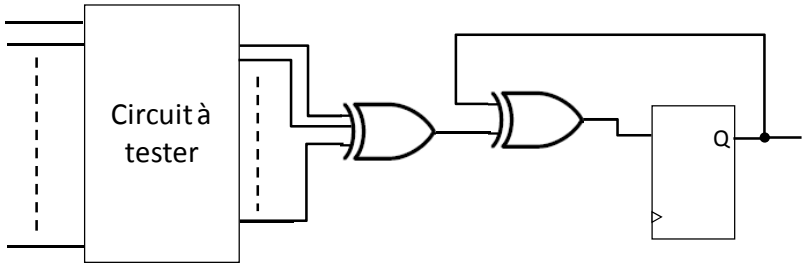


FIGURE 4-2 : COMPACTION PAR VERIFICATION DE LA PARITE POUR CIRCUIT A SORTIES MULTIPLE

Les sorties du circuit sont additionnées modulo 2 avant d'alimenter le vérificateur de parité.

Dans les deux cas, que ce soit un circuit ayant une seule sortie ou un circuit ayant plusieurs sorties, la parité obtenue après avoir appliqué la séquence de test est comparée à celle obtenue avec un circuit sain.

b) Comptage :

Deux techniques de comptage ont été proposées, le comptage des « uns » dans la séquence de sortie du circuit sous test et le comptage des transitions [Hay76].

Comme son nom l'indique, la technique de comptage des « uns » compte le nombre de 1 obtenu en sortie du circuit. Par exemple, pour un circuit (ayant une seule sortie) auquel N vecteurs de test sont appliqués, le nombre de 1 compté en sortie va varier de 0 à N. Le résultat obtenu à la fin sera comparé avec celui obtenu avec un circuit sain.

La technique de comptage des transitions consiste à compter le nombre de transitions (0 vers 1 et 1 vers 0) obtenues en sortie du circuit et à le comparer au nombre attendu pour le circuit sain.

c) Utilisation des LFSRs :

Les structures de compaction de réponses les plus classiques sont le SISR (« Single Input Shift Register ») et le MISR (« Multiple Input Shift Register »), respectivement utilisées quand le circuit possède une et plusieurs sorties. Ces deux structures sont basées sur l'utilisation de LFSR.

Le SISR est réalisé à partir d'un simple LFSR à une entrée. La signature est constituée par le contenu du registre une fois que le dernier bit d'entrée a été envoyé. Une structure de SISR est représentée sur la figure 4-3.

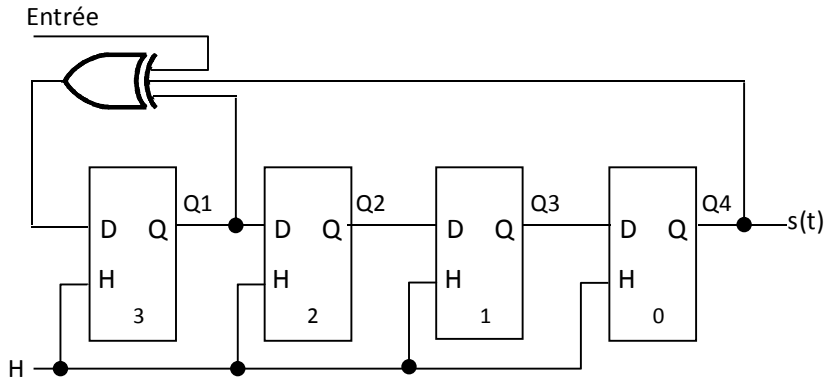


FIGURE 4-3 : STRUCTURE D'UN SISR

La signature obtenue correspond au reste de la division de la séquence d'entrée (considérée comme un polynôme) par le polynôme caractéristique du LFSR.

Le MISR proposé par [Kone80] est présenté sur la figure 4-4 :

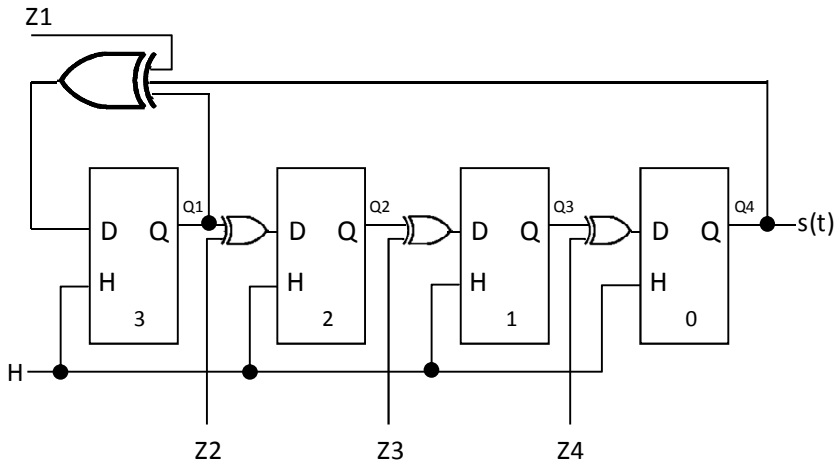


FIGURE 4-4 : STRUCTURE D'UN MISR

Celui présenté ici est composé de 4 bascules, reliées en série, subissant une rétroaction à travers une porte XOR à 3 entrées (LFSR de type 1 implanté suivant le polynôme $p(x) = x^4 + x + 1$). Les entrées Zi du MISR correspondent aux sorties du circuit sous test. En pratique, la taille du MISR (nombre de bascules ou d'étages) est supérieure ou égale au nombre de sorties du circuit sous test.

Récemment, des structures également basées sur l'utilisation de portes « ou-exclusif » et de bascules ont été proposées afin de tolérer des valeurs inconnues dans les réponses [Raj08] [Mit04] et [Mit04bis].

4.1.2 MISE EN ŒUVRE DANS LE CAS D'UN CŒUR DE CHIFFREMENT

L'approche que nous proposons consiste à utiliser le cœur de chiffrement comme compacteur de réponses. Les cœurs de chiffrement, utilisés en mode de chiffrement dit : « à chaînage de blocs (CBC pour Cipher Block Chaining) » ont comme propriété de propager une erreur se produisant sur l'entrée [Sch97]. Ainsi, une erreur dans un texte en clair affecte le texte chiffré correspondant et tous les textes chiffrés qui suivent. La structure du mode CBC est illustrée sur la figure 4-5.

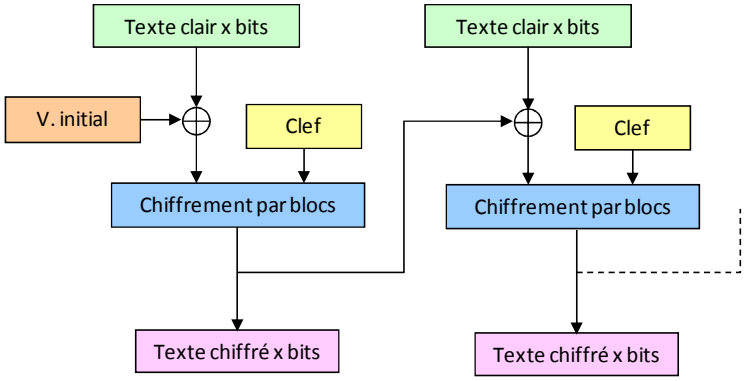


FIGURE 4-5 : MODE CBC AVEC UN ALGORITHME DE CHIFFREMENT PAR BLOC DE TAILLE M

Le mode CBC est employé pour du chiffrement par blocs. C'est-à-dire que les données à chiffrer sont traitées sous forme de bloc de m bits. Ce mode de chiffrement utilise une méthode de rétroaction permettant de chiffrer le bloc courant à l'aide du résultat du chiffrement du bloc précédent. En d'autres termes, le texte en clair est ajouté (« ou-exclusif ») avec le bloc chiffré précédent avant d'être chiffré à son tour. Ainsi le chiffrement de chaque bloc dépend de tous les messages précédents.

Cette propriété de propagation de toute modification altérant un bloc à chiffrer est particulièrement intéressante dans notre cas. En effet, si un seul bit est modifié dans le bloc de texte en clair alors le bloc de texte chiffré correspondant et les blocs de texte chiffré suivants sont affectés. Dans notre étude le bloc à chiffrer correspond à un ensemble de réponses du circuit sous test qu'il nous faut compacter et toute modification de ces réponses (présence d'une erreur) doit être visible en fin de processus.

Une solution consisterait donc à utiliser le cœur de chiffrement en mode CBC pour réaliser la compaction des réponses du circuit sous test.

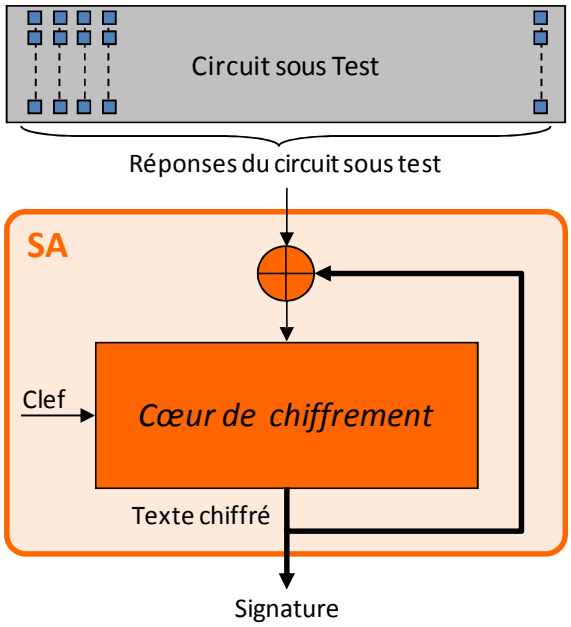


FIGURE 4-6 : COEUR DE CHIFFREMENT EN MODE COMPACTION DE REPONSES

Chaque réponse du circuit est additionnée modulo 2 avec le résultat du précédent chiffrement (voir figure 4-6). Après avoir compacté l'ensemble des réponses de test, le dernier texte chiffré obtenu correspond à la signature. Comme pour toutes les structures de compaction de réponses de test présentées, la signature obtenue doit être comparée avec celle préalablement déterminée par simulation du circuit sain.

Cependant cette solution nécessite un cycle de chiffrement par réponse du circuit sous test. Or un cycle de chiffrement est composé de plusieurs cycles de rondes (16 pour un DES et 10 pour un AES), donc de plusieurs cycles d'horloge. La synchronisation avec l'acquisition des réponses de test à compacter n'est donc pas directe (attente de plusieurs cycles de chiffrement avant l'acquisition d'une nouvelle réponse) et le processus de chiffrement dans ce cas ralentit le test.

Une deuxième solution consiste à modifier le module de chiffrement pour compacter une nouvelle réponse du circuit sous test à chaque ronde. La nouvelle structure de compaction de réponses de test est illustrée sur la figure 4-7.

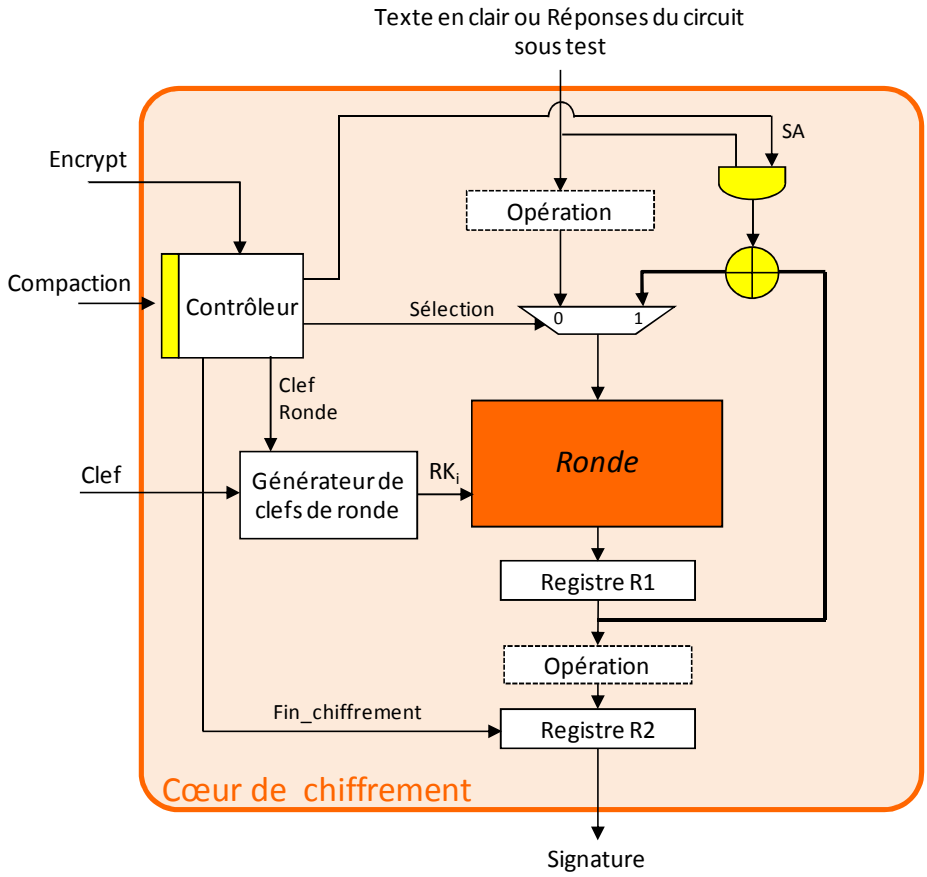


FIGURE 4-7 : MODE COMPACTION DE REPONSES, UTILISANT LA RETROACTION DE RONDE

A chaque cycle, la sortie de la ronde est additionnée modulo 2 avec une réponse du circuit sous test. A la fin de la procédure de test, le dernier résultat de ronde est récupéré en sortie du registre R2. Ce résultat (signature) est comparé avec la signature attendue. Cette solution permet de compacter les réponses du circuit sous test à chaque cycle d'horloge (cycle de ronde) comme le font les modules de compaction de réponses de type MISR.

Afin d'utiliser le cœur de chiffrement en tant que compacteur de réponses, les modifications à lui apporter sont :

- Modification du contrôleur, afin d'exécuter un nombre de rondes supérieur au nombre normalement requis pour un chiffrement. La ronde de chiffrement ne subit aucune modification. Dans le cas de l'AES, par exemple, le motif de 9 rondes composées de quatre opérations, suivi d'une ronde composée de 3 opérations est respecté.
- Ajout d'une opération de « et logique » entre l'entrée (texte en clair ou réponses du circuit sous test) et le signal SA. Ce signal SA est rajouté pour le mode d'analyse de signature. Il est commandé par le contrôleur.

- Modification de la rétroaction de ronde, une opération d'« ou-exclusif » entre le résultat de la précédente ronde et la sortie du « et logique » est rajoutée.

Les modifications sont faites pour gérer les deux modes : le mode correspondant à un chiffrement (dit « mode normal ») et le mode correspondant à l'utilisation du cœur de chiffrement comme compacteur de réponses. En mode normal, le signal SA est toujours affecté à 0, ce qui inhibe l'effet du « ou-exclusif » rajouté dans la rétroaction. Et en mode compaction de réponses, les signaux SA et Sélection sont toujours affectés à 1. Ainsi la réponse courante du circuit sous test est compactée avec les précédentes réponses (sortie de la ronde).

Notre architecture pourra être utilisée pour des circuits ayant un nombre de sorties au plus égal à la taille du bloc de données traité par le cœur de chiffrement utilisé, c'est-à-dire 128 sorties pour un AES et 64 pour un DES. Si par contre le circuit sous test étudié a un nombre de sorties inférieur à la taille du bloc (m), la solution consiste à appliquer sur les entrées restantes toujours la même valeur : soit 1, soit 0.

Cette solution demande néanmoins de s'assurer que, comme dans le cas du mode CBC, toute erreur, ou ensemble d'erreurs, présente(nt) dans le flot de réponses à compacter restera visible dans la signature finale. Ce point est discuté dans le chapitre suivant.

4.2 EVALUATION DE LA PROBABILITE DE MASQUAGE

4.2.1 DESCRIPTION DU PHENOMENE DE MASQUAGE

Intrinsèquement, la génération de la signature entraîne une perte d'informations. En effet seule la signature est observée et non pas l'ensemble des réponses du circuit sous test. Par conséquent, la présence d'erreurs peut être masquée par le module de compaction, les sorties erronées se compensant pour donner la signature attendue à la fin du test.

Le masquage porte atteinte à la qualité du test, dans le sens où les vecteurs de test sensibilisent bien la faute mais la compaction perd cette information. Le choix du module de compaction devra être fait afin de réduire le masquage d'erreurs.

Nous fournissons pour chacun des analyseurs de signature présentés précédemment la probabilité de masquage. Ces résultats sont énoncés rapidement, le détail des calculs étant disponible dans la littérature.

a) Techniques par vérification de la parité (cas d'un circuit a une seule sortie) :

Parmi toutes les séquences possibles de longueur m (soit 2^m séquences), la moitié a une parité paire et l'autre moitié a une parité impaire. Il existe donc, quelle que soit la parité du circuit sain, $(2^m/2)$-1 séquences erronées conduisant à la même signature. La probabilité de masquage est donc égale à :

$$P = \frac{\frac{2^m}{2}-1}{2^m-1}.$$

Pour des valeurs de m importantes, la probabilité tend vers ½.

b) Techniques de comptage :

Supposons que le circuit sain ait une séquence de réponses S de longueur m et dont le nombre de transition est r. Considérons une séquence de réponse T quelconque de longueur m. Le nombre de transitions de T peut être compris entre 0 et m-1. Or si T doit contenir r transitions alors le nombre de séquences ayant r transitions vaut C^r_{m-1}. De plus, la séquence $\overline{T}$ (complément de T) a le même nombre de transition que T. Il y a donc $2C^r_{m-1}$ séquences possibles conduisant à la même signature. Par conséquent la probabilité de masquage est donc égale à :

$$P = \frac{2C^r_{m-1} - 1}{2^m - 1}.$$

c) Techniques utilisant des LFSRs :

Que ce soit pour le SISR ou pour le MISR, l'analyse de la probabilité de masquage repose sur l'hypothèse que toutes les réponses fausses sont équiprobables.

1. SISR :

Pour un flot de données entrant de longueur n, la probabilité pour qu'un générateur de signature de taille m ne détecte pas l'erreur est :

$$P = \frac{2^{n-m} - 1}{2^n - 1}.$$

Si m est très grand par rapport à n alors la probabilité peut être réduite à $P \approx \frac{1}{2^m}$.

Pour un grand nombre de réponses de test (n), la probabilité de déclarer un circuit sain alors que celui est fautif (masquage) dépend de la taille m du SISR.

2. MISR :

Concernant le MISR, supposons n le nombre de réponses de test et m la taille du MISR, la probabilité de masquage s'écrit :

$$P = \frac{2^{n-1} - 1}{2^{m+n-1} - 1} \approx \frac{1}{2^m}.$$

La probabilité de générer une signature correcte alors qu'une erreur s'est produite sur une des sorties du circuit est alors égale à la probabilité de générer n'importe quelle signature. Sous l'hypothèse que toutes les erreurs sont équiprobables pour un grand nombre de réponses de test, la probabilité de déclarer sain un circuit fautif est de $1/2^m$.

4.2.2 CALCUL DE LA PROBABILITE DE MASQUAGE D'UNE STRUCTURE AES

Dans cette partie, nous détaillerons le calcul de la probabilité de masquage d'une structure de compaction utilisant un cœur de chiffrement AES. Avec la même approche, nous avons établi la probabilité de masquage d'une structure utilisant le cœur de chiffrement DES.

La figure 4-8 est un schéma simplifié de la structure étudiée pour calculer la probabilité de masquage. Cette structure présente les deux parties importantes qui sont la ronde et l'opération d'« ou-exclusif » entre le résultat de ronde et la réponse du circuit sous test.

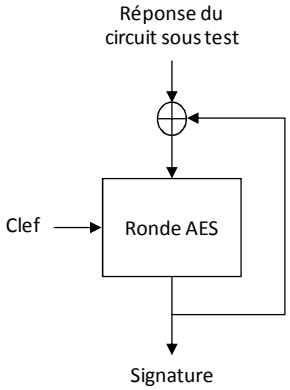


FIGURE 4-8 : STRUCTURE DE L'AES EN MODE DE COMPACTION DE REPONSES

La signature obtenue à l'aide d'un compacteur de réponses peut être égale à celle préalablement calculé si : le circuit est sain donc s'il n'y a pas d'erreur ou si le circuit contient une ou plusieurs erreurs compensées par le module de compaction.

Nous utiliserons pour la démonstration les notations suivantes : S_n la signature attendue à l'instant n si le circuit est sain, où n représente le nombre de cycles de test qui ont été accomplis. La réponse à l'instant n est notée R_n. Les termes S_n^* et R_n^* représentent respectivement la signature réelle obtenue en appliquant la séquence de n vecteurs de test et la réponse courante du circuit sous test. La fonction de compaction réalisée par notre structure de compaction de réponses est notée Comp. A l'instant n, on peut alors écrire que : $S_n = \mathrm{Comp}(S_{n-1}, R_n)$.

Dans la suite, Comp sera l'abréviation de $\mathrm{Comp}(S_{n-1}, R_n)$ et Comp^* sera l'abréviation de $\mathrm{Comp}(S_{n-1}^*, R_n^*)$.

Le phénomène de masquage c'est-à-dire $S_n = S_n^*$ alors qu'il existe au moins une réponse telle que $R_i^* \neq R_i$, peut se produire dans une des 4 situations suivantes :

1) La signature courante et la réponse de test sont respectivement égales à celles attendues, mais le phénomène de masquage a eu lieu durant les n-1 premiers cycles de test, c'est-à-dire que $(S_{n-1}^*, R_n^*) = (S_{n-1}, R_n)$.

2) La signature courante est différente de celle attendue. Par contre, la réponse de test est égale à celle attendue mais le phénomène de masquage se produit au cycle n, c'est-à-dire que $S_{n-1}^* \neq S_{n-1}$ et $R_n^* = R_n$.

3) La réponse de test courante est différente de celle attendue. Par contre, la signature est égale à celle attendue mais le phénomène de masquage se produit au cycle n, c'est-à-dire que $R_n^* \neq R_n$ et $S_{n-1}^* = S_{n-1}$.

4) La signature courante et la réponse de test sont différentes de celles attendues mais le phénomène de masquage se produit au cycle n, c'est-à-dire que $S_{n-1}^* \neq S_{n-1}$ and $R_n^* \neq R_n$.

Pour simplifier la démonstration, notons les différentes conditions sous la forme suivante :

Condition R: $(R_n = R_n^*)$ et condition $\overline{R}$: $(R_n \neq R_n^*)$

Condition S: $(S_{n-1} = S_{n-1}^*)$ et condition $\overline{S}$: $(S_{n-1} \neq S_{n-1}^*)$.

En conséquence, la probabilité que la signature courante soit égale à celle attendue peut s'écrire de la façon suivante :

$$\begin{aligned} P(S_n = S_n^*) = {} & P(S \cap R) \cdot P\left[(Comp = Comp^*)/(S,R)\right] \\ & + P(\bar{S} \cap R) \cdot P\left[(Comp = Comp^*)/(\bar{S},R)\right] \\ & + P(S \cap \bar{R}) \cdot P\left[(Comp = Comp^*)/(S,\bar{R})\right] \\ & + P(\bar{S} \cap \bar{R}) \cdot P\left[(Comp = Comp^*)/(\bar{S},\bar{R})\right] \end{aligned}$$

Le terme $P\left[(Comp = Comp^*)/(S,R)\right]$, revient à calculer la probabilité que le module de compaction donne la même signature sachant que la signature à l'instant n-1 est égale à la signature attendue et que la réponse du circuit à l'instant n soit égale à celle attendue. Ce terme $P\left[(Comp = Comp^*)/(S,R)\right]$ sera toujours égal à 1.

La propriété de bijection inhérente au cœur de chiffrement AES permet de simplifier les autres termes de l'équation (c'est aussi le cas pour le DES [Mur02, You05]). La ronde réalisée durant le chiffrement est une opération bijective. En conséquence, s'il existe une différence sur l'entrée de ronde entre la valeur courante et celle attendue, la différence sera toujours visible sur les valeurs obtenues en sortie de la ronde.

Pour illustrer cette propriété, considérons l'entrée de ronde attendue égale à : 10 22 A9 BF 90 17 B0 CD CC D1 24 56 13 DF AA B4 (valeur en hexadécimal) et l'entrée courante à : 10 22 A**B** BF 90 17 B0 CD CC D1 24 56 13 DF AA B4. A présent, regardons les résultats obtenus en sortie de ronde. Pour cela on suppose une clef de ronde égale à RKi = A0 FA FE 17 88 54 2C B1 23 A3 39 39 2A 6C 76 05.

Premièrement, considérons la propagation de l'erreur à travers une des 9 premières rondes du cœur AES. Les structures des 9 premières rondes étant identiques, la propagation de l'erreur suivra le même modèle quelle que soit la ronde. Sur la figure 4-9, nous voyons que la différence en entrée se retrouve en sortie et qu'elle est même amplifiée. La partie supérieure de la figure représente les étapes de ronde pour la réponse attendue et la partie inférieure celles pour la réponse obtenue. La différence en entrée était sur un octet (octet coloré, voir partie gauche de la figure), on la retrouve en sortie sur 4 octets (partie droite de la figure). Pour rappel c'est la propriété qui est utilisée pour réaliser l'attaque décrite au chapitre 2.

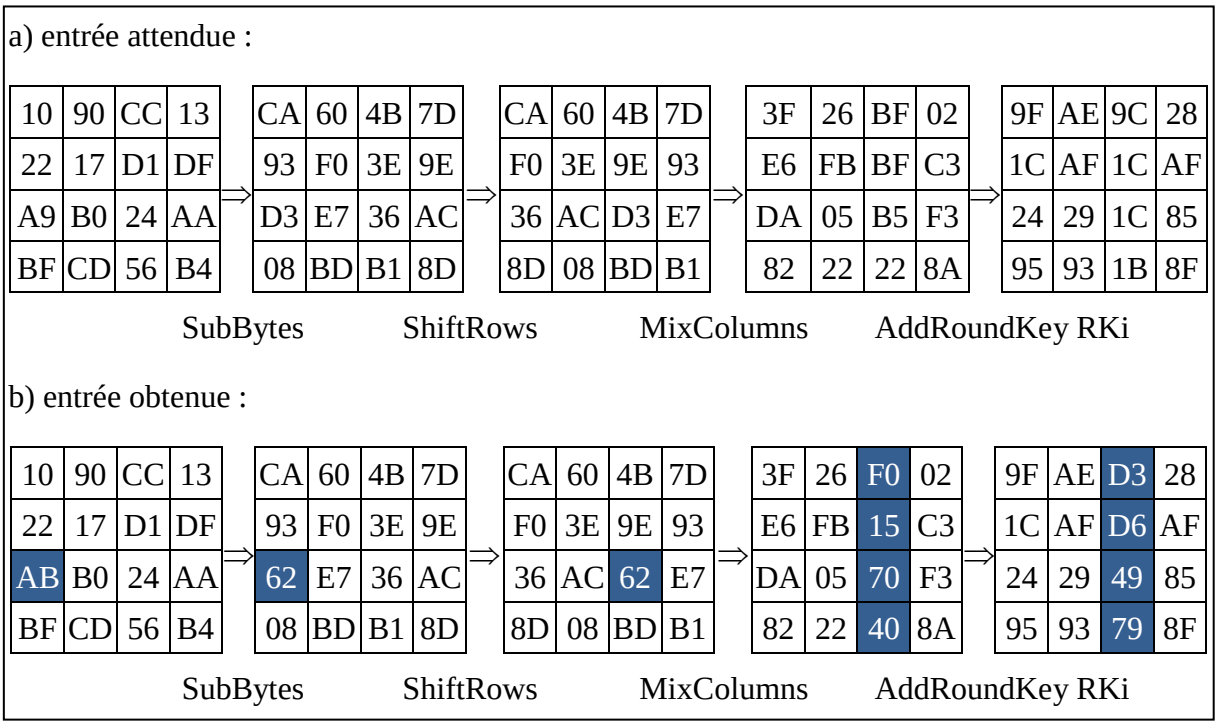


FIGURE 4-9 : PROPAGATION DE L'ERREUR A TRAVERS UNE DES 9 PREMIERES RONDES

Maintenant si l'on considère la 10ème ronde de l'AES, une différence sur un octet en entrée va se retrouver en sortie sur un seul octet car l'opération MixColumns n'est pas réalisée (voir la figure 4-10).

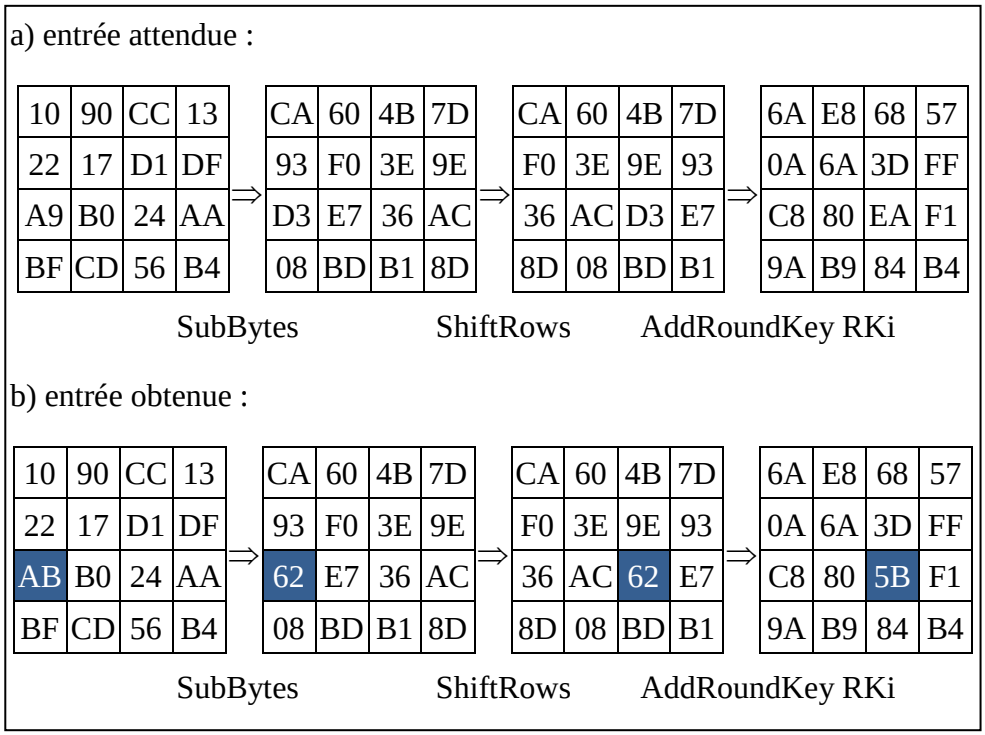


FIGURE 4-10 : PROPAGATION DE L'ERREUR A TRAVERS LA DIXIEME RONDE

Etant donné que la ronde est bijective, la probabilité d'avoir deux signatures identiques en sortie de l'AES est équivalente à la probabilité de masquage du « ou-exclusif », qui est rajouté sur la rétroaction de ronde.

Donc, la probabilité de masquage de la fonction de compaction peut s'écrire :

$$P(\text{Comp}=\text{Comp}^*)=P(\text{xor}_{\text{masquage}})$$
$$=P\left[\left(R_n \oplus S_{n-1}\right)=\left(R_n^* \oplus S_{n-1}^*\right)\right].$$

En conséquence, les termes $P\left[(\text{Comp}=\text{Comp}^*)/(\bar{S},R)\right]$ et $P\left[(\text{Comp}=\text{Comp}^*)/(S,\bar{R})\right]$ sont égaux à 0, car une porte « ou-exclusif » ne peut pas fournir en sortie le même résultat si seul un des opérandes est différent (ex : $0 \oplus 0 = 0$, $0 \oplus 1 = 1$).

Concernant le dernier terme $P\left[(\text{Comp}=\text{Comp}^*)/(\bar{S},\bar{R})\right]$, cela revient à déterminer :

$P(\text{xor}_{\text{masquage}}) = P[(a_i \oplus b_i)=(c_i \oplus d_i)]$, sachant que $a_i \neq c_i$ et $b_i \neq d_i$.

Si les valeurs sont codées sur :

- Un bit : $P(\text{xor}_{\text{masquage}}) = 1$.
- Deux bits :

Considérons l'exemple suivant : $a_i = b_i = \{00\}$, d'où $a_i \oplus b_i = \{00\}$. Les 9 combinaisons envisageables pour le couple (c_i, d_i) sachant que $a_i \neq c_i$ et $b_i \neq d_i$ sont :

c_i	d_i	**Résultat** $c_i \oplus d_i$
01	**01**	**00**
01	10	11
01	11	10
10	01	11
10	**10**	**00**
10	11	01
11	01	10
11	10	01
11	**11**	**00**

Parmi ces 9 combinaisons, seules 3 (soulignées) fournissent le même résultat que $a_i \oplus b_i$.

En résumé, il y a 16 combinaisons possibles pour le couple (a_i, b_i). Le choix d'un couple implique un choix entre 9 couples (c_i, d_i) tel que $a_i \neq c_i$ et $b_i \neq d_i$. D'où 16×9 combinaisons sont possibles. Or pour un couple (a_i, b_i) seul 3 couples de (c_i, d_i) fournissent le même résultat. Alors 16×3 combinaisons fournissent le même résultat. La probabilité de masquage vaut donc :

$$P\left(xor_{masquage}\right) = \frac{3 \times 16}{9 \times 16} = \frac{3}{9}$$

- De façon générale, pour m bits, la probabilité de masquage vaut :

$$P\left(xor_{masquage}\right) = \frac{2^m - 1}{(2^m - 1)^2} = \frac{1}{2^m - 1}$$

La probabilité d'avoir deux signatures identiques s'écrit :

$$P\left(S_n = S_n^*\right) = P(S \cap R) + P(\bar{S} \cap R) \cdot 0 + P(S \cap \bar{R}) \cdot 0 + P(\bar{S} \cap \bar{R}) \cdot \frac{1}{2^m - 1}$$

$$= P(S \cap R) + P(\bar{S} \cap \bar{R}) \cdot \frac{1}{2^m - 1}$$

De plus, les deux évènements $S_{n-1} = S_{n-1}^*$ et $R_n = R_n^*$ (ou $S_{n-1} \neq S_{n-1}^*$ et $R_n \neq R_n^*$) étant indépendants, la probabilité d'avoir deux signatures égales s'écrit :

$$P\left(S_n = S_n^*\right) = P(S) \times P(R) + P(\bar{S}) \times P(\bar{R}) \times \frac{1}{2^m - 1}.$$

La probabilité que deux réponses de test soient identiques est égale à $1/2^m$ où m est le nombre de sorties du cœur AES (m = 128) :

$$P\left(S_n = S_n^*\right) = P(S) \times \frac{1}{2^m} + P(\bar{S}) \times \left(1 - \frac{1}{2^m}\right) \times \frac{1}{2^m - 1}.$$

La probabilité de masquage est la probabilité que la signature courante soit égale à celle attendue alors qu'au moins une réponse erronée a été chargée dans le compacteur de réponses. Elle vaut donc :

$$P(\text{masquage}) = P\left(S_n = S_n^*\right) - P(R)^n = P\left(S_n = S_n^*\right) \times \left(\frac{1}{2^m}\right)^n.$$

Supposons S_0 fixe et connue d'où $P(S_0 = S_0^*) = 1$.

- Calculons $P(S_1 = S_1^*)$:

$$\begin{aligned} P(S_1 = S_1^*) &= P(S_0 = S_0^*) \cdot P(R_1 = R_1^*) + P(S_0 \neq S_0^*) \cdot P(R_1 \neq R_1^*) \cdot \frac{1}{2^m - 1} \\ &= P(S_0 = S_0^*) \cdot P(R_1 = R_1^*) \\ &= P(R_1 = R_1^*) = \frac{1}{2^m} \end{aligned}$$

La probabilité de masquage au rang 1 est donc égale à :

$$\begin{aligned} P(\text{masquage1}) &= P(S_1 = S_1^*) - \left(\frac{1}{2^m}\right)^1 = \frac{1}{2^m} - \frac{1}{2^m} \\ &= 0. \end{aligned}$$

- Calculons $P(S_2 = S_2^*)$: pour cela, notons $P(S_1 = S_1^*) = P(E_1)$

$$\begin{aligned} P(S_2 = S_2^*) &= P(E_1) \cdot P(R_2 = R_2^*) + (1 - P(E_1)) \cdot P(R_2 \neq R_2^*) \cdot \frac{1}{2^m - 1} \\ &= P(E_1) \cdot \frac{1}{2^m} + (1 - P(E_1)) \cdot \left(1 - \frac{1}{2^m}\right) \cdot \left(\frac{1}{2^m - 1}\right) \\ &= P(E_1) \cdot \left[\frac{1}{2^m} - \frac{1}{2^m}\right] + \frac{1}{2^m} = \frac{1}{2^m} \end{aligned}$$

La probabilité de masquage au rang 2 est donc égale à :

$$\begin{aligned} P(\text{masquage2}) &= \frac{1}{2^m} - \left(\frac{1}{2^m}\right)^2 = \frac{1}{2^m} - \frac{1}{2^{2m}} \\ P(\text{masquage2}) &\to \frac{1}{2^m}. \end{aligned}$$

- Calculons $P(S_n = S_n^*)$: pour cela, notons $P(S_{n-1} = S_{n-1}^*) = P(E_{n-1})$

$$\begin{aligned} P(S_n = S_n^*) &= P(E_{n-1}) \cdot P(R_n = R_n^*) + (1 - P(E_{n-1})) \cdot P(R_n \neq R_n^*) \cdot \frac{1}{2^m - 1} \\ &= P(E_{n-1}) \cdot \frac{1}{2^m} + (1 - P(E_{n-1})) \cdot \left(1 - \frac{1}{2^m}\right) \cdot \left(\frac{1}{2^m - 1}\right) \\ &= \frac{1}{2^m}. \end{aligned}$$

La probabilité de masquage au rang n est donc égale à :

$$P(\text{masquage n}) = \frac{1}{2^m} \cdot \left(\frac{1}{2^m}\right)^n = \frac{1}{2^m} \cdot \frac{1}{2^{n \cdot m}}.$$

Quand n est grand, la probabilité de masquage tend vers $\frac{1}{2^m} = \frac{1}{2^{128}} \approx 0{,}29387 \times 10^{-38}$.

En conclusion, cette structure a une probabilité de masquage de l'ordre de $\frac{1}{2^m}$ où m est la taille du bloc de l'AES soit 128 bits.

Afin d'illustrer ce résultat théorique nous avons réalisé plusieurs expériences.

1) Première simulation : une séquence de 3000 vecteurs ayant une taille de 128 bits a été considérée pour la compaction. Elle a été choisit de façon aléatoire. Cette séquence représente la séquence des réponses du circuit sous test. Cette séquence a été compactée à l'aide du cœur de chiffrement. Les signatures obtenues sont présentées pour 5 configurations différentes (voir figure 4-11) :

1. sans erreur
2. avec un bit faux sur le 175ième vecteur, le vecteur a été choisi au hasard.
3. avec un bit faux sur le 2999ième vecteur
4. avec un bit faux sur le 175ième et le 1159ième vecteur (le même bit)
5. avec un bit faux sur le 175ième et le 1159ième vecteur (bits différents)

F0	06	23	53
8A	67	85	3F
2C	BD	2A	7B
A6	5A	5E	7E

1

4A	5A	20	CA
A4	B7	68	3B
1C	2A	4C	71
C4	A5	B7	F3

2

F0	06	23	CD
8A	67	30	3F
2C	E7	2A	7B
92	5A	5E	7E

3

39	B0	04	25
1C	CD	93	5F
F6	2B	93	C8
A1	FF	38	9B

4

C7	72	CA	BA
E4	F0	B2	F0
CC	D6	C9	69
34	2E	FE	66

5

FIGURE 4-11 : RESULTATS DE COMPACTION DE REPONSES DE TEST

Quelles que soient les configurations 2, 3, 4 ou 5, la signature obtenue représentée sous forme matricielle, est différente de celle attendue, fournie en 1. L'effet de l'erreur ou des erreurs est préservé dans la signature. En effet pour les configurations 2, 4 et 5, aucun octet obtenu en sortie n'est identique à celui attendu. Pour la configuration 3, se sont quatre octets qui sont différents de ceux attendus. Ainsi, la conclusion serait que le circuit sous test est défectueux.

2) Seconde simulations : les séquences de réponses correspondent à un circuit sous test ayant moins de 128 sorties. Les sorties du circuit sous test sont connectées au compacteur de réponses soit via des entrées adjacentes, soit via des entrées choisies de façon aléatoire. Dans tous les cas, les autres entrées non connectées du compacteur ne reçoivent que des 0. Nous avons considéré un circuit à 8 sorties et le nombre de réponses traitées est de 200.

Premier cas, les 8 sorties du circuit sous test alimentent les 8 premières entrées du cœur de chiffrement, les signatures obtenues sont présentées sur la figure 4-12.

65	49	CD	A5
58	F3	71	77
9A	37	47	27
D0	20	BC	1A

Sans erreur

B3	1B	AE	7B
DE	C3	FC	C6
63	95	FF	88
52	DF	93	DD

1 bit faux

A6	1B	EB	D0
9C	7A	30	C7
57	61	60	49
EC	E7	70	41

2 fautes sur 2 vecteurs différents

FIGURE 4-12 : RESULTATS DE COMPACTION DE REPONSES CODEES SUR 8 BITS

En présence de faute, le circuit sous test fournit des signatures différentes de celles attendues. Dans les deux cas avec un ou deux bits fautifs, l'ensemble des octets obtenus sont différents.

Deuxième cas, les sorties alimentent non plus les 8 premières entrées du cœur mais n'importe quelles autres, les signatures obtenues sont présentées sur la figure 4-13.

DA	33	2E	59
65	08	51	25
BC	03	1A	70
94	A8	84	13

Sans erreur

53	C2	F7	4E
63	9F	4D	98
1E	16	2C	41
51	13	AB	70

1 bit faux

4C	DC	48	7B
C1	52	A6	DB
B8	72	6F	13
92	12	09	DA

2 fautes sur 2 vecteurs différents

FIGURE 4-13 : RESULTATS DE COMPACTION DE REPONSES, ALIMENTANT 8 ENTREES CHOISIT AU HASARD PARMI LES 128 POSSIBLES

Les signatures obtenues en présence de faute sont différentes de celles attendues. En conclusion, que les sorties du circuit sous test alimentent des entrées adjacentes ou n'importe quelles entrées du compacteur, la présence d'erreur(s) sera visible dans la signature obtenue dans les deux cas.

Notre structure fournit donc de bons résultats en termes de masquage. Pour comparaison, la probabilité de masquage $\frac{1}{2^m}$, où m vaut 128, obtenue avec notre structure de compaction de réponses de test est équivalente à la probabilité de masquage d'un MISR composé de 128 étages.

4.2.3 LA PROBABILITE DE MASQUAGE D'UNE STRUCTURE DES

La probabilité de masquage d'une structure de compaction utilisant un cœur de chiffrement DES est égale à $\frac{1}{2^m}$ où m représente la taille du bloc entrant c'est-à-dire 64. La procédure utilisée pour déterminer cette probabilité de masquage d'une structure de compaction DES est identique à celle utilisée dans le cas de l'AES. Comme pour l'AES, l'expression de la probabilité de masquer une erreur peut être simplifiée car la ronde du DES est une bijection.

L'expérience ci-après est décrite à titre d'exemple. Considérons que l'entrée de ronde attendu est égale à A8 85 4D FF 12 B5 EF 09 (en hexadécimal), la clef de ronde égale à 24 A6 B9 27 F6 94 et que l'entrée courante contient une erreur. Du fait de la structure du DES (voir chapitre 1), nous considérerons deux cas : dans le premier l'erreur affecte la partie droite (32 bits de droite) de la réponse et dans le deuxième l'erreur affecte la partie gauche (32 bits de gauche) de la réponse.

- 1er cas :

L'entrée courante valant A8 85 4D FF 1**3** B5 EF 09 donne en sortie de ronde le résultat 13 B5 EF 09 36 71 7D C0.

- 2ième cas :

L'entrée courante valant A8 85 4D **E**F 12 B5 EF 09 donne en sortie de ronde le résultat 12 B5 EF 09 76 70 7C C0.

Sans erreur, le résultat de ronde attendu est 12 B5 EF 09 76 70 7C D0. Dans les deux cas, en présence d'erreur, le résultat de ronde obtenu est différent de celui attendu.

4.3 CONCLUSION

Dans ce chapitre, nous avons étudié l'utilisation d'un cœur de chiffrement comme compacteur de réponses dans un mode original qui consiste à compacter la réponse d'un module sous test avec les réponses précédentes à chaque cycle de ronde.

La compaction de réponses entraîne une perte d'information, et un masquage éventuel de la présence d'une erreur. Nous avons démontrer que ces structures présentaient d'excellents générateurs de signatures avec une probabilité de masquage très faible et équivalente au meilleurs procédés actuels basés sur l'utilisation de MISR. Plusieurs expériences ont permis d'illustrer ces résultats théoriques.

De plus, il faut noter que les modifications faites sur la structure du cœur de chiffrement pour son utilisation en tant que compacteur de réponses de test n'ont pas d'impact sur ces performances en mode normal de chiffrement.

Chapitre 5 : L’autotest d’un cœur de chiffrement

Chapitre 5 : L'AUTOTEST D'UN CŒUR DE CHIFFREMENT

Dans les deux précédents chapitres, nous avons montré que le cœur de chiffrement symétrique, présent dans le circuit sécurisé, pouvait être utilisé comme ressource de test pour tester les autres cœurs embarqués, soit en tant que générateur des vecteurs de test, soit en tant que compacteur de réponses. Tout ceci permet de réduire le coût du test du circuit sécurisé si et seulement si le cœur de chiffrement ne requiert pas pour son propre test l'ajout de ressource de test.

Dans ce chapitre, le test, et plus particulièrement l'autotest du cœur de chiffrement est présenté. Son principe et sa mise en œuvre sont décrits ainsi que des résultats de simulations de fautes. En dernier lieu ce chapitre présente une architecture globale permettant d'utiliser un cœur de chiffrement en tant que générateur de données de test, compacteur de réponses et en autotest.

5.1 INTRODUCTION

Nous avons montré dans le chapitre 2 que d'un point de vue sécuritaire, le test intégré est le plus adapté au contexte des circuits sécurisés. Des implantations de cœurs de chiffrement, avec leurs ressources de test, sont présentées dans [Derv, Naz05]. Dans [Naz05], les ressources de test implantées sont un générateur de vecteurs de test (LFSR de 32-bits) et un compacteur de réponses (SISR de 32-bits) représentant un coût additionnel de 0, 35% en nombre de portes équivalentes.

Une solution utilisant le cœur de chiffrement en mode d'autotest a été présenté par Bo Yang et Ramesh Karri dans [Yan05bis]. Le cœur de chiffrement étudié est un AES à 128 bits de clef. La solution d'autotest proposée par les auteurs consiste à reboucler la sortie du circuit sur l'entrée. Cette rétroaction permet de tester les fautes du module de ronde. Concernant le module de génération de clefs, la solution évoquée pour tester l'ensemble des fautes s'appuie sur l'alimentation du module avec la sortie du circuit. Ainsi à chaque cycle d'horloge, une clef

de ronde est dérivée du précédent texte chiffré. Les fautes du contrôleur ne sont pas prises en compte. Les auteurs ont évalué à 0,75% le coût en surface de leur solution et annoncent qu'avec 12 cycles de chiffrement l'ensemble des fautes sont testées.

Le manque d'informations sur l'architecture proposée dans cet article notamment sur l'implantation du cœur de chiffrement, sur la procédure d'autotest ainsi que sur les fautes testées ne nous a pas permis de reproduire ces expériences.

La solution d'autotest que nous proposons ici ne requiert aucun ajout de matériel. Cette solution repose sur l'aspect itératif du processus de chiffrement ainsi que sur les propriétés lui permettant d'être utilisé comme générateur de vecteurs aléatoires et comme compacteur de réponses.

De plus, dans [Schu99], un test pseudo-aléatoire est décrit comme une technique efficace pour le test de cœurs de chiffrement symétriques. Des taux de couvertures élevés peuvent être obtenus avec une séquence de test pseudo-aléatoire courte, car les opérations traditionnellement utilisées en cryptographie (xor, substitution, modulo…) sont facilement testées avec des données aléatoires.

D'une part, une propriété inhérente à ces opérations est de propager les données aléatoires à travers le circuit. D'autre part, les cœurs de chiffrement ont, par nature, de bonnes propriétés d'observabilité et de contrôlabilité nécessaires à leur test. En effet, le principe de diffusion, sur lequel reposent les algorithmes à clef privée, a des propriétés intéressantes pour le test d'une implantation matérielle d'un cœur de chiffrement :

- Il implique que chaque bit d'entrée d'une ronde influence plusieurs bits en sortie, c'est-à-dire que chaque ligne d'entrée d'une ronde appartient au cône de logique de plusieurs sorties. Autrement dit, une erreur causée par une faute dans la ronde est facilement propagée sur les sorties. Donc, le circuit est observable.

- De plus, une ronde est une opération de bijection, l'entrée du cône de logique de chaque sortie contient plusieurs entrées. Donc, les fautes sont fortement contrôlables.

La technique d'autotest proposée utilise la propriété que possède le cœur de chiffrement à délivrer des données aléatoires. Ces données serviront de vecteurs de test. Ainsi, en rebouclant le cœur de chiffrement, celui-ci recevra des vecteurs aléatoires en entrée. La

signature sera disponible dans le registre de sortie du cœur de chiffrement. Cette signature sera observée et comparée à celle attendu uniquement à la fin du test (de l'autotest).

L'architecture d'un cœur de chiffrement (type DES ou AES) utilisé en mode d'autotest est illustrée sur la figure 5-1.

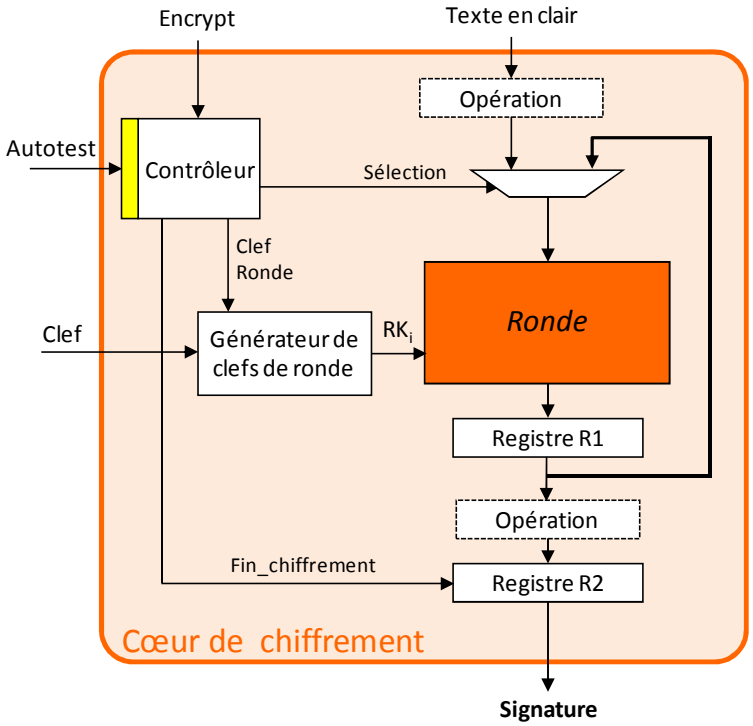


FIGURE 5-1 : MODE AUTOTEST, UTILISANT LA RETROACTION DE RONDE

La rétroaction de ronde, nécessaire au chiffrement, sera utilisée pour acheminer les vecteurs de test de la sortie de ronde à son entrée. Cependant, le nombre limité de rondes (10 pour un chiffrement AES et 16 pour un chiffrement DES) n'est pas suffisant pour fournir un taux de couverture acceptable. Par conséquent pour fournir un taux de couverture élevé, le nombre de rondes exécuter devra être augmenté. Le contrôleur sera modifié de sorte à autoriser un nombre de rondes supérieur à celui normalement requis pour un chiffrement et à charger le registre de sortir qu'à la fin de l'autotest.

Le modèle de fautes considéré est le modèle de collage décrit au chapitre 2. De plus l'hypothèse de panne unique est considérée.

Afin de tester l'ensemble des fautes du cœur de chiffrement, la procédure d'autotest proposée est la suivante :

1) Appliquer un texte en clair T_0 et une clef secrète K_0, exécuter le nombre de cycles de rondes nécessaires au test de celle-ci.
2) Appliquer le texte $\overline{T_0}$ (complément de T_0) et la clef K_0 et exécuter un cycle de rondes. Ceci permet de tester les fautes de collage restantes sur l'entrée de données.
3) Appliquer le texte T_0 et la clef $\overline{K_0}$ (complément de K_0) et exécuter un cycle de ronde. Ceci permet de tester les fautes de collage restantes sur l'entrée de clef.
4) Appliquer un texte T_1 et une clef K_1 afin d'avoir en sortie un texte chiffré qui soit le complément du texte chiffré obtenu en 1, 2 ou 3 et exécuter un cycle de ronde. Ceci permet de tester les fautes de collage restantes sur la sortie de la ronde.

Le coût en surface de la solution proposée est nul, car le cœur de chiffrement sera utilisé comme générateur de vecteurs de test et compacteur de réponses pour son propre test.

5.2 TEST ALEATOIRE DE L'AES

Dans cette partie, l'autotest du cœur de chiffrement AES sera étudié. La difficulté du test peut se résumer en deux principaux points.

Premièrement il faut s'assurer que le circuit puisse être testé avec des données aléatoires. Deuxièmement il faut déterminer la longueur de la séquence de test.

5.2.1 TEST DES DIVERSES OPERATIONS

Dans le but de s'assurer que le cœur de chiffrement AES est testé avec des données aléatoires, la complexité du test des différents modules ou différentes opérations est étudiée. L'architecture de l'AES est décrite en VHDL et synthétisée à l'aide de l'outil de Synopsys Design Compiler. La librairie utilisée pour la synthèse est la librairie CMOS 0,35µm fournie par AMS. Le nombre de cellules et la surface (en μm^2) nécessaires à l'implantation des différents modules sont fournis dans le tableau 5-1.

		AES	
		# cellules	Surface en µm²
Ronde	-SubBytes	10192	803 734,113
	-ShiftRows	0	0
	-MixColumns	301	59 847,461
	-AddRoundKey	423	49 945,051
Contrôleur		63	6 345,601
Générateur de clefs		3159	301 162,8121
Logique		735	153 620,587
TOTAL		**14873**	**1 374 655,625**

TABLEAU 5-1 : COUT EN CELLULES DE L'IMPLANTATION DE L'AES

Premièrement, la complexité du test de l'opération SubBytes, du module ronde, est évaluée. Cette opération occupe à elle seule 60% de la surface totale d'un cœur AES et 90% de la surface du module ronde. Pour rappel, l'opération SubBytes agit indépendamment sur chaque octet. Mathématiquement, cette opération comprend deux étapes. La première consiste à prendre l'inverse de l'octet dans GF(2^8) sachant que l'octet {00} (en hexadécimal) est son propre inverse. La seconde lui applique une transformation affine dans GF(2). Les deux étapes sont couramment combinées en une table de substitution (voir chapitre 1) appelée S-Box. Une S-Box AES possède 8 entrées et 8 sorties.

Plusieurs solutions d'implantation sont possibles pour cette opération :

1. Implanter la S-Box sous forme de mémoire type ROM (Read Only Memory).
2. Implanter la S-Box sous forme de logique combinatoire, en optimisant sa synthèse logique grâce à des outils de CAO.
3. Implanter, sous forme de logique combinatoire, les opérations « Mathématiques » suivantes : la transformation affine, son inverse et le calcul de l'inverse dans GF(2^8) [Rij00].

Dans notre implantation de l'AES, l'opération SubBytes est composée de 16 S-Box implantées sous forme de logique (solution 2). En mode normal, les 16 S-Box fonctionnent en parallèle et de façon indépendante (voir figure 5-2, les S-Box sont notées S).

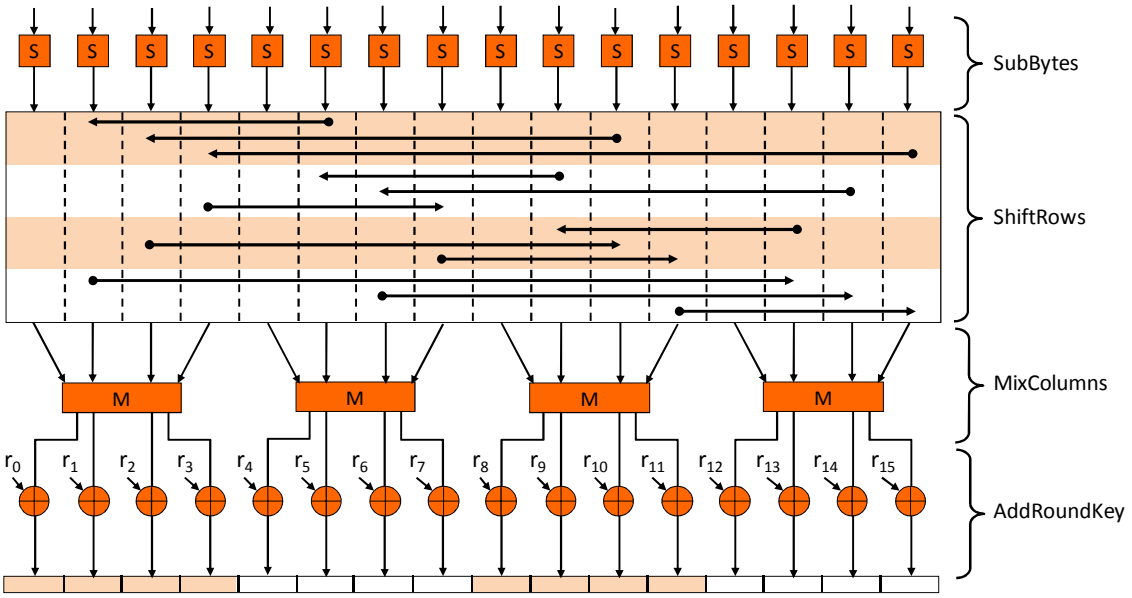


FIGURE 5-2 : RONDE DU COEUR DE CHIFFREMENT AES

Nous nous sommes d'abord intéressés au test d'une S-Box et en particulier au nombre minimal de vecteurs de test à lui appliquer pour tester toutes ces fautes de collage (3184 fautes pour une S-Box). Pour déterminer ce nombre minimal de vecteurs, un dictionnaire de fautes : association faute et vecteur(s) détectant(s), a été établi. Les résultats ont été obtenus à l'aide d'un outil développé au laboratoire. Cet outil est une association entre un simulateur de fautes [Bos08] et une heuristique de recherche de « recouvrement minimal ». Le nombre minimal de vecteurs de test à appliquer pour tester une S-Box est de 203 vecteurs. Suivant l'implantation réalisée le nombre de vecteur minimal varie entre 170 et 256. Appliquer 256 vecteurs correspond à un test exhaustif de la S-Box. Cette séquence est utilisée pour une implantation sous forme de mémoire ROM.

En résumé, pour couvrir les 3184 fautes de collage d'une S-Box, une séquence composée de 203 vecteurs doit être appliquée.

Deuxièmement, la complexité du test de l'opération SubBytes étant traitée, évaluons à présent la complexité du test des trois autres opérations de rondes, en sachant qu'en mode autotest celles-ci seront testées par les données sortant du module SubBytes :

- L'opération *ShiftRows* (voir figure 5-2) est une simple permutation des octets sortant des S-Box. Elle est implantée uniquement avec des fils (il n'y pas de porte). Donc pour tester les fautes de collage chacun des fils doit recevoir une fois un 0 et une fois un 1.

- Les opérations *MixColumns* et *AddRoundKey* sont implantées sous forme d'arbre de portes « ou-exclusif ». Pour rappel, le test d'une porte « ou-exclusif » à 2 entrées requiert 3 vecteurs de test.

Une séquence de test, pour les 5878 fautes de collage de ces trois opérations, a été générée à l'aide de l'outil Tetramax de Synopsys. Cette séquence de test est composée de 15 vecteurs. D'autre part, ces 5878 fautes de collage ont été simulées avec des vecteurs aléatoires (option Fault Sim de Tetramax). Toutes les fautes sont testées quand la séquence de test contient au moins 50 vecteurs aléatoires.

Par conséquent, compte tenu de leurs implantations (fils et portes « ou-exclusif ») et du faible nombre de vecteurs aléatoire de test requis, ces opérations seront testées par les vecteurs sortant des S-Box.

Les 56822 fautes de collage d'une ronde ont été simulées avec 203 vecteurs de test. Chaque vecteur est composé de 16 octets. Les valeurs sur le premier octet sont celles de la séquence utilisée pour le test d'une S-Box. Celles du second octet sont identiques à celles du premier mais décalées d'un cycle, et ainsi de suite… Ceci afin d'éviter que l'opération MixColumns n'ait les mêmes valeurs sur ces 4 entrées.

Le résultat de la simulation est présenté sur la figure 5-3.

```
-----------------------------------------------
#patterns     #faults      test      process
simulated  detect/active  coverage  CPU time
---------  -------------  --------  --------
32          35072  21750    61.72%      0.20
64           6224  15526    72.68%      0.25
96           4356  11170    80.34%      0.28
128          3757   7413    86.95%      0.33
160          3784   3629    93.61%      0.36
192          2981    648    98.86%      0.39
203           648      0   100.00%      0.41
Fault simulation completed: #patterns=203, CPU time=0.41
```

FIGURE 5-3 : RESULTAT DE LA SIMULATION DE FAUTE SUR LA RONDE

En conclusion, une séquence composée des 203 vecteurs de test permet de tester l'ensemble des fautes de collage d'une ronde de chiffrement AES.

5.2.2 CALCUL DE LA LONGUEUR MINIMALE DE LA SEQUENCE DE TEST

En mode d'autotest, les vecteurs de test sont ceux produits par le cœur de chiffrement. Or comme nous l'avons montré au chapitre 3 le cœur de chiffrement est un générateur aléatoire. Par conséquent, la longueur de la séquence aléatoire de test doit être déterminée afin de

contenir les 203 vecteurs requis pour le test d'une ronde de chiffrement AES. L'approche envisagée est pessimiste car nous faisons l'hypothèse que seule une séquence de 203 vecteurs spécifiques permet de tester l'ensemble des fautes. Or il est possible qu'il existe plusieurs séquences de 203 vecteurs permettant de tester l'ensemble des fautes.

Comme précédemment, nous nous intéresserons dans un premier temps au cas d'une S-Box avant de généraliser au cas da la ronde. Pour une S-Box, un ensemble de 256 vecteurs de test est possible, car une S-Box possède 8 entrées ce qui fait 2^8 combinaisons possibles. A l'entrée de chaque S-Box, la séquence générée à l'aide du cœur de chiffrement est considérée comme aléatoire c'est-à-dire que chaque vecteur entrant d'une S-Box à la même probabilité d'apparition et peut être répété. D'où une séquence aléatoire est similaire à une séquence de « tirages avec remise ».

La question qui se pose est : « Quelle est la probabilité d'avoir tiré les vecteurs requis pour le test d'une S-Box après avoir effectué n tirages avec remise ? »

Notons : - n : le nombre de vecteurs aléatoires,

- k : le nombre de vecteurs de test requis pour tester une S-Box ici k = 203,

- m : nombre total de vecteurs possibles en entrée d'une S-Box ici m = 256.

Ce problème est connu en mathématiques sous le nom du problème du « Collectionneur de coupons » [Foa01, Shi07, Bard87]. Ainsi pour déterminer la longueur de la séquence aléatoire de test d'une S-Box, nous utilisons l'équation donnée dans [Shi07] :

$$P[X \leq n] = 1 - \sum_{i=1}^{k} (-1)^{i+1} C_k^i (1- ip)^n,$$

où X représente le nombre de tirages requis pour obtenir les k vecteurs. La probabilité $P[X \leq n]$ est donc la probabilité que le nombre de tirages requis pour obtenir les k vecteurs soit plus petit que n. La probabilité de tirage d'un vecteur est représenté par p. Les vecteurs étant équiprobables, la probabilité p est égale à 1/m. Nous fixons la valeur de $P[X \leq n]$ à 0,99, c'est-à-dire que nous sommes sur à 99% que la séquence contiendra les k vecteurs souhaités.

En remplaçant les indices k et m par leurs valeurs, l'équation s'écrit :

$$0,99 = 1 - \sum_{i=1}^{203} (-1)^{i+1} C_{203}^i \left(1-i\frac{1}{256}\right)^n.$$

Le résultat de cette équation est représenté sous forme graphique. La figure 5-4 illustre ce résultat, l'intersection entre les courbes $y_1 = 0{,}99$ et $y_2 = 1 - \sum_{i=1}^{203} (-1)^{i+1} C_{203}^{i} \left(1-i\frac{1}{256}\right)^{n}$ donne le nombre n de vecteurs aléatoires que doit contenir la séquence de test.

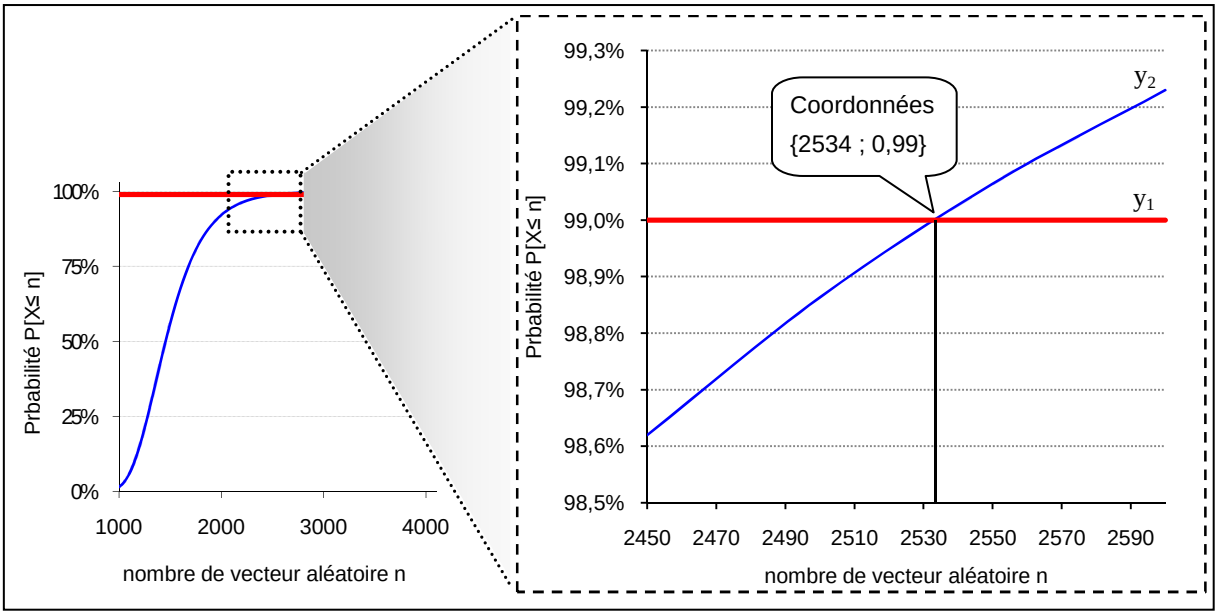


FIGURE 5-4 : LONGUEUR MINIMALE DE LA SEQUENCE ALEATOIRE DE VECTEURS DE TEST

En conclusion, en appliquant 2534 vecteurs de test aléatoires à une S-Box, la probabilité d'avoir appliqué les 203 vecteurs de test est de 99%. L'expérience a été réalisée avec d'autres S-Box synthétisées différemment. Dans tous les cas traités, la valeur de k varie entre 170 et 256 vecteurs, ce qui donne une longueur de séquence de test aléatoires comprise entre 2488 et 2600 vecteurs. La borne supérieure de 2600 vecteurs aléatoires correspond au test exhaustif d'une S-Box (256 vecteurs).

En appliquant 2534 vecteurs sur l'entrée de ronde, on est sur à 99% de tester les fautes de collage des 4 opérations SubBytes, ShiftRows, MixColumns et AddRoundKey.

Concernant la gestion des clefs de rondes, il existe deux solutions suivant l'utilisation envisagée :

1. Les 10 clefs de rondes nécessaires au chiffrement peuvent être pré-calculées et stockées en mémoire. Cette solution est envisagée quand la clef secrète ne doit pas changer.

2. Un générateur de clefs de rondes est implanté. Cette solution est envisagée quand plusieurs clefs secrètes vont être utilisées durant la vie du circuit.

L'architecture générale du générateur de clef AES est présentée sur la figure 5-5.

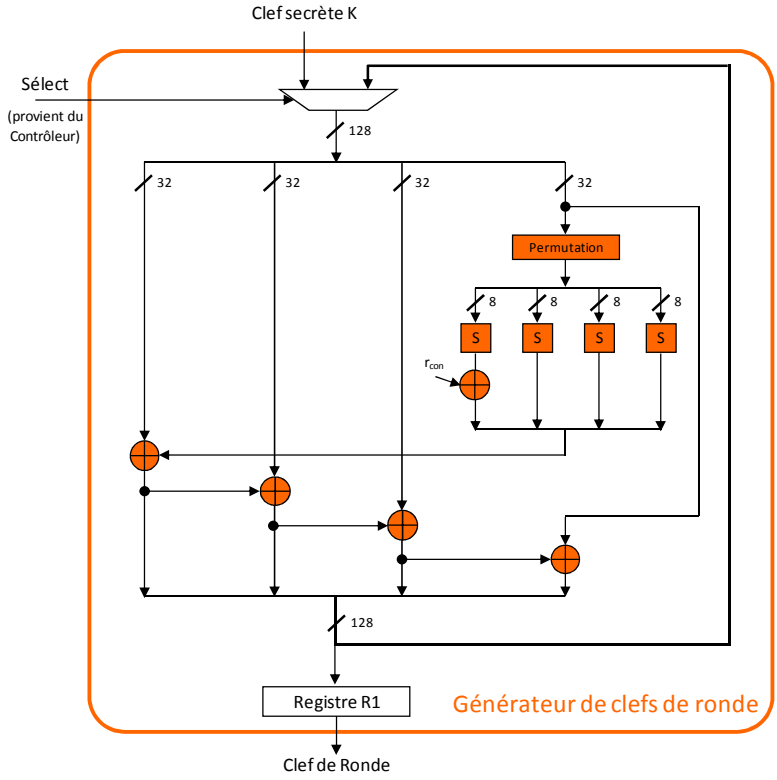


FIGURE 5-5 : ARCHITECTURE DU GENERATEUR DE CLEFS DE L'ALGORITHME AES

Le principe de génération des clefs de rondes est donné au chapitre 1. Pour rappel les clefs de rondes dérivent de la clef secrète. La première clef de ronde est déduite de la clef secrète, et la seconde clef de ronde de la première et ainsi de suite.

Si les clefs de rondes sont stockées dans une mémoire, en appliquant les 2534 vecteurs nécessaires au test des opérations de ronde, les erreurs sur les clefs seront testées. Par contre, si un générateur de clefs est implanté sans aucune modification celui–ci ne sera pas entièrement testé. En effet, dans la structure du générateur de clefs, 4 S-Box (noté S sur la figure 5-5) identiques à celles de l'opération SubBytes sont implantées. Pour rappel, le test d'une S-Box requiert 203 vecteurs déterministes (ou 2534 vecteurs aléatoires). Au regard du premier point de la procédure d'autotest (*1. Appliquer un texte en clair T_0 et une clef secrète K_0, exécuter le nombre de cycles de rondes nécessaires au test de celle-ci*) et sans modification du générateur de clefs, les 10 mêmes clefs de rondes seront toujours utilisées. Par conséquent les S-Box du générateur de clefs recevront seulement 10 vecteurs de test et ne seront pas testées.

Pour pallier ce problème, la solution proposée est d'utiliser la rétroaction déjà présente dans le module de génération de clefs afin de dériver une onzième clef à partir de la dixième et une douzième à partir de la onzième et ainsi de suite… Sachant que les erreurs pouvant se produire sur l'entrée primaire « clef secrète » seront testées par le troisième point de la procédure d'autotest (*3. Appliquer la clef* $\overline{K_0}$ *et le texte en clair* T_0 *et exécuter un cycle de ronde. Ceci permet de tester les fautes de collage restantes sur l'entrée de clef*). Notre solution n'induit aucun surcoût au niveau du générateur de clef, seul le contrôleur est modifié pour gérer de façon adéquate le signal de Sélection.

5.3 SIMULATION DE FAUTES

5.3.1 VALIDATION DE L'AUTOTEST POUR UN CŒUR DE CHIFFREMENT AES

La figure 5-6 illustre la structure d'un cœur de chiffrement AES en mode d'autotest. L'architecture de l'AES a été décrite en VHDL et synthétisée à l'aide de l'outil de Synopsys Design Compiler. Pour la synthèse, la librairie CMOS 0,35µm fournit par AMS est utilisée. Le module de contrôle est modifié afin de gérer deux modes de fonctionnement : le mode normal et le mode d'autotest. La ronde et le générateur de clef ne subissent aucune modification.

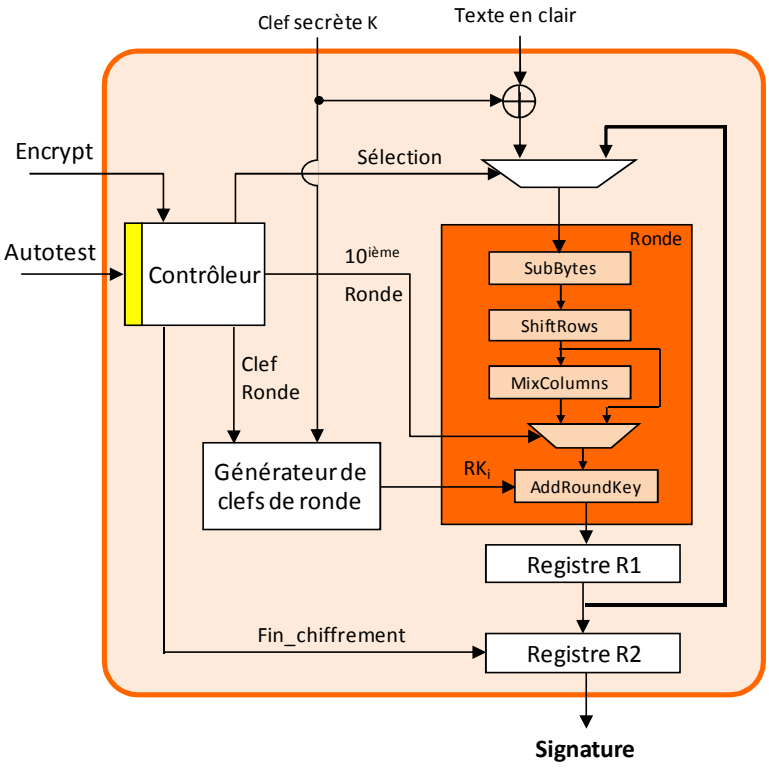


FIGURE 5-6 : STRUCTURE DE L'AES EN MODE AUTOTEST

L'architecture synthétisée est une architecture 128 bits. Le coût en surface de l'implantation est donné dans le tableau 5-2. La colonne de gauche représente le nombre de cellules nécessaires à l'implantation et celle de droite représente la surface requise en µm². Dans ce tableau, le coût de l'implantation de l'AES combinant les deux modes : chiffrement et Autotest est comparé au coût de l'implantation de l'AES en mode chiffrement uniquement. Pour les deux implantations, la synthèse des S-Box est réalisée afin d'avoir des S-Box identiques et testables avec 2534 vecteurs aléatoires.

		AES		AES Autotest	
		# cellules	Surface en µm²	# cellules	Surface en µm²
Ronde	-SubBytes	10192	803 734,113	10192	803 734,113
	-ShiftRows	0	0	0	0
	-MixColumns	301	59 847,461	301	59 847,461
	-AddRoundKey	423	49 945,051	423	49 945,051
Contrôleur		63	6 345,601	69	6 763,201
Générateur de clefs		3159	301 162,812	3160	301 208,187
Logique		735	153 620,587	736	153 684,111
TOTAL		**14873**	**1 374 655,625**	**14881**	**1 375 182,125**

TABLEAU 5-2 : COUT DE L'IMPLENTATION EN NOMBRE DE CELLULES

En conclusion, après synthèse, l'architecture combinant les deux modes a un coût négligeable. En effet, la solution engendre un coût en nombre de cellules de 0,05% et en surface de 0,04%.

Afin de valider le résultat théorique obtenu, des simulations de fautes sont réalisées sur le cœur de chiffrement AES configuré en mode d'autotest. L'outil Tetramax a été utilisé pour effectuer les simulations de fautes. Dans un premier temps, seules les fautes de la ronde sont considérées et dans un deuxième temps toutes les fautes de l'implantation de l'AES seront considérées.

1. Autotest de la Ronde

Dans cette partie, l'autotest de l'AES n'est pas traité en entier. En effet les modules "générateur de clefs de rondes" et "contrôleur" ne sont pas étudiés. Par conséquent la description au niveau porte fourni au simulateur ne contient pas la description de ces deux modules. D'autre part seules les fautes de collage du module ronde sont étudiées.

La séquence de test fourni au simulateur de fautes est créée de sorte que le signal "Fin_chiffrement" force l'écriture du registre de sortie au dernier cycle, pour se rapprocher du fonctionnement en mode d'autotest où seul le dernier résultat est observé.

Cinq couples (texte en clair, clef secrète) différents choisis de façon aléatoire ont été appliqués. Le nombre de fautes simulées sur le chemin de ronde est de 56 052. Les résultats obtenus sont présentés dans le tableau 5-3.

	# cycles de ronde	100	200	500	700	1000	1200	1500	1700	2000	2200	2534
Couple 1	# fautes non testées	8974	4634	1055	464	113	55	11	4	1	0	0
	TC	83,99	91,73	98,12	99,17	99,80	99,90	99,98	99,99	100	100	100
Couple 2	# fautes non testées	8858	4641	1004	458	105	42	24	15	2	0	0
	TC	84,20	91,72	98,21	99,18	99,81	99,93	99,96	99,97	100	100	100
Couple 3	# fautes non testées	8856	4724	973	424	112	63	12	6	6	0	0
	TC	84,20	91,57	98,26	99,24	99,80	99,89	99,98	99,99	99,99	100	100
Couple 4	# fautes non testées	9109	4693	1034	443	140	76	42	27	15	5	0
	TC	83,75	91,63	98,16	99,21	99,75	99,86	99,93	99,95	99,97	99,99	100
Couple 5	# fautes non testées	9079	4603	1026	409	113	48	13	11	2	2	0
	TC	83,80	91,79	98,17	99,27	99,80	99,91	99,98	99,98	100	100	100

TABLEAU 5-3 : RESULTAT DE L'AUTOTEST DE L'AES : PARTIE CHEMIN DE RONDE

Dans ce tableau, le nombre de fautes restant à tester et le taux de couverture obtenu sont donnés pour chacun des 5 couples étudiés. Par exemple, lorsque 100 cycles de ronde sont effectués, le taux de couverture est de l'ordre de 83-84% et le nombre de fautes non testées est de l'ordre de 9000. Maintenant si les 2534 vecteurs aléatoires sont appliqués (ou en d'autres termes les 2534 cycles de rondes sont effectués), alors quelque soit le texte en clair et la clef secrète, la totalité des fautes sont détectées par la séquence de test. Ce résultat est en accord avec le calcul théorique de la longueur minimale de la séquence de test, à savoir que la ronde est entièrement testée après 2534 cycles de ronde.

Le nombre minimal de rondes à exécuter pour tester l'ensemble des fautes a été déterminé. Pour le premier couple, 2056 cycles de ronde sont requis, pour le deuxième, le troisième, le quatrième et le cinquième couple respectivement 2005, 1979, 2203 et 2213 cycles de ronde sont requis.

Les résultats présentés dans le tableau montrent que le taux de couverture est une fonction croissante du nombre de vecteurs.

2. Autotest du cœur de chiffrement complet

Dans un premier temps, considérons uniquement les fautes de collage des trois modules : « ronde », « contrôleur » et « générateur de clefs », les fautes liées aux multiplexeurs, aux registres et à l'opération de « ou-exclusif » ne sont pas considérées. Etant donné les résultats de simulation de fautes obtenus sur la ronde, seuls les cas où 2200 et 2534 cycles de rondes sont effectués seront considérés.

L'architecture de l'AES est légèrement modifiée afin que le signal « Fin_chiffrement » soit commandé depuis l'extérieur et que l'observation puisse être effectuée seulement au dernier cycle. Le nombre total de fautes équivalentes à tester est de 41 037 dont 30 906 fautes du module ronde, 9 742 du module de génération de clef et enfin 389 fautes du module de contrôle. Parmi ces fautes, certaines sont non-testables. Il y en a 142 dans le module de génération de clef dues au signal gérant la rétroaction et 15 dans le module contrôleur liées au signal de sélection du mode. Ces fautes ne sont pas considérées dans les résultats de la simulation de fautes.

Dans le tableau 5-4, les résultats de simulation de fautes sur le cœur de chiffrement AES sont présentés. Ce tableau met en évidence le nombre de fautes non testées dans les différents modules de l'AES.

	# cycles de ronde		2200	2534
Couple 1	# fautes non testées	Ronde	0	0
		Contrôleur	1	1
		Générateur de clefs	190	190
	TC		98,73	98,73
Couple 2	# fautes non testées	Ronde	0	0
		Contrôleur	1	1
		Générateur de clefs	193	191
	TC		98,73	98,73
Couple 3	# fautes non testées	Ronde	0	0
		Contrôleur	1	1
		Générateur de clefs	191	191
	TC		98,73	98,73
Couple 4	# fautes non testées	Ronde	2	0
		Contrôleur	1	1
		Générateur de clefs	195	195
	TC		98,72	98,73
Couple 5	# fautes non testées	Ronde	1	0
		Contrôleur	1	1
		Générateur de clefs	192	192
	TC		98,73	98,73

TABLEAU 5-4 : RESULTAT DE L'AUTOTEST DE L'AES : RONDE, CONTROLEUR ET GENERATEUR DE CLEFS

Même après avoir effectué les 2534 cycles de rondes, des fautes n'ont pas été testées dans les modules : générateur de clefs et contrôleur. Ces fautes seront testées par la suite en appliquant l'intégralité de la procédure de test.

Dans un deuxième temps, considérons l'ensemble des fautes du circuit AES (c'est-à-dire les fautes des 3 modules ainsi que les fautes de la logique qui les entourent). La liste de fautes réduites contient 46 231 fautes.

La procédure de test appliquée à notre cœur AES est la suivante :

1. Appliquer T_0 et K_0, exécuter 2534 cycles de rondes.
2. Appliquer $\overline{T_0}$ et K_0, exécuter un cycle de ronde, résultat C_1.
3. Appliquer T_0 et $\overline{K_0}$, exécuter un cycle de ronde.
4. Appliquer T_1 et $\overline{K_0}$, exécuter un cycle de ronde, résultat $\overline{C_1}$.

En moyenne après avoir effectué le premier point de la procédure, 96,46% des fautes du circuit sont testées. Après le second point 98,27%, après le troisième, 99,44% et enfin après le dernier point, 100% des fautes sont testées.

En conclusion, la procédure de test décrite dans ce chapitre permet de tester l'ensemble des fautes d'un cœur de chiffrement AES. Pour appliquer la procédure, nous avons besoin d'un texte en clair et d'une clef secrète, de leurs compléments, et enfin d'un texte en clair permettant de générer un texte chiffré qui soit le complément d'un précédent texte chiffré. Pour tester l'intégralité des fautes d'un AES, 2537 cycles de rondes devront être exécutés.

5.3.2 VALIDATION DE L'AUTOTEST POUR UN CŒUR DE CHIFFREMENT DES

La même approche d'autotest a été étudiée sur un cœur de chiffrement DES. Comme pour l'AES, le test d'une S-Box est traité en premier. Une S-Box du DES possède 6 entrées et 4 sorties. La longueur de la séquence aléatoire de test est déterminée à l'aide de l'équation du « Collectionneur de coupons », en considérant une probabilité $P[X \leq n]$ égale à 99%.

Si l'approche de test des S-Box DES est une approche exhaustive, le nombre de vecteurs aléatoires requis pour les tester est de 560. La longueur de la séquence dépend de l'implantation des S-Box. Elle peut contenir de 440 à 560 vecteurs aléatoires ce qui équivaut à un nombre de cycle de chiffrement variant de 27 à 35.

Afin de valider ce résultat théorique, des opérations de simulations de fautes ont été réalisées sur le cœur de chiffrement DES. Quatre couples (texte en clair, clef secrète) différents ont été étudiés. Les fautes de collage simulées sont celles du module de ronde et du contrôleur. Les fautes du module de génération de clefs ne sont pas considérées. A la différence de l'AES, l'implantation du module de génération de clef est constituée uniquement de fils. Donc pour tester les fautes de collage de ce module, un 0 puis un 1 doivent être appliqué sur chacun des fils. Par conséquent, pour tester toutes les fautes de collage de ce module, il faut appliquer une première clef secrète K_0 et son complément $\overline{K_0}$. Les résultats de simulation montrent qu'avec environ 400 cycles de rondes (25 cycles de chiffrement) et quelques soit le texte en clair et la clef secrète les fautes de collage du module de ronde et du module de contrôle sont testées.

En résumé, en mode d'autotest au plus 560 cycles de rondes (ou 35 cycles de chiffrement) sont suffisant pour tester la structure du DES.

Par mesure de sécurité le cœur de chiffrement DES est souvent implanté de sorte à combiner les modes de chiffrement et de déchiffrement. Une utilisation connue est le Triple-DES (cf. chapitre 1). Pour rappel, les opérations réalisées durant le chiffrement sont identiques à celles réalisées durant le déchiffrement. La seule différence est l'ordre des clefs

de rondes : elles sont utilisées dans l'ordre inverse pour le déchiffrement. Donc après que le circuit ait été entièrement testé en mode de chiffrement, seules les fautes liées au mode de déchiffrement du générateur de clef ne sont pas testées. Afin de tester l'ensemble des fautes, nous proposons d'appliquer la procédure suivante, sachant que le circuit reste positionné toute la procédure en mode d'autotest :

- Appliquer un texte en clair T_0 et une clef secrète K_0. Exécuter 35 cycles de chiffrement. Noter le résultat T_1. Exécuter un cycle de chiffrement supplémentaire.
- Exécuter un cycle de déchiffrement avec la clef K_0. Le résultat de cette opération doit être T_1.
- Comparer la valeur finale obtenue avec la signature attendue (préalablement pré-calculée), c'est-à-dire qu'il faut s'assurer d'avoir obtenu le résultat T_1.

Cette procédure permet de tester toutes les fautes de collage d'un dispositif implantant à la fois le chiffrement et le déchiffrement DES. En appliquant cette procédure au circuit, le taux de couverture obtenu est bien de 100%. L'intégralité de la procédure d'autotest (les 4 points), décrite en début de chapitre, est appliquée afin de tester les fautes restantes sur les entrées et sorties primaires.

5.3.3 Optimisation de l'autotest d'un cœur de chiffrement AES

L'approche d'autotest présentée est généralisable à tous les cœurs de chiffrement symétriques dont l'opération de ronde est une bijection. Concernant les deux standard DES et AES, l'autotest nécessite environ 25 cycles de chiffrement pour un DES et 254 cycles de chiffrement pour un AES.

Dans cette partie, une optimisation de l'autotest du cœur de chiffrement AES est présentée. Le but de cette optimisation est de réduire le temps de test. Pour rappel, la longueur de la séquence de test est fixée par le test d'une S-Box. La solution envisagée cible une approche de test exhaustive pour les S-Box, ce qui permettrait d'avoir une séquence de test de longueur égale à 256 vecteurs. L'idée est de reboucler les S-Box sur elles-mêmes de façon individuelle (c'est-à-dire que la sortie d'une S-Box devient au cycle suivant son entrée) de sorte que la séquence ainsi générée contienne tous les états possibles (256 états possibles).

Or en rebouclant une S-Box sur elle-même on n'obtient jamais la séquence composée des 256 états. En effet, une S-Box rebouclée peut engendrée cinq séquences différentes suivant la

graine de départ. Les cinq séquences sont listées dans le tableau 5-5. La deuxième colonne donne le nombre d'états composant la séquence et la troisième liste les états (en hexadécimal) de la séquence.

	Longueur	Etats composant la séquence
1	59	63, FB, 0F, 76, 38, 07, C5, A6, 24, 36, 05, 6B, 7F, D2, B5, D5, 03, 7B, 21, FD, 54, 20, B7, A9, D3, 66, 33, C3, 2E, 31, C7, C6, B4, 8D, 5D, 4C, 29, A5, 06, 6F, A8, C2, 25, 3F, 75, 9D, 5E, 58, 6A, 02, 77, F5, E6, 8E, 19, D4, 48, 52, 00
2	81	7C, 10, CA, 74, 92, 4F, 84, 5F, CF, 8A, 7E, F3, D, D7, 0E, AB, 62, AA, AC, 91, 81, 0C, FE, BB, EA, 87, 17, F0, 8C, 64, 43, 1A, A2, 3A, 80, CD, BD, 7A, DA, 57, 5B, 39, 12, C9, DD, C1, 78, BC, 65, 4D, E3, 11, 82, 13, 7D, FF, 16, 47, A0, E0, E1, F8, 41, 83, EC, CE, 8B, 3D, 27, CC, 4B, B3, 6D, 3C, EB, E9, 1E, 72, 40, 09, 01
3	87	F2, 89, A7, 5C, 4A, D6, F6, 42, 2C, 71, A3, A, 67, 85, 97, 88, C4, 1C, 9C, DE, 1D, A4, 49, 3B, E2, 98, 46, 5A, BE, AE, E4, 69, F9, 99, EE, 28, 34, 18, AD, 95, 2A, E5, D9, 35, 96, 90, 60, D0, 70, 51, D1, 3E, B2, 37, 9A, B8, 6C, 50, 53, ED, 55, FC, B0, E7, 94, 22, 93, DC, 86, 44, 1B, AF, 79, B6, 4E, 2F, 15, 59, CB, 1F, C0, BA, F4, BF, 08, 30, 04
4	27	2B, F1, A1, 32, 23, 26, F7, 68, 45, 6E, 9F, DB, B9, 56, B1, C8, E8, 9B, 14, FA, 2D, D8, 61, EF, DF, 9E, 0B
5	2	8F, 73

TABLEAU 5-5 : LES 5 SEQUENCES GENEREES PAR UNE S-BOX AES REBOUCLEE

En résumé, on constate qu'en rebouclant simplement la sortie sur l'entrée, tous les états ne sont pas accessibles à partir de l'état initial. En d'autres termes, au maximum 87 états (sur 256) donc 87 vecteurs peuvent être obtenus.

Par conséquent, il est nécessaire de modifier l'approche envisagée. La nouvelle approche intègre une fonction g dans la rétroaction (voir figure 5-7).

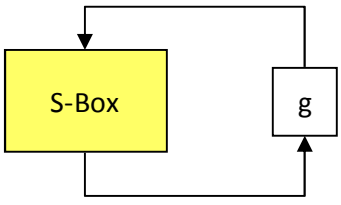


FIGURE 5-7 : FONCTION g DANS LA RETROACTION DE LA S-BOX

Cette fonction g, composée de 5 portes inverseurs, garantit l'exploration de tous les états possibles en entrée d'une S-Box. L'emplacement des portes inverseurs a été déterminé par une approche exhaustive. Deux dispositions sont possibles $118 = (01110110)_2$ et $234 = (11101010)_2$, les 1 représentant l'emplacement des portes inverseurs. Le coût d'implantation de la fonction g est faible.

En résumé, cette solution permet un test exhaustif des S-Box, mais contrairement à la solution précédente où la rétroaction englobe toute la ronde, la solution de rétroaction autour des S-Box nécessite l'implantation d'un compacteur de réponses (type MISR). L'architecture de la ronde est décrite sur la figure 5-8.

Afin de réduire le coût de l'implantation, le registre de sortie est modifié de sorte à être utilisé comme MISR pendant la phase d'autotest. Ainsi les vecteurs de test des S-Box testent également, en se propageant, les fautes de collage des trois opérations : ShiftRows, MixColumns et AddRoundKey.

Les fautes de collage des opérations de rondes sont testées mais la ronde n'est pas entièrement testée car ni les fautes de la rétroaction de ronde, ni les fautes des multiplexeurs n'ont été testées. Par conséquent, en plus des 256 cycles de rétroaction autours des S-Box, 16 cycles de ronde doivent être exécutés. Ce nombre de cycle de ronde a été déterminé par simulation de fautes.

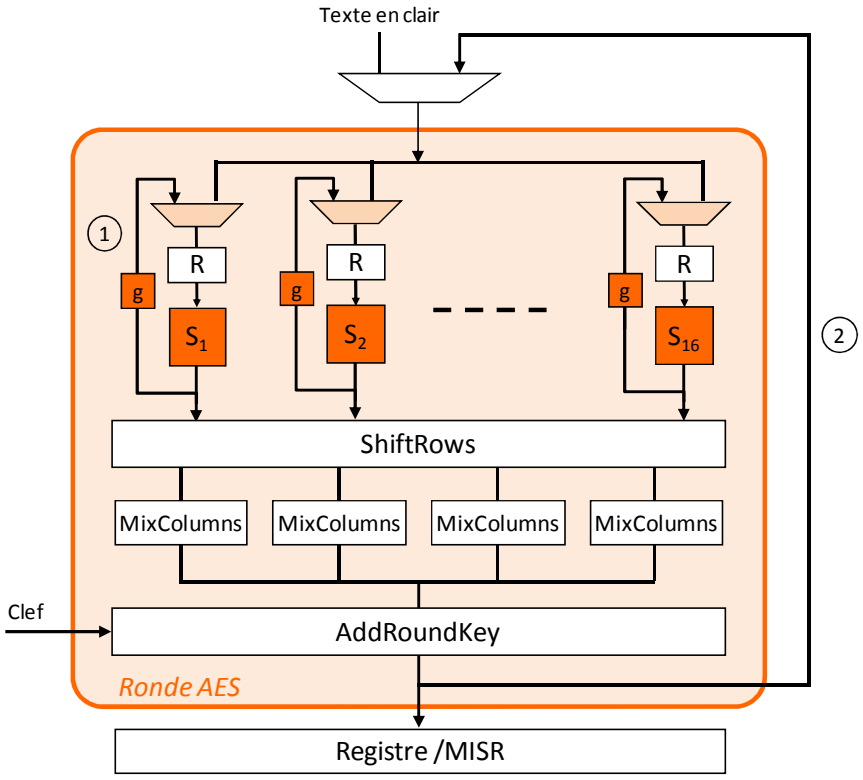


FIGURE 5-8 : OTIMISATION DE L'AUTOTEST DE L'AES

En résumé, la procédure de test est la suivante : la rétroaction noté 1 (sur la figure 5-8) est répété 256 fois. A chaque cycle le résultat est stocké et compacté dans le MISR. Après ces

256 cycles, 16 cycles de ronde (rétroaction noté 2) sont exécutés pour tester les fautes restantes.

Cette solution permet de tester toutes les S-Box avec seulement 256 cycles. Cette solution utilise les propriétés d'une S-Box et peut donc être envisagée pour ces diverses implantations. La surface additionnelle engendrée par l'implantation de la solution est de 3,61%. Cette solution permet de couvrir toutes les fautes de collage de l'AES en seulement 272 cycles d'horloge. De plus cette solution est également applicable aux S-Box du générateur de clefs.

5.4 ARCHITECTURE GENERALE ET COUT

5.4.1 DESCRIPTIF DE L'ARCHITECTURE

Dans cette partie est présentée l'architecture d'un cœur de chiffrement par bloc, implantée de façon itérative et pouvant fonctionner dans les 4 modes suivant : mode normal ou de chiffrement, mode générateur de vecteurs aléatoires de test, mode compaction de réponses et mode d'autotest (sans optimisation).

Quelque soit le mode d'utilisation : « ***générateur de vecteur de test*** », « ***compaction de réponses*** » ou encore « ***autotest*** », la modification majeure et commune à ces trois modes consiste à exécuter un nombre de cycles de ronde supérieur au nombre normalement requis pour une opération de chiffrement. Cette modification affecte le contrôleur. L'autre modification importante est induite par le mode « compaction de réponses », puisqu'il est nécessaire de modifier la rétroaction de ronde afin d'insérer une opération d'« ou-exclusif ». Par conséquent, s'il n'est pas nécessaire d'implanter le mode de « compaction de réponses » alors le coût en surface, nécessaire à l'implantation des modes « générateurs de vecteurs de test » et « autotest », est négligeable.

L'architecture générale d'un cœur de chiffrement pouvant combiner les 4 modes énoncés précédemment, est illustrée sur la figure 5-9.

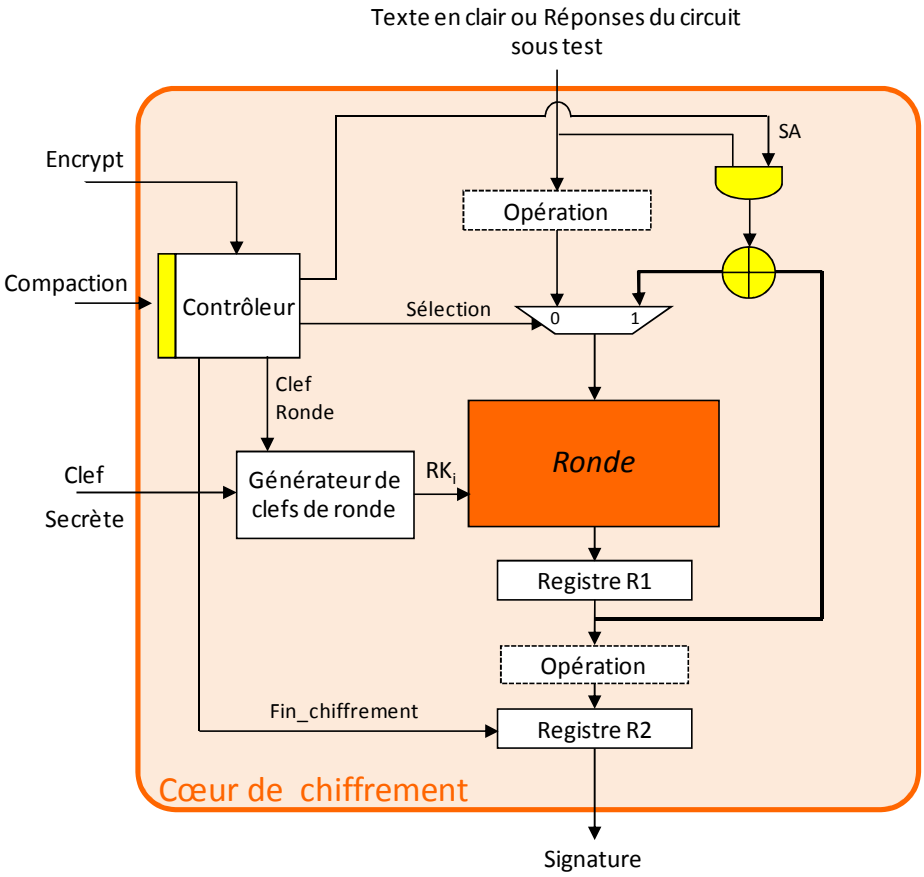


FIGURE 5-9 : IMPLANTATION DE L'AES FONCTIONNANT SUIVANT 4 MODES

Le principe de fonctionnement de ces différents modes est :

- En mode ***normal*** : le signal SA est figé à 0 tandis que le signal Sélection est d'abord mis à 0 avant de commuter à 1 pour effectuer les cycles de rondes nécessaires au chiffrement. Au dernier cycle de ronde le signal « Fin-chiffrement » autorise l'écriture du registre de sortie R2.

- En mode ***générateur de vecteurs de test*** : le signal SA est figé à 0. Le signal « Sélection » est dans un premier temps mis à 0 pour charger la « graine » du générateur, puis le signal commute à 1 pour toute la durée de la génération. Le signal « Fin-chiffrement » autorise l'écriture du registre de sortie R2 à chaque cycle de ronde afin d'avoir un vecteur de test à chaque cycle d'horloge.

- En mode ***compaction de réponses***, les signaux Sélection et SA sont figés à 1 pour qu'une opération d'« ou-exclusif » soit exécutée entre la réponse du circuit sous test et le précédent résultat de ronde. La signature finale, obtenue après compaction de l'ensemble des réponses du circuit sous test, est chargée dans le registre R2.

De plus, le signal « Fin-chiffrement » peut être utilisé pour observer toutes les signatures intermédiaires afin de faciliter le diagnostic.

- En mode d'***autotest***, le signal Sélection est mis à 0 pour charger le texte en clair puis il est mis à 1 pour réaliser les cycles de rondes nécessaires au test. Le signal SA est mis à 0 durant toute la phase de test.

Dans le cas du chiffrement AES, si le module de génération de clefs est implanté, celui-ci est légèrement modifié pour être testé. En effet la dixième clef est utilisée pour dériver les 10 clefs de rondes suivantes. Cette modification ne change pas le fonctionnement en mode normal, seul les modes additionnels : ***génération de vecteurs*** et ***compaction de réponses*** sont affectés. Dans le cas du mode de ***génération des vecteurs*** de test, la séquence générée avec le générateur de clef modifié a les mêmes propriétés aléatoires que celle générée sans la modification du générateur (cf. chapitre 3). Concernant le mode de ***compaction de réponses*** de test, la modification du générateur de clefs n'affecte pas la probabilité de masquage, car l'opération de ronde est toujours bijective.

5.4.2 COUT

L'architecture proposée en mode ***génération de vecteurs*** de test requiert, pour assurer un bon niveau de sécurité, l'utilisation d'une clef secrète différente de celle utilisée en fonctionnement normal. Sans cette sécurité, la tâche d'un attaquant pourrait être facilitée. En effet s'il arrivait à récupérer deux résultats de rondes consécutifs il lui serait alors possible de recalculer la clef secrète de chiffrement. Concernant les deux autres modes ***compacteur de réponses*** de test et ***autotest***, aucune faille n'est engendrée car aucun résultat intermédiaire n'est exploité directement.

Les modifications nécessaires à la mise en œuvre de ces 3 modes n'ont aucun impact sur la durée d'un chiffrement. En effet, un chiffrement sera toujours effectué en 11 cycles (10 de rondes +1 de déchargement). De plus ces modifications sont intégrées dès la première étape du flot de conception.

		AES	AES générateur	AES compacteur	AES autotest	AES 4 modes
Ronde	-SubBytes	803 734,113	.	.	.	.
	-ShiftRows	0	.	.	.	.
	-MixColumns	59 847,461	.	.	.	.
	-AddRoundKey	49 945,051	.	.	.	.
Contrôleur		6 345,601	+ 5,72%	+ 8,72%	+ 6,58%	+ 9,58%
Générateur de clefs		301 162,812	+ 0,015%	+ 0,015%	+ 0,015%	+ 0,015%
Logique		153 620,587	+ 0,04%	+ 17,95%	+ 0,04%	+ 18,36%
TOTAL		**1 374 655,625**	**+ 0,03 %**	**+ 2,05%**	**+ 0,04%**	**+ 2,10%**

TABLEAU 5-6 : COUT EN SURFACE (EN µm²) DE L'INTEGRATION DES DIFFERENTS MODES

Dans le tableau 5-6, le surcoût en surface de l'intégration des différents modes proposés est donné en pourcentage par rapport à la surface (en µm²) d'un AES utilisable uniquement en mode de chiffrement. Toutes les synthèses ont été faites avec la librairie CMOS 0,35µm d'AMS. Le mode qui influence le plus le coût en surface est le mode de ***compaction de réponses*** avec un surcoût total de 2,05%. Les modes de ***génération de vecteur*** et d'***autotest*** quant à eux ont un faible impact sur la surface initiale du cœur AES. Ainsi, le surcoût en surface de 2,10% de la structure composée des 4 modes : chiffrement, génération, compaction et autotest est majoritairement dû au mode de compaction.

Pour comparaison, la structure BILBO (Built-In Logic Block Observer) [Bard87] utilisée en test pour la génération et la compaction de réponses de test, engendrerait pour l'implantation d'une structure équivalente à l'AES (128 bits) un surcoût en surface de 7,64% (105 029,406µm²). De plus, le surcoût de 7,64% ne tient pas compte des contraintes en surface liées à l'intégration du registre BILBO dans le circuit (multiplexeurs, connectique...).

5.5 CONCLUSION

Dans ce chapitre, nous avons proposé une solution de test intégré à faible coût pour les cœurs de chiffrement symétrique. Cette solution utilise les propriétés de génération et de compaction de données inhérentes à ces algorithmes de chiffrement pour leur propre test.

Après avoir présenté la procédure de test, nous avons établi, pour un chiffrement AES et DES, la longueur de la séquence aléatoire de test pour tester l'intégralité des fautes du circuit. Pour un DES la longueur de la séquence est de 400 (25 cycles de chiffrement) et pour un AES de 2537 (254 cycles de chiffrement). Ces résultats ont été validés par plusieurs simulations de

fautes. Concernant le chiffrement AES une optimisation de la solution d'autotest a été présentée. Celle-ci permet de réduire la longueur de la séquence à 272 cycles (28 cycles de chiffrement).

De plus, la structure générale d'un cœur de chiffrement symétrique combinant les 4 modes de fonctionnement : ***chiffrement***, ***générateur de vecteurs aléatoires***, ***compacteurs de réponses*** de test et ***autotest***, a été présentée. Le coût en surface de l'intégration des différents modes a été évalué et comparé à l'implantation d'une structure classique de test.

En conclusion, utiliser le cœur de chiffrement comme ressource de test intégré à la fois pour le test des autres cœurs et pour lui-même, est une solution peu coûteuse en surface avec un surcoût de 2,10% contre 7,64% pour une structure BILBO. Il est à noter que le coût en surface du contrôleur de test intégré n'a pas été considéré ni pour le cœur de chiffrement ni pour la structure BILBO.

Chapitre 6 : Test en ligne

Chapitre 6 : TEST EN LIGNE

Dans ce chapitre nous proposons une solution de test en ligne destinée aux implantations matérielles d'un cœur de chiffrement AES. L'approche développée cible un test en ligne continu, c'est-à-dire un test effectué en parallèle de l'application. Cette solution permet de s'assurer de la fiabilité du cœur de chiffrement en vérifiant la validité des données traitées en cours de fonctionnement. Cette solution n'est pas orientée vers le test de production quoiqu'éventuellement elle puisse être utilisée dans ce cadre.

Dans une première partie, plusieurs solutions de test en ligne existantes pour des implantations de l'AES sont présentées. Ces solutions utilisent les techniques de détection d'erreurs basées sur la redondance d'information ou de matériel. Dans une deuxième partie, la solution de test en ligne proposée est décrite. Cette solution de test en ligne permet de détecter les fautes permanentes simples et multiples dans un AES. Dans la dernière partie, le coût de la solution proposée est évalué.

6.1 ETAT DE L'ART

Lors du fonctionnement de la puce, des erreurs transitoires ou permanentes peuvent apparaître. Elles sont dues à la dégradation physique du matériel ou à des interactions environnement-puce ou homme-puce, ou encore, dans le contexte des circuits sécurisés, à des attaques malveillantes... Comme leurs noms l'indiquent les fautes permanentes endommagent de façon irréversible le composant tandis que les fautes transitoires endommagent rarement le composant affecté même si elles peuvent induire un comportement erroné généralement de courte durée.

Diverses solutions de test en ligne existent pour ces fautes : celles détectant les fautes permanentes, celles détectant les fautes transitoires et celle détectant les deux types de fautes. Ces solutions sont divers compromis entre le taux de couverture, le coût en surface et l'impact sur les performances.

Quelque soit le type de fautes permanentes ou transitoires, les techniques de détection d'erreurs les plus couramment utilisées sont les codes détecteurs d'erreurs et le doublement-comparaison. Les codes détecteurs d'erreurs sont basés sur une redondance d'information : soit des bits de contrôles sont ajoutés aux données, soit une nouvelle représentation utilisant plus de bits est utilisée. Le choix du code détecteur dépend du type d'erreurs : erreurs simples, erreurs unidirectionnelles (commutation 0 vers 1 ou commutation 1 vers 0) ou erreurs multiples. Le contrôle de parité est la solution de codage la plus connue pour la détection d'erreurs simples. Les erreurs unidirectionnelles sont détectées par les codes de Berger [Berg61] et les codes « m parmi n ». Quant aux erreurs multiples, elles sont détectées, par exemple, avec le code double-rail.

Le doublement-comparaison est une technique basée sur la redondance matérielle. Cette solution consiste à dupliquer le module dont on veut détecter les erreurs, puis à comparer les sorties de ces deux modules. Cette solution est fondée sur l'hypothèse que le processus d'activation ou de génération des erreurs agit de façon indépendante sur les deux modules. En effet, si l'erreur ou les erreurs affectent de la même façon les deux modules, le résultat de la comparaison est identique à celui attendu alors que les deux modules sont fautifs.

Dans cette partie, nous exposons des solutions de détection d'erreurs proposées dans la littérature pour le nouveau standard de chiffrement symétrique AES. Ces solutions sont regroupés en trois catégories suivant quelles permettent de détecter les fautes permanentes, les fautes transitoires ou bien les deux types de fautes.

6.1.1 DETECTION DE FAUTES PERMANENTES ET TRANSITOIRES

Les solutions présentées sont basées sur les deux principales techniques qui sont les codes détecteurs d'erreurs et la redondance matérielle.

1. Code détecteur d'erreur

Une grande majorité des solutions trouvées dans la littérature utilise le contrôle de parité pour la détection d'erreurs. Le principe général du contrôle de parité consiste à ajouter un bit supplémentaire (appelé bit de parité) aux bits de données afin de vérifier leurs conformités. Le bit ajouté est un 0 si le nombre de bits de données valant 1 est pair sinon c'est un 1. Ce bit de parité est comparé avec celui préalablement calculé, et une erreur est détectée s'ils diffèrent. La difficulté dans l'implantation de ces solutions est le calcul prédictif de la parité. De plus, il

est à noter que le contrôle de parité ne permet de détecter qu'un nombre impair de bits en erreur dans un mot. Par ailleurs cette solution ne permet pas de corriger les erreurs détectées.

La mise en œuvre de ces solutions, pour des implantations de cœur AES, nécessite de déterminer la parité attendue en sortie de chacune des opérations de ronde : SubBytes, ShiftRows, MixColumns et AddRoundKey. Concernant l'opération ShiftRows la parité attendue en sortie (sur les 128 bits) est égale à la parité entrante. De même pour l'opération MixColumns il a été montré dans [Wu04], que cette opération ne modifie pas la parité si l'on considère les données 32 bits par 32 bits (c'est-à-dire colonne par colonne pour une représentation matricielle). Pour l'opération AddRoundKey la parité attendue est déterminée en ajoutant la parité des données entrantes à la parité de la clef de ronde correspondante. Enfin, concernant l'opération SubBytes, l'approche est plus délicate car cette opération est une opération non linéaire, c'est-à-dire que la parité entrante n'est pas forcément préservée en sortie de cette opération. Diverses solutions ont été proposées [Ber03, Wu04, DiN07], afin de déterminer la parité attendue en sortie d'une S-Box.

- Première solution

La solution proposée par Bertoni et al. dans [Ber03] décrit, pour chaque opération de ronde, la procédure permettant de prédire la parité attendue. Seule la mise en œuvre de cette solution pour l'opération SubBytes est détaillée. Cette solution consiste à traiter les données non pas par bloc de 128 bits mais sous forme d'octets. A chaque octet de données est associé un bit de parité. Le mot constitué de l'octet et de son bit de parité est obligatoirement un mot pair. Ainsi, pour détecter la présence d'une faute il suffit d'observer la parité des mots obtenus en sortie de l'opération SubBytes, si ils sont pair il n'y a pas de fautes autrement le circuit est fautif.

Cette solution nécessite de modifier l'implantation de la S-Box. Au lieu d'une mémoire 256×8 bits, la taille de la mémoire requise est de 512×9, car les *mots* sont codés sur 9 bits. Les données stockées en mémoire seront pour moitié les résultats attendus, c'est-à-dire l'octet résultant de la substitution et son bit de parité pré-calculé (8+1), et l'autre moitié contiendra des données erronées permettant de détecter la présence d'une erreur (voir figure 6-1). Les données erronées sont des *mots* (9 bits) de parité impaire, par exemple 0000000 1.

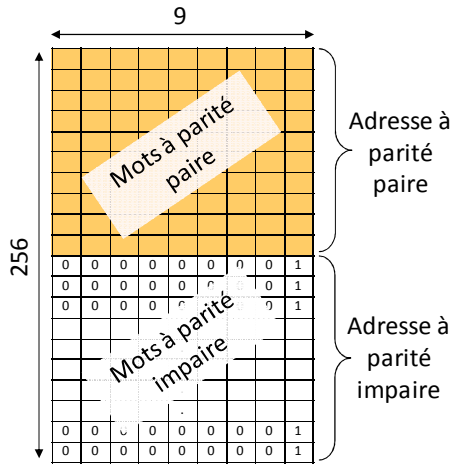


FIGURE 6-1 : DETECTION D'ERREURS DANS LES S-BOX (SOLUTION NUMERO 1)

Si l'erreur se produit sur une des entrées de la S-Box alors l'adresse (qui n'est autre que le mot entrant) aura une parité impaire. Ainsi, le mot en sortie de la S-Box aura également une parité impaire. Par conséquent l'erreur sera détectée. Si l'erreur affecte le comportement de la mémoire, c'est-à-dire si l'adresse est correcte mais que le mot correspondant est erroné, alors l'erreur sera détectée car la parité du mot sortant sera impaire au lieu d'être paire.

En conclusion, cette solution permet de couvrir toutes les fautes de collage simples et multiples engendrant un nombre d'erreurs impair et 99,997% des fautes multiples engendrant un nombre d'erreurs pair. Par contre cette solution est très coûteuse en surface, la taille de la mémoire étant doublée.

- Deuxième solution

La solution proposée dans [Wu04] consiste à ajouter un bit de parité en sortie de la S-Box. La valeur de ce bit de parité est le résultat de l'opération de « ou-exclusif » entre la parité des 8 bits d'entrées et la parité des 8 bits de sorties correspondants. Concrètement, elle est déterminée à partir de la table de vérité des S-Box (originelles). Par conséquent, la valeur de ce bit de parité est indépendante du fonctionnement de la S-Box.

La figure 6-2 représente la structure générale de la ronde AES intégrant le contrôle de parité.

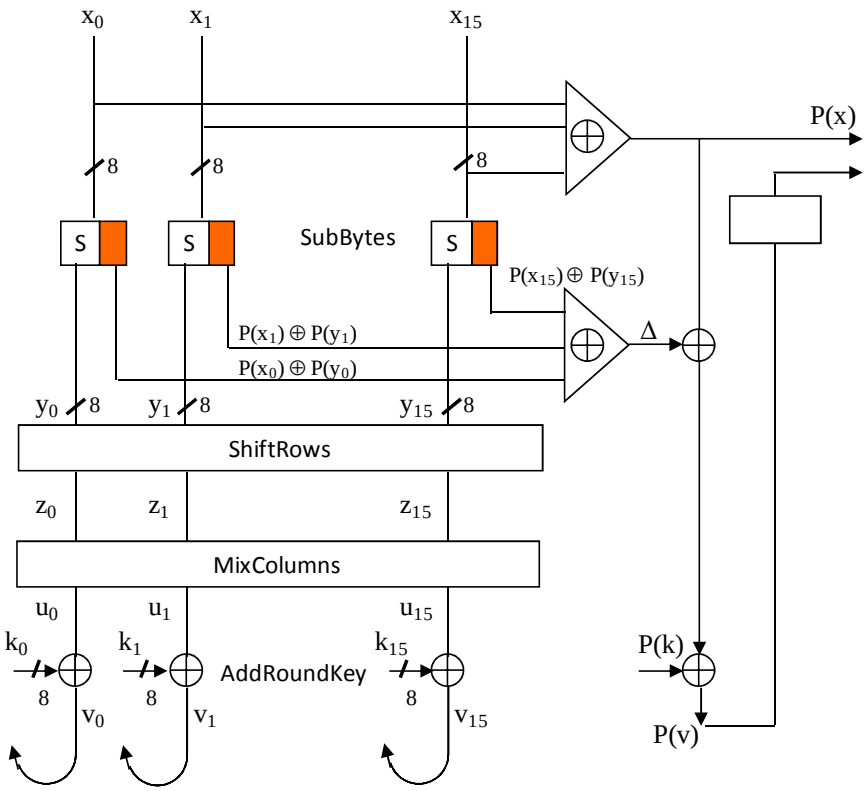


FIGURE 6-2 : DETECTION D'ERREURS SOLUTION NUMERO 2

A la fin de chaque ronde, la parité de sortie notée P(v) est stockée dans un registre afin d'être comparée avec la nouvelle parité entrante notée P(x), car la rétroaction de ronde suppose que la parité P(x) entrante est la parité de sortie de la ronde précédente. En résumé, P(x) représente la parité réellement obtenue en sortie d'une ronde et P(v) représente la parité qui doit être obtenue sans l'erreur.

En conclusion, cette solution permet de détecter 98,7% des fautes simples de collage, c'est-à-dire des erreurs affectant un seul bit. La présence d'une erreur est détectée après un cycle de ronde et le coût en surface, pour implanter cette solution, est de 35%.

- Troisième solution

La solution proposée dans [DiN07] consiste à évaluer à la fois la parité des données entrantes dans la S-Box et des données sortantes. Cette solution est illustrée sur la figure 6-3.

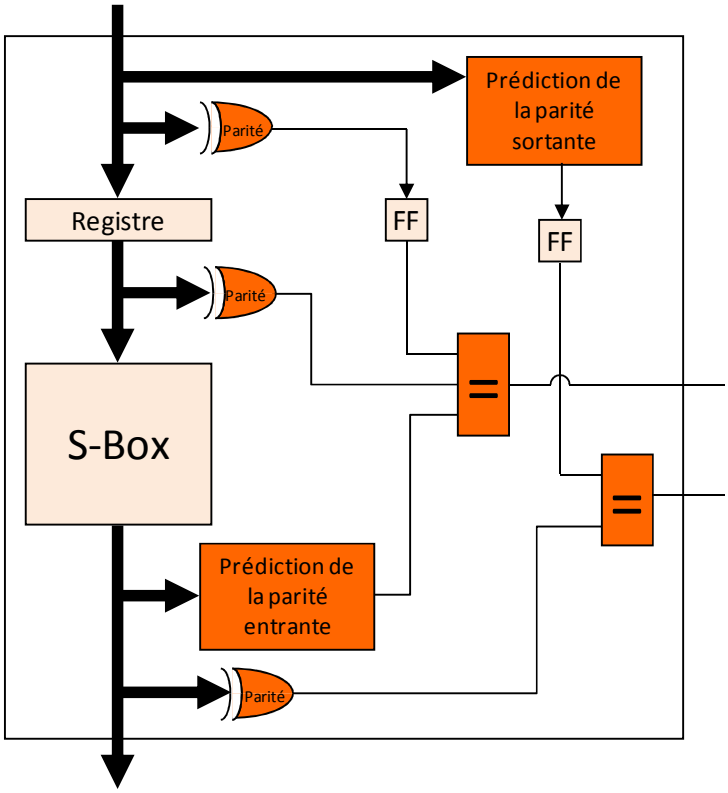


FIGURE 6-3 : DETECTION D'ERREURS SOLUTION NUMERO 3

La procédure permettant de prédire la parité des données entrantes ou sortantes est basée sur la table de vérité de la S-Box et sur la parité de chacune de ces valeurs. En effet, pour prédire la parité entrante (resp. sortante), on détermine à partir du résultat en sortie (resp. en entrée) de la S-Box et de la table de vérité, la parité de l'entrée (resp. de la sortie) correspondante. La présence d'une erreur sera observée au moment de la comparaison.

En conclusion, cette solution permet de tester 99,55% des fautes simples de collage. La présence d'une erreur est détectée à la fin de l'opération SubBytes et le coût en surface, pour implanter cette solution, est de 38,33%.

- Conclusion

Les techniques de détection d'erreurs utilisant le contrôle de parité permettent de détecter toutes les fautes simples avec un coût en surface limité de l'ordre de 30-35% (hors mis la première solution). Par contre concernant la détection de fautes multiples, ces solutions ne sont pas adaptées au cas où le nombre de fautes est pair. De plus le coût en surface de la solution est généralement plus important pour détecter les fautes multiples.

2. Redondance matérielle

La redondance matérielle fait référence à l'utilisation de plusieurs composants matériels pour effectuer des tâches identiques. La présence d'erreurs est détectée par la comparaison des résultats obtenus. Ces techniques sont souvent très coûteuses en surface pour leur implantation. Contrairement aux solutions utilisant le contrôle de parité présentées précédemment, ces solutions couvrent l'intégralité des fautes simples et multiples.

Une solution de redondance matérielle couramment utilisée est la solution de doublement-comparaison. Cette solution n'est pas envisageable dans le cas d'un cœur de chiffrement car elle est trop coûteuse en surface.

La solution présentée par Karri et al. dans [Kar02], utilise la relation qui existe dans l'AES entre le chiffrement et le déchiffrement pour réduire le coût en surface de la structure nécessaire à la détection d'erreurs. Par conséquent, cette solution peut être utilisée quand le chiffrement et le déchiffrement sont déjà implantés.

La solution proposée consiste à réaliser un chiffrement suivi d'un déchiffrement et de s'assurer que les données obtenues en sortie sont bien celles du départ. Les propriétés d'inversion de l'algorithme AES ont permis aux auteurs de proposer trois approches de détection d'erreurs différentes :

- Première approche « *niveau algorithme* » : un chiffrement entier est réalisé, suivi d'un déchiffrement.

- Deuxième approche « *niveau ronde* » : une ronde de chiffrement est exécutée, suivi d'une ronde de déchiffrement.

- Troisième approche « *niveau opération de ronde* » : une opération de ronde de chiffrement (SubBytes, ShiftRows, MixColumns ou AddRoundKey) est effectuée, suivi d'une opération de ronde de déchiffrement ($SubBytes^{-1}$, $ShiftRows^{-1}$, $MixColumns^{-1}$ ou $AddRoundKey^{-1}$).

Quelle que soit l'approche utilisée, les auteurs ont montré que l'intégralité des fautes est détectée, les erreurs dans le module de génération de clefs de rondes n'étant pas considérées. Le tableau 6-1 présente les avantages et inconvénients des différentes approches.

	niveau algorithme	*niveau ronde*	*niveau opération de ronde*
Durée d'un chiffrement	2 × Temps d'un chiffrement	Temps d'un chiffrement + 1cycle de ronde	Temps d'un chiffrement + Temps d'exécution d'une opération
Latence à la détection	2 × Temps d'un chiffrement	2 × Temps d'une Ronde	2 × Temps d'exécution d'une opération
Coût en surface	+ +	+	+ + +

TABLEAU 6-1 : COMPARAISON DES APPROCHES DE DETECTION

L'approche de détection au *niveau ronde* est le meilleur compromis temps, latence et surface. Par contre, cette approche ne permet pas de localiser avec précision l'endroit où l'erreur s'est produite, contrairement à l'approche au *niveau opération de ronde,* qui permet de localiser l'opération de ronde où l'erreur s'est produite.

En conclusion, les solutions utilisant les propriétés d'inversion liées au chiffrement et déchiffrement permettent de détecter toutes les fautes de collage mais modifie obligatoirement la durée d'un chiffrement.

6.1.2 DETECTION DE FAUTES TRANSITOIRES

La solution proposée par Maistri et al. dans [Mai07] permet de tester les fautes transitoires. Cette solution est basée sur la redondance temporelle et plus particulièrement cette solution s'appuie sur la technique de DDR (Double Data Rate). Les techniques de DDR permettent de traiter les données sur les fronts montant et descendant d'horloge, par conséquent cette technique a pour effet de doubler le débit.

Le but de cette solution de détection à base de redondance temporelle DDR est de doubler l'exécution d'une ronde sans modifier la durée du chiffrement. Ainsi une ronde qui est normalement exécuté en 6 cycles d'horloge (un domaine d'horloge : front montant) sera exécuter en 3 cycles d'horloges avec la solution DDR (deux domaines d'horloge : front montant plus front descendant) et ainsi pour un nombre de cycles d'horloge identique une ronde sera exécutée deux fois.

La procédure de détection consiste à dupliquer les données entrantes dans la ronde avant d'exécuter les opérations de rondes. En effet, les opérations de rondes sont d'abord exécutées

sur la première copie et le résultat obtenu est mémorisé, ensuite les mêmes opérations de rondes sont exécutées sur la deuxième copie puis le résultat obtenu est comparé avec le précédent. Si ces résultats sont différents alors une erreur a été détectée autrement le circuit est sain.

Cette solution est efficace uniquement sur les fautes transitoires de courte durée (de 1cycle à 3 cycles), les fautes permanentes étant indétectable avec une telle solution.

En conclusion, cette solution permet de détecter 100% des fautes transitoires affectant la ronde de chiffrement. Au vue de la structure DDR, la présence de l'erreur est détectée à la fin d'une opération de ronde classique (sur 6 cycles). Le coût en surface, pour implanter cette solution, est de 36%. De plus dans le cas des circuits sécurisé, cette solution de détection d'erreurs permet de contrer les attaques par injection de fautes (voir chapitre 1).

Dans la partie suivante, nous proposons une solution de détection d'erreurs, basée sur de la redondance matérielle. Cette solution a été développée de sorte à ne pas modifier le temps d'exécution d'un chiffrement.

6.2 APPROCHE PROPOSEE : DETECTION DE FAUTES PERMANENTES

6.2.1 ARCHITECTURE PROPOSEE

L'architecture proposée a été conçue pour des algorithmes AES dont l'implantation de l'opération SubBytes est composée de 16 S-Box et dont les clefs de rondes utilisées pour le chiffrement ou le déchiffrement sont pré-calculées et stockées en mémoire. Ces implantations favorisent le temps d'exécution d'une procédure de chiffrement ou de déchiffrement au coût en surface de son implantation. En effet, si les 16 S-Box sont implantées alors l'opération SubBytes est effectuée au sein du même cycle d'horloge.

La mise en œuvre de l'architecture de détection des fautes permanentes nécessite de modifier la structure de la ronde. La partie haute de la figure 6-4 représente la structure d'une ronde AES sans modification et la partie basse illustre cette même ronde mais avec la modification. Les notations RS et MA représentent respectivement l'association Registre S-Box et l'association MixColumns AddRoundKey. Il est à noté que pour simplifier la figure, seulement 8 blocs RS et 2 blocs MA sont représentés au lieu de respectivement 16 et 4.

Dans une ronde AES sans modification les opérations sont parcourues dans l'ordre suivant SubBytes, ShiftRows, MixColumns et AddRoundKey. La modification consiste à permuter l'opération ShiftRows avec l'opération Registre-SubBytes.

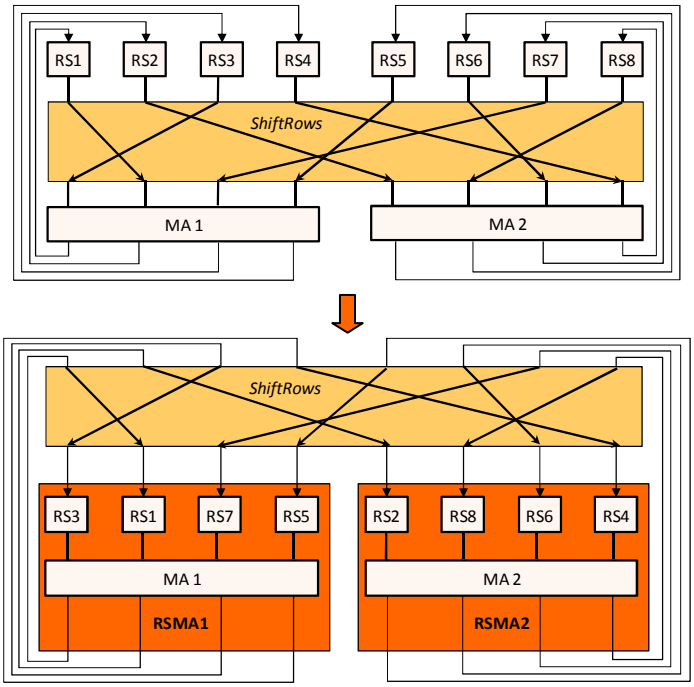


FIGURE 6-4 : MODIFICATION DE L'ORDRE DES OPERATIONS DE RONDE

Cette modification ne change pas le résultat ni en sortie de ronde ni en sortie du chiffrement. Puisque l'instant de permutation des octets (avant ou après la phase de substitution) n'a aucune incidence, les données entrants dans le bloc MA restent identiques.

Après la permutation de ces deux opérations, la partie centrale de l'algorithme peut être divisée en 4 blocs identiques qui opèrent chacun sur 32 bits de données. Ces blocs sont appelés RSMA, ces lettres représentant leurs compositions : 32 bits de **R**egistre, 4 **S**-Box, 1 **M**ixColumns et 32 « ou-exclusif » pour l'opération **A**ddRoundKey.

La solution proposée consiste à ajouter un bloc RSMA supplémentaire dans la ronde. Par conséquent, la ronde contiendra 5 blocs RSMA au lieu de 4. Cette redondance autorise le même fonctionnement de deux blocs simultanément. A chaque cycle d'horloge deux blocs sont alimentés avec la même entrée et les sorties obtenues à la fin du processus sont comparées. Cette comparaison permet de détecter les fautes éventuelles affectant un des deux blocs.

L'architecture de la ronde intégrant les 5 blocs RSMA (sans l'opération ShiftRows) est présentée sur la figure 6-5.

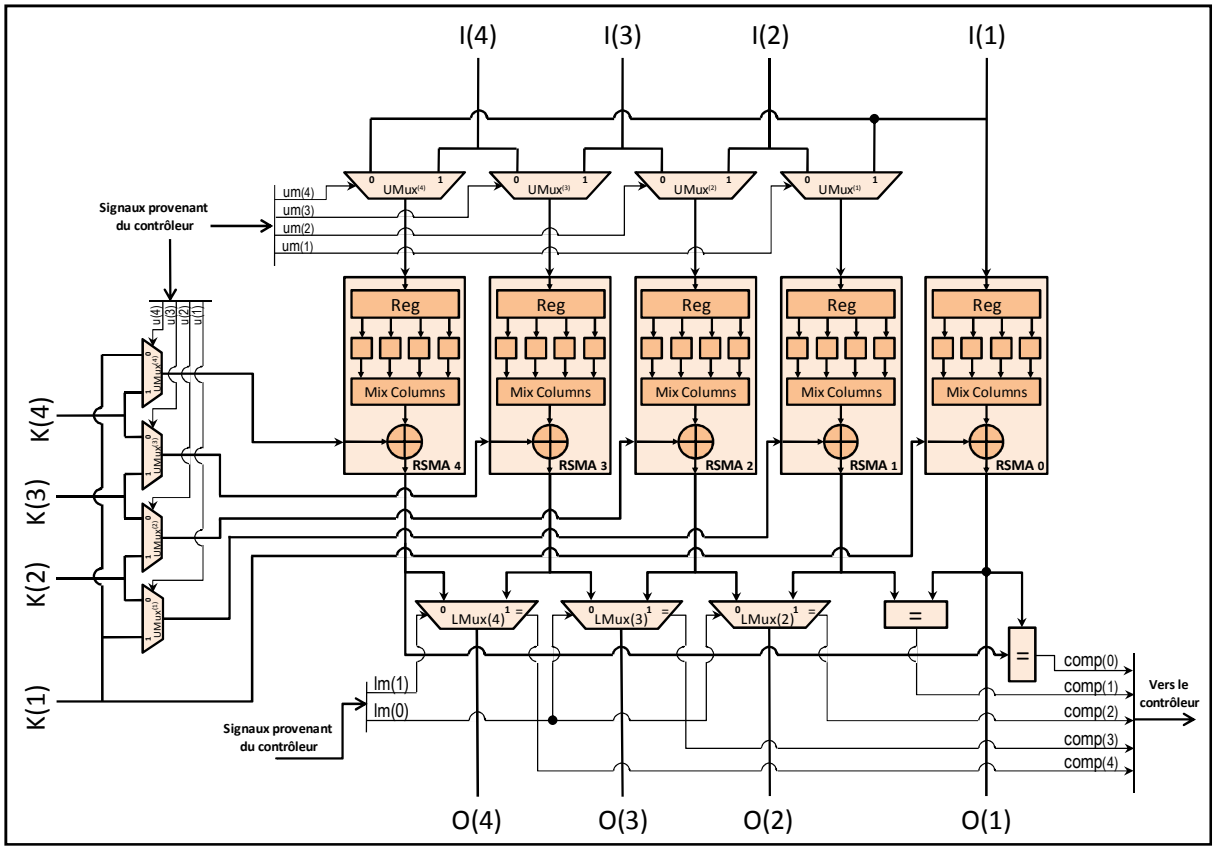


FIGURE 6-5 : TEST EN LIGNE DE LA RONDE

Sur la figure, les trois LMux sont trois multiplexeurs auxquels sont ajoutés une sortie additionnelle correspondant au résultat de la comparaison des 2 signaux entrants. En d'autre terme, il s'agit d'une association multiplexeur-comparateur.

Considérons, par exemple, que les blocs RSMA4 et RSMA3 soient ceux comparés. A l'aide du signal Um, on force les entrées des différents modules de sorte que les entrées de ces deux blocs soient identiques (c'est-à-dire Um(4) = 1 et Um(3) = 0 ce qui implique Um(2) = Um(1) = 0). A la fin du cycle de ronde, la valeur du signal comp(4) est observée, celle-ci fournit l'information sur la présence d'erreurs.

Le tableau 6-2 détaille la gestion des signaux de contrôles entrants et sortants suivant les différents blocs comparés. Avec l'architecture proposée, il y a au total 5 configurations différentes de paires de blocs RSMA à comparer.

Modules comparés	Um	Lm	Sortie à observée
RSMA4 - RSMA3	1000	11	comp(4)
RSMA3 - RSMA2	1100	01	comp(3)
RSMA2 - RSMA1	1110	00	comp(2)
RSMA1 - RSMA0	1111	00	comp(1)
RSMA0 - RSMA4	0000	11	comp(0)

TABLEAU 6-2 : CONTROLE DU TEST EN LIGNE

La solution de détection proposée repose sur une bonne gestion des diverses configurations. Un chiffrement AES est exécuté en 10 cycles d'horloge et il y a 5 configurations différentes. Par conséquent, il est possible d'utiliser les 5 configurations deux fois au cours d'un chiffrement. Grâce à ces 5 configurations, chaque bloc RSMA est comparé à deux reprises, une fois avec le bloc de gauche et une fois avec le bloc de droite. Ainsi, si les 5 configurations sont exécutées deux fois, alors chaque bloc RSMA est comparé 4 fois durant un chiffrement. Un compteur peut être utilisé pour s'assurer de la gestion des diverses configurations.

La sous-partie suivante est consacrée à l'étude de la structure de détection, sa capacité à détecter les erreurs permanentes, sa latence, son coût en surface.

6.2.2 RESULTATS ET COUT DE LA STRUCTURE

L'efficacité de la solution proposée, pour la détection de fautes permanentes, diffère des solutions classiques de doublement-comparaison. En effet, une architecture classique permet de détecter toutes les erreurs simples et multiples qui induisent un comportement erroné sur les sorties. En d'autres termes, dès que le vecteur entrant sensibilise la faute, celle-ci sera détectée. Dans tous les cas, l'architecture doublement-comparaison ne délivrera jamais de réponse erronée sans avoir détectée la présence d'erreurs.

La solution proposée est capable de détecter toutes les fautes simples ou multiples, qui engendrent une réponse fausse en sortie d'un bloc RSMA mais seulement quand ce bloc défectueux est comparé à un autre bloc. Contrairement à la solution de doublement-comparaison où les blocs sont comparés à chaque cycle, avec la solution proposée, la reconfiguration dynamique des blocs implique que chaque bloc n'est comparé que 2 fois sur 5 cycles.

La question qui se pose alors est : « Qu'elle est la probabilité de ne pas détecter une réponse fausse en sortie de l'AES ? »

Cette probabilité peut être analysée en calculant la probabilité (notée $P_{err}(f)$) de ne pas détecter une erreur en sortie de l'AES alors qu'une faute (notée f) affecte le circuit. L'hypothèse de faute permanente unique sera considérée pour calculer la probabilité $P_{err}(f)$. De plus seules les fautes de collage seront traitées.

Au vu de l'architecture proposée, $P_{err}(f)$ est la probabilité que la faute f soit activée (c'est-à-dire sensibilisée et propagée de sorte à engendrer une réponse fausse en sortie) pendant au moins un des 6 cycles d'horloge où le bloc RSMA défectueux n'est pas comparé avec un autre, et qu'elle ne soit pas activé pendant un des 4 cycles d'horloge où le bloc RSMA défectueux est comparé.

La probabilité d'activation d'une faute f dans un bloc RSMA (notée P_f) est la probabilité de sensibiliser une erreur à l'aide d'un vecteur aléatoire et de propager la faute en sortie. Dans l'hypothèse que plusieurs entrées fonctionnelles distinctes sont utilisées alors on peut considérer que le circuit sera alimenté par une source aléatoire. De plus, les propriétés intrinsèques de l'AES impliquent que la séquence, composée des résultats de rondes d'un chiffrement, peut être considéré comme aléatoire. En résumé, les divers blocs RSMA sont alimentés par des données aléatoires. Par conséquent, la probabilité P_f est égale au rapport nombre de vecteurs entrants testant f sur le nombre total de vecteurs entrants. Le nombre total de vecteur entrant pour un bloc RSMA est de 2^{32}.

Généralement, les outils de simulation de fautes permettent de déterminer la valeur de P_f mais dans le cas d'un bloc RSMA une approche exhaustive n'est pas envisageable. Afin de résoudre ce problème, le bloc RSMA est traité en deux parties : la partie RS et la partie MA. Concernant les S-Box, elles possèdent seulement 8 entrées d'où une analyse exhaustive est possible et ainsi P_f peut être déterminée pour chaque faute par simulation. De plus les opérations MixColumns et AddRoundKey sont inversibles, par conséquent les réponses erronées en sortie des S-Box seront propagées jusqu'en sortie du bloc RSMA. Concernant les autres opérations, une approche exhaustive n'est pas envisageable (2^{32}), néanmoins l'opération MixColumns ne représente que 3 à 4% du circuit. Dans cette étude, la probabilité P_f de l'opération MixColumns n'est pas déterminée et par conséquent le résultat final sera légèrement impacté par une erreur de 3 à 4%.

Pour l'architecture proposée, la probabilité que f ne soit pas activée au cours des cycles d'horloge de comparaison est égale à $(1\text{-}P_f)^4$ et la probabilité que f soit activée au moins une fois pendant un cycle d'horloge sans comparaison est égale à $1\text{-}(1\text{-}P_f)^6$.

Par conséquent $P_{err}(f)$ s'écrit :

$$P_{err}(f) = (1\text{-}P_f)^4 \times (1\text{-}(1\text{-}P_f)^6) .$$

L'évolution de $P_{err}(f)$ en fonction de P_f est représentée sur la figure 6-6.

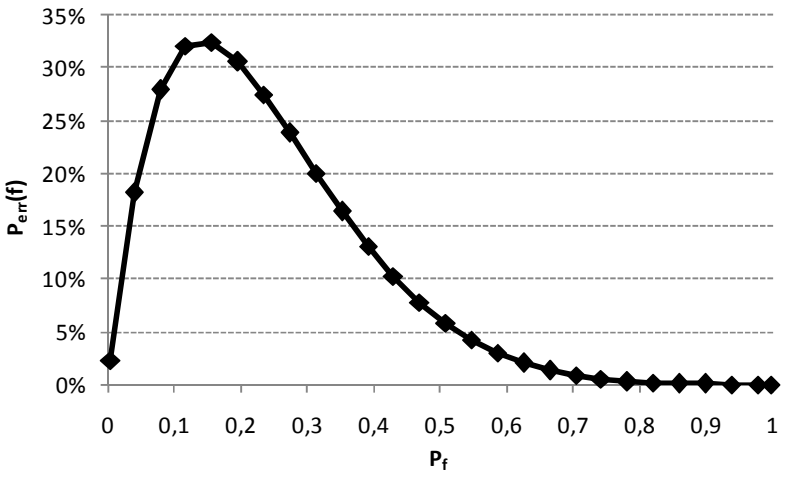


FIGURE 6-6 : $P_{err}(f)$ EN FONCTION DE P_f

Il est à noter que les fautes dites dure-à-tester ($P_f \sim 0$) et facile-à-tester ($P_f \sim 1$), ne sont pas les fautes qui devraient produire des erreurs indétectables. Au contraire, la probabilité la plus élevée de ne pas détecter la réponse erronée correspond à des erreurs ayant une probabilité P_f de 0,14.

Afin de calculer la probabilité d'erreurs sur l'ensemble du circuit (en fait 97% du circuit, l'opération MixColumns étant omise), la distribution des probabilités d'activation des fautes (P_f) dans les S-Box a été déterminée par simulation de fautes. Pour ce faire le nombre de fautes qui sont activées par un seul vecteur ($P_f = 1/256$) a été calculé, puis celles activées par deux vecteurs et ainsi de suite…

La figure 6-7 résume, pour chaque probabilité P_f, le nombre de fautes $FD(P_f)$ qui sont activées.

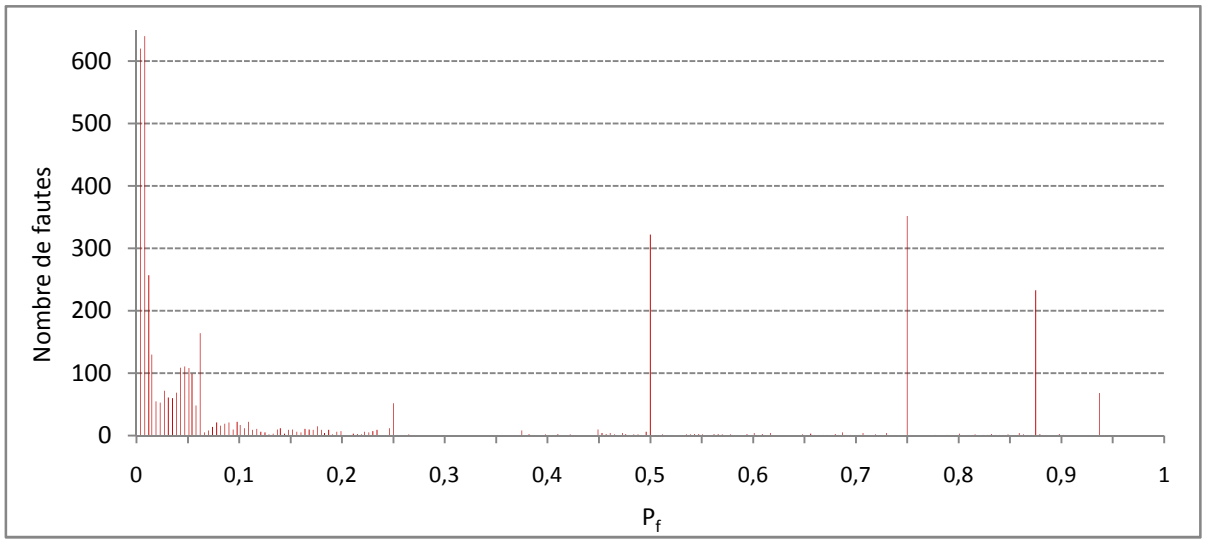


FIGURE 6-7 : DISTRIBUTION DES FAUTES ACTIVEES EN FONCTION DE P_f NOTE $FD(P_f)$

En faisant l'hypothèse que chaque faute a la même probabilité d'apparaître dans le circuit, la probabilité d'erreurs P_{ERR} du bloc RSMA est calculée comme la moyenne pondérée des valeurs $P_{err}(f)$ en fonction de la distribution $FD(P_f)$:

$$P_{ERR} = \frac{1}{\#\,\text{Fautes}} \sum_{i=1}^{256} \left\{ \left[FD\left(\frac{i}{256}\right) \right] \times \left[\left(1-\frac{i}{256}\right)^4 \times \left(1-\left(1-\frac{i}{256}\right)^6\right) \right] \right\}.$$

D'où P_{ERR} est égal à 10,20%, c'est-à-dire que l'architecture proposée permet de détecter environ 90% des erreurs en un cycle de chiffrement (10 cycles d'horloge).

L'évolution de cette probabilité, en fonction du nombre de cycles de chiffrement exécutés, peut être déterminée à partir de l'équation suivante :

$$P_{err}(E, f) = (1-P_f)^{4\times E} \times (1-(1-P_f)^{6\times E}),$$

où E représente le nombre de cycle de chiffrement exécuté. En effet, si E cycles de chiffrement sont exécutés alors le bloc RSMA est comparé 4×E cycles d'horloge et n'est pas comparé 6×E cycles d'horloge.

A l'aide de la distribution de faute FD donnée sur la figure 6-7, la probabilité d'erreurs globale P_{ERR} du bloc RSMA peut être déterminée en fonction des cycles de chiffrement exécutés (voir figure 6-8).

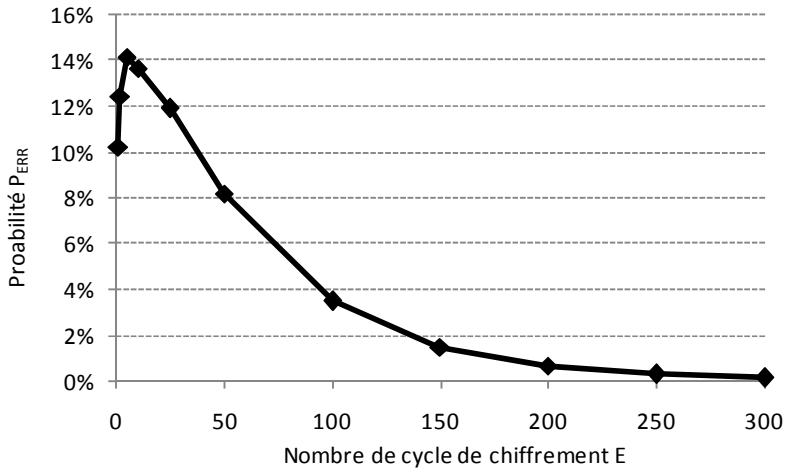


FIGURE 6-8 : PROBABILITE D'ERREURS EN FONCTION DU NOMBRE DE CYCLES DE CHIFFREMENT EXECUTES

La probabilité d'erreurs est maximum (~ 14%) quand 5 cycles de chiffrement sont exécutés tandis que pour des nombres plus élevé de cycles de chiffrement, elle tend vers 0. A savoir, pour 300 cycles de chiffrement, la probabilité de détecter une erreur est de 99,9%.

- *Coût de l'implantation :*

Cette sous-partie est dédiée au coût en surface engendré par l'implantation de la solution de test en ligne proposée.

L'architecture proposée a été décrite en VHDL et synthétisée à l'aide de Design Compiler en utilisant la librairie CMOS 130nm fournit par STM [STM]. Le tableau 6-3 présente les coûts d'implantation en termes de cellules et de surface.

		AES		AES Redondance	
		# cellules	Surface en µm²	# cellules	Surface en µm²
Ronde	-SubBytes	8384	40338,9961	10480	50423,7451
	-ShiftRows	0	0	0	0
	-MixColumns	348	3933,79834	435	4917,24793
	-AddRoundKey	431	3243,40477	635	4362,95711
	-Registres	128	2388,37516	160	2985,46948
	-Mux + Comp…	0	0	1003	5905,08818
Contrôleur		49	275,497589	72	510,384009
Logique		320	2781,323551	299	2252,27712
TOTAL		**9660**	**52961,39551**	**13084**	**71357,169**

TABLEAU 6-3 : COUT DE L'ARCHITECTURE REDONDANTE DE TEST EN LIGNE

La surface du circuit originel est de 52961,39551µm² (ce qui correspond à 9660 cellules) tandis que la surface de l'architecture de test en ligne proposée est 71357,169µm² (ce qui correspond à 13084 cellules), ce qui représente un surcoût en surface de 34,73%.

- *Evaluation de notre architecture au vue des erreurs transitoires*

Bien que l'architecture proposée ait été développée pour détecter les fautes permanentes, il est à noter que celle-ci peut détecter les fautes transitoires sous certaine contraintes. En effet, supposons une faute transitoire de courte durée (1 cycle d'horloge) affectant un bloc RSMA sensibilisée par le vecteur entrant, cette faute a deux chances sur 5 d'être détectée (soit 40%) Ces deux chances sont liées aux cycles de comparaison des blocs RSMA, en effet chaque bloc est comparé deux fois en cinq cycles. Par conséquent, cette architecture ne peut pas être envisagée comme solution pour contrer les attaques par injection de fautes.

6.3 CONCLUSION

Dans ce chapitre, nous avons proposé une solution de test en ligne pour le cœur de chiffrement AES. Cette solution permet de détecter toutes les fautes permanentes simples et multiples affectant le circuit durant son fonctionnement.

L'architecture proposée est basée sur les techniques de redondance matérielle. En effet un bloc RSMA supplémentaire est ajouté. Il faut noter que cette solution n'a aucun impact sur la latence de calcul. De plus la solution proposée est très efficace en termes de latence et de détection de fautes, mais également le coût de sa mise en œuvre est faible (de l'ordre de 34,73%). Un des avantages de cette architecture est de pourvoir envisager, après une phase de diagnostic isolant le bloc RSMA défectueux, la correction (ou réparation) en utilisant la redondance matérielle.

D'autre part, dans le contexte des circuits sécurisés, les solutions de test en ligne permettent généralement de protéger le circuit contre les attaques non-invasives. L'architecture proposée augmente la complexité de mise en œuvre de l'attaque en consommation (DPA). En effet la reconfiguration dynamique des blocs RSMA implique que pour une même entrée il existe 5 courbes en courant différentes. Cette architecture ayant été développée uniquement pour les fautes permanentes, elle ne permet pas de protéger le circuit des attaques par injection de fautes.

Conclusion générale

Conclusion générale

Avec l'accroissement de la numérisation des données et des besoins en confidentialité, les circuits sécurisés se sont particulièrement développés ces dernières années. Ces circuits impliquent non seulement des spécificités de conception pour garantir la sécurité des données, mais également des particularités de test. En effet, l'approche de test utilisée pour ces circuits ne doit pas laisser révéler d'information liées aux données secrètes.

L'approche de test externe combinée avec une technique de conception en vue du test « scan-path » est une solution de test couramment utilisée dans l'industrie car elle fournit de bons résultats. Cependant, cette technique a l'inconvénient dans le contexte des circuits sécurisés de permettre à un éventuel attaquant de retrouver la clef secrète de chiffrement. Ces conclusions sur l'approche de test externe ont orienté nos travaux vers une approche de test intégré. L'objectif de cette thèse a été de développer des solutions de test intégré permettant de réduire le coût en surface d'intégration des ressources de test sans ouvrir la voie à de nouvelles attaques.

Pour répondre à cette contrainte de surface, nous avons proposé des solutions de test intégré hors ligne et en ligne. Le test hors ligne est orienté vers le test de production tandis que le test en ligne ne permet que de s'assurer de la validité des données en cours de fonctionnement.

Tout d'abord, nous avons développé des solutions pour le test hors ligne. Ces solutions reposent sur les propriétés des cœurs de chiffrement à générer des données aléatoires et à propager des erreurs potentielles sur le message entrant. Le premier aspect concerne l'utilisation du cœur de chiffrement comme générateur de vecteurs de test. En pratique, la solution proposée consiste à ajouter une rétroaction autour de la ronde du cœur de chiffrement et à exécuter un nombre de rondes supérieur au nombre de rondes requis pour un chiffrement. Les résultats de rondes obtenus en sortie forment la séquence de vecteurs de test. Pour vérifier

que ces séquences puissent être utilisées dans le contexte d'un test aléatoire, nous avons étudié leurs propriétés à l'aide des tests statistiques du NIST et les avons comparées à celles d'une séquence générée à l'aide de LFSR. Les résultats obtenus ont montré que les séquences engendrées par les cœurs de chiffrement DES et AES ne présentent pas plus de corrélation que celles obtenues à l'aide d'un LFSR et que les taux de couverture de fautes sont au moins équivalents. La deuxième solution que nous avons développée consiste à utiliser le cœur de chiffrement comme compacteur de réponses de test. Pour ce faire, la réponse courante du circuit sous test est additionné modulo deux avec le résultat de la ronde courante de chiffrement. Nous avons démontré que la probabilité de masquage liée à ce type compacteur est égale à celle d'un MISR. Par conséquent, le cœur de chiffrement peut être utilisé comme compacteur de réponse avec une faible probabilité de masquer la présence d'une erreur.

Toujours dans un contexte de réduction de la surface nécessaire à l'intégration des ressources de test, nous avons ensuite proposé une solution d'autotest pour les cœurs de chiffrement. Nous avons tout d'abord établi une estimation de la longueur minimale de la séquence de test pour couvrir toutes les fautes. Nous avons ensuite montré expérimentalement que l'ensemble des fautes du cœur de chiffrement est effectivement détecté en 2534 cycles. La modification requise pour implanter l'autotest est identique à celle permettant d'utiliser le cœur de chiffrement en tant que générateur de vecteurs de test.

Ensuite, une étude sur le coût d'implantation de ces trois solutions de test hors ligne (génération de vecteurs, compaction de réponses et autotest) a été réalisée. La solution la plus coûteuse à implanter est l'utilisation du cœur de chiffrement en mode de compaction de réponses avec un coût de 2,05%. Les modes « générateur de vecteurs de test » et « autotest » engendrent eux un coût en surface négligeable de respectivement 0,03% et 0,04%.

Pour finir, nous avons proposé une solution de test en ligne. Les solutions de test en ligne doivent garantir la fiabilité du circuit durant son fonctionnement normal. Nous avons développé une solution de test en ligne basée sur de la redondance matérielle (rajout d'un bloc RSMA). Cette solution permet de détecter 90% des fautes possibles en un cycle de chiffrement, pour un surcoût en surface faible (de l'ordre de 35%).

Le bilan de cette étude nous permet de conclure que, l'utilisation d'une approche de test intégré étant essentielle pour garantir la sécurité des données, nos solutions permettent d'envisager une telle approche de test tout en réduisant le coût en surface de son intégration.

Les perspectives de ce travail peuvent s'orienter suivant trois directions.

D'un point de vue théorique, la question concernant la longueur du cycle de la séquence aléatoire obtenue avec un générateur utilisant un cœur de chiffrement symétrique reste à ce jour ouverte. Au cours de cette thèse, nous avons considéré la longueur moyenne du cycle comme valant 2^{m-1} [Sch97] où m est la taille du bloc traité par le cœur de chiffrement. En pratique, lors de nos travaux, sur des séquences de plus de 10^6 vecteurs, nous n'avons jamais constaté de cycles. Il serait également intéressant de voir plus précisément si la rétroaction autour du module de génération de clefs, comme nous l'avons utilisé dans notre étude, influe sur la longueur des cycles.

Ensuite, une solution utilisant le cœur de chiffrement de façon simultanée comme générateur de vecteurs de test et compacteurs de réponses pour le test d'un même circuit semble envisageable. En effet, la structure développée pour la compaction de réponses pourrait de prime abord être utilisée dans ce cas. Toutefois, cette solution nécessite d'être plus profondément étudiée afin de s'assurer que la présence d'une erreur est bien détectée.

Finalement, il serait intéressant d'envisager une étude similaire sur les cœurs de chiffrement asymétrique. En effet, si pour l'instant ces cœurs sont encore implantés sous forme logicielle, on peut raisonnablement imaginer, compte tenu de l'évolution technologique (diminution de la taille des transistors, augmentation de la densité d'intégration, augmentation des fréquences de fonctionnement…) qu'ils soient implantés sous forme matérielle dans un futur proche. Se poseront alors les mêmes questions que pour les cœurs de chiffrement symétrique « Qu'en est-il du test de ces cœurs ? » « Peut-on les utiliser comme ressource de test pour les autres cœurs du circuit ? » Les réponses à ces questions ne peuvent être directement déduites des travaux présentés dans ce mémoire. En effet, les opérations impliquées dans ces algorithmes sont sensiblement différentes de celles des algorithmes symétriques. Toutefois, ces opérations reposent sur un formalisme mathématique beaucoup plus solide que celles de l'AES ou du DES, ce qui devrait rendre l'établissement de caractéristiques utiles pour le test (bijection, de longueurs de cycle, etc…) plus aisé.

Annexe A : Les tests du NIST

Annexe A : Les tests du NIST

- **1. Test de fréquence** « Monobit Test »

Ce test s'intéresse à la proportion de « zéros » et de « uns » dans la séquence entière. Le but de ce test est de déterminer si le nombre de « zéros » et de « uns » dans la séquence est approximativement égal à celui prévu pour une vraie séquence aléatoire. Le test évalue la déviation de la proportion des « uns » par rapport à 1/2.

Illustrons le principe général du test

Supposons une séquence ε = 11001010110011

Remplacer 1 par (+1) et 0 par (-1) et faire la somme S bit à bit de la séquence.

S = 1+1-1-1+1-1+1-1+1+1-1-1+1+1 = 2

Si S est très grand (trop de 1) ou très petit (trop de 0) alors la séquence est considérée comme non aléatoire.

- **2. Test de fréquence dans un bloc «** Block Frequency Test »

Ce test étudie la proportion de « uns » et de « zéro » dans un bloc de M bits. Dans notre cas M vaut 128. Le but de ce test est de déterminer si le nombre de « uns » dans un bloc de M bits est approximativement $M/2$, comme ce serait le cas pour une séquence aléatoire.

Illustrons le principe général du test

Supposons une séquence ε = 11001010110011

Notons n la longueur totale de la séquence (ici n=14), M la taille des blocs (supposons M=3), N= $\left\lfloor \frac{n}{M} \right\rfloor$ (ici 4) représente le nombre de blocs traités.

Déterminer la proportion de 1 dans chaque bloc, puis comparer la distribution des proportions avec celle attendu pour une séquence aléatoire.

Blocs : $B_1 = 110$ $B_2 = 010$ $B_3 = 101$ $B_4 = 100$

Proportions : $p_1 = \frac{2}{3}$ $p_2 = \frac{1}{3}$ $p_3 = \frac{2}{3}$ $p_4 = \frac{1}{3}$

- **3. Test des suites homogènes «** Runs Test »

Ce test porte sur le nombre de suites homogènes (de « salves ») sur la totalité de la séquence. Une suite homogène de longueur k est une séquence ininterrompue de bits identiques encadrée par des bits de valeur opposée. Le but de ce test est de déterminer si le nombre de suites homogènes de longueur k correspond à celui d'une suite véritablement aléatoire.

Illustrons le principe général du test

Supposons une séquence ε = 11001010110011

Calculer le nombre de changement de valeur (noté Nc), et comparer avec celui d'une séquence aléatoire. Si $\varepsilon_i = \varepsilon_{i+1}$ alors Nc = Nc, et si $\varepsilon_i \neq \varepsilon_{i+1}$ alors Nc = Nc +1

Nc = 0+1+0+1+1+1+1+1+0+1+0+1+0 = 8.

Si la valeur de Nc est petite (oscillation très lente : peu de changement) alors la séquence est considérée comme non aléatoire.

- **4. Test sur la plus longue suite de « un » dans un bloc** « Long Runs of Ones Test »

Ce test porte sur la plus longue suite homogène de « un » dans un bloc de taille M. Le but est de vérifier si la longueur de la plus longue suite homogène de « un » correspond à la longueur de la plus longue suite homogène de « un » rencontré dans une suite véritablement aléatoire.

Illustrons le principe général du test

Supposons une séquence ε = 11001100000101010110110001 01

Diviser la en blocs de taille M (ici 8) et déterminer pour chaque bloc la longueur de la plus longue suite homogène de « un » et comparer la distribution de ces longueurs avec celle d'une séquence aléatoire.

Blocs	longueur max de « un »
11001100	2
00010101	1
01101100	2 …

- **5. Test sur le rang de la matrice binaire aléatoire** « Rank Test »

L'objectif du test est de calculer le rang des sous-matrices disjointes formant la séquence toute entière. Les sous-matrices sont de taille M×Q avec M et Q égaux à 32 bits. Le but de ce

test est de contrôler la dépendance linéaire entres des sous-suites de longueur k de la séquence originale.

Illustrons le principe général du test

Supposons une séquence ε = 01011001001010101101

Ecrire la séquence sous forme de matrice M×Q (ici 3×3). Déterminer le rang binaire de chaque sous-matrice. Distribuer les rangs de ces sous-matrices par rapport au rang plein et comparer le résultat avec celui d'une séquence aléatoire.

$S\text{-}M_1 = \begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix}$ et $S\text{-}M_2 = \begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix}$ le rang de $S\text{-}M_1$ vaut 2 et le rang $S\text{-}M_2$ de vaut 3.

Distribuer les rangs de ces sous-matrices par rapport au rang plein (ici 3) et comparer le résultat avec celui d'une séquence aléatoire.

- **6. Test sur la transformée de Fourier discrète** « Discrete Fourier Transform (Spectral) Test (DFT) »

Ce test porte sur la hauteur des pics dans la transformée de Fourier discrète de la séquence. Le but est de détecter la périodicité de certains motifs dans la séquence testée.

Illustrons le principe général du test

Supposons une séquence ε = 1001010011, noté X = 1, -1, -1, 1, -1, 1, -1, -1, 1, 1.

Notons n la longueur de la séquence. Calculer sa transformée de Fourier Discrète S= DFT(X) à l'aide de :

$$f_j = \sum_{k=1}^{10} x_k e^{-2\pi i(k-1)j/n}$$

Calculer $M = |S'|$, où S' est la sous-chaîne formée par les $n/2$ premiers éléments de S.

Déterminer le nombre de pic dans M, puis le comparer avec celui attendu pour une séquence aléatoire.

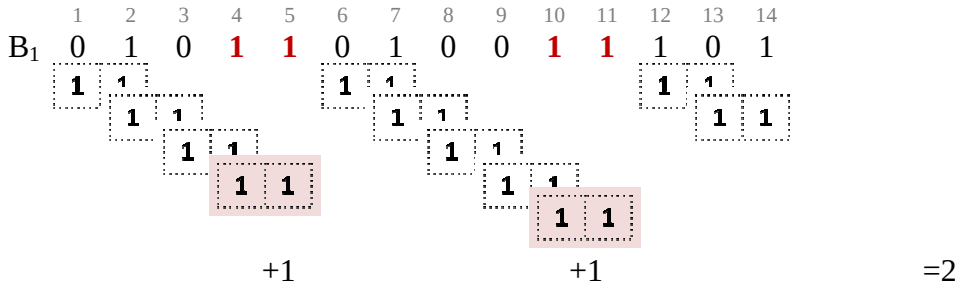
- **7. Recherche d'un motif apériodique** « Aperiodic Templates Test »

Ce test porte sur le nombre d'occurrence d'une sous-suite donnée à l'intérieur de la séquence. Le but de ce test est de rejeter les séquences présentant un trop grand nombre d'occurrences d'un motif apériodique. La recherche se fait en utilisant une fenêtre de m bits glissant sur la séquence à la recherche d'un motif de taille m. Lorsque ce motif a été trouvé, la fenêtre de recherche est replacée sur le premier bit suivant le motif découvert.

Illustrons le principe général du test

Supposons une séquence ε = 01011010011101101110100010 10. Considérons des blocs de taille 14 bits (B_1 et B_2). Prenons un motif de taille 2 bits '11'.

La fenêtre contenant le motif '11' parcours le bloc B_1 (B_1 = 01011010111001). Le 4ième et le 5ième bits correspondent au motif, la variable servant au comptage du nombre d'occurrence du motif dans le bloc est incrémenté. Puis la fenêtre reprend son parcours du bloc au 6ième bit jusqu'à ré-rencontrer le motif et ainsi de suite.



Le motif '11' est rencontré 2 fois dans le bloc B_1, le bloc B_2 est traité de la même façon.

Le test rejettera les séquences qui ont un trop grand nombre d'occurrence d'un motif.

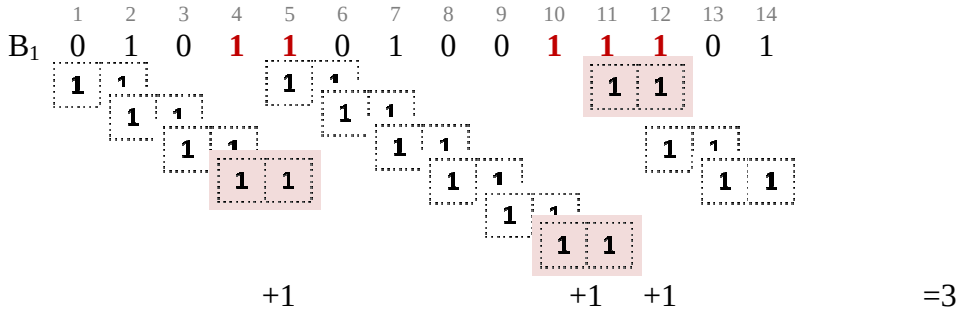
- **8. Recherche d'un motif périodique** « Periodic Template Test »

Comme le test précédent, ce test recherche un motif particulier dans la séquence. Cependant, lorsque ce motif est trouvé dans la suite, la fenêtre de recherche n'est pas déplacée à la fin de celui-ci, mais continue à avancer normalement bit par bit.

Illustrons le principe général du test

Supposons une séquence ε = 01011010011101101110100010 10. Considérons des blocs de taille 14 bits (B_1 et B_2). Prenons un motif de taille 2 bits '11'.

Le principe est identique au test de « Recherche d'un motif apériodique » hormis que la fenêtre reprend son parcours du bloc B_1 uniquement un bit après le précédent résultat de la recherche.



Le motif '11' est rencontré 3 fois dans le bloc B_1, le bloc B_2 est traité de la même façon.

Le test rejettera les séquences qui ont un trop grand nombre d'occurrence d'un motif.

- **9. Test statistique universel de Maurer** « Maurer's Universal Statistical Test »

Ce test détermine le nombre de bits entre deux motifs identiques. Le but est de détecter si la séquence peut être compressée de manière significative sans perte d'information. Une séquence très compressible est considérée comme non aléatoire.

Illustrons le principe général du test

Supposons une séquence ε = 01011010011101010111 de longueur n =20 bits. Les n bits de la séquence sont partitionnés en 2 parties : une partie initiale composée de Q bloc de L-bits non recouvrant et une partie de test composée de K bloc de L-bits non recouvrant.

Si L=2 et Q=4 alors $K = \left\lfloor \frac{n}{L} \right\rfloor - Q = 6$ d'où :

Partie d'initialisation : 01, 01, 10, 10.

Partie test : 01, 11, 01, 01, 01, 11.

La partie initialisation est écrite sous la forme d'un tableau A-1 dans lequel le numéro du bloc correspond à la dernière occurrence des L-bits. Pour le L-bits = 01, la dernière occurrence correspond au deuxième bloc. Ce tableau est ensuite complété avec les 6 blocs (5 à 10) de la partie test, la position du bloc remplace la valeur précédente dans la table. Par exemple le bloc 5 vaut '01' donc le 2 est remplacé par un 5 (dans la ligne suivante).

	Valeur possible pour les L-bits			
	00	01	10	11
Initialisation	0	2	4	0
Bloc 5	0	5	4	0
Bloc 6	0	5	4	6
Bloc 7	0	7	4	6
Bloc 8	0	8	4	6
Bloc 9	0	9	4	6
Bloc 10	0	9	4	10

TABLEAU A-1 : TEST MAURER

Le test consiste à calculer l'écart entre deux valeurs d'une colonne du tableau. Par exemple au bloc 5 l'écart est égal à 5-2, au bloc 6 l'écart est de 6-0…

Enfin la statistique basée sur ces différences est calculée et comparée à celle attendue pour une séquence aléatoire.

- **10. Test de la complexité linéaire** « Linear Complexity Test »

Ce test détermine la longueur d'un registre à décalage (type LFSR) produisant la séquence analysée. Il permet de déterminer si la séquence est suffisamment complexe pour être considérée comme aléatoire. Si le registre est trop court, la suite n'est pas aléatoire.

Illustrons le principe général du test

La séquence de n bits est partagée en N blocs indépendants de Mbits. A l'aide de l'algorithme de Berlekamp-Massey, la complexité linaire de chaque bloc (L_i) est calculée. L_i représente la longueur minimale du registre à décalage permettant de générer l'ensemble des bits du bloc. Si L_i est trop faible, la séquence est considérée comme non aléatoire.

- **11. Test série** « Serial Test (serial1 & serial2) »

Ce test porte sur la fréquence de chaque sous-séquence de m bits (séquences superposées). Le but de ce test est de déterminer si le nombre d'occurrences des 2^m séquences de m bits est approximativement égal à celui attendu pour une vraie suite aléatoire. En résumé, une séquence aléatoire doit être uniforme, ce qui signifie que chaque sous-séquence de m bits doit avoir la même probabilité d'apparaître que chaque autre sous-séquence de m bits. Si m est choisit égal à 1 alors ce test est équivalent au Test de fréquence (numéro 1).

Illustrons le principe général du test

Supposons une séquence ε = 0011011101, la longueur de la séquence est notée n (ici n=10).
Choisissons la taille des sous-séquences m = 2 bits. Ainsi nous pouvons considérer ε' qui correspond à ε auquel nous ajoutons m-1 bits à la fin provenant du début de ε. Dans l'exemple m = 2, le premier bit de ε (0) est rajouté à la fin de ε pour former ε'= 00110111010.
La fréquence de chaque sous-bloc (recouvrant) est déterminée. Les sous-blocs de m-bits sont cherchés dans ε' et ceux de m-1 bits sont cherchés dans ε. Les combinaisons de m-bits à chercher sont 00, 01, 10, 11 et les combinaisons de m-1 bits à chercher sont 0 et 1.
Soit dans ε = 0011011101 il y a #0s= 4 et #1s = 6
Et dans ε'= 00110111010 il y a #00s= 1, #01s= 3, #10s= 3 et #11s= 3.
Puis ces fréquences sont comparées avec celles attendues pour une séquence aléatoire.

- **12. Test d'entropie approchée** « Approximate Entropy Test »

Il s'agit d'un test de fréquence sur toutes les sous-séquences de m bits. Le but est de comparer la fréquence dans 2 blocs superposés de longueur consécutive (m et m+1) à celles rencontrées dans une suite aléatoire.

Illustrons le principe général du test

Le principe est similaire au test série, hormis que les blocs traités sont de taille m puis m=m+1.

Supposons ε = 0100110101, et m = 3 alors :

m = 3 $\Rightarrow$ ε'= 010011010101, motifs cherchés de taille m

#000 = 0, #001 = 1, #010 = 3, #011 = 1, #100 = 1, #101 = 3, #110 = 1 et #111 = 0.

Puis m = m+1= 4 $\Rightarrow$ ε'= 0100110101010, motifs cherchés de taille m

#0011 = 1, #0100 = 1, #0101 = 2, #0110 = 1, #1001 = 1, #1010 = 3, #1101 = 1.

Puis ces fréquences sont comparées avec celles attendues pour une séquence aléatoire.

- **13. Test de la somme cumulée** « Cumulative Sums Test »

Ce test vérifie l'excursion maximale (depuis 0) lors d'une marche aléatoire définie par la somme cumulée des bits de la séquence ajustée à (−1,+1). Le but est de déterminer si la somme cumulée des séquences partielles est trop grande ou trop petite par rapport à celle attendue pour une séquence aléatoire.

Illustrons le principe général du test

Supposons ε = 1011010111, noté encore X = 1, -1, 1, 1, -1, 1, -1, 1, 1, 1.

Deux approches sont possibles pour ce test notés mode 0 et mode 1. Dans les deux approches le test consiste à faire la somme des bits consécutifs.

Mode 0	Mode 1
$S_1 = X_1$	$S_1 = X_n$
$S_2 = X_1 + X_2$	$S_1 = X_n + X_{n-1}$
⋮	⋮
$S_n = X_1 + X_2 + \ldots + X_n$	$S_n = X_n + X_n1 + \ldots + X_1$

Soit pour l'exemple :

Mode 0			Mode 1		
$S_1 = 1$	$S_4 = 2$	$S_7 = 1$	$S_1 = 1$	$S_4 = 2$	$S_7 = 3$
$S_2 = 0$	$S_5 = 1$	$S_8 = 2$	$S_2 = 2$	$S_5 = 3$	$S_8 = 4$
$S_3 = 1$	$S_6 = 2$		$S_3 = 3$	$S_6 = 2$	

Pour les deux modes, la plus grande somme partielle (en valeur absolue) est cherchée. Ici pour le mode 0, il s'agit de S_{10} et pour le mode 1, S_8 et S_{10}.
Pour que la séquence soit considérée comme aléatoire, il ne faut pas que cette valeur soit trop grande ou trop petite.

- **14. Test d'excursions aléatoires** « Random Excursion Test »

Ce test s'intéresse au nombre de cycles visités exactement K fois lors d'une marche aléatoire. La marche aléatoire est trouvée en effectuant la somme cumulée des séquences de (0, 1) ramenés à (−1, +1). Un cycle (ou excursion aléatoire) consiste en une séquence de n pas, de longueur unité pris au hasard, commençant et finissant à l'origine. Le but de ce test est de déterminer si le nombre de visites d'un état lors d'un parcours aléatoire dépasse celui attendu pour une séquence aléatoire.

Illustrons le principe général du test

Supposons ε = 0110110101, noté encore X = -1, 1, 1, -1, 1, 1, -1, 1, -1, 1. Sur le même principe que le test de la somme cumulée, les sommes partielles sont calculées (ici S = {-1, 0, 1, 0, 1, 2, 1, 2, 1, 2}. Représenter sous forme de graphe le résultat voir figure A-1 :

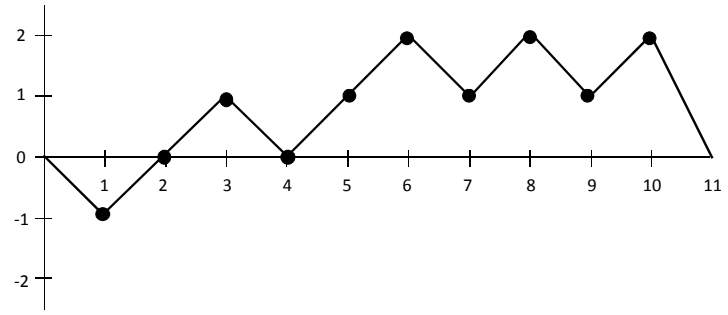


FIGURE A-1 : EXEMPLE DE MARCHE ALEATOIRE

Par principe la marche commencera et finira toujours par la valeur 0. Pour l'exemple, il y a 3 cycles différents : le premier (0, - 1, 0), le deuxième (0, 1, 0) et enfin le dernier (0, 1,2, 1, 2, 1, 2, 0). Enfin pour chacun des cycles, il faudra déterminer le nombre de visite du même état, soit pour le premier et le deuxième cycle un seul état a été visité respectivement -1 et 1. Par contre pour le dernier cycle, plusieurs états sont visités : l'état 1 trois fois et l'état 2 trois fois également. Ensuite, à chaque état est affecté le nombre de cycles qui l'ont visité. En fonction de ce résultat, la séquence est dite aléatoire ou non aléatoire.

- **15. Variante du test d'excursions aléatoires** « Random Excursion Variant Test »

Ce test porte sur le nombre de fois où un même état est rencontré lors d'une marche aléatoire. Le but est de détecter les écarts par rapport au nombre d'occurrence normal des différents états lors d'une marche aléatoire.

Illustrons le principe général du test

Comme pour le test d'excursions aléatoires, il faut tracer la marche aléatoire de la séquence étudiée. Supposons la même séquence ε = 0110110101. Le test va consister à déterminer le nombre de visites d'un même état. Pour l'exemple, le nombre d'occurrence de la valeur (-1) est de 1, de la valeur (1) est 4 et de la valeur (2) est 3. Puis ces valeurs d'occurrence sont comparées avec celles attendues pour une séquence aléatoire.

Références

Références

[Abr91] : D.G. Abraham, G.M. Dolan, G.P. Double, and J.V. Stevens, "*Transaction security systems*", IBM Systems Journal, 1991, 30(2), pp. 206-229.

[Abra94] : M.Abramovici, M.A. Breuer and A.D. Friedman, "*Digital Systems Testing and Testable Design*", IEEE Press, 1990.

[Agr93] : V.D. Agrawal, C.R. Kime and K.K. Saluja, "*A tutorial on built-in self-test. 1. Principles*", IEEE Design & Test of Computers, Mars 1993, Volume 10, Issue 1, pp. 73-82.

[Agr93bis] : V.D. Agrawal, C.R. Kime and K.K Saluja, "*A tutorial on built-in self-test. 2. Applications*", IEEE Design & Test of Computers, June 1993, Volume 10, Issue 2, pp. 69-77.

[Alya03] : A.A. Al-Yamani and E.J. McCluskey, "*Built-in reseeding for serial BIST*", Proceedings VLSI Test Symposium, 2003, pp. 63-68.

[And98] : R. Anderson, E. Biham and L. Knudsen, "*Serpent : A Proposal for the Advanced Encryption Standard*", http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf, 1998.

[Andl82] : D.Andleman and J. Reeds, "*On Cryptanalysis of Rotor Machines and Substitution-Permutation Networks*", IEEE Transactions on Information Theory, 1982, IT-28(4), 578-584.

[Bard84] : P. H Bardell, G. S. Ditlow and J. Savir, "*Random pattern testability*", IEEE Trans. on Computers, January 1984, Vol. C-33, No. 1, pp.79-90.

[Bard87] : P. H Bardell, W. H. McAnney and J. Savir, "*Built-In Test for VLSI: Pseudorandom Techniques*", John Wiley & Sons, 1987.

[Berg61] : J.M. Berger, "*A note on an error detection code for asymmetric channels*", Information and Control, March 1961, vol 4, pp. 68-73.

[Ber03] : G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "*Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard*", IEEE Trans. Computers, Apr. 2003, vol. 52, no. 4, pp.492-505.

[Bih97] : E. Biham and A. Shamir, "*Differential fault analysis of secret key cryptosystems*", Advances in Cryptology - Crypto '97, Lecture Notes in Computer Science, Springer, 1997, vol. 1294, pp. 513--525.

[Bir04] : A. Biryukov, "*The boomerang attack on 5 and 6-round reduced AES*", Congrès AES 2004 : advanced encryption standard, Advanced encryption.

[Bos08] : A. Bosio and G. Di Natale, "*LIFTING: a Flexible Open-Source Fault Simulator*", IEEE International Asian Test Symposium (ATS 2008), 2008, à paraître.

[Brgl89] : F. Brglez, D. Bryant and K. Kozminski, "*Combinational Profiles of Sequential Benchmark Circuits*", IEEE Int. Symp. on Circuits and Systems, 1989, pp. 1929-1934.

[Bus00] : M.L. Bushnell and V.D. Agrawal, "*Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*", Kluwer Academic Publishers, 2000.

[Cart82] : W.C. Carter, "*Signature Testing with Guaranteed Bounds for Fault Coverage*", Proc. of International Test Conference, 1982, pp. 75-82.

[Cha86] : D. Chaum and J.-H. Evertse, "*Cryptanalysis of DES With a Reduced Number of Rounds ; Sequences of Linear Factors in Block Ciphers*", Advances in Cryptology| CRYPTO '85, Springer-Verlag, 1986, pp. 192-211.

[Che89] : W.T. Cheng and V. D. Agrawal, "*An Economical scan Design for Sequential Logic Test Generation*", Proceedings of the Fault-Tolerant Computing Symposium, 1989, pp. 28-35.

[Dav86] : R. David, "*Signature Analysis for Multiple Output Circuits*", Transactions on Computers, Vol. C-35, N° 9, September, 1986, pp. 830-837.

[Dav98] : R. David, "*Random Testing of Digital Circuits: Theory and Applications*", New-York: Marcel Dekker, 1998.

[Dcomp] : http://www.synopsys.com/products/test/dftcompiler_ds.htm

[Derv] : http://deversys.com/?action=download&id=44

[DIEH] : George Marsaglia's Diehard tests
http://stat.fsu.edu/~geo/diehard.html

[Dif76] : W. Diffie and M. Hellman, "*New directions in cryptography*", IEEE TIT Transactions on Information Theory, 1976, vol. 22, pp. 644-654.

[DiN07] : G. Di Natale, M-L. Flottes and B Rouzeyre, "*An On-Line Fault Detection Scheme for SBoxes in Secure Circuits*", Proc. IEEE Int. On-Line Testing Symposium, 2007, pp. 57-62.

[ElG85] : T. ElGamal, "*A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*", IEEE Transactions on Information Theory, 1985, Vol. IT-31, N° 4, pp469–472.

[ENT] : http://www.fourmilab.ch/random/

[Fag03] : C. Fagot, O. Gascuel, P. Girard and C. Landrault, "*A Ring Architecture Strategy for BIST Test Pattern Generation*", Journal of Electronic Testing: Theory and Applications, June 2003, Vol. 19, N° 3, pp. 223-231.

[FIPS46a] : NIST, "*Data Encryption Standard (DES)"*, National Institut of Standard and Technology FIPS PUB46-2, December 1993.

[FIPS46b] : NIST, "*Data Encryption Standard (DES and Triple-DES)*", National Institut of Standard and Technology FIPS PUB46-3, October 1999.

[FIPS140] : http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[FIPS197] : Announcing the ADVANCED ENCRYPTION STANDARD (AES), 2001 November 26.

[Foa01] : D. Foata, G-N. Han and B. Lass, "*Les nombres hyperharmoniques et la fratrie du collectionneur de vignettes*", Séminaire Lotharingien de Combinatoire, 2001, vol. 47, B47a, pp. 20.

[Gir04] : C. Giraud, "*DFA on AES*", AES Conference 2004, pp. 27-41.

[Goe81] : P. Goel, "*An Implicit Enumeration Algorithm to Generate Tests for Combinational Circuits*", IEEE Transactions on Computers, Mar. 1981, vol. C-30, no. 3, pp. 215-222.

[Hass83] : S.Z. Hassan and E. J. McCluskey, "*Increased Fault Coverage Through Multiple Signatures*", Proc. of International Fault-Tolerant Computing Symposium, 1984, pp. 354-359.

[Hay76] : J. Hayes, "*Transition count testing of combinational logic circuits*", IEEE Transactions on Computers, June 1976, VOL C–25, no 6.

[Hell95] : S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "*Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers*", IEEE Trans. on Computers, February 1995, Vol. 44, N° 2, pp. 223-233.

[Helle03] : P. Hellekalek and S. Wegenkittl, "*Empirical evidence concerning AES*", ACM Transactions on Modeling and Computer Simulation, October 2003, Volume 13, Issue 4, pp. 322-333.

[Hel04] : D. Hély, F. Bancel, M-L. Flottes, B. Rouzeyre, M. Renovell and N. Bérard, "SCAN Design and Secure Chip" International On-Line Test Symposium (IOLTS 2004), Funchal, Portugal, July 12-14, pp. 219-224.

[Hel05] : D. Hély, F. Bancel, M-L. Flottes and B. Rouzeyre, "*Test Control for Secure Scan Designs*", European Test Symposium (ETS), May 2005, pp. 190-195.

[Hel06] : D. Hély, F. Bancel, M-L. Flottes and B. Rouzeyre, "*A secure Scan Design Methodology*", 7th IEEE Latin American Test Workshop, 2006, pp. 81-86.

[Hel06bis] : D. Hély, F. Bancel, M-L. Flottes and B. Rouzeyre, "*Scan Pattern Watermarking*", IEEE Latin American Test Workshop, 2006.

[Hort89] : P.D. Hortensius, R.D. McLeod, W. Pries, M. Miller, and H.C. Card, "*Cellular Automata Based Pseudorandom Number Generators for Built-In Self-Test*", Transactions on Computer-Aided Design, August 1989, Vol. 8, N° 8, pp. 842-858.

[Hort90] : P.D. Hortensius, R.D. McLeod and B.W. Podaima, "*Cellular Automata Circuits for Built-In Self-Test*", IBM Journal of Research and Development, March/May 1990, Vol. 34, N°2/3, pp. 389-405.

[How81] : M.J. Howes and V.D. Morgan, "*Reliability and Degradation-Semiconductor Devices and Circuits*", Wiley-Interscience, 1981.

[Kar01] : R. Karri and al, "*Fault-Based Side-Channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture*", IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2001.

[Kar02] : R. Karri, K. Wu, P. Mishra and Y. Kim, "*Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers*", IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Dec. 2002, vol. 21, no. 12, pp. 1509-1517.

[Ker83] : A. Kerckhoffs, "*La cryptographie militaire*", Journal des sciences militaires, Janvier 1883, vol. IX, pp. 538, pp. 161-191.

[Kie00] : G. Kiefer; H. Vranken, E.J. Marinissen and H.J. Wunderlich, "*Application of Deterministic Logic BIST on Industrial Circuits*", Journal of Electronic Testing: Theory and Applications, June-August 2001, Vol. 17, N° 3-4, pp. 351-362.

[Kob84] : N. Koblitz, "*Introduction to Elliptic Curves and Modular Forms*", Graduate Texts in Mathematics, Springer, 1984, N° 97, ISBN 3-540-96029-5.

[Koc96] : P. Kocher, "*Timing Attacks on Implementations of Die-Hellman, RSA, DSS, and Other Systems*", LNCS 1109, Advances in Cryptology, Proceedings of Crypto'96, Springer-Verlag, 1996, pp. 104-113.

[Koc99] : P. Kocher, J. Jaffe, and B. Jun, "*Differential Power Analysis*", CRYPTO '99 Proceedings, Springer–Verlag, 1999, pp. 388-397.

[Kom99] : O. Kömmerling and M. G. Kuhn, "*Design Principles for Tamper-Resistant Smartcard Processors*", presented at USENIX Workshop on Smartcard Technology (Smartcard 99), May 1999, pp. 9-19.

[Kone80] : B. Konemann, J. Mucha, and G. Zwiehoff, "*Built-in Test for Complex Digital Integrated Circuits*", Journal of Solid State Circuits, June 1980, Vol. SC-15, N° 3, pp. 315-318.

[Kris01] : C.V. Krishna, A. Jas and N.A. Touba, "*Test vector encoding using partial LFSR reseeding*", Proceedings. International Test Conference ITC, 2001, pp. 885- 893.

[Kris02] : C.V. Krishna and N.A Touba, "*Reducing test data volume using LFSR reseeding with seed compression*", Proceedings International Test Conference ITC, 2002, pp. 321- 330.

[Lee06] : J. Lee, M. Tehranipoor and J. Plusquellic, "*A Low-Cost Solution for Protecting IPs Against Scan-Based Side-Channel Attacks*", VTS, 2006, pp. 94-100.

[Mas94] : L. Massey, "*SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm*", Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 1-17.

[McC86] : E. Mc CLUSKEY, "*Logic Design Principles*", Prentice–Hall, Englewood Cliffs, New Jersey, 1986.

[Mit04] : S. Mitra and K. S. Kim, "*X-Compact: An efficient response compaction technique*", IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., Mars 2004, vol. 23, no. 3, pp. 421–432.

[Mit04bis] : S. Mitra, S. S. Lumetta, and M. Mitzenmacher, "*X-tolerant signature analysis*", in Proc. ITC, 2004, pp. 432–441.

[Mor06] : A. Moradi, M. T. Manzuri Shalmani and M. Salmasizadeh, "*A Generalized Method of Differential Fault Attack Against AES Cryptosystem*", Proceedings LNCS CHES, 2006, , vol. 4249, pp. 91-100.

[Mur02] : S. Murphy and M. J.B. Robshaw, "*Essential algebraic structure within the AES*", Advances in Cryptology - CRYPTO 2002 (M. Yung, ed.), Lecture Notes in Computer Science, Springer, 2002, vol. 2442, pp. 1–16.

[Naz05] : M. Nazm-Bojnordi, N. Sedaghati-Mokhtari and S. Mehdi Fakhraie, "*A Self-Testing Fully Pipelined Implementation for the Advanced Encryption Standard*", 17th International Conference on Microelectronics (ICM), 2005, pp. 260- 263.

[NIST 800-22] : "*A statistical test suite for random and pseudorandom number generators for cryptographic applications*", NIST Special Publication 800-22 (with revisions dated May 15, 2001).

[Qui00] : J. J. Quisquater and D. Samyde, "*A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions, the SEMA and DEMA methods*", presented at the rump session of EUROCRYPT'2000, May 2000.

[Raj08] : J. Rajski, J. Tyszer, G. Mrugalski, W-T. Cheng, N. Mukherjee and M. Kassab, "*X-Press: Two-Stage X-Tolerant Compactor With Programmable Selector*", IEEE Trans. on CAD of Integrated Circuits and Systems, 2008, Vol 27 no.1, pp. 147-159.

[Rij00] : V. Rijmen, "*Efficient Implementation of the Rijndael SBox*", http://www.esat.kuleuven.ac.be/rijmen/rijndael/sbox.pdf, 2000.

[Riv78] : R. Rivest, A. Shamir and L. Adleman, "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*", Communications of the ACM, 1978, Vol. 21 (2), pp.120–126.

[Rot66] : JP. Roth, "*Diagnosis of Automata Failures: A Calculus and a Method*", IBM Journal Research and Development, Jul. 1966, vol. 10, no. 4, pp. 278-291.

[Sam02] : D. Samyde, S. Skorobogatov, R. Anderson, and J. J. Quisquater, "*On a New Way to read Data from Memory*", presented at First International IEEE Security in Storage Workshop, 2002.

[Sap90] : G. Saporta, "*Probabilités, analyse des données et statistique*", Technip Publishing Company, Paris, 1990.

[Savi85] : J. Savir and W.H. McAnney, "*On the Masking Probability with Ones Counts and Transition Count*", Proc. of International Conference on Computer-Aided Design, 1985, pp. 111-113.

[Sch97] : B. Schneier, "*Cryptographie appliquée : protocoles, algorithmes et codes sources en C*", J. Wiley, 1997.

[Schu99] : A. Schubert and W. Anheier, "*On Random Pattern Testability of Cryptographic VLSI Cores*", Journal of Electronic Testing: Theory and Applications archive, June 2000, Volume 16, Issue 3, pp 185–192.

[Sha48] : C. E. Shannon, "*A mathematical theory of communication*", Bell System Technical Journal, July-October 1948, Vol 27 No4, pp. 379-423, 623-656.

[Sha49] : C. Shannon, "*Communication theory of secrecy systems*", Bell System Technical Journal, 1949, Vol 28 No4, pp. 656-715.

[Shi07] : S. Shioda, "*Some upper and lower bounds on the coupon collector problem*", Journal of Computational and Applied Mathematics, March 2007, Volume 200, Issue 1, pp. 154-167.

[Sko02] : S. Skorobogatov and R. Anderson, "*Optical Fault Induction Attacks*", Cryptographic Hardware and Embedded Systems Workshop (CHES-2002), de Lecture Notes in Computer Science, Springer-Verlag, 2002, Vol. 2523, pp. 2–12.

[Soto00] : J. Soto and L. Bassham, "*Randomness Testing of the Advanced Encryption Standard Finalist Candidates1*", National Institute of Standards and Technology, 2000.

[STM] : http://www.st.com

[Tmax] : www.synopsys.com/products/test/tetramax_ds.html

[Tou01] : N.A. Touba and E.J. McCluskey, "*Bit-fixing in pseudorandom sequences for scan BIST*", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Apr 2001, Vol. 20, N° 4, pp. 545-555.

[Wu04] : K. Wu, R. Karri, G. Kuznetsov and M. Goessel, "*Low Cost Concurrent Error Detection for the Advances Encryption Standard*", Proc. Int'l Test Conference, 2004, pp. 1242-1248.

[Wun96] : H.-J. Wunderlich and G. Kiefer, "*Bit-flipping BIST*", IEEE/ACM International Conference on Computer-Aided Design ICCAD-96, Nov 1996, pp. 337-343.

[Yan04] : B. Yang, K. Wu and R. Karri, "*Scan-based Side-Channel Attack on Dedicated Hardware Implementations on Data Encryption Standard*", International Test Conference (ITC), October 2004, pp. 339-344.

[Yan05] : B. Yang, K. Wu and R. Karri, "*Secure Scan: A Design-for-Test Architecture for Crypto Chips*", Design Automation Conference (DAC), July 2005, pp. 135-140.

[Yan05bis] : B. Yang and R. Karri, "*Crypto BIST: A Built-In Self Test Architecture for Crypto Chips*", 2nd Workshop on fault diagnosis and tolerance in cryptography (FDTC), 2005, pp. 95-108.

[You05] : A. M. Youssef and S. E. Tavares: "*Affine equivalence in the AES round function*". Discrete Applied Mathematics, 2005, 148(2), pp. 161-170.

Publications

Publications

REVUES INTERNATIONALES AVEC COMITE DE LECTURE :

[TVLSI] : G. Di Natale, M. Doulcier, M.-L. Flottes, B. Rouzeyre, "*Self-Test Techniques for Crypto-Devices*". TVLSI: IEEE Transactions on VLSI Systems, accepté.

[JETTA] : G. Di Natale, M. Doulcier, M. L. Flottes, B. Rouzeyre, "*A Reliable Architecture for parallel implementations of Advanced Encryption Standard*", Journal of Electronic Testing (JETTA), soumis.

CONFÉRENCES INTERNATIONALES AVEC ACTES :

[LATW'07] : M. Doulcier, M.-L. Flottes, B. Rouzeyre, "*AES vs LFSR Based Test Pattern Generation: A Comparative Study*". LATW'07: 8th IEEE Latin-American Test Workshop, Cuzco, Peru, 2007.

[DELTA'08] : M. Doulcier, M.-L. Flottes, B. Rouzeyre, "*AES-based BIST: Self-test, Test Pattern Generation and Signature Analysis"*, DELTA'08: 4th IEEE International Symposium on Electronic Design, Test & Applications, Hong-Kong, 2008, Pages : 314-321.

[ETS'08] : G. Di Natale, M. Doulcier, M.-L. Flottes, B. Rouzeyre, "*A Reliable Architecture for the Advanced Encryption Standard"*, ETS'08: European Test Symposium, Italy, 2008, Pages : 13-18.

[WDSN'08] : G. Di Natale, M. Doulcier, M.-L. Flottes, B. Rouzeyre, "*Low Cost Self-Test of Crypto-Devices"*, WDSN'08: 2nd Workshop on Dependable and Secure Nanocomputing, Anchorage : États-Unis d'Amérique, 2008, Pages : 41-46.

CONFÉRENCES INTERNATIONALES SANS ACTES :

[SETS'06] : M. Doulcier, M-L Flottes, B. Rouzeyre, "*Secure Circuit and Test"***,** South European Test Seminar, Autriche, 2006**.**

[SETS'07] : M. Doulcier, M-L Flottes, B. Rouzeyre, "*AES as test pattern generation for secure circuit"***,** South European Test Seminar, Italie, 2007**.**

[CryptArchi'07] : G. Di Natale, M. Doulcier, M.-L. Flottes, B. Rouzeyre, "*Test and Security"*, CryptArchi'07: Cryptographic Architectures Embedded in Reconfigurable Devices, France, 2007.

CONFERENCES NATIONALES AVEC OU SANS ACTES

[GDR'07] : M. Doulcier, M-L Flottes, B. Rouzeyre, "*Utilisation de ressources cryptographiques pour le test des circuits sécurisés"*, GDR SOC-SIP, Paris 2007.

[JNRDM'08] : M. Doulcier, M-L Flottes, B. Rouzeyre, "*L'auto-test d'un cœur de chiffrement AES"*, Journées Nationales du Réseau Doctoral en Microélectronique, Bordeaux, 2008.

Liste des symboles et acronymes

Liste des symboles et acronymes

AES	Advanced Encryption Standard
ATE	Automatic Test Equipment
BILBO	Built-In Logic Block Observer
BIST	Built In Self Test
CAO	Conception Assistée par Ordinateur
CBC	chiffrement avec chaînage de blocs « Cipher Block Chaining »
CFB	chiffrement à rétroaction « Cipher FeedBack »
CMOS	Complementary Metal-Oxide Semiconductor
DEMA	Differential Electro Magnetic Analysis
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DPA	Differential Power Analysis
ECB	carnet de codage électronique « Electronic codeBook »
EEPROM	Electrical Erasable Programable Read Only Memory
FIPS	Federal Information Processing Standards
LFSR	Linear Feedback Shift Registers
MISR	Multiple Input Shift Register
NIST	National Institute of Standard and Technology
OFB	rétroaction de sortie « Output-FeedBack »
PRPG	Pseudo Random Pattern Generator
RAM	Random Access Memory
ROM	Read Only Memory
RSA	Rivest Shamir Adleman
RSMA	Bloc : Register-SubBytes-MixColumns-AddRoundKey
SAFER	Secure And Fast Encryption Routine
SEMA	Simple Electro Magnetic Analysis
SISR	Single Input Shift Register

SPA	Simple Power Analysis
SRSG	Shift Register Sequence Generator
STIL	Standard Test Interface Language
STS	Statistical Test Suite
STUMPS	Self Test Using MISR / Parallel SRSG
TC	Taux de Couverture
TPG	Test Pattern Generator
VLSI	Very Large Scale Integration

Liste des figures

Liste des figures

Liste des tableaux

Liste des tableaux

TEST INTEGRE DE CIRCUITS CRYPTOGRAPHIQUES

RESUME

Parce que les architectures de test classiques visent principalement à accroître la contrôlabilité et l'observabilité des données manipulées par le système matériel, elles sont identifiées comme sources potentielles de manipulations frauduleuses lorsqu'elles sont mises en œuvre dans des systèmes traitant de sécurité numérique. Les dispositifs sécurisés demandent donc de développer des moyens de test adaptés.

Ce rapport de thèse présente des solutions de test pour systèmes intégrés de chiffrement en s'attachant à la fois aux tests exécutés en fin de production ou en maintenance, et aux tests effectués en cours de fonctionnement.

En ce qui concerne les tests exécutés hors fonctionnement normal, l'approche préconisée s'appuie sur un autotest intégré. Il présente les avantages cumulés de limiter l'accès aux moyens de test intégrés au système, il préserve donc la sécurité des données, d'effectuer un test de qualité, il garantit donc un bon fonctionnement du système, et enfin de ne demander que très peu de ressources additionnelles. Profitant des propriétés inhérentes aux algorithmes de chiffrement (diffusion, confusion, itération) et des implantations matérielles qui en découlent (architectures rebouclées), des solutions d'autotest sont proposées pour des cœurs DES et AES. Il est aussi démontré comment les réutiliser pour générer les vecteurs de test d'autres ressources matérielles du système et analyser leurs réponses.

Pour ce qui concerne les tests exécutés en cours de fonctionnement, l'architecture particulière des cœurs de chiffrement est à nouveau mise à profit pour de la détection de fautes en ligne basée sur de la redondance d'information ou de matériel.

MOTS-CLES : test intégré, carte à puce, cryptographie, autotest, test en ligne

BIST OF CRYPTOGRAPHIC CIRCUITS

ABSTRACT

Because the conventional test architectures are mainly designed to increase the controllability and observability of the signals, they are identified as potential sources of attacks when implemented in systems dealing with digital security. It is then necessary to develop appropriate test methods.

This thesis presents test solutions for encryption systems focusing on both tests performed at the end of production or maintenance, and tests carried out during the mission mode.

Regarding off-line tests performed after production or in-situ, the approach relies on an integrated self-test schemes. It presents the combined advantages of limiting the access to internal data, and thus preserves data security, conducting a test of high quality, thus it guarantees the proper system behavior, and finally requiring only very little additional resources. Taking advantage of inherent properties of encryption algorithms (diffusion, confusion, iteration) and their physical implementations (feedback architectures), self-test solutions are proposed for DES and AES cores. It is also demonstrated how such crypto-cores can be used as test resources for other cores in the system.

Regarding the tests performed during the functional mode, the proposed approach allows the detection of faults using different forms of duplication (information or hardware redundancies).

KEYWORDS: Built-In Self Test, smartcard, cryptography, self test, on-line test

DISCIPLINE : Microélectronique

Université de Montpellier II: Sciences et Techniques du Languedoc
LIRMM : Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier
161 Rue Ada - 34392 Montpellier Cedex 5