Étude de la sécurité des T-fonctions

Thomas Peyrin

D.C.S.S.I.

Stage M.P.R.I.

Septembre 2005

Responsable
Guillaume Poupard
D.C.S.S.I.

Superviseur
Frédéric Muller
D.C.S.S.I.

Résumé

Nous étudions dans ce rapport de nouvelles primitives cryptographiques introduites par Klimov et Shamir : les Triangular-fonctions (T-fonctions). Ces fonctions ont de nombreuses applications dans le domaine de la cryptographie symétrique, des stream ciphers aux block ciphers en passant par les fonctions de hachage. Nous rappelons la théorie suivie par Klimov et Shamir et les diverses applications potentielles.

Nous nous attachons tout particulièrement à la première utilisation des T-fonctions : la construction de nouveaux stream ciphers. Nous passons en revue les principaux algorithmes conçus et introduisons une nouvelle méthode d'attaque contre les stream ciphers fondés sur des T-fonctions. Cette attaque générique pouvant être appliquée à de nombreux schémas existants, nous étudions dans quelle mesure elle affecte la sécurité des primitives.

Remerciements

Je tiens à remercier tout d'abord Guillaume Poupard pour m'avoir fait confiance et m'avoir accepté dans le laboratoire de cryptographie de la DCSSI durant mon stage.

Je tiens aussi à remercier Frédéric Muller, mon maître de stage, pour m'avoir guidé tout au long de ces quatre mois.

Enfin, je remercie toute l'équipe et les stagiaires du laboratoire de cryptographie de la DCSSI pour l'excellente ambiance quotidienne.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

La cryptographie peut être divisée hiérarchiquement en plusieurs sous parties. Tout au dessus de la pyramide, les protocoles fournissent des solutions pour des problèmes tels que l'envoi de messages privés, l'organisation d'élections sécurisées, etc. Pour cela, les protocoles peuvent utiliser plusieurs mécanismes cryptographiques comme le chiffrement, la signature, etc. On trouve ensuite les primitives cryptographiques comme un algorithme de chiffrement par blocs (block cipher), un algorithme de chiffrement pas flot (stream cipher), une fonction à sens unique (one way function), etc. Enfin, au niveau le plus bas, les primitives utilisent des briques cryptographiques telles que des boîtes de substitution (S-Box) pour les block ciphers, des registres linéaires à décalage (LFSR) pour les stream ciphers, etc. Ici, nous étudions une famille récente de briques cryptographiques appelées les Triangular-fonctions (T-fonctions) introduites par Klimov et Shamir [13, 14]. Nous nous attacherons tout particulièrement à la sécurité des ces briques cryptographiques pour les stream ciphers.

Tout comme les block ciphers, les stream ciphers sont une famille importante de primitives de chiffrement symétrique. Ils opèrent en générant une longue séquence pseudo-aléatoire à partir d'une clé symétrique courte. Ensuite, un message est chiffré simplement en appliquant un XOR entre la séquence pseudo-aléatoire et le message. Le déchiffrement se réalise grâce à la même opération (voir Figure 1.1). Il faut noter que les attaques considérées contre les stream ciphers sont des attaques à clair connu, c'est à dire que l'on attaque la séquence pseudo-aléatoire elle-même. Ainsi, l'attaquant a gagné s'il réussit à retrouver l'état initial du registre[1]. De plus, pour pouvoir considérer l'algorithme comme sûr, on souhaite que la séquence soit indistinguable d'une séquence réellement aléatoire, en un temps raisonnable. Cependant, même si la sécurité cryptographique est le principal but, le concepteur doit garder à l'esprit l'efficacité de son schéma. En effet, la vitesse est un des principaux avantages des stream ciphers sur les block ciphers.

[1] Il n'est pas toujours nécessaire de retrouver la clé puisque l'attaquant pourra générer la séquence pseudo-aléatoire entière en connaissant seulement l'état initial des registres

De nos jours, construire un stream cipher est assez risqué et l'existence de bons block ciphers tels que l'AES a amené les cryptographes à se poser des questions au sujet du futur des stream ciphers [2, 24]. Cependant, certains domaines particuliers continuent de se montrer très actifs ; par exemple, beaucoup de schémas de stream ciphers très rapides et orientés vers des applications software ont été proposés récemment [26, 7, 6, 3] et des schémas plus légers orientés vers des applications hardware peuvent aussi être utiles, notamment pour les modules à faibles ressources tels que les RFID. Tout récemment, un appel pour des propositions de nouveaux algorithmes de stream ciphers a été lancé par le projet européen ECRYPT [5] et de nombreux nouveaux algorithmes ont été proposés.

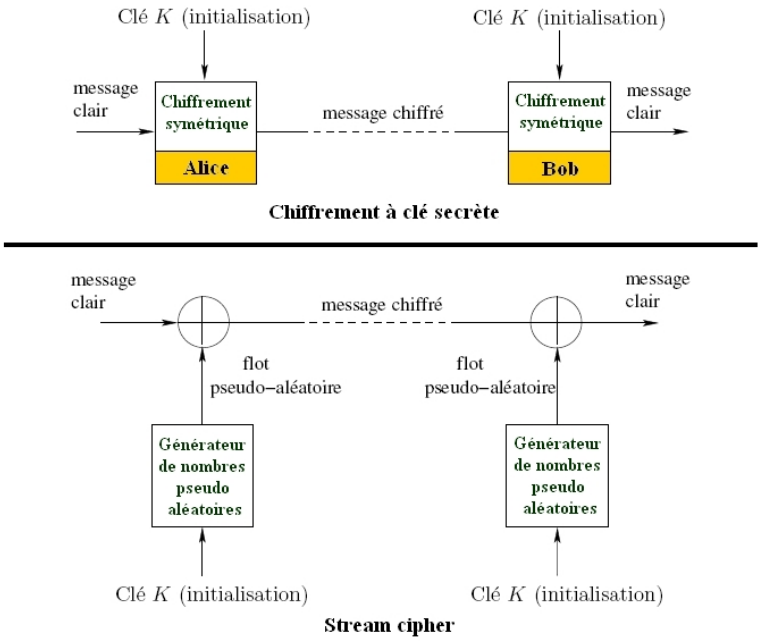


FIG. 1.1 – Schéma d'un stream cipher

Il existe deux approches diamétralement opposées pour concevoir des algorithmes cryptographiques : on peut tout d'abord utiliser des objets très simples avec un comportement très facilement analysable. De nombreux théorèmes sur leurs propriétés cryptographiques seront prouvables, ce qui permet d'éviter des mauvaises surprises, mais la structure mathématique de tels objets peut aussi aider l'attaquant à concevoir des attaques. Inversement, on peut utiliser des primitives assez complexes et difficilement analysables mais qui peuvent présenter des failles difficilement décelables. Nous allons voir comment les T-fonctions comblent cet écart. En effet, ces primitives permettent l'utilisation d'opérations complexes tout en gardant la possibilité de prouver plusieurs théorèmes intéres-

sants sur leurs propriétés cryptographiques.

Dans la lignée de la première approche, de nombreux schémas utilisant des Linear Feedback Shift Registers (LFSR) existent. De telles primitives doivent être combinées avec des fonctions booléennes non linéaires pour casser leur linéarité. A cause de l'apparition de nouvelles attaques (comme les attaques algébriques [4, 1]), de nouvelles primitives ont été introduites pour remplacer les LFSR, comme les T-fonctions. Par définition, ces nouvelles primitives sont calculables des bits de poids faible (LSB) aux bits de poids fort (MSB), et ceci est bénéfique pour l'implémentation, car beaucoup d'opérations disponibles sur les processeurs (comme $+$, $\times$, XOR, OR, AND) sont des T-fonctions. Elles ne sont pas (nécessairement) linéaires et, pour certains choix appropriés, elles peuvent représenter des permutations avec un cycle unique, ce qui peut être très utile pour la conception de stream ciphers. Klimov et Shamir ont de plus étendu leur théorie aux T-fonctions multi-mots et ont apporté plusieurs résultats sur divers domaines tels que les block ciphers et les fonctions de hachage [12, 15, 16, 17].

Les premiers Stream Ciphers Fondés sur des T-fonctions (SCFTF), furent proposés dans les premiers papiers de Klimov et Shamir (famille TF-i). Plus récemment, Hong *et al.* ont trouvé une nouvelle classe de T-fonctions à cycle unique ayant la propriété d'utiliser des S-Box [9, 10], ce qui leur permet de décrire trois nouveaux algorithmes (famille TSC-i). Le premier, TSC-1, a été conçu pour un environnement hardware et le second, TSC-2, peut être implémenté très efficacement en software. Plusieurs attaques contre les SCFTF ont été publiées. A Asiacrypt 2004, Mitra et Sarkar [21] ont décrit une attaque par compromis temps-mémoire qui casse plusieurs algorithmes proposés par Klimov et Shamir. Junod, Künzli et Meier ont récemment trouvé des attaques par distingueur applicables à plusieurs SCFTF [18].

Nous introduisons ici une nouvelle méthode de cryptanalyse pour les SCFTF. Notre idée est de réaliser une attaque statistique en utilisant des approximations linéaires du stream cipher. Tout d'abord, nous linéarisons le comportement de la T-fonction en considérant plusieurs itérations consécutives. Ensuite, nous linéarisons d'autres composants, tels que la fonction de sortie. Enfin, nous décrivons comment retrouver la clé secrète en combinant toutes ces approximations linéaires. Cette attaque générique est fortement liée aux attaques par corrélation contre les stream ciphers fondés sur des LFSR [25, 20] et aussi à la cryptanalyse linéaire contre les block ciphers [19].

Notre attaque s'applique très efficacement à la famille TSC-i. En effet, nous arrivons à casser TSC-1 avec une complexité de $2^{25.4}$ en temps et de $2^{21.4}$ en données. De la même façon, TSC-2 et TSC-3 peuvent être cassés avec une complexité de $2^{44.1}$ en données et $2^{48.1}$ en temps et de 2^{34} en données et 2^{66} en temps respectivement. Nous avons réalisé une implémentation de la première attaque contre TSC-1 pour retrouver totalement la clé secrète initiale. Ces attaques sont les premières attaques contre la famille TSC-i permettant de retrouver la clé et nous pensons de plus que ces failles révèlent quelques points importants à prendre en compte pour la conception de SCFTF. En particulier, l'existence de bonnes approximations linéaires de la T-fonction sur plusieurs itérations doit

être évitée.

Tout d'abord nous verrons quelques propriétés basiques des T-fonctions après une introduction à ce nouveau concept. Ensuite, nous verrons les diverses applications de telles primitives, en insistant sur l'aspect stream cipher. Dans la Section 4.1, nous explicitons notre attaque générique contre les SCFTF. Ensuite, nous décrivons les attaques pratiques contre tous les représentants de la famille TSC-i, en expliquant les divers biais sur les S-Box et l'état interne nous permettant d'attaquer ces stream ciphers. Enfin, avant la conclusion, nous donnerons quelques recommandations pour éviter ce genre d'attaques.

Chapitre 2

Introduction aux T-fonctions

Nous présentons dans ce chapitre une introduction succincte aux T-fonctions et à leurs applications. Le lecteur recherchant une introduction plus complète peut se reporter au prochain chapitre.

Les T-fonctions

Une T-fonction est une fonction faisant correspondre un mot de n bits à un mot de n bits, pour laquelle chaque bit i du mot de la sortie (pour $0 \leq i < n$) ne dépend que des bits $0, 1, \ldots, i$ du mot de l'entrée. Un paramètre est une fonction faisant correspondre un mot de n bits à un mot de n bits, pour laquelle chaque bit i du mot de la sortie (pour $0 \leq i < n$) ne dépend que des bits $0, 1, \ldots, i-1$ du mot de l'entrée. Un exemple est donné Figure 2.1, la fonction considérée étant $f(x) = x + y$ où y est une constante.

Toutes les opérations booléennes, la plupart des opérations arithmétiques disponibles dans les processeurs modernes et leur compositions sont des T-fonctions. Par exemple,

$$x \longrightarrow (x \oplus x^2) + (x \wedge ((3x \lll 5) \vee (x - 1)))$$

où $\oplus$, $\wedge$, $\vee$ représentent respectivement le XOR, le ET, le OU et où $\lll$ représente le décalage des bits vers la gauche, est une T-fonction. Les opérations arithmétiques sont calculées naturellement modulo 2^n.

On peut étendre ce concept à des fonctions sur plusieurs mots : une T-fonction multi-mots est une fonction faisant correspondre k mots de n bits à k mots de n bits, pour laquelle chaque bit i des k mots de la sortie (pour $0 \leq i < n$) ne dépend que des bits $0, 1, \ldots, i$ des k mots de l'entrée. Ainsi,

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \longrightarrow \begin{pmatrix} x_0 & + & x_1 \\ x_1 & \cdot & x_2 \\ x_2 & \vee & x_3 \\ x_3 & \wedge & x_1 \end{pmatrix}$$

est une T-fonction à $k = 4$ mots (voir Figure 2.2).

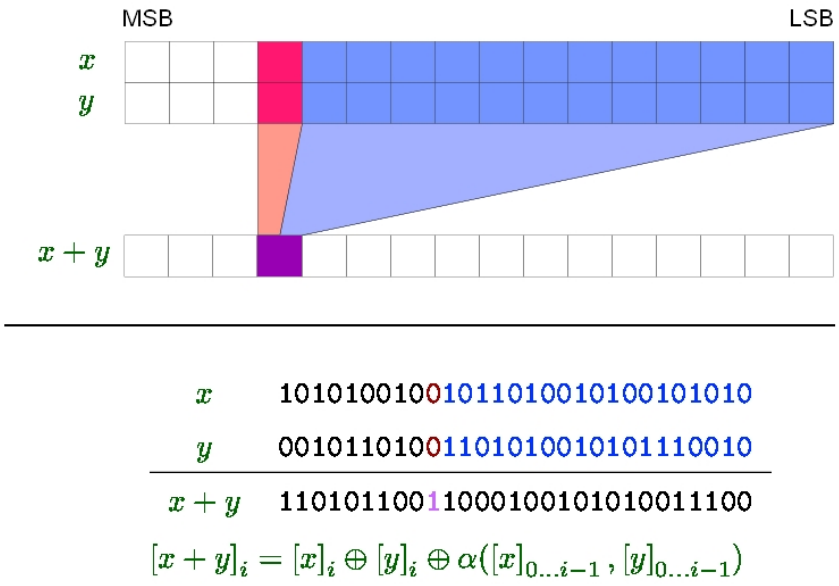


FIG. 2.1 – Exemple d'une T-fonction

L'inversibilité

Une fonction $x \longrightarrow \phi(x)$ est inversible si l'égalité $\phi(x) = \phi(y)$ est vraie si et seulement si $x = y$. Étant donné que nous ne considérons que les fonctions dont l'ensemble de départ et égal à l'ensemble d'arrivée, une fonction sera inversible si elle est injective. Les fonctions inversibles (aussi appelées permutations) ont de nombreuses applications en cryptographie : dans les block ciphers pour permettre un déchiffrement unique, dans les fonctions de hachage et les stream ciphers pour remettre à jour un état interne (pour éviter une perte d'entropie ce qui faciliterait la cryptanalyse). Nous explicitons dans la Section 3.2 une technique permettant de savoir si une T-fonction est inversible ou non.

La propriété de cycle unique

Une fonction $x \longrightarrow \phi(x)$ est une fonction à cycle unique si la séquence produite par ses itérations

$$x_0,\ x_1 = \phi(x_0),\ x_2 = \phi(x_1) = \phi(\phi(x_0)),\ \ldots$$

a une période de taille maximale 2^n pour des mots de n bits. Cette propriété est très importante pour les stream ciphers car on a ainsi la certitude de ne

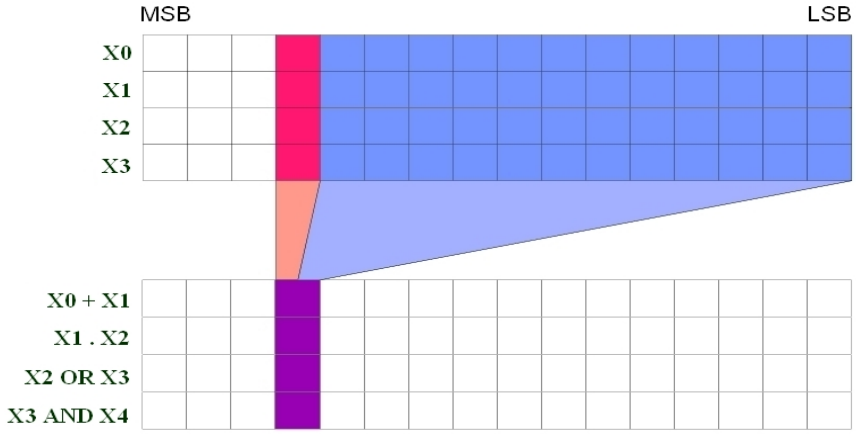


FIG. 2.2 – Exemple d'une T-fonction multi-mots

jamais tomber dans des "petits" cycles, très dangereux pour la sécurité d'un algorithme de chiffrement.

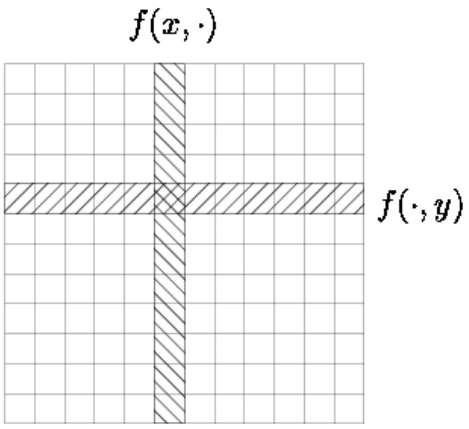
Les carrés latins

Une fonction $f(x, y)$ est un carré latin si $f(x, \cdot)$ et $f(\cdot, y)$ sont toutes deux inversibles (voir Figure 2.3). Les carrés latins sont de bonnes primitives cryptographiques, notamment dans les domaines des fonction de hachage et des block ciphers. Ils permettent d'éviter certains types de collision pour les fonctions de hachage ($f(x, y) = f(x', y)$ ou $f(x, y) = f(x, y')$). Dans le domaine des block ciphers, les multipermutations (un certain type de carrés latins) sont de très bonnes primitives cryptographiques présentant d'excellentes propriétés de diffusion et de confusion [23]. La théorie de Klimov et Shamir permet de construire des carrés latins et des multipermutations.

Les fonctions MDS

Une fonction inversible sur m variables est une fonction MDS si toute variation sur k entrées se propage en une variation sur au moins $(m + 1) - k$ sorties. La cryptanalyse différentielle étant une technique très efficace pour attaquer de nombreux types d'algorithmes cryptographiques, les fonctions MDS sont importantes car elles assurent que n'importe quelle caractéristique différentielle contient au moins $m + 1$ S-Box actives pour chaque paire de tours consécutifs. Grâce aux T-fonctions, Klimov et Shamir nous permettent de construire une grande classe de fonctions MDS non linéaires et non algébriques.

Pour alléger le contenu de ce rapport nous ne nous intéresserons pas aux carrés latins et aux fonctions MDS, nous invitons le lecteur intéressé à se reporter à



Une fonction $f(x, y)$ est un carré latin si chaque ligne et chaque colonne de la matrice définissant ses valeurs est une permutation.

FIG. 2.3 – Carrés Latins

[12].

Les SCFTF

Grâce à la propriété de cycle unique vérifiée par certaines familles de T-fonctions, ces primitives sont assez prometteuses pour la conception de stream ciphers. Intuitivement, nous allons les utiliser comme remplacement des LFSR, ce qui nous permettra de se protéger des attaques algébriques tout en gardant de bonnes propriétés de cycle : nous obtenons des Stream Ciphers Fondés sur des T-fonctions (SCFTF). On utilise un registre remis à jour par une T-fonction sur lequel on applique une fonction de filtre pour obtenir les bits de sortie. Un schéma générique est donné Figure 2.4.

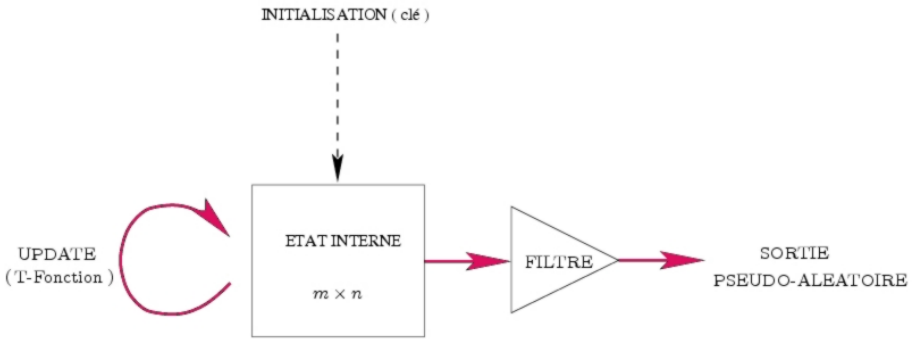


FIG. 2.4 – Schéma de fonctionnement d'un SCFTF

Chapitre 3

Les T-fonctions

Dans ce chapitre nous donnons les principaux résultats de la théorie des T-fonctions. Le lecteur intéressé pourra se reporter à la thèse d'Alexander Klimov [12].

3.1 Notations et définitions

Introduisons tout d'abord quelques notations. On note par $\mathbb{B}$ l'ensemble $\{0,1\}$. $[x]_i$ représente le bit i du mot x (avec $[x]_0$ étant le LSB). Indistinctement, on notera x le vecteur de n bits $([x]_{n-1},\ldots,[x]_0) \in \mathbb{B}^n$ et l'entier correspondant est $x = \sum_{i=0}^{n-1} 2^i [x]_i$. Une collection $\mathbf{x}$ de m mots de n bits peut être décrite comme une matrice $m \times n$. On note $\mathbf{x}_{i-1}$ la i-ème ligne $\mathbf{x}_{i-1,*}$ et on note $[\mathbf{x}]_{j-1}$ la j-ème colonne $\mathbf{x}_{*,j-1}$:

$$\mathbf{x} = ([\mathbf{x}]_{n-1},\ldots,[\mathbf{x}]_0) = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{m-1} \end{pmatrix}$$

Nous n'allons considérer que certains types d'opérations : les opérations primitives.

Définition 1. (Opération primitive) *Soit x et y deux variables de n bits. Une fonction*

$$\phi : \mathbb{B}^{k\times n} \to \mathbb{B}^n$$

est une opération primitive si :

– *$k = 1$ et $\phi(x)$ est une des opérations de*

$$\begin{array}{ll} \text{négation :} & \phi(x) = -x \pmod{2^n}, \\ \text{complémentation :} & [\phi(x)]_i = [\bar{x}]_i, \text{ ou} \end{array}$$

– $k = 2$ *et* $\phi(x, y)$ *est l'une des opérations suivantes :*

$$\begin{array}{ll}
\text{addition :} & \phi(x,y) = x + y \pmod{2^n}, \\
\text{soustraction :} & \phi(x,y) = x - y \pmod{2^n}, \\
\text{multiplication :} & \phi(x,y) = x \cdot y \pmod{2^n}, \\
\text{xor :} & [\phi(x,y)]_i = [x]_i \oplus [y]_i, \\
\text{et :} & [\phi(x,y)]_i = [x]_i \wedge [y]_i, \\
\text{ou :} & [\phi(x,y)]_i = [x]_i \vee [y]_i.
\end{array}$$

On remarque que l'opération de décalage des bits vers la gauche (contrairement à droite) est une opération primitive puisque cela revient à multiplier par 2. Ces opérations primitives représentent toutes les T-fonctions directement accessibles sur un processeur usuel.

Définition 2. (T-fonction) *Une fonction* f *de* $\mathbb{B}^{m\times n}$ *vers* $\mathbb{B}^{m\times n}$ *est une T-fonction si la* k*-ième colonne de la sortie* $[f(\mathbf{x})]_{k-1}$ *ne dépend que des* k *premières colonnes* $[\mathbf{x}]_{k-1}, \ldots, [\mathbf{x}]_0$ *de l'entrée :*

$$\begin{pmatrix} [\mathbf{x}]_0 \\ [\mathbf{x}]_1 \\ \vdots \\ [\mathbf{x}]_{n-1} \end{pmatrix}^T \longrightarrow \begin{pmatrix} f_0([\mathbf{x}]_0) \\ f_1([\mathbf{x}]_0, [\mathbf{x}]_1) \\ \vdots \\ f_{n-1}([\mathbf{x}]_0, \ldots, [\mathbf{x}]_{n-1}) \end{pmatrix}^T . \tag{3.1}$$

On comprend maintenant en regardant la forme de (3.1) le nom de "Fonctions Triangulaires" (T-fonctions). On remarque que toute composition de T-fonctions est aussi une T-fonction. Le lemme suivant nous montre que ces primitives sont en fait des fonctions assez communes.

Lemme 1. *Toute opération primitive est une T-fonction.*

Preuve. Il est trivial que le i-ème bit de $x \circ y$, où $\circ$ est une opération logique bit à bit, ne dépend que des i-èmes bits de x et y. En ce qui concerne les opérations algébriques, nous avons

$$x \circ y = (x \bmod 2^i) \circ (y \bmod 2^i) \pmod{2^i},$$

où $\circ$ représente l'addition, la soustraction ou la multiplication. □

Pour introduire la notion de paramètre, nous avons besoin des fonctions paramétriques :

Définition 3. (Fonction paramétrique) *Une fonction paramétrique est une fonction*

$$g(x_1, \ldots, x_a; \alpha_1, \ldots, \alpha_b)$$

dont les arguments sont séparés entre les entrées (les x_i*) et les paramètres (*α_j*).*

Définition 4. (Paramètre) *Un paramètre est une fonction paramétrique ne dépendant pas de ses entrées.*

Par exemple, $f(x;\alpha,\beta) = \alpha + \beta$ est un paramètre. Dans le cas des T-fonctions, on peut considérer ainsi la i-ème colonne de la sortie comme une fonction dont la i-ème colonne des arguments est considérée comme entrée et les colonnes précédentes comme paramètres. Cela nous donne une nouvelle définition des T-fonctions et des paramètres.

Définition 5. (T-fonction) *Une T-fonction est une fonction*

$$\begin{pmatrix} [\mathbf{x}]_0 \\ [\mathbf{x}]_1 \\ \vdots \\ [\mathbf{x}]_{n-1} \end{pmatrix}^T \longrightarrow \begin{pmatrix} f_0([\mathbf{x}]_0) \\ f_1([\mathbf{x}]_1; [\mathbf{x}]_0) \\ \vdots \\ f_{n-1}([\mathbf{x}]_{n-1}; [\mathbf{x}]_0, \ldots, [\mathbf{x}]_{n-2}) \end{pmatrix}^T$$

où les f_i sont des fonctions paramétriques arbitraires.

Définition 6. (Paramètre) *Une fonction f est un paramètre (noté par une lettre grecque) si chaque f_i est un paramètre*

$$\begin{pmatrix} [\mathbf{x}]_0 \\ [\mathbf{x}]_1 \\ \vdots \\ [\mathbf{x}]_{n-1} \end{pmatrix}^T \longrightarrow \begin{pmatrix} f_0([\mathbf{x}]_0) \\ f_1([\mathbf{x}]_1; [\mathbf{x}]_0) \\ \vdots \\ f_{n-1}([\mathbf{x}]_{n-1}; [\mathbf{x}]_0, \ldots, [\mathbf{x}]_{n-2}) \end{pmatrix}^T = \begin{pmatrix} \alpha_0 \\ \alpha_1([\mathbf{x}]_0) \\ \vdots \\ \alpha_{n-1}([\mathbf{x}]_0, \ldots, [\mathbf{x}]_{n-2}) \end{pmatrix}^T.$$

Beaucoup de propriétés d'une T-fonction f peuvent être déduites des propriétés des fonctions paramétriques f_i.

Définition 7. (Fonction de couche) *Soit $f(x)$ une T-fonction. Une fonction paramétrique*

$$[f(x)]_i = f_i([x]_i; [x]_0, \ldots, [x]_{i-1}).$$

est appelée fonction de couche i de $f(x)$.

Les propriétés de ces fonctions de couche sont souvent indépendantes de la forme exacte des paramètres. Ainsi, grâce au théorème suivant, nous pouvons dériver récursivement à partir d'une représentation compliquée des fonctions de couche, une composition d'opérations primitives (sans se soucier de la forme exacte des paramètres). Ces représentations nous servirons par la suite à démontrer des propriétés sur les T-fonctions.

Théorème 1. *Pour $i > 0$, nous avons les égalités suivantes :*

$$[x \times y]_0 = [x]_0 \wedge [y]_0 \qquad [x \times y]_i = [x]_i \alpha_{[y]_0} \oplus \alpha_{[x]_0}[y]_i \oplus \alpha_{xy}$$

$$[x \pm y]_0 = [x]_0 \oplus [y]_0 \qquad [x \pm y]_i = [x]_i \oplus [y]_i \oplus \alpha_{x \pm y}$$

$$[x \oplus y]_0 = [x]_0 \oplus [y]_0 \qquad [x \oplus y]_i = [x]_i \oplus [y]_i$$

$$[x \wedge y]_0 = [x]_0 \wedge [y]_0 \qquad [x \wedge y]_i = [x]_i \wedge [y]_i$$

$$[x \vee y]_0 = [x]_0 \vee [y]_0 \qquad [x \vee y]_i = [x]_i \vee [y]_i$$

où les α représentent des paramètres.

Preuve. Les preuves des autres égalités étant similaires, nous ne traiterons que l'exemple $[x \times y]_i = [x]_i \alpha_{[y]_0} \oplus \alpha_{[x]_0}[y]_i \oplus \alpha_{xy}$. Nous avons :

$$\begin{array}{rcl} xy \bmod 2^{i+1} & = & (2^i[x]_i + \ldots + 2^0[x]_0)(2^i[y]_i + \ldots + 2^0[y]_0) \bmod 2^{i+1} \\ & = & 2^i([x]_i[y]_0 + [x]_{i-1}[y]_1 + \ldots + [x]_0[y]_i) \\ & + & 2^{i-1}([x]_{i-1}[y]_0 + [x]_{i-2}[y]_1 + \ldots + [x]_0[y]_{i-1}) \\ & + & \ldots + [x]_0[y]_0. \end{array}$$

En considérant la couche i, on note que $[x]_0, \ldots, [x]_{i-1}$ et $[y]_0, \ldots, [y]_{i-1}$ sont des paramètres. Donc

$$xy \bmod 2^{i+1} = 2^i(([x]_i \alpha_{[y]_0} + \alpha_{[x]_0}[y]_i + \alpha') \bmod 2) + \alpha'',$$

où α' et α'' sont des paramètres. $[xy]_i$ étant le MSB de $xy \bmod 2^{i+1}$, on peut conclure que

$$[x \times y]_i = [x]_i \alpha_{[y]_0} \oplus \alpha_{[x]_0}[y]_i \oplus \alpha_{xy}$$

où α_{xy} est un paramètre. □

Pour illustrer, essayons de "dériver" $f(x) = x + (x^2 \vee 5)$:

$$[x + (x^2 \vee 5)]_0 = [x]_0 \oplus [x^2 \vee 5]_0 = [x]_0 \oplus ([x^2]_0 \vee [5]_0) = [x]_0 \oplus 1$$

et pour $i > 0$:

$$\begin{array}{rcl} [x + (x^2 \vee 5)]_i & = & [x]_i \oplus [x^2 \vee 5]_i \oplus \alpha_{x+(x^2 \vee 5)} \\ & = & [x]_i \oplus ([x^2]_i \vee [5]_i) \oplus \alpha_{x+(x^2 \vee 5)} \\ & = & [x]_i \oplus ([x]_i \alpha_{[x]_0} \oplus \alpha_{[x]_0}[x]_i \oplus \alpha_{x^2}) \vee [5]_i \oplus \alpha_{x+(x^2 \vee 5)} \\ & = & [x]_i \oplus \alpha_{x^2} \vee [5]_i \oplus \alpha_{x+(x^2 \vee 5)} = [x]_i \oplus \alpha. \end{array}$$

Avant de commencer à étudier les propriétés des couches, nous devons introduire les T-fonctions multi-mots. En effet, en cryptographie, la taille des mots traités par les microprocesseurs (32 ou 64 bits) est trop petite. Utiliser des T-fonctions sur un seul mot de 128 ou 256 bits serait assez inefficace. On souhaite donc pouvoir utiliser des états internes composés d'une concaténation de l mots de taille standard. Concrètement, nous allons devoir augmenter la largeur de la matrice de n à nl.

3.2 Propriétés des T-fonctions

Ici, nous nous intéresserons à deux propriétés très importantes pour l'aspect cryptographique des T-fonctions : l'inversibilité et la longueur du cycle généré par la T-fonction.

3.2.1 Propriété d'inversibilité

Grâce au Théorème 1, nous pouvons étudier l'inversibilité par couche, en faisant abstraction des dépendances compliquées et inutiles (dans notre cas) des paramètres pour une couche i.

Définition 8. (Application inversible) *Une application $\phi : \mathbb{B}^k \longrightarrow \mathbb{B}^k$ est inversible si l'équation $\phi(x) = \phi(y)$ est vérifiée si et seulement si $x = y$.*

Définition 9. (Fonction Paramétrique Inversible (FPI)) *Une fonction paramétrique inversible est une fonction paramétrique pour laquelle la relation entrée/sortie est inversible pour toute valeur admissible des paramètres :*

$$\forall \alpha, x, y : \quad g(x;\alpha) = g(y;\alpha) \iff x = y.$$

Par exemple, $f(x;\alpha) = \alpha x$ avec $x \in \mathbb{R}$ est une FPI pour tout $\alpha \neq 0$.

Théorème 2. *Une T-fonction est inversible si et seulement si chacune de ses couches est une FPI.*

Preuve. Considérons la Définition 5 d'une T-fonction où les f_i sont des fonctions paramétriques arbitraires. Puisque f_0 est inversible et ne contient aucun paramètre, on peu calculer $[x]_0$ de façon unique. Comme f_1 est inversible et ne contient que des paramètres déjà connus ($[x]_0$), nous pouvons calculer $[x]_1$ de façon unique. On continue de la même manière pour toutes les couches.

Cela prouve donc qu'une T-fonction est inversible si chacune des couches est une FPI. Pour prouver l'autre implication, nous utilisons le lemme suivant :

Lemme 2. *Soit A et B deux ensembles finis, et soit $\phi : A \times B \longrightarrow A \times B$ une application inversible de la forme :*

$$(a, b) \longrightarrow (f(a), g(b;a))$$

où

$$\begin{aligned} f : &\quad A &&\longrightarrow A \\ g : &\quad B \times A &&\longrightarrow B. \end{aligned}$$

Alors f est inversible et $g(b;a)$ est une FPI.

Preuve. Supposons f non inversible, donc $|\{f(a)|a \in A\}| < |A|$. Alors

$$|\{(f(a), \phi(a,b))|a \in A, b \in B\}| < |A| \cdot |B|,$$

ce qui contredit l'inversibilité de ϕ.

Ensuite, supposons que g n'est pas une FPI, alors il existe a, b_1 et b_2 tels que $g(b_1;a) = g(b_2;a)$. Ce qui signifierait que ϕ envoie (a, b_1) et (a, b_2) vers la même sortie et contredit l'hypothèse. □

Étant donné une T-fonction inversible f, on applique le lemme en commençant par le MSB. On représente f comme :

$$(f_{n-1}([x]_{n-1}; [x]_{0,\ldots,n-2}), f'([x]_{0,\ldots,n-2})).$$

Grâce au lemme, on déduit que f_{n-1} est une FPI et que f' est inversible. On continue alors ce processus pour démontrer l'inversibilité de chaque couche. □

En analysant couches par couches, il est donc possible de savoir si une expression définit une T-fonction inversible. De plus, avec la technique de simplification du Théorème 1, on peut facilement analyser ces couches. Pour illustrer, notre précédent exemple nous donnait

$$[x + (x^2 \vee 5)]_0 = [x]_0 \oplus 1$$

et pour $i > 0$

$$[x + (x^2 \vee 5)]_i = [x]_i \oplus \alpha.$$

Comme chaque couche est inversible, nous avons bien obtenu une T-fonction inversible. La technique est la même pour des T-fonctions multivariées.

Après avoir décrit un test d'inversibilité, on peut ensuite se poser le problème de la construction de T-fonctions inversibles. Au lieu de tirer des T-fonctions aléatoires et de tester leur inversibilité, Klimov et Shamir ont mis au point une technique de construction de telles fonctions assez naturelle : on choisit tout d'abord une forme inversible pour une couche i de la de la T-fonction, puis on vérifie (et on corrige si nécessaire) que l'on a toujours la propriété d'inversibilité pour la couche 0. On pourra se reporter à [12] pour une étude détaillée de la construction de T-fonctions inversibles.

3.2.2 Propriété de cycle unique

Dans cette section, nous souhaitons étudier la longueur du cycle d'une permutation définie par une T-fonction. Ici, nous considérons uniquement les T-fonctions dont l'espace d'entrée et celui de sortie sont identiques. Considérons la séquence suivante :

$$\mathbf{x}^{(0)},\ \mathbf{x}^{(1)} = f(\mathbf{x}^{(0)}),\ \mathbf{x}^{(2)} = f(\mathbf{x}^{(1)}),\ \ldots$$

où $x^{(i)} \in \mathbb{B}^{m \times k}$. Puisque $\mathbb{B}^{m \times k}$ est un ensemble fini, la séquence est périodique :

$$\exists s, t : \forall i \geq 0, \mathbf{x}^{(s+i)} = \mathbf{x}^{(s+i+t)}$$

où t est la période et s est le point de départ pour le cycle atteint par $\mathbf{x}^{(0)}$ (on considère t et s les plus petits possibles). Soit le graphe induit sur $\mathbb{B}^k$ par la fonction f. Si f est inversible, tout noeud possède un prédécesseur, en d'autres termes le graphe d'une fonction inversible n'est constitué que de cycles.

Définition 10. (T-fonction à cycle unique) *Une T-fonction dont le graphe induit est isomorphique à un cycle unique est appelée T-fonction à cycle unique.*

Le fait que l'on puisse tester facilement si un LFSR correspond à une application à cycle unique explique en partie leur popularité pour les stream ciphers, malgré les mauvaises propriétés cryptographiques induites par la linéarité de la fonction de transition. De plus, les LFSR ont comme défaut l'existence inévitable d'un point fixe (l'état où tous les bits sont à 0) et les mécanismes permettant à l'initialisation d'éviter de tomber dans cet état peuvent aboutir à des attaques. En choisissant certains types de T-fonctions, on peut obtenir des applications où tous les états font partis d'un cycle unique et maximal, sans point fixe.

Cas univarié

Tout d'abord, Klimov et Shamir ont étudié le cas univarié. Une T-fonction univariée est inversible si et seulement si la couche i est de la forme $[f(x)]_i = [x]_i \oplus \alpha$ (il y a quatre cas possibles dont les deux derniers sont inversibles : 0, 1, x, $x \oplus 1$). On peut montrer récursivement que les seules longueurs de cycle possibles pour une T-fonction sont des puissances de 2, donc la séquence $\{x^{(i+1)} = f(x^{(i)}) \bmod 2^n\}$ a une période de 2^n si et seulement si $x^{(i)} \neq x^{(i+2^{n-1})} \pmod{2^n}$. Puisque pour toute T-fonction on a $x^{(i)} = x^{(i+2^{n-1})} \pmod{2^{n-1}}$, on en déduit que $f(x)$ défini un cycle unique si et seulement si $x^{(i)}$ et $x^{(i+2^{n-1})}$ diffèrent sur leur bit de poids fort :

$$[x^{(i)}]_{n-1} \neq [x^{(i+2^{n-1})}]_{n-1}$$

et on a

$$[x^{(i+2^{n-1})}]_{n-1} = [x^{(i)}]_{n-1} \oplus \bigoplus_{j=0}^{2^{n-1}-1} \alpha(x^{(i+j)}),$$

où α est un paramètre, donc $\alpha(x + 2^{n-1}) = \alpha(x) \pmod{2^n}$. Puisque $\{x^{(i)} \bmod 2^{n-1}\}$ a une période de 2^{n-1}, l'ensemble $\{\alpha(x^{(i+j)})\}_{j=0}^{2^{n-1}}$ est identique à $\{\alpha(j)\}_{j=0}^{2^{n-1}}$. On arrive ainsi à la caractérisation suivante :

Théorème 3. *Une T-fonction $f(x)$ définit un cycle unique modulo 2^n si et seulement si pour tout $i < n$, la i-ème couche de l'application peut être représentée par*

$$[f(x)]_i = [x]_i \oplus \alpha([x]_{0,\ldots,i-1})$$

avec

$$\bigoplus_{j=0}^{2^i-1} \alpha(j) = 1. \tag{3.2}$$

Preuve. Si $f(x)$ définit un cycle unique alors (3.2) est vraie. Pour l'implication opposée, on prouve par induction. Tout d'abord, pour $n = 1$ dans (3.2), on obtient que $[f(x)]_0 = [x]_0 \oplus 1$ et donc $f(x)$ définit un cycle unique modulo 2^1. Ensuite, si on suppose qu'il existe un cycle unique modulo 2^{n-1} alors (3.2) nous donne que $[x^{(i)}]_{n-1} \neq [x^{(i+2^{n-1})}]_{n-1}$ et donc il existe un cycle unique modulo 2^n. □

Ce théorème caractérise complètement les T-fonctions définissant un cycle unique mais son application est délicate. Nous allons donc nous intéresser à un cas particulier de celui-ci. Supposons que $\alpha(x)$ soit un paramètre, soit

$$\alpha(x + 2^{n-1}) = \alpha(x) \pmod{2^n}.$$

On a alors $\alpha(x) = \alpha(x + 2^{n-1}) + 2^n b(x) \pmod{2^{n+1}}$. Ainsi, suivant la valeur de

$$B[\alpha, n] = 2^{-n} \sum_{i=0}^{2^{n-1}-1} (\alpha(i + 2^{n-1}) - \alpha(i)) \pmod 2 = \bigoplus_{i=0}^{2^{n-1}-1} b(i). \tag{3.3}$$

on introduit les notions de paramètres pairs et impairs.

Définition 11. (Paramètre pair) *Un paramètre $\alpha(x)$ est appelé pair si on peut trouver un entier N_0 tel que $B[\alpha, n] = 0$ pour tout $n \geq N_0$.*

Définition 12. (Paramètre impair) *Un paramètre $\alpha(x)$ est appelé impair si on peut trouver un entier N_0 tel que $B[\alpha, n] = 1$ pour tout $n \geq N_0$.*

Par exemple, le paramètre $\alpha(x) = 2x$ est pair pour $n \geq 2$. Inversement, le paramètre définit par $[\alpha(x)]_i = [x]_0 \wedge [x]_1 \wedge \ldots \wedge [x]_{i-1}$ est un paramètre impair. Nous avons déjà vu que dans le cas univarié, une couche d'une T-fonction inversible peut être écrite sous la forme $[f(x)]_i = [x]_i \oplus \alpha$ où α est un paramètre. Grâce au Théorème 1, on s'aperçoit que l'on ne doit traiter que les cas suivants :

$$\begin{aligned} f_1(x) &= x + \alpha_1(x), \\ f_2(x) &= x \oplus \alpha_2(x), \\ f_3(x) &= x \cdot \alpha_3(x) \end{aligned}$$

où $\alpha_1(x), \alpha_2(x), \alpha_3(x)$ sont des paramètres et $[\alpha_3]_0 = 1$ (pour garantir l'inversibilité de f_3). Seules f_1 et f_2 peuvent être retenues car $[f_3(x)]_0 = [x]_0[\alpha_3(x)]_0 = [x]_0$, ce qui implique que f_3 comporte deux cycles modulo 2 et ne peux pas être une T-fonction à cycle unique modulo 2^n. Grâce à la notion de paramètres pairs et impairs, Klimov et Shamir ont pu traiter les cas de f_1 et f_2.

Théorème 4. *Soit N_0 tel que*

$$x \longrightarrow x + \alpha(x) \tag{3.4}$$

définisse un cycle unique modulo 2^{N_0}. L'application (3.4) définit un cycle unique modulo 2^n pour tout n si et seulement si pour tout $n \geq N_0$ la fonction $\alpha(x)$ est un paramètre pair.

Preuve. Voir [12]. □

Théorème 5. *Soit N_0 tel que*

$$x \longrightarrow x \oplus \alpha(x) \tag{3.5}$$

définisse un cycle unique modulo 2^{N_0}. L'application (3.5) définit un cycle unique modulo 2^n pour tout n si et seulement si pour tout $n \geq N_0$ la fonction $\alpha(x)$ est un paramètre impair.

Preuve. Voir [12]. □

Cas multivarié

On s'intéresse maintenant au cas multivarié, beaucoup plus compliqué, pour tenter de construire des T-fonctions multi-mots inversibles et à cycle unique. Pour cela, on définit la transformation suivante :

$$f_i(x_0, \ldots, x_{m-1}) = x_i \oplus (\alpha_i(x_0, \ldots, x_{m-1}) \wedge x_0 \wedge \ldots \wedge x_{i-1}), \tag{3.6}$$

où chaque α_i est un paramètre impair :

$$\bigoplus_{(x_0,\ldots,x_{m-1})=(0,\ldots,0)}^{(2^n-1,\ldots,2^n-1)} [\alpha_i(x_0,\ldots,x_{m-1})]_n = 1$$

avec $[\alpha_i]_0 = 1$.
Sur chaque tranche, cette construction se comporte comme une sorte de compteur. On peut par exemple constater que si on remplace α_i par 1, on retombe sur un compteur.

Théorème 6. *Toute application définie par (3.6) définie un cycle unique de longueur* 2^{nm}.

Preuve. Voir [12]. □

On peut généraliser le précédent théorème.

Théorème 7. *Soit*
- $\alpha_i(x_0,\ldots,x_{m-1})$ *pour* $i = 0,\ldots,m-1$ *des paramètres impairs tels que* $[\alpha_i]_0 = 1$,
- $\beta_0(x_0,\ldots,x_{m-1})$ *un paramètre pair tel que* $[\beta_0]_0 = 0$, *et*
- $\beta_i(x_0,\ldots,x_{m-1})$ *pour* $i = 1,\ldots,m-1$ *des paramètres arbitraires,*

alors l'application suivante définit un cycle unique de longueur 2^{mn} *:*

$$\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{m-1} \end{pmatrix} \longrightarrow \begin{pmatrix} x_0 & \oplus & \alpha_0 & \oplus & \beta_0 \\ x_1 & \oplus & (\alpha_1 \wedge x_0) & \oplus & \beta_1 \\ \vdots & & & & \\ x_{m-1} & \oplus & (\alpha_{m-1} \wedge x_0 \wedge \cdots \wedge x_{m-2}) & \oplus & \beta_{m-1} \end{pmatrix}.$$

Preuve. Voir [12]. □

Après la découverte de cette famille de T-fonctions multi-mots inversibles et à cycle unique, Hong *et al.* ont à leur tour trouvé une nouvelle famille [9, 10]. Pour cela, ils ont étudié le remplacement du système de compteurs par tranche de Klimov et Shamir par une fonction plus générale. Au lieu d'ajouter 1 à certaines tranches, suivant la valeur du paramètre, nous pouvons définir, grâce à une S-Box de m bits vers m bits, $S : \{0,1\}^m \mapsto \{0,1\}^m$, la fonction :

$$\begin{cases} \mathbf{S} : (\{0,1\}^n)^m & \longrightarrow & (\{0,1\}^n)^m \\ \mathbf{x} & \longmapsto & \mathbf{S}(\mathbf{x}), \text{ avec } [\mathbf{S}(\mathbf{x})]_i = S([\mathbf{x}]_i) \end{cases}$$

Une S-Box est à **cycle unique** si, à partir d'un point quelconque, l'itération de S passe par tous les éléments possibles de $\{0,1\}^m$ avant de revenir au point initial. Notons que $\mathbf{S}$ ne définit par forcément une T-fonction à cycle unique même si on choisit S à cycle unique. Pour décrire la nouvelle famille, il nous

faut introduire quelques opérations logiques. Soit $\mathbf{x} = (x_k)_{k=0}^{m-1}$ et $\mathbf{y} = (y_k)_{k=0}^{m-1}$ deux multi-mots, on définit $\mathbf{x} \oplus \mathbf{y}$ en posant

$$\mathbf{x} \oplus \mathbf{y} = (x_k \oplus y_k)_{k=0}^{m-1}.$$

De plus pour le mot α, on définit le multi-mot

$$\alpha \cdot \mathbf{x} = (\alpha \wedge x_k)_{k=0}^{m-1}.$$

On notera de plus $\sim \alpha$ le complémentaire bit à bit de α. On peut maintenant introduire cette nouvelle famille de T-fonctions à cycle unique.

Théorème 8. *Soit S une S-Box à cycle unique et soit α un paramètre impair. Si S^o est une puissance impaire de S et S^e est une puissance paire de S, la fonction*

$$T(\mathbf{x}) = (\alpha(\mathbf{x}) \cdot \mathbf{S^o}(\mathbf{x})) \oplus (\sim \alpha(\mathbf{x}) \cdot \mathbf{S^e}(\mathbf{x}))$$

définit une T-fonction à cycle unique.

En prenant 0 comme puissance paire et 1 comme puissance impaire, on a le corollaire suivant :

Corollaire 1. *Étant donné une S-Box S à cycle unique et un paramètre impair α, la fonction suivante définit une T-fonction à cycle unique :*

$$\mathbf{x} \longmapsto \mathbf{x} \oplus (\alpha(\mathbf{x}) \cdot (\mathbf{x} \oplus \mathbf{S}(\mathbf{x}))).$$

Nous verrons dans la prochaine section comment ces deux familles de T-fonctions ont été utilisées pour construire des stream ciphers fondés sur des T-fonctions multi-mots.

3.3 Les Stream Ciphers Fondés sur des T-fonctions (SCFTF)

Ici, nous donnons les spécifications des deux familles de SCFTF : TF-i et TSC-i. Ces stream ciphers sont construits de la même façon que ceux fondés sur des LFSR, la T-fonction jouant le rôle de fonction de mise à jour du registre (voir Figure 2.4).

3.3.1 La famille TF-i

Les stream ciphers TF-i ont été la première application cryptographique proposés pour les T-fonctions. Les algorithmes ainsi proposés ont de très bonnes performances pour les applications software.

TF-0

Le premier algorithme décrit par Klimov et Shamir est surtout destiné à la génération d'aléa (en remplacement des générateurs congruentiels). Il utilise la T-fonction univariée non-linéaire à cycle unique la plus simple possible.

On part d'un état x de 64 bits et on utilise la T-fonction :

$$T(x) = x + (x^2 \vee C)$$

où C est une constante telle que $C < 2^{32}$ et $C \bmod 8 = 5$ ou 7, par exemple $C = 5$. Ensuite, on sort les $k = 1$, $k = 8$, $k = 16$ ou $k = 32$ bits de poids fort de x suivant la rapidité d'exécution souhaitée.

Ce générateur n'est pas sûr car il est vulnérable aux attaques par force brute en 2^{64-k} et à d'autres attaques plus sophistiquées. Il est donc recommandé de ne pas l'utiliser pour des applications cryptographiques.

TF-1

TF-1 est le premier SCFTF réellement proposé. Il vise une sécurité de 2^{128} et garantit une période de 2^{256}. On utilise $m = 4$ mots x_0, x_1, x_2, x_3 chacun de $n = 64$ bits. L'état interne $\mathbf{x}$ comporte ainsi 256 bits.

Grâce à leur théorie sur les T-fonctions, Klimov et Shamir ont réussi à construire une T-fonction multi-mots inversible et à cycle unique. Soit C_0 un nombre impair, $C_1 = 0\text{x}12481248$ et $C_3 = 0\text{x}48124812$. On pose $a_0 = x_0$ et $a_{i+1} = a_i \wedge x_{i+1}$ pour $i = 0, 1, 2$. On a de plus $\alpha = \alpha(\mathbf{x}) = (a_3 + C_0) \oplus a_3$. On utilise alors la T-fonction suivante :

$$\mathbf{T}\begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \longmapsto \begin{pmatrix} x_3 \oplus (\alpha \wedge a_2) \oplus (2x_0(x_1 \vee C_1)) \\ x_2 \oplus (\alpha \wedge a_1) \oplus (2x_0(x_3 \vee C_3)) \\ x_1 \oplus (\alpha \wedge a_0) \oplus (2x_2(x_3 \vee C_3)) \\ x_0 \oplus \alpha \oplus (2x_2(x_1 \vee C_1)) \end{pmatrix} \tag{3.7}$$

Pour motiver les chercheurs à éprouver leur nouveau stream cipher, Klimov et Shamir décidèrent d'utiliser une fonction de sortie extrêmement simple : on produit 128 bits à chaque itération en prenant les 32 bits de poids fort des 4 mots du registre. Cette fonction de sortie étant trop simple, Mitra et Sarkar[21] ont trouvé une attaque en compromis temps-mémoire contre TF-1 ayant une complexité de 2^{40} en temps et nécessitant au plus 5 blocs de 128 bits de la sortie.

TF-2

TF-2 fut conçu en même temps que TF-1, l'idée étant d'utiliser une T-fonction différente. Ici aussi on vise une sécurité de 2^{128} avec une période garantie de 2^{256}. On utilise toujours $m = 4$ mots x_0, x_1, x_2, x_3 chacun de $n = 64$ bits pour un état interne $\mathbf{x}$ de 256 bits.

Soit $M = 1 \ldots 1110_2$. On garde $a_0 = x_0$ et $a_{i+1} = a_i \wedge x_{i+1}$ pour $i = 0, 1, 2$ mais on a maintenant $\alpha = \alpha(\mathbf{x}) = (a_3 + 1) \oplus a_3$. On utilise alors la T-fonction suivante :

$$\mathbf{T}\begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \longmapsto \begin{pmatrix} x_3 & \oplus & (& \alpha & \wedge a_2) & \oplus x_0^2 \wedge M \\ x_2 & \oplus & (& \alpha & \wedge a_1) & \oplus x_3^2 \wedge M \\ x_1 & \oplus & (& \alpha & \wedge a_0) & \oplus x_2^2 \wedge M \\ x_0 & \oplus & & \alpha & & \oplus x_1^2 \wedge M \end{pmatrix} \tag{3.8}$$

La fonction de sortie est la même que celle de TF-1, donc l'attaque de Mitra et Sarkar[21] s'applique encore avec cette fois une complexité de 2^{21} en temps et nécessitant au plus 5 blocs de 128 bits de la sortie..

TF-3

TF-3 a été conçu suite à l'attaque contre TF-1 pour cause de fonction de sortie trop faible. La T-fonction utilisée est donc la même, seule la fonction de filtre change, on utilise maintenant :

$$f(x_0, x_1, x_2, x_3) = ((x_0 + x_2)_{>>>32})(((x_1 + x_3)_{>>>32}) \vee 1).$$

où $>>>$ représente le décalage des bits vers la droite. La sortie est donc non-linéaire et dépend de tout l'état interne. Elle n'est pas une T-fonction elle-même mais reste efficace. Le décalage sert à éviter l'utilisation des bits de poids faible qui peuvent être mieux décrits linéairement que ceux de poids fort. Le $\vee 1$ permet de garder uniformes tous les bits du produit.

Une optimisation est possible en utilisant certaines opérations sur les nouveaux processeurs de 64 bits et en traitant deux registres en même temps pour augmenter le débit du stream cipher.

3.3.2 La famille TSC-i

Hong *et al.* utilisent naturellement le Théorème 8 et le Corollaire 1 pour leur nouveau schéma de stream ciphers. Pour éviter de révéler trop d'information sur l'état interne et pour ainsi éviter le même type d'attaque que celle de Mitra et Sarkar contre TF-1, les stream ciphers TSC-i utilisent une fonction de filtre pour la sortie (un peu comme TF-3).

TSC-1

Le premier algorithme [9] s'appelle TSC-1 et est directement basé sur le Théorème 8.

Pour la T-fonction, les concepteurs ont choisi $m = 4$ mots x_0, x_1, x_2, x_3 chacun de $n = 32$ bits. L'état interne $\mathbf{x}$ comporte ainsi 128 bits. On définit ensuite une 4×4 S-Box S_1 à cycle unique :

$$S_1[16] = \{3, 5, 9, 13, 1, 6, 11, 15, 4, 0, 8, 14, 10, 7, 2, 12\}; \tag{3.9}$$

On définit de plus un paramètre impair :

$$\alpha(\mathbf{x}) = (p + C) \oplus p \oplus 2s, \tag{3.10}$$

où C = 0x12488421, $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ et $s = x_0 + x_1 + x_2 + x_3$. Toutes les additions sont réalisées modulo 2^{32}. La constante C a une densité de bits à 1 supérieure pour les bits de poids fort pour ainsi rapidement sortir de l'état particulier où tous les registres seraient à zéro. Nous pouvons maintenant utiliser le Théorème 8 avec $S^o = S_1$ et $S^e = S_1^2$. La T-fonction suivante est donc single-cycle :

$$T(\mathbf{x}) = (\alpha(\mathbf{x}) \cdot S_1(\mathbf{x})) \oplus (\sim \alpha(\mathbf{x}) \cdot S_1^2(\mathbf{x})). \tag{3.11}$$

Il faut encore définir la fonction de filtre qui produit 32 bits de sortie après une application de $\mathbf{T}$:

$$f(\mathbf{x}) = (x_{0 \lll 9} + x_1)_{\lll 15} + (x_{2 \lll 7} + x_3), \tag{3.12}$$

où le symbole $\lll$ représente la rotation gauche et les additions sont encore à réaliser modulo 2^{32}.

On sait déjà que la période de l'état interne de TSC-1 est de 2^{128} mais il est possible de prouver que la période de la sortie de ce stream cipher est aussi de 2^{128}. De manière générale, pour une T-fonction à cycle unique, on peut toujours montrer qu'au moins un bit de l'état aura une période égale à celle de la T-fonction. En complétant la T-fonction par un filtre moyennement compliqué, on a de fortes chances de retrouver cette propriété pour la sortie du stream cipher.

Notons que TSC-1 vise un niveau de sécurité de 96 bits. En effet, deviner trois des quatre registres et regarder la sortie détermine uniquement le dernier registre.

En plus d'être à cycle unique, la S-Box S_1 vérifie les conditions suivantes :

1. Pour une application de S_1 à tout mot x, chacun des 4 bits de x a une probabilité de changer égale à $\frac{1}{2}$,
2. Ceci est vrai aussi pour toutes les puissances S_1^i telles que $i \bmod 16 \neq \{0, 4, 8, 12\}$.

En termes plus exacts, on peut reformuler la première condition :

$$\forall i, \sharp\{0 \leq t \leq 16 | \text{ le } i\text{-ème bit de } t \oplus S(t) \text{ est } 1\} = 8.$$

Les rotations dans la fonction de filtre ont deux buts. Tout d'abord, s'assurer que la sortie d'une même S-Box (les bits d'une même tranche de l'état) ne contribue pas directement au même bit de sortie. On désire en effet que les contributions à un bit de sortie proviennent de bits changeants indépendamment les uns des autres. La deuxième raison est que l'on souhaite éviter la possibilité de relier une partie de la sortie avec une partie de l'état interne, compliquant ainsi les attaques par corrélation (on force ainsi l'attaquant à devoir deviner une grande partie des registres pour prédire l'évolution de l'état).

Des tests [22] vérifient les bonnes propriétés statistiques de TSC-1. De plus, les S-Box étant non linéaires, les attaques algébriques sont écartées.
On peut noter que la famille TF-i de Klimov et Shamir est très orientée software (avec l'utilisation de multiplications entières), tandis que la famille TSC-i utilise des S-Box. En particulier, TSC-1 serait mieux adapté pour des besoins de stream ciphers orientés hardware.

TSC-2

Le deuxième algorithme [9], TSC-2, est basé sur le Corollaire 1.

Tout comme TSC-1, les concepteurs ont choisi $m = 4$ mots, chacun de $n = 32$ bits, pour un état interne de 128 bits. On utilise la S-Box à cycle unique suivante :

$$S_2[16] = \{5, 2, 11, 12, 13, 4, 3, 14, 15, 8, 1, 6, 7, 10, 9, 0\}; \tag{3.13}$$

et on définit le paramètre impair suivant :

$$\alpha_2(\mathbf{x}) = (p + 1) \oplus p \oplus 2s. \tag{3.14}$$

où $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ et $s = x_0 + x_1 + x_2 + x_3$. Grâce au Corollaire 1 avec S_2, la T-fonction suivante est prouvée à cycle unique :

$$\mathbf{T}(\mathbf{x}) = \mathbf{x} \oplus (\alpha_2(\mathbf{x}) \cdot (\mathbf{x} \oplus S_2(\mathbf{x}))). \tag{3.15}$$

Enfin, après application de $\mathbf{T}$, les 32 bits de sortie sont obtenus par la fonction de filtre suivante :

$$f_2(\mathbf{x}) = (x_{0 \lll 11} + x_1)_{\lll 14} + (x_{0 \lll 13} + x_2)_{\lll 22} + (x_{0 \lll 12} + x_3) \tag{3.16}$$

avec les additions à réaliser modulo 2^{32}.

Tout comme pour TSC-1, il est prouvé que la période de TSC-2 est de 2^{128}. Ce stream cipher vise lui aussi un niveau de sécurité de 96 bits. Les mêmes arguments de sécurité que pour TSC-1 sont valables pour TSC-2. La seule différence étant les probabilités de changement des bits après application de S_2. Avec une T-fonction construite suivant le modèle du Corollaire 1, il est difficile d'obtenir des probabilités de changement des bits des registres équilibrées. Il est facile d'obtenir une attaque par distingueur si cette caractéristique arrive jusqu'au flot de sortie. Pour éviter cela, les concepteurs ont utilisé une S-Box S_2 à cycle unique telle que :

1. Pour une application de S_2 à tout mot x, le LSB de x est changé avec probabilité 1.

Pour savoir comment est présente cette caractéristique pour $\mathbf{T}$ elle même, il nous faut savoir combien de fois S_2 est appliquée à chaque tranche de l'état.

Lemme 3. *Le paramètre impair (3.14) satisfait*

$$\left| \frac{1}{2} - \Pr_{\mathbf{x}}([\alpha(\mathbf{x})]_i = 1) \right| = \frac{1}{2^{4i}}$$

pour tout $i > 0$.

Preuve. Voir [9]. □

Donc, à part pour les bits de poids faible, α est assez uniformément distribué. On conclut donc que les bits du registre x_0 ont une probabilité de changer proche de $\frac{1}{2}$, à part pour les bits de poids faible. Chaque addition dans la fonction de filtre compte forcément x_0 dans un de ces termes, ce qui permet d'avoir à la sortie de bit dont la probabilité de changer est proche de $\frac{1}{2}$. Ceci nous évite donc une attaque par distingueur utilisant cette caractéristique.

TSC-3

Le troisième algorithme [10], TSC-3, est une amélioration des deux précédents stream ciphers de la même famille, prenant en compte les attaques par distingueur de Künzli *et al.*. TSC-3 a été proposé au projet ECRYPT [5].

Ici, les concepteurs ont choisi $m = 4$ mots, chacun de $n = 40$ bits, pour un état interne de 160 bits. On utilise la même S-Box à cycle unique que TSC-1 :

$$S_3[16] = S_1[16] = \{3, 5, 9, 13, 1, 6, 11, 15, 4, 0, 8, 14, 10, 7, 2, 12\}; \tag{3.17}$$

et on définit les paramètres :

$$\begin{aligned} p(\mathbf{x}) &= x_0 \wedge x_1 \wedge x_2 \wedge x_3 \\ s(\mathbf{x}) &= p(\mathbf{x}) \oplus (p(\mathbf{x}) + C) \\ e_1(\mathbf{x}) &= (x_0 + x_1)_{\ll 1} \oplus (x_2 + x_3)_{\ll 8} \\ e_0(\mathbf{x}) &= (x_0 + x_1)_{\ll 8} \oplus (x_2 + x_3)_{\ll 1} \\ f_1(\mathbf{x}) &= s(\mathbf{x}) \oplus e_1(\mathbf{x}) \\ f_0(\mathbf{x}) &= s(\mathbf{x}) \oplus e_0(\mathbf{x}) \end{aligned} \tag{3.18}$$

où les opérations logiques s'opèrent sur des mots de 40 bits et les additions sont réalisées modulo 2^{40}. $\ll$ dénote le décalage vers la gauche. Enfin, on définit le paramètre multi-mots comme suit :

$$\alpha(\mathbf{x}) = (f_k(\mathbf{x}))_{k=0}^{1}$$

ce qui, par tranche, peut être vu comme une entier compris entre 0 et 3 :

$$[\alpha(\mathbf{x})]_i = 2 \cdot [f_1(\mathbf{x})]_i + [f_0(\mathbf{x})]_i.$$

La T-fonction suivante, définie tranches par tranches, est ainsi prouvée à cycle

unique :

$$[T(\mathbf{x})]_i = \begin{cases} S_3([\mathbf{x}]_i) \text{ si } [\alpha(\mathbf{x})]_i = 3 \\ S_3^2([\mathbf{x}]_i) \text{ si } [\alpha(\mathbf{x})]_i = 2 \\ S_3^5([\mathbf{x}]_i) \text{ si } [\alpha(\mathbf{x})]_i = 1 \\ S_3^6([\mathbf{x}]_i) \text{ si } [\alpha(\mathbf{x})]_i = 0 \end{cases} \tag{3.19}$$

Pour la fonction de sortie, il nous faut tout d'abord définir $\mathbf{y} = (y_k)_{k=0}^3$ comme étant les 32 bits de poids fort des 4 registres de $\mathbf{x}$ (c.à.d. $[y_k]_i = [x_k]_{i+8}$ pour $i \in [0, 31]$). Le processus se déroule comme suit : après application de $\mathbf{T}$, on mélange les 4 registres y_k selon les LSB des 4 registres x_k :

- on échange le contenu de y_0 et de y_1 si $[x_0]_0 = 1$,
- on échange le contenu de y_2 et de y_3 si $[x_2]_0 = 1$,
- on échange le contenu de y_1 et de y_2 si $[x_1]_0 = 1$,
- on échange le contenu de y_0 et de y_3 si $[x_3]_0 = 1$.

On applique donc de façon cyclique 16 fonctions de sortie différentes (le cycle de la première couche de bit des registres étant égal à 16). Ensuite, les 32 bits de sortie sont obtenus par la fonction de filtre suivante :

$$f_3(\mathbf{y}) = (y_{0 \lll 7} + y_{1 \ggg 2}) \lll 8 + (y_{2 \lll 7} + y_3) \ggg 9 \tag{3.20}$$

avec les additions réalisées modulo 2^{32}.

Dans TSC-1 et TSC-2, l'état initial est lui même la clé secrète. TSC-3 comporte donc une autre particularité : un remplissage de l'état initial des registres grâce à la clé secrète et un IV a été défini (pour satisfaire aux conditions des propositions de stream ciphers ECRYPT). Nous ne nous attarderons pas sur ce point étant donné que cela n'a aucune incidence sur nos attaques. En effet, la clé secrète peut être retrouvée très facilement à partir de l'état initial car la fonction d'initialisation est inversible. Le processus complet est décrit dans [10].

Il est possible de prouver que TSC-3 a une période de 2^{160}. Il vise une sécurité de 2^{80} bits. Les tests statistiques [22] donnent de bons résultats et la non linéarité de la T-fonction utilisée permet de se prémunir des attaques algébriques. Les rotations dans la fonction de sortie ont le même rôle que dans les autres stream ciphers de la famille TSC-i et la S-Box utilisée étant la même que dans TSC-1, les mêmes conclusions concernant la sécurité sont valides. Une attention particulière a été portée aux attaques par distingueur de Künzli *et al.* Tout d'abord, les biais exploités dans ces attaques ont été diminués en utilisant une T-fonction plus complexe que pour TSC-1 et TSC-2. Ensuite, la fonction de sortie utilise cycliquement 16 fonctions de filtre différentes, ce qui contraint l'attaquant à utiliser beaucoup plus de données pour arriver à distinguer la sortie d'une séquence aléatoire.

Chapitre 4

Cryptanalyse des T-fonctions

4.1 Cryptanalyse linéaire contre les SCFTF

4.1.1 Contexte

Les attaques utilisant des approximations linéaires ont de nombreuses applications en cryptanalyse. Par exemple, l'attaque de Matsui est la meilleure cryptanalyse connue du DES [19] et plus généralement, la cryptanalyse linéaire a de nombreuses applications pour les block ciphers. Dans le domaine des stream ciphers, des attaques assez classiques utilisant des approximations linéaires ont été développées pour les schémas fondés sur des LFSR et sont connues sous le nom d'**attaques par corrélation** [25, 20]. De plus, quelques attaques linéaires ont été proposées contre d'autres stream ciphers, par exemple l'attaque de Golic contre RC4 [8].

Dans le cas des SCFTF, l'idée d'utiliser des approximations linéaires fut d'abord introduite par Junod, Künzli et Meier. Durant la rump session de Fast Software Encryption 2005 (FSE'05), ils ont présenté une attaque par distingueur contre TSC-1 nécessitant à peu près 2^{22} blocs de sortie connus [11]. Cette idée fut ensuite développée dans [18].

4.1.2 Un cadre générique d'attaque contre les SCFTF

L'attaque que nous proposons est composée de trois étapes :

1. trouver une **approximation linéaire de la T-fonction**. Ceci nous donne une relation probabiliste entre des bits de l'état interne du stream cipher à différents moments.
2. trouver une **approximation linéaire de la fonction de sortie**. Ceci nous donne une relation probabiliste entre les bits de sortie et ceux de l'état interne.
3. **combiner les deux approximations**. Une possibilité serait de trouver des relations ne faisant intervenir que des bits de sortie pour obtenir un

distingueur. Mais une idée plus intéressante est de deviner quelques bits de l'état interne pour éliminer quelques termes dans les approximations et ainsi **augmenter le biais**.

L'idée générale de cette attaque est de retirer la non-linéarité induite par la T-fonction. Les deux premières étapes sont toujours possibles dans une certaine mesure mais il peut être dur de combiner les approximations obtenues à chaque étape.

Formellement, soit $[x_j]_i^t$ représentant la valeur du bit i du registre j à l'instant t. Dans la première étape, on cherche des équations de la forme :

$$\Pr[[x_j]_i^t \oplus [x_{j'}]_{i'}^{t+\delta} = 1] = \frac{1}{2}(1+\epsilon) \tag{4.1}$$

pour un certain δ et avec $|\epsilon|$ aussi grand que possible (ce que nous noterons par la suite $|\epsilon| \gg 0$). Le plus souvent, on est amené à considérer le cas où $(i,j) = (i',j')$. Cela peut être obtenu grâce à une analyse générale de la fonction de remise à jour de l'état interne.

La deuxième étape est similaire aux problèmes rencontrés avec les stream ciphers utilisant des LFSR filtrés par une fonction de sortie. Il est bien connu que la transformée de Fourier rapide peut être utilisée pour trouver de bonnes approximations linéaires. Dans le cas de filtres plus complexes, des astuces supplémentaires sont requises. Par exemple, une idée simple serait de regarder les LSB de la sortie.

Durant la troisième étape, on essaie de combiner les deux approximations et de monter des attaques par distingueur ou des attaques retrouvant la clé. Par exemple, supposons que nous avons trouvé une relation linéaire probabiliste entre plusieurs bits de l'état interne $[x_j]_i^t$ et plusieurs bits du flot de sortie $[s]_k^t$, à l'instant t. Alors, il est possible de combiner cette relation avec la première étape pour obtenir :

$$\Pr[\bigoplus_{i,j}([x_j]_i^t \oplus [x_j]_i^{t+\delta}) \bigoplus_k ([s]_k^t \oplus [s]_k^{t+\delta}) = 1] = \frac{1}{2}(1+\epsilon) \tag{4.2}$$

On espère que $|\epsilon| \gg 0$. Alors, on peut deviner une portion de la clé pour prédire quelques $[x_j]_i$. On augmente ainsi $|\epsilon|$ et on peut alors utiliser (4.2) pour trouver le bon candidat. La structure des T-fonctions est potentiellement utile. Si on devine les i LSB de chaque registre de l'état initial, il nous est possible de prédire ces i LSB à chaque instant à cause de la structure triangulaire des T-fonctions.

4.2 Le cas TSC-1

Dans cette partie, nous appliquons notre attaque générique à TSC-1 et nous exhibons une attaque efficace retrouvant la clé. Nous expliquons cette attaque en suivant les trois étapes de notre cadre d'attaque générique.

4.2.1 Première étape

Nous devons ici approximer le comportement de la fonction de mise à jour de l'état interne entre deux instants t et $t+\delta$. Expérimentalement, Künzli *et al.* ont trouvé que $\delta = 8$ ou $\delta = 11$ seraient de bonnes valeurs pour trouver des biais sur TSC-1. Cependant, $\delta = 3$ est en fait un meilleur choix.

Pour expliquer pourquoi, regardons de plus près la fonction de mise à jour (3.11). On peut observer qu'elle utilise un paramètre $\alpha(\mathbf{x})$ et la S-box S_1. En se focalisant seulement sur la i-ème couche, on obtient

$$[\mathbf{T}(\mathbf{x})]_i = \begin{cases} S_1^2([\mathbf{x}]_i) & \text{si } [\alpha(\mathbf{x})]_i = 0 \\ \\ S_1([\mathbf{x}]_i) & \text{si } [\alpha(\mathbf{x})]_i = 1 \end{cases} \tag{4.3}$$

Comme nous l'avons déjà vu, les concepteurs ont choisi S_1 telle que :

1. Pour une application de S_1 à tout mot x, chacun des 4 bits de x a une probabilité de changer égale à $\frac{1}{2}$,
2. Ceci est vrai aussi pour toutes les puissances S_1^i telles que $i \bmod 16 \neq \{0, 4, 8, 12\}$.

Nous faisons l'hypothèse que le paramètre comme étant uniformément distribué :

$$\Pr[[\alpha(\mathbf{x})]_i = 0] = \Pr[[\alpha(\mathbf{x})]_i = 1] = \frac{1}{2} \text{ avec } i \neq 0,$$

ce qui a été vérifié expérimentalement. On construit alors un arbre binaire pour décrire la mise à jour de la i-ème couche (voir Figure 4.1). Cet arbre commence par une valeur inconnue a de 4 bits. Chaque branche correspond à une valeur de $[\alpha(\mathbf{x})]_i$, soit 0 et 1.

Soit K_i^j le nombre d'occurrences de S_1^i à la profondeur j de l'arbre. Les coefficients K_i^j sont calculés par la formule :

$$K_i^j = \binom{j}{i-j} \text{ avec } i \geq j. \tag{4.4}$$

En utilisant ces coefficients, il est facile de calculer la probabilité pour chaque bit de la i-ème couche de changer après j itérations. Nous souhaitons que les puissances "faibles" de la S-Box (*c.à.d.* $S_1^0, S_1^4, S_1^8, S_1^{12}$) apparaissent avec des coefficients importants[1]. En regardant l'arbre de la Figure 4.1, il nous est possible de calculer théoriquement la probabilité de changer des 4 bits pour plusieurs valeurs de j (on considère un arbre par valeur initiale de a). Ces résultats sont listés dans le Tableau A.1 de l'appendice. Par symétrie, les probabilités sont les mêmes pour x_0, x_1, x_2, x_3 et le Tableau A.1 ne contient donc qu'une seule probabilité par profondeur. On observe que les biais les plus importants sont obtenus avec $j = 3, 5, 8, 11$. Par exemple, une bonne approximation linéaire est :

$$\Pr[[x_i]_1^t \oplus [x_i]_1^{t+3} = 1] \simeq 0.64 = \frac{1}{2}(1 + 0.28). \tag{4.5}$$

[1] Les autre puissances de S_1 donnent des probabilités pour les 4 bits de changer non biaisées.

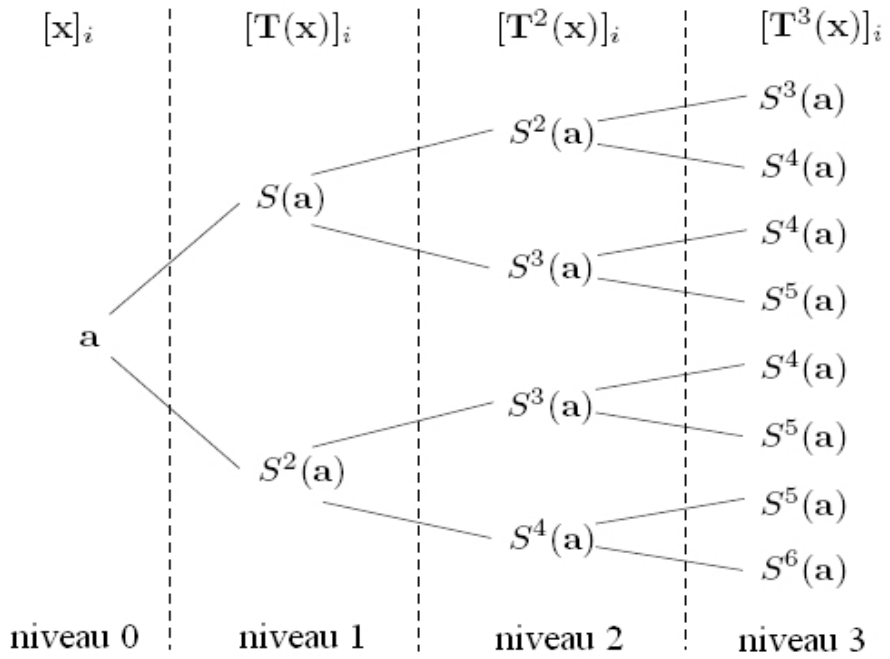


FIG. 4.1 – Évolution possible de la i-ème couche pour TSC-1

pour tout $i = 0, \ldots, 3$. En particulier, cette analyse explique les résultats de Künzli *et al.* concernant $j = 8$ et $j = 11$ mais de meilleurs biais existent avec $j = 3$.
Le Tableau A.2 de l'appendice montre que les biais qui sont observés expérimentalement aux instants t et $t + 3$ concordent avec la théorie. Ainsi on peut considérer **l'hypothèse initiale concernant la distribution uniforme du paramètre comme satisfaite**. La seule exception concerne les LSB des registres. En effet, le bit du paramètre à la position 0 est constant et l'hypothèse n'est plus vérifiée (le même raisonnement nous permet quand même le calcul pour la couche 0).

4.2.2 Deuxième étape

Pour cette étape, nous souhaitons linéariser le comportement de la fonction de sortie de TSC-1 définie par (3.12). Ce fonction utilise l'addition et la rotation gauche sur des mots de 32 bits. La rotation gauche est déjà linéaire, donc nous devons seulement linéariser les additions. Cela peut être fait naturellement en introduisant une **retenue**. Par exemple, en ajoutant deux entiers a_0 et a_1, nous

pouvons exprimer le i-ème bit du résultat par l'expression linéaire :

$$[a_0]_i \oplus [a_1]_i \oplus R_i$$

où le terme R_i dépend des couches $< i$ et est généralement non équilibré[2]. Dans cette section, nous voulons étendre cette observation à plus de deux termes, mais aussi commencer à connecter ces approximations linéaires avec celles de la première étape.

Tout d'abord, nous introduisons quelques notations. Considérons l'addition de m entiers de n bits notés $a_0, \ldots, a_{m-1}$. On désignera par $A = \sum_{k=0}^{m-1} a_k$ le résultat et par $R(i)$ la i-ème retenue. Pour $n = 2$ termes la retenue n'est représentée que par un seul bit, mais plus généralement c'est un entier défini formellement par :

$$R(i) = \frac{\sum_{k=0}^{m-1}(a_k \bmod 2^i) - (\sum_{k=0}^{m-1}(a_k \bmod 2^i)) \bmod 2^i}{2^i}. \tag{4.6}$$

Notons que $R(0) = 0$. La Figure 4.2 explicite notre modélisation.

$$\begin{array}{ccccccccccc}
[a_0]_{n-1} & \oplus & \cdots & \oplus & [a_k]_{n-1} & \oplus & \cdots & \oplus & [a_{m-1}]_{n-1} & & \\
 & & & & & & & & & \Uparrow & R(n-1) \\
\vdots & & & & \vdots & & & & \vdots & & \\
 & & & & & & & & & \Uparrow & R(i+2) \\
[a_0]_{i+1} & \oplus & \cdots & \oplus & [a_k]_{i+1} & \oplus & \cdots & \oplus & [a_{m-1}]_{i+1} & & \\
 & & & & & & & & & \Uparrow & R(i+1) \\
[a_0]_i & \oplus & \cdots & \oplus & [a_k]_i & \oplus & \cdots & \oplus & [a_{m-1}]_i & & \\
 & & & & & & & & & \Uparrow & R(i) \\
[a_0]_{i-1} & \oplus & \cdots & \oplus & [a_k]_{i-1} & \oplus & \cdots & \oplus & [a_{m-1}]_{i-1} & & \\
 & & & & & & & & & \Uparrow & R(i-1) \\
\vdots & & & & \vdots & & & & \vdots & & \\
 & & & & & & & & & \Uparrow & R(1) \\
[a_0]_0 & \oplus & \cdots & \oplus & [a_k]_0 & \oplus & \cdots & \oplus & [a_{m-1}]_0 & &
\end{array}$$

FIG. 4.2 – Modélisation d'une addition

On exprime la valeur du i-ème bit de A par :

$$[A]_i = \left[\sum_{k=0}^{n-1} a_k\right]_i = [R(i)]_0 \oplus \bigoplus_{k=0}^{n-1} [a_k]_i. \tag{4.7}$$

En regardant attentivement la fonction de sortie, on observe qu'elle est composée d'une addition à deux termes ($E = x_{0 \lll 9} + x_1$) et d'une addition à trois termes

[2]On peut noter que la retenue est un "paramètre" comme défini au Chapitre 3

$(S = E_{\lll 15} + x_{2\lll 7} + x_3)$ où S représente la sortie. Ainsi, en prenant (4.7), pour tout bit i on a :

$$\begin{cases} [E]_i = [x_0]_{(i+23)} \oplus [x_1]_{(i)} \oplus [R_E(i)]_0 \\ [S]_i = E_{(i+17)} \oplus [x_2]_{(i+25)} \oplus [x_3]_{(i)} \oplus [R_S(i)]_0 \end{cases} \tag{4.8}$$

où R_E et R_S représentent les retenues pour les additions à deux et trois termes respectivement. Tous les index sont calculés modulo 32. Notons que $R_E(i) \in \{0, 1\}$ et $R_S(i) \in \{0, 1, 2\}$. Finalement, on obtient :

$$[S]_i = [x_0]_{(i+8)} \oplus [x_1]_{(i+17)} \oplus [R_E(i+17)]_0 \oplus [x_2]_{(i+25)} \oplus [x_3]_{(i)} \oplus [R_S(i)]_0. \tag{4.9}$$

Cette relation est une **approximation linéaire de la fonction de sortie**. Pour connecter cette relation avec l'étape 1, le problème est maintenant de déterminer la probabilité pour les bits de retenues de changer entre les instants t et $t + 3$, sachant que chaque bit des termes des additions change avec une probabilité connue.

4.2.3 Probabilité de changer des bits de retenues

On définit la **retenue générale** $[R_G(i)] = [R_E(i + 17)] \oplus [R_S(i)]$. On note aussi $X_{R_G}(i) = [R_G(i)]_0^t \oplus [R_G(i)]_0^{t+3}$ et $X_j(i) = [x_j]_i^t \oplus [x_j]_i^{t+3}$. Grâce aux précédentes sections, on a :

$$[S]_i^t \oplus [S]_i^{t+3} = X_{R_G}(i) \oplus X_0(i + 8) \oplus X_1(i + 17) \oplus X_2(i + 25) \oplus X_3(i) \tag{4.10}$$

Maintenant, nous devons connaître la probabilité que cette relation soit vérifiée. On sait déjà grâce à la première étape que $\Pr[X_j(i) = 1] = \frac{1}{2}(1+\epsilon_i^j)$ avec $|\epsilon_i^j| \gg 0$ et nous connaissons les valeurs de tous les ϵ_i^j listées dans le Tableau A.2. Le terme restant est $X_{R_G}(i)$. Sa probabilité est liée à **la probabilité de changer des bits des retenues** R_S et R_E, dans les additions à deux et trois termes. Nous avons observé expérimentalement que

$$\Pr[X_{R_G}(i) = 1] = \frac{1}{2}(1 + \epsilon_i^G) \text{ avec } |\epsilon_i^G| \gg 0 \tag{4.11}$$

et nous essayons de calculer le ϵ_i^G de la relation. Malheureusement, on observe encore expérimentalement que les deux retenues $R_S(i)$ et $R_E(i)$ ne sont pas indépendantes, il est donc impossible de les traiter séparément.

Cependant, les retenues arrivant à la couche $i + 1$ sont calculées en connaissant seulement les retenues arrivant à la couche i et les termes de l'addition. Ce phénomène est une **chaîne de Markov**, puisqu'il n'est pas nécessaire de se rappeler de ce qui est arrivé aux couches inférieures ou égales à $i - 1$. Les cas simples (comme l'addition à deux termes) peuvent être facilement traités, mais dans le cas de la fonction de sortie de TSC-1, il y a deux additions enchevêtrées par des rotations gauche ce qui aboutit à un problème plus compliqué. Il est difficile d'exprimer $\Pr[X_{R_G}(i) = 1]$ avec une expression simple. Il est par contre

possible d'implémenter un algorithme récursif pour évaluer cette probabilité, en utilisant l'approche de la chaîne de Markov : grâce aux probabilités de changer des bits de la couche i, on évalue celles des bits de la couche $i+1$.

En premier lieu, pour tous les $a, b, c, d \in \{0,1,2\}^2 \times \{0,1\}^2$ possibles, on évalue récursivement :

$$\begin{aligned} \Pr_i(a,b,c,d) \quad = \Pr\ [\quad & (R_S(i)^t = a) \wedge (R_S(i)^{t+3} = b) \\ & \wedge (R_E(i+17)^t = c) \wedge (R_E(i+17)^{t+3} = d)] \end{aligned} \tag{4.12}$$

en commençant par la couche $i = 0$. L'algorithme est simple : pour calculer $\Pr_{i+1}(a,b,c,d)$, on examine tous les cas pour la couche i : on essaie toutes les valeurs des termes de l'addition, toutes celles des retenues arrivant à la couche i, et on calcule les nouvelles retenues. Chaque événement pour la couche i est associé avec sa probabilité correspondante, et nous incrémentons en conséquence les probabilités de la couche $i+1$. Après avoir examiné tous les cas, on obtient $\Pr_{i+1}(a,b,c,d)$. Alors, on incrémente i et on passe à la couche suivante[3].

Ensuite, nous calculons directement la probabilité que les bits de la retenue générale $X_{R_G}(i)$ changent avec :

$$\Pr[X_{R_G}(i) = 1] = \sum_{a,b,c,d | a \oplus b \oplus c \oplus d = 1} \Pr_i(a,b,c,d). \tag{4.13}$$

Les essais expérimentaux sur TSC-1 ont fourni les mêmes probabilités que les résultats de nos calculs utilisant l'approche de la chaîne de Markov. Ces résultats sont donnés dans le Tableau A.2 de l'appendice. Nous avons maintenant des expressions linéaires biaisées ne faisant intervenir que des bits de la sortie et de l'état interne. Nous pouvons passer à la troisième étape.

4.2.4 Troisième étape

Attaques par distingueur

Les approximations linéaires sont souvent utilisées pour des attaques par distingueur, en regardant si un certain biais est vérifié ou non. Pour un biais ε, il est connu qu'environ ε^{-2} échantillons sont requis. Par exemple, Künzli *et al.* ont trouvé un distingueur nécessitant 2^{22} données pour TSC-1 [11], en considérant t avec $t+8$, et en observant un biais de l'ordre de 2^{-11}. De la même façon, notre attaque générique nous permet d'obtenir des attaques par distingueur.

Par exemple, focalisons nous sur la couche $i = 1$ de la sortie. Nous avons

$$[S]_1^t \oplus [S]_1^{t+3} = X_{R_G}(1) \oplus X_0(9) \oplus X_1(18) \oplus X_2(26) \oplus X_3(1) \tag{4.14}$$

[3]Nous n'avons pas mentionné un petit problème technique venant du fait que la couche 0 dépend de la couche 31 à cause des rotations gauche, ce qui nous empêche de savoir comment initialiser la récurrence. En fait, les probabilités deviennent rapidement indépendantes des valeurs initiales, nous pouvons donc traiter facilement ce cas particulier.

En supposant les termes indépendants, les biais sont juste multipliés, ainsi le biais théorique ϵ_D de $\Pr[[S]_1^t \oplus [S]_1^{t+3} = 1]$ est :

$$\epsilon_D = \epsilon(X_0(9)) \times \epsilon(X_1(18)) \times \epsilon(X_2(26)) \times \epsilon(X_3(1)) \times \epsilon(X_{R_G}(1)) \qquad (4.15)$$

Le Tableau A.2 de l'appendice nous donne finalement $\epsilon_D = 0.2834 * 0.2824 * 0.2732 * 0.2812 * (-0.0874) = -2^{-10.86}$. Ce résultat est légèrement meilleur que ceux de l'attaque de Künzli *et al.* :

$$\epsilon_D^{-2} \simeq 2^{21.7}$$

bits de données seront nécessaires. Tous les i ne sont pas utiles pour une attaque par distingueur. En effet, dès qu'un des termes a un biais nul, $\epsilon_D = 0$ et aucune attaque par distingueur n'est possible. C'est ce qui arrive par exemple aux positions $0, 7, 15, 24$.

Cependant, des attaques par distingueur sont aussi possibles **après avoir deviné certaines portions de l'état, dans le but d'augmenter le biais**. Cette idée est étudiée plus en profondeur ci-dessous pour aboutir à des attaques retrouvant la clé.

Attaques retrouvant la clé

Comme indiqué dans la section 4.1.2, si on devine les i LSB de chaque registre dans l'état initial, on peut alors prédire ces bits pour n'importe quel instant t. Cette idée peut être utilisée pour **éliminer de nombreux termes dans les expressions linéaires biaisées**.

Tout d'abord, essayons de deviner les LSB de chaque registre. Il y a $2^4 = 16$ cas différents. Pour tout instant t, on peut prédire ces LSB et ainsi éliminer tous les termes de la forme $X_i(0)$ dans les approximations linéaires telles que (4.14). Par effet de bord, comme un terme est retiré, le biais augmente. En effet, nous n'observons plus $[S]_0^t \oplus [S]_0^{t+3}$ mais $[S]_0^t \oplus [S]_0^{t+3} \oplus X_3(0)$. Avec une quantité suffisante de données, on peut trouver lequel des 16 cas est le bon candidat : on mesure $\Pr[[S]_0^t \oplus [S]_0^{t+3} \oplus X_3(0) = 1]$ pour toutes les 16 valeurs possibles, et on garde seulement le candidat ayant le plus fort biais.

Cette observation nous fournit une attaque par distingueur avec une complexité améliorée par rapport à celle de la section précédente. Mais nous pouvons aussi répéter le processus pour les couches suivantes et successivement deviner toutes les couches jusqu'à finalement obtenir tout l'état initial[4].

La complexité pour deviner chaque couche dépend du meilleur biais que l'on pourra trouver à chaque étape. Pour la première, on a $\epsilon = -0.2826 * 0.2818 * 0.2826 * 0.1906 = -2^{-7.86}$ et ainsi nous avons besoin de

$$M = \epsilon^{-2} = 2^{15.72}$$

sorties pour trouver le bon candidat pour les LSB. La complexité en temps correspondante est d'environ

$$T = 2^{15.72} \times 2^4 = 2^{19.72}$$

[4]deviner plusieurs couches à la fois ne donne pas de meilleurs résultats.

étapes. Il faut noter qu'à chaque étape nous avons le choix entre plusieurs approximations linéaires. La meilleure pour la couche i est souvent celle provenant du bit i de la sortie, mais ce n'est pas toujours le cas. Par exemple, $X_{R_G}(2)$ est quasiment non biaisé, on préfère donc utiliser une autre approximation linéaire pour deviner la couche 2 de l'état initial. Tous les biais, positions de bits attaqués et complexités sont donnés dans le Tableau A.3 de l'appendice.

Il a des possibilités d'amélioration : pour la couche 7, nous pouvons augmenter le biais de l'approximation linéaire. En effet,

$$[S]_7^t \oplus [S]_7^{t+3} = X_{R_G}(7) \oplus X_0(15) \oplus X_1(24) \oplus X_2(0) \oplus X_3(7) \tag{4.16}$$

et puisque l'on connaît $X_2(0)$, on élimine deux termes. On observe $[S]_7^t \oplus [S]_7^{t+3} \oplus X_3(7) \oplus X_2(0)$ ce qui nous permet d'augmenter le biais à détecter. Pour les couches suivantes, on peut toujours éliminer deux termes, jusqu'à la couche 15 où l'on peut éliminer trois termes, et jusqu'à la couche 23 où l'on peut éliminer quatre termes. En fait, les premiers tours sont les plus coûteux en terme de complexité dans notre attaque : le tour 2 a la plus grosse complexité et nous obtenons une complexité finale d'environ $2^{21.4}$ en données et $2^{25.4}$ en temps (une unité de temps représentant le temps d'une itération de TSC-1).

4.3 Le cas TSC-2

Comme pour la section précédente, nous appliquons notre attaque générique à TSC-2 en suivant les trois étapes discernées. L'idée générale est essentiellement identique à TSC-1.

4.3.1 Première étape

Pour TSC-2, une S-Box et une T-fonction différentes de TSC-1 ont été choisies. La nouvelle forme de la fonction de mise à jour peut être mieux comprise en écrivant :

$$[\mathbf{T}(\mathbf{x})]_i = \begin{cases} [\mathbf{x}]_i & \text{si } [\alpha(\mathbf{x})]_i = 0 \\ \\ S_2([\mathbf{x}]_i) & \text{si } [\alpha(\mathbf{x})]_i = 1 \end{cases} \tag{4.17}$$

Tout d'abord, supposons que le paramètre soit uniformément distribué. Alors, la couche $[\mathbf{x}]_i$ reste inchangée avec probabilité $\frac{1}{2}$. On peut donc logiquement penser que l'approche "arbre binaire" éprouvée contre TSC-1 va nous donner des résultats pour TSC-2. Cependant, les concepteurs ont ajouté une propriété spéciale à S_2 : à chacune de ses applications, le LSB de l'entrée change avec probabilité 1. Ainsi, tous les bits du registre 0 ont une probabilité de changer d'exactement $\frac{1}{2}$. L'idée ici est de garantir au moins un registre "non biaisé" et de l'utiliser plusieurs fois dans la fonction de sortie pour "masquer" les autres registres "biaisés". Malheureusement, **le paramètre lui même n'est pas équilibré**. En effet, nous avons vu que

$$\alpha(\mathrm{x}) = (p + 1) \oplus p \oplus s$$

où $s = x_0 + x_1 + x_2 + x_3$. Le terme $(p+1) \oplus p$ est généralement égal à 0 (sauf pour les positions de poids faibles), tandis que le terme s a une probabilité de changer non négligeable. Ceci implique que la probabilité

$$\Pr[[\alpha(\mathbf{x})]_i^t \oplus [\alpha(\mathbf{x})]_i^{t+1} = 1]$$

dévie légèrement de $\frac{1}{2}$. Si on considère le même arbre que dans la Figure 4.1 pour TSC-2, les branches ne sont pas parfaitement équilibrées. On peut donc écrire des approximations linéaires entre les bits à différents instants, comme dans le cas de TSC-1. Nous avons cherché pour plusieurs profondeurs δ, et nous avons observé expérimentalement les biais. Les meilleurs résultats sont obtenus pour $\delta = 2$ (voir Tableau A.4 de l'appendice).

4.3.2 Deuxième étape

Dans cette partie de l'attaque, nous utilisons exactement la même technique que pour TSC-1 puisque les fonctions de sortie des deux stream ciphers se ressemblent. Pour TSC-2, la fonction de sortie utilise deux additions à deux termes ($E_1 = x_{0 \lll 11} + x_1$, $E_2 = x_{0 \lll 13} + x_2$) et une addition à quatre termes ($S = E_{1 \lll 14} + E_{2 \lll 22} + x_{0 \lll 12} + x_3$). On a alors :

$$\begin{aligned}
[S]_i = \quad & [x_0]_{(i+7)} \oplus [x_1]_{(i+18)} \oplus [R_{E_1}(i+18)]_0 \\
\oplus \quad & [x_0]_{(i+29)} \oplus [x_2]_{(i+10)} \oplus [R_{E_2}(i+10)]_0 \\
\oplus \quad & [x_0]_{(i+20)} \oplus [x_3]_{(i)} \oplus [R_S(i)]_0
\end{aligned} \tag{4.18}$$

On redéfinit la retenue générale $[R_G(i)] = [R_{E_1}(i+18)] \oplus [R_{E_2}(i+10)] \oplus [R_S(i)]$ et $X_{R_G}(i) = [R_G(i)]_0^t \oplus [R_G(i)]_0^{t+2}$. En posant $X_j(i) = [x_j]_i^t \oplus [x_j]_i^{t+2}$ et en combinant la première et la deuxième étape, on obtient :

$$\begin{aligned}
[S]_i^t \oplus [S]_i^{t+2} = \quad & X_1(i+18) \oplus X_2(i+10) \oplus X_3(i) \oplus X_{R_G}(i) \\
\oplus \quad & X_0(i+7) \oplus X_0(i+29) \oplus X_0(i+20)
\end{aligned} \tag{4.19}$$

On utilise la méthode de la Section 4.2.3 pour prédire les probabilités de changer des bits des retenues, grâce à un algorithme récursif similaire à celui pour TSC-1. Les résultats sont donnés dans le Tableau A.4 de l'appendice.

4.3.3 Troisième étape

Pour cette étape, nous pouvons aboutir à une attaque par distingueur ou une attaque retrouvant la clé. Le cheminement est similaire à celui de TSC-1 : en utilisant les approximations linéaires biaisées, on attaque une par une les 32 couches de 4 bits. La difficulté est de trouver quelle approximation linéaire est la meilleure pour chaque étape. Nous donnons dans le Tableau A.5 de l'appendice un choix possible pour chaque étape et les complexités correspondantes. Nous

obtenons finalement une attaque nous permettant de retrouver la totalité de l'état initial avec une complexité de $2^{44.1}$ en données et de $2^{48.1}$ en temps (une unité de temps représentant le temps d'une itération de TSC-2).

4.4 Le cas TSC-3

TSC-3 a été conçu en tenant compte des attaques par distingueur de Künzli *et al.*, en ajoutant notamment plusieurs contre-mesures pour éviter ce type de failles. Pourtant, notre attaque générique permet tout de même de casser ce stream cipher. Pour cela, nous utilisons des approximations linéaires différentes de celles de TSC-1 et TSC-2.

4.4.1 Première étape

Nous devons, dans cette section, trouver une approximation linéaire du comportement de la fonction de mise à jour. On peut déjà remarquer que l'on utilise ici la même S-Box que dans TSC-1, nous ne reviendrons pas donc sur les propriétés de cette primitive cryptographique, déjà étudiées dans la Section 4.2. Nous ne pouvons plus utiliser exactement la même technique que précédemment étant donnée la protection mise en place, l'idée est cependant très proche. Nous allons toujours regarder la mise à jour d'une couche $[\mathbf{x}]_i$ du registre par un arbre, mais un arbre non binaire cette fois. En effet, nous obtenons $[\mathbf{T}(\mathbf{x})]_i$ pour une couche i, qui selon la valeur du paramètre (que nous considérons uniformément distribué), est égal à $S^j([\mathbf{x}]_i)$, $j = 1, 2, 5, 6$ avec chacun une probabilité égale à $\frac{1}{4}$. Notre arbre aura donc 4 branches sortant de chaque noeud. De même, on peut facilement calculer la probabilité que $[\mathbf{T}^t(\mathbf{x})]_i$ soit égal à chacun des $S^j([\mathbf{x}]_i)$ avec $j = 0, \ldots, 15$ (voir Tableau 4.3).

$j =$	0	1	2	3	4	5	6	7
$t = 0$	1	0	0	0	0	0	0	0
$t = 1$	0	1/4	1/4	0	0	1/4	1/4	0
$t = 2$	0	0	1/16	1/8	1/16	0	1/8	1/4
$t = 3$	0.047	0.047	0.016	0.016	0.047	0.047	0.016	0.047
$t = 4$	0.039	0.063	0.094	0.063	0.023	0.031	0.047	0.031
...								

$j =$	8	9	10	11	12	13	14	15
$t = 0$	0	0	0	0	0	0	0	0
$t = 1$	0	0	0	0	0	0	0	0
$t = 2$	1/8	0	1/16	1/8	1/16	0	0	0
$t = 3$	0.141	0.141	0.047	0.047	0.141	0.141	0.047	0.016
$t = 4$	0.023	0.063	0.094	0.063	0.039	0.094	0.141	0.094
...								

FIG. 4.3 – Probabilité que $[\mathbf{T}^t(\mathbf{x})]_i = S^j([\mathbf{x}]_i)$ pour TSC-3

Ainsi, en énumérant toutes les entrées possibles, il est possible de calculer la probabilité de chaque sortie à la couche i en moyennant sur toutes les feuilles possibles de cette couche. Cependant, contrairement aux cas TSC-1 et TSC-2, on ne considère plus la probabilité qu'un certain bit change mais la relation entre un des 4 bits d'entrée et un des 4 bits de sortie. Ceci est nécessaire à cause de la permutation de registres dans la fonction de sortie, ce qui nous oblige à considérer des bits de différents registres dans les prochaines étapes de l'attaque. On se concentre sur la probabilité que :

$$[x_j]_i^t = [x_{j'}]_i^{t+\delta} \tag{4.20}$$

pour deux registres j et j' différents $\in \{0, 1, 2, 3\}$, pour une certaine profondeur δ et pour une certaine couche i. Nous observons d'intéressants résultats étant donné que les concepteurs n'ont pas protégé TSC-3 contre ce type d'approximations linéaires.

Grâce au Tableau 4.3, on peut calculer la probabilité que l'équation (4.20) soit vérifiée pour n'importe quelle paire (j, j'). Ces résultats sont indépendants de la couche considérée, sauf pour les couches de poids faibles lesquelles ne sont pas utilisées dans la fonction de sortie. Les probabilités obtenues pour quelques valeurs de δ sont listées dans le Tableau A.6 de l'appendice. Tous ces résultats ont été vérifiés expérimentalement, ce qui valide notre hypothèse concernant la distribution uniforme du paramètre.

4.4.2 Deuxième étape

Dans le cas de TSC-3, la fonction de sortie n'est pas directement appliquée aux 4 registres d'état, mais à quatre registres y_0, y_1, y_2 et y_3 qui sont des versions tronquées et permutées des registres x_0, x_1, x_2 et x_3. Tout d'abord, linéarisons la fonction de sortie comme nous l'avons fait pour TSC-1 et TSC-2 :

$$[S]_i = [y_0]_{i+17} \oplus [y_1]_{i+26} \oplus [y_2]_{i+2} \oplus [y_3]_{i+9} \oplus [R_G(i)]_0$$

où $R_G(i)$ est la retenue générale, définie comme précédemment. Ensuite, on remplace les bits des registres y_i par les bits appropriés des registres x_i. A cause de la troncation et de la permutation, on a $[y_j]_i = [x_{\pi(j)}]_{i+8}$ où π est une permutation de 4 bits déterminée par la couche 0 de l'état interne. L'approximation linéaire dépend de cette permutation. Supposons que nous sommes dans le cas particulier où :

$$[\mathbf{x}]_0^t = 4.$$

Alors la prochaine valeur de cette couche est :

$$[\mathbf{x}]_0^{t+1} = 1.$$

En regardant la permutation π associée avec ces valeurs particulières, on obtient

$$[S]_i^t = [x_0]_{i+25}^t \oplus [x_1]_{i+34}^t \oplus [x_2]_{i+10}^t \oplus [x_3]_{i+17}^t \oplus [R_G(i)]_0^t$$

et

$$[S]_i^{t+1} = [x_0]_{i+25}^{t+1} \oplus [x_1]_{i+34}^{t+1} \oplus [x_2]_{i+10}^{t+1} \oplus [x_3]_{i+17}^{t+1} \oplus [R_G(i)]_0^{t+1}.$$

En utilisant le Tableau A.6 de l'appendice, on observe que :

$$\begin{aligned}
\Pr([x_0]_{(i+25)}^t = [x_1]_{(i+25)}^{t+1}) &= 0.46875 \\
\Pr([x_1]_{(i+34)}^t = [x_0]_{(i+34)}^{t+1}) &= 0.53125 \\
\Pr([x_3]_{(i+10)}^t = [x_2]_{(i+10)}^{t+1}) &= 0.53125 \\
\Pr([x_2]_{(i+17)}^t = [x_3]_{(i+17)}^{t+1}) &= 0.46875
\end{aligned}$$

Ces quatre probabilités sont de la forme $\frac{1}{2}(1 \pm 2^{-4})$. Nous avons essayé de regarder d'autres valeurs initiales de $[\mathbf{x}]_0$ que 4, mais il semble que ce soit le meilleur choix puisque ce sont pour ces paramètres qu'apparaissent les meilleures probabilités dans le Tableau A.6.

En combinant les deux relations aux instants t et $t+1$, on obtient :

$$[S]_i^t \oplus [S]_i^{t+1} = [R_G(i)]_0^t \oplus [R_G(i)]_0^{t+1}$$

avec probabilité $\frac{1}{2}(1 \pm \epsilon)$ et $|\epsilon| = (2^{-4})^4 = 2^{-16}$.

Comme pour TSC-1 et TSC-2, les retenues des additions qui interviennent dans la fonction de sortie ne sont pas indépendantes les unes des autres. Il n'est donc pas facile d'exprimer simplement la probabilité que $[R_G]$ change entre t et $t+1$. Comme auparavant, modéliser ce phénomène par une chaîne de Markov nous donnerait des résultats plus précis mais pour des raisons de simplicité nous avons choisi de les mesurer expérimentalement. Les résultats obtenus pour différentes valeurs de i sont listés dans le Tableau A.7 de l'appendice. Pour certaines positions bien choisies (typiquement celles pour lesquelles certaines retenues sont égales à 0 de façon certaine), la probabilité est largement différente de $\frac{1}{2}$. Nous avons observé des biais jusqu'à $\epsilon \simeq 2^{-3}$ pour certaines "bonnes" positions telles que $i = 8$ ou $i = 23$. En conséquence,

$$[S]_{23}^t \oplus [S]_{23}^{t+1}$$

est égal à 0 avec probabilité $\frac{1}{2}(1 + \epsilon)$ et $|\epsilon| \simeq 2^{-16} \times 2^{-3} = 2^{-19}$.

Ce biais est seulement valide quand $[\mathbf{x}]_0^t = 4$, ce qui est le cas pour exactement une position sur 16 dans la séquence cyclique de la couche de poids faible de l'état interne. On peut alors connaître de façon déterministe quelles positions analyser en connaissant les 4 bits de poids faible de l'état initial. Dans la section suivante, nous étudions comment exploiter ces biais pour des attaques par distingueur ou des attaques retrouvant la clé.

4.4.3 Troisième étape

En exploitant les bits 8 et 23 du mot de sortie, nous avons vu dans la section précédente que nous obtenons un biais de l'ordre de $\epsilon = 2^{-19}$. On peut ainsi utiliser ce biais pour distinguer la sortie de TSC-3 d'une séquence aléatoire, avec

$\epsilon^{-2} = 2^{38}$ données. Mais il faut tenir compte du fait que seulement une valeur sur 16 est utile dans la séquence de sortie, ce qui nous oblige à utiliser M mots de sortie, avec :

$$M = 16 \times 2^{38} = 2^{42}.$$

De plus, il nous faut tester toutes les valeurs de la couche de poids faible de l'état initial, ce qui nous donne une complexité en temps de :

$$T = 2^4 \times 2^{38} = 2^{42}.$$

Pour une attaque retrouvant la clé, on commence par deviner les 9 tranches de poids faible de l'état initial, soit $4 \times 9 = 36$ bits au total. Ceci nous permet de prédire la couche $[\mathbf{x}]_8^t$ qui est aussi la couche de poids faible des registres y_i. Cette couche est une de celles intervenant dans notre "meilleure" approximation : $[S]_{23}^t \oplus [S]_{23}^{t+1}$.

Ainsi, on élimine un terme dans cette approximation ce qui fait augmenter le biais de 2^{-19} à 2^{-15}. Une fois trouvés les bits de ces 9 couches, il est très facile de continuer pour le reste de l'état interne, comme nous l'avons fait pour TSC-1 et TSC-2. En fait, la première étape est largement la plus coûteuse car nous devons deviner 36 bits en une seule étape. On arrive à une complexité de

$$T = 2^{36} \times (2^{15})^2 = 2^{66}$$

en temps (une unité de temps représentant le temps d'une itération de TSC-3) et de

$$M = 16 \times (2^{15})^2 = 2^{34}$$

en données. On rappelle que la clé secrète peut être retrouvée rapidement à partir de l'état initial, le processus d'initialisation étant inversible.

4.5 La famille TF-i

Contrairement à la famille TSC-i, notre attaque générique s'applique mal pour la famille TF-i. En effet, les multiplications utilisées dans la T-fonction ne nous permettent de trouver des approximations linéaires que pour les bits de poids faible de l'état interne. Or les différentes fonctions de sortie définies pour la famille TF-i n'utilisent que les bits de poids fort des registres. Ainsi, nous arrivons à passer la première et la deuxième étape, mais durant la dernière phase, la combinaison de ces deux étapes est difficile. Donc la famille TF-i semble bien protégée contre les attaques par approximation linéaire.

Il faut cependant noter que Mitra et Sarkar [21] ont réussi à casser TF-1 et TF-2 en utilisant un compromis temps-mémoire. Il permet grâce à quelques mots de sortie de l'algorithme de retrouver la clé avec une complexité inférieure à celle d'une recherche exhaustive. Ils utilisent pour cela une modélisation de la multiplication (pour TF-1) et de la mise au carré (pour TF-2) utilisées dans

les T-fonctions de la famille TF-i. A cause d'une fonction de sortie très simple pour TF-1 et TF-2, ils sont en mesure de deviner rapidement quels bits de poids faible peuvent aboutir à une sortie donnée (faisant intervenir les bits de poids fort). Ils retrouvent ainsi l'état initial des registres. Cette faille a été corrigée dans TF-3 en utilisant une fonction de sortie plus compliquée, ne dévoilant pas directement les bits de poids fort des registres.

4.6 Les résultats

Nous résumons nos résultats concernant la famille TSC. Le Tableau 4.1 liste ces complexités, ainsi que celles des précédents travaux de Künzli *et al.* concernant les attaques par distingueur.

Tab. 4.1 – Résumé des attaques contre la famille TSC

Algorithme	Type d'attaque	Temps	Données
TSC-1	Distingueur [11, 18]	2^{22}	2^{22}
TSC-1	Distingueur	2^{19}	2^{15}
TSC-1	Attaque retrouvant la clé	$2^{25.4}$	$2^{21.4}$
TSC-2	Distingueur[18]	2^{34}	2^{34}
TSC-2	Attaque retrouvant la clé	$2^{48.1}$	$2^{44.1}$
TSC-3	Distingueur	2^{42}	2^{42}
TSC-3	Attaque retrouvant la clé	2^{66}	2^{34}

4.7 Critères pour conceptions futures

Tout d'abord, on peut remarquer que les trois étapes distinctes de notre attaque générique par cryptanalyse linéaire sont toujours possibles, dans une certaine mesure.

– La périodicité des couches de poids faibles dans les T-fonctions multi-mots est toujours petite, par construction. La périodicité de la i-ème couche est d'au plus 2^{mi} pour un état de m mots. Ainsi la relation linéaire suivante est toujours vérifiée avec probabilité 1 :

$$[x_j]_i^t \oplus [x_j]_i^{t+2^{mi}} = 0$$

D'autres approximations existent selon la nature de T-fonction, comme nous l'avons illustré avec la famille TSC.

- Pour tout choix de la fonction de sortie, il existe des approximations linéaires entre les bits de l'état interne et ceux de la sortie. A moins que la fonction de sortie ne soit vraiment complexe (ce qui n'est généralement pas le cas car la fonction de sortie se doit d'être rapide), il est très probable que des approximations avec bonne probabilité existent.
- Si les approximations des étapes 1 et 2 peuvent être combinées, il est généralement possible d'exploiter ces relations biaisées pour aboutir à une attaque retrouvant la clé.

De ce fait, la difficulté ne réside pas dans la recherche d'approximations linéaires ou dans la façon de les exploiter, mais plus dans la combinaison de toutes les approximations pour décrire l'algorithme de chiffrement dans sa totalité. Il est très probable que les T-fonctions soient amenées à jouer un rôle important dans le futur de la conception des stream ciphers. Pour éviter l'application de la cryptanalyse linéaire, nous suggérons plusieurs recommandations.

- Ne jamais utiliser la moitié de poids faible des registres dans la fonction de sortie à cause de la faible périodicité (cette contre-mesure a déjà été appliquée par Klimov et Shamir dans plusieurs conceptions).
- Utiliser des rotations dans la fonction de sortie pour combiner les bits de tous les registres. La fonction de sortie de TSC-1 est sans doute trop simple, ce qui facilite l'analyse.
- Essayer d'éviter les approximations linéaires simples pour les T-fonctions sur plusieurs étapes consécutives. Pour les T-fonctions basées sur des S-Box proposées par Hong *et al.*, le problème de savoir si cela est possible est encore ouvert. Peut-être, pour une instantiation appropriée de la S-Box et du paramètre, l'existence de bonnes approximations linéaires peut être évitée. Ce point est un sujet intéressant pour des recherches futures.
- Essayer de profiter le plus possible des avantages que nous fournissent les opérations "complexes" disponibles sur les processeurs. Par exemple, nous pensons que l'utilisation de multiplication entière, lorsqu'elle est possible, est une bonne idée, même dans la fonction de sortie.

Toutes ces contre-mesures peuvent avoir un impact négatif sur la vitesse de chiffrement, mais ceci doit être mis en relation avec l'augmentation du niveau de sécurité.

Chapitre 5

Conclusion

Dans ce rapport, nous avons étudié la nouvelle famille de primitives cryptographiques introduites par Klimov et Shamir que sont les T-fonctions. Nous avons vu leurs nombreuses applications dans le domaine de la cryptographie symétrique, notamment en ce qui concerne les stream ciphers. Nous avons enfin décrit et implémenté un modèle d'attaque efficace contre les stream ciphers fondés sur des T-fonctions. Nous arrivons à la conclusion que les stream ciphers de la famille TSC-i n'offrent pas la sécurité qu'ils visent, et doivent donc être évités pour une utilisation sensible.

Plusieurs axes de recherches futures sont possibles. Tout d'abord, il serait intéressant de chercher une nouvelle famille de T-fonctions inversibles et à cycle unique, peut être même pouvoir caractériser toutes ces T-fonctions à l'aide d'un théorème facilement utilisable pour la construction de stream ciphers. La conception d'un nouveau stream cipher fondé sur des T-fonctions serait alors possible, et dans ce but, la cryptanalyse décrite dans ce rapport apporte quelques éléments pour éviter certains écueils. Il serait aussi intéressant de trouver une faille dans les derniers stream ciphers de la famille TF-i, contre lesquels aucune attaque n'a encore été décrite. Pour cela, une extension de l'attaque de Mitra et Sarkar, une adaptation de celle décrite dans ce rapport ou même un nouveau type d'attaque, sont autant de voies d'exploration possibles. Enfin, dans une optique plus large, de nombreuses autres applications des T-fonctions sont sans doute encore inconnues, notamment dans le domaine actuellement très sensible des fonctions de hachage.

Bibliographie

[1] F. Armknecht and M. Krause. Algebraic Attacks on Combiners with Memory. In D. Boneh, editor, *Advances in Cryptology – Crypto'03*, volume 2729 of *Lectures Notes in Computer Science*, pages 162–175. Springer, 2003.

[2] Steve Babbage. Stream Ciphers : What Does the Industry Want ? In *State of the Art of Stream Ciphers* workshop (SASC'04), 2004.

[3] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scaveius. Rabbit : A New High-Performance Stream Cipher. In T. Johansson, editor, *Fast Software Encryption – 2003*, volume 2887 of *Lectures Notes in Computer Science*, pages 307–329. Springer, 2003.

[4] N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *Advances in Cryptology – Eurocrypt'03*, volume 2656 of *Lectures Notes in Computer Science*, pages 345–359. Springer, 2003.

[5] ECRYPT. ECRYPT stream cipher project. http ://www.ecrypt.eu.org/stream/.

[6] P. Ekdahl and T. Johansson. A New Version of the Stream Cipher SNOW. In *Selected Areas in Cryptography – 2002*, Lectures Notes in Computer Science, pages 47–61. Springer, 2002.

[7] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno. Helix, Fast Encryption and Authentication in a Single Cryptographic Primitive. In T. Johansson, editor, *Fast Software Encryption – 2003*, volume 2887 of *Lectures Notes in Computer Science*, pages 195–209. Springer, 2003.

[8] J. Golić. Linear Statistical Weakness of Alleged RC4 Keystream Generator. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97*, volume 1233 of *Lectures Notes in Computer Science*, pages 226–238. Springer, 1997.

[9] J. Hong, D. Lee, Y. Yeom, and D. Han. A New Class of Single Cycle T-functions. In *Fast Software Encryption – 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2005.

[10] J. Hong, D. Lee, Y. Yeom, D. Han, and S. Chee. T-function based streamcipher TSC-3. In *ECRYPT Call for Stream Cipher Primitives*, 2005. Submission.

[11] P. Junod, S. Künzli, and W. Meier. Attacks Against TSC. Rump Session at Fast Software Encryption – 2005, 2005.

[12] A. Klimov. *Applications of T-functions in Cryptography*. PhD thesis, Weizmann Institute of Science, 2004. http ://www.wisdom.weizmann.ac.il/ ask/.

[13] A. Klimov and A. Shamir. A New Class of Invertible Mappings. In B. Kaliski, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, volume 2523 of *Lectures Notes in Computer Science*, pages 470–483. Springer, 2002.

[14] A. Klimov and A. Shamir. Cryptographic Applications of T-functions. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – 2003*, volume 3006 of *Lectures Notes in Computer Science*, pages 248–261. Springer, 2004.

[15] A. Klimov and A. Shamir. New Cryptographic Primitives Based on Multiword T-Functions. In B. Roy and W. Meier, editors, *Fast Software Encryption – 2004*, volume 3017 of *Lectures Notes in Computer Science*, pages 1–15. Springer, 2004.

[16] A. Klimov and A. Shamir. The TFi Family of Stream Ciphers, 2004. Handout given at the SASC'04 workshop.

[17] A. Klimov and A. Shamir. New Applications of T-functions in Block Ciphers and Hash Functions. In *Fast Software Encryption – 2005*, 2005. To appear.

[18] S. Künzli, W. Meier, and P. Junod. Distinguishing Attacks on T-Functions. In *International Conference on Cryptology in Malaysia* (MyCrypt'05), 2005. To appear.

[19] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt'93*, volume 765 of *Lectures Notes in Computer Science*, pages 386–397. Springer, 1993.

[20] Willi Meier and Othmar Staffelbach. Fast Correlations Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, pages 159–176. Springer, 1989.

[21] J. Mitra and P. Sarkar. Time-Memory Trade-Off Attacks on Multiplications and T-functions. In P. Lee, editor, *Advances in Cryptology - Asiacrypt'04*, volume 3329 of *Lectures Notes in Computer Science*, pages 468–482. Springer, 2004.

[22] NIST. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22.

[23] Claus-Peter Schnorr and Serge Vaudenay. Black Box Cryptoanalysis of Hash Networks Based on Multipermutations. In Alfredo De Santis, editor, *Advances in Cryptology – Eurocrypt'94*, volume 950 of *Lectures Notes in Computer Science*, pages 47–57. Springer, 1994.

[24] A. Shamir. Stream Ciphers : Dead or Alive ? Invited talk presented at Asiacrypt'04, 2004.

[25] Thomas Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30, pages 776–780, 1984.

[26] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel. A New Key Stream Generator MUGI. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption – 2002*, volume 2365 of *Lectures Notes in Computer Science*, pages 179–194. Springer, 2002.

Annexe A

Application de l'attaque à la famille TSC-i

TAB. A.1 – TSC-1 : probabilité de changer des bits pour différentes profondeurs j de l'arbre

j	P	$\lvert\epsilon\rvert$
1	0.5000	0.0000
2	0.5937	0.1874
3	0.6406	0.2812
4	0.5078	0.0156
5	0.4219	0.1562
6	0.4473	0.1054
7	0.5479	0.0958
8	0.5996	0.1992
9	0.5264	0.0528
10	0.4143	0.1714
11	0.3993	0.2014
12	0.4849	0.0302
13	0.5587	0.1174
14	0.5507	0.1014
15	0.4972	0.0056
16	0.4717	0.0566

TAB. A.2 – TSC-1 pour t/t+3 : probabilité de changer des bits des registres et du LSB de la retenue générale R_G

bit	registre 0	registre 1	registre 2	registre 3	LSB(R_G)
0	0.5000	0.5000	0.5000	0.5000	0.5953
1	0.6406	0.6406	0.6406	0.6406	0.4563
2	0.6479	0.6479	0.6479	0.6479	0.4948
3	0.6446	0.6446	0.6446	0.6446	0.5247
4	0.6442	0.6442	0.6442	0.6442	0.5328
5	0.6427	0.6427	0.6427	0.6427	0.5343
6	0.6356	0.6356	0.6356	0.6356	0.5356
7	0.6412	0.6412	0.6412	0.6412	0.5352
8	0.6413	0.6413	0.6413	0.6413	0.5263
9	0.6417	0.6416	0.6416	0.6417	0.5337
10	0.6417	0.6417	0.6417	0.6417	0.5355
11	0.6364	0.6364	0.6364	0.6364	0.5357
12	0.6408	0.6408	0.6409	0.6409	0.5354
13	0.6412	0.6411	0.6412	0.6412	0.5352
14	0.6416	0.6416	0.6415	0.6416	0.5354
15	0.6417	0.6416	0.6416	0.6417	0.4348
16	0.6364	0.6364	0.6364	0.6363	0.4952
17	0.6408	0.6409	0.6409	0.6408	0.5253
18	0.6412	0.6412	0.6412	0.6412	0.5332
19	0.6416	0.6416	0.6417	0.6416	0.5349
20	0.6364	0.6364	0.6364	0.6364	0.5355
21	0.6409	0.6408	0.6408	0.6409	0.5359
22	0.6412	0.6412	0.6412	0.6413	0.5355
23	0.6365	0.6365	0.6365	0.6365	0.5360
24	0.6408	0.6408	0.6408	0.6409	0.5349
25	0.6413	0.6413	0.6413	0.6413	0.5266
26	0.6366	0.6365	0.6366	0.6366	0.5333
27	0.6409	0.6408	0.6408	0.6409	0.5351
28	0.6414	0.6413	0.6413	0.6412	0.5357
29	0.6365	0.6366	0.6365	0.6365	0.5357
30	0.6408	0.6409	0.6408	0.6408	0.5355
31	0.6412	0.6412	0.6412	0.6412	0.5351

Tab. A.3 – Schéma d'attaque possible contre TSC-1

étape	bit attaqué	registre attaqué	nombre de termes connus	biais obtenu	complexité en temps
0	0	3	0	$2^{-7.86}$	$2^{19.72}$
1	1	3	0	$2^{-9.03}$	$2^{22.06}$
2	9	2	0	$2^{-9.41}$	$2^{22.82}$
3	10	2	0	$2^{-9.29}$	$2^{22.58}$
4	11	2	0	$2^{-9.33}$	$2^{22.66}$
5	12	2	0	$2^{-9.39}$	$2^{22.78}$
6	13	2	0	$2^{-9.31}$	$2^{22.62}$
7	7	3	1	$2^{-7.48}$	$2^{18.96}$
8	0	0	1	$2^{-6.04}$	$2^{16.08}$
9	1	0	1	$2^{-7.21}$	$2^{18.42}$
10	10	3	1	$2^{-7.47}$	$2^{18.94}$
11	11	3	1	$2^{-7.46}$	$2^{18.92}$
12	19	2	1	$2^{-7.49}$	$2^{18.98}$
13	20	2	1	$2^{-7.51}$	$2^{19.02}$
14	21	2	1	$2^{-7.50}$	$2^{19.00}$
15	15	3	2	$2^{-4.81}$	$2^{13.62}$
16	8	0	2	$2^{-6.07}$	$2^{16.14}$
17	9	0	2	$2^{-5.76}$	$2^{15.52}$
18	10	0	2	$2^{-5.64}$	$2^{15.28}$
19	11	0	2	$2^{-5.63}$	$2^{15.26}$
20	12	0	2	$2^{-5.69}$	$2^{15.38}$
21	13	0	2	$2^{-5.66}$	$2^{15.32}$
22	14	0	2	$2^{-5.64}$	$2^{15.28}$
23	15	0	3	$2^{-2.94}$	$2^{9.88}$
24	24	3	3	$2^{-3.84}$	$2^{11.68}$
25	0	2	3	$2^{-2.39}$	$2^{8.78}$
26	1	2	3	$2^{-3.52}$	$2^{11.04}$
27	10	1	3	$2^{-3.82}$	$2^{11.64}$
28	11	1	3	$2^{-3.81}$	$2^{11.62}$
29	12	1	3	$2^{-3.82}$	$2^{11.64}$
30	13	1	3	$2^{-3.83}$	$2^{11.66}$
31	14	1	3	$2^{-3.82}$	$2^{11.64}$

TAB. A.4 – TSC-2 pour t/t+2 : probabilité de changer des bits des registres et du LSB de la retenue générale R_G

bit	registre 0	registre 1	registre 2	registre 3	LSB(R_G)
0	0.0000	0.5000	0.5000	0.6250	0.4929
1	0.5000	0.4062	0.4062	0.4453	0.4960
2	0.5312	0.3848	0.3848	0.4146	0.4942
3	0.4980	0.3744	0.3744	0.4057	0.4943
4	0.4957	0.3739	0.3739	0.4054	0.4894
5	0.4858	0.3715	0.3715	0.4036	0.4940
6	0.4811	0.3703	0.3703	0.4027	0.4955
7	0.4792	0.3698	0.3698	0.4023	0.4949
8	0.4785	0.3697	0.3697	0.4022	0.4952
9	0.4782	0.3695	0.3695	0.4021	0.4948
10	0.4780	0.3695	0.3695	0.4022	0.4947
11	0.4781	0.3694	0.3695	0.4022	0.4949
12	0.4781	0.3695	0.3695	0.4021	0.4944
13	0.4781	0.3696	0.3696	0.4021	0.4892
14	0.4781	0.3695	0.3696	0.4021	0.4877
15	0.4781	0.3695	0.3694	0.4021	0.4963
16	0.4781	0.3695	0.3696	0.4021	0.4952
17	0.4781	0.3695	0.3696	0.4021	0.4949
18	0.4781	0.3696	0.3696	0.4022	0.4948
19	0.4781	0.3695	0.3695	0.4021	0.4945
20	0.4782	0.3695	0.3696	0.4021	0.4947
21	0.4781	0.3695	0.3695	0.4020	0.4944
22	0.4781	0.3696	0.3695	0.4022	0.4875
23	0.4782	0.3695	0.3695	0.4021	0.4945
24	0.4781	0.3696	0.3696	0.4022	0.4953
25	0.4781	0.3696	0.3696	0.4021	0.4946
26	0.4781	0.3695	0.3695	0.4022	0.4896
27	0.4781	0.3695	0.3696	0.4022	0.4942
28	0.4781	0.3695	0.3695	0.4022	0.4955
29	0.4781	0.3695	0.3695	0.4022	0.4951
30	0.4782	0.3696	0.3696	0.4022	0.4949
31	0.4781	0.3695	0.3695	0.4022	0.4951

Tab. A.5 – Schéma d'attaque possible contre TSC-2

étape	bit attaqué	registre attaqué	nombre de termes connus	biais obtenu	complexité en temps
0	3	0	0	$2^{-21.76}$	$2^{47.52}$
1	13	0	0	$2^{-20.79}$	$2^{45.58}$
2	14	0	1	$2^{-18.66}$	$2^{41.32}$
3	3	3	1	$2^{-19.36}$	$2^{42.72}$
4	4	3	1	$2^{-18.46}$	$2^{40.92}$
5	5	3	1	$2^{-19.28}$	$2^{42.56}$
6	6	3	1	$2^{-19.70}$	$2^{43.40}$
7	7	3	1	$2^{-19.52}$	$2^{43.04}$
8	8	3	1	$2^{-19.60}$	$2^{43.20}$
9	12	0	1	$2^{-17.22}$	$2^{38.44}$
10	3	0	2	$2^{-14.84}$	$2^{33.68}$
11	4	0	2	$2^{-13.95}$	$2^{31.90}$
12	5	0	2	$2^{-14.77}$	$2^{33.54}$
13	3	2	3	$2^{-12.91}$	$2^{29.82}$
14	14	3	3	$2^{-11.80}$	$2^{27.60}$
15	5	2	3	$2^{-12.83}$	$2^{29.66}$
16	6	2	3	$2^{-13.25}$	$2^{30.50}$
17	7	2	3	$2^{-13.07}$	$2^{30.14}$
18	8	2	3	$2^{-13.15}$	$2^{30.30}$
19	9	2	3	$2^{-13.04}$	$2^{30.08}$
20	0	0	4	$2^{-10.65}$	$2^{25.30}$
21	14	0	4	$2^{-7.28}$	$2^{18.56}$
22	15	0	4	$2^{-9.01}$	$2^{22.02}$
23	3	0	5	$2^{-6.45}$	$2^{16.90}$
24	4	0	5	$2^{-5.56}$	$2^{15.12}$
25	5	0	5	$2^{-6.38}$	$2^{16.76}$
26	6	0	5	$2^{-6.80}$	$2^{17.60}$
27	7	0	5	$2^{-6.62}$	$2^{17.24}$
28	8	0	5	$2^{-6.70}$	$2^{17.40}$
29	9	0	5	$2^{-6.59}$	$2^{17.18}$
30	10	0	5	$2^{-6.56}$	$2^{17.12}$
31	11	0	5	$2^{-6.62}$	$2^{17.24}$

TAB. A.6 – TSC-3 : probabilité que $[x_j]_i^t = [x_{j'}]_i^{t+\delta}$

Cas $\delta = 1$

Input \ Output	$[x_0]_i^t$	$[x_1]_i^t$	$[x_2]_i^t$	$[x_3]_i^t$
$[x_0]_i^{t+1}$	0.5	0.53125	0.46875	0.5
$[x_1]_i^{t+1}$	0.46875	0.5	0.5	0.46875
$[x_2]_i^{t+1}$	0.5	0.53125	0.5	0.53125
$[x_3]_i^{t+1}$	0.53125	0.5	0.46875	0.5

Cas $\delta = 2$

Input \ Output	$[x_0]_i^t$	$[x_1]_i^t$	$[x_2]_i^t$	$[x_3]_i^t$
$[x_0]_i^{t+2}$	0.515625	0.515625	0.5	0.5
$[x_1]_i^{t+2}$	0.5	0.515625	0.46875	0.5
$[x_2]_i^{t+2}$	0.5	0.5	0.515625	0.515625
$[x_3]_i^{t+2}$	0.5	0.5	0.5	0.515625

Cas $\delta = 3$

Input \ Output	$[x_0]_i^t$	$[x_1]_i^t$	$[x_2]_i^t$	$[x_3]_i^t$
$[x_0]_i^{t+3}$	0.51172	0.49610	0.51172	0.50391
$[x_1]_i^{t+3}$	0.50391	0.51172	0.49610	0.51172
$[x_2]_i^{t+3}$	0.49610	0.48828	0.51172	0.49610
$[x_3]_i^{t+3}$	0.48828	0.49610	0.50391	0.51172

TAB. A.7 – TSC-3 : biais mesurés expérimentalement pour la retenue générale R_G

Position	$\Pr([R_G]_i^t \oplus [R_G]_i^{t+1})$
$i = 0$	0.5001
$i = 1$	0.4921
$i = 2$	0.4968
$i = 3$	0.4989
$i = 4$	0.5001
$i = 5$	0.5003
$i = 6$	0.5004
$i = 7$	0.4999
$i = 8$	0.4442
$i = 9$	0.4871
$i = 10$	0.4967
$i = 11$	0.4999
$i = 12$	0.4996
$i = 13$	0.4997
$i = 14$	0.5002
$i = 15$	0.4997
$i = 16$	0.4993
$i = 17$	0.4998
$i = 18$	0.4997
$i = 19$	0.4998
$i = 20$	0.5003
$i = 21$	0.5002
$i = 22$	0.4996
$i = 23$	0.4452
$i = 24$	0.4862
$i = 25$	0.4962
$i = 26$	0.4995
$i = 27$	0.5003
$i = 28$	0.5005
$i = 29$	0.4997
$i = 30$	0.4996
$i = 31$	0.5007