Rapport à diffusion publique du stage de fin d'études
réalisé du 1er mars au 31 août 2013
en vue de l'obtention du diplôme de

MASTER de Cryptologie de l'Université de Limoges

Amélioration des attaques différentielles sur *KLEIN*

Cryptanalyse de la version complète de *KLEIN*-64

Virginie Lallemand

Stage effectué à Inria Paris Rocquencourt

Sous la direction de María Naya-Plasencia

Le présent rapport porte sur le stage de fin d'études réalisé en seconde année de Master CRYPTIS parcours Mathématiques, Cryptologie, Codage et Applications (MCCA) de l'Université de Limoges [1]. Il s'est déroulé du 1er mars au 31 août 2013 au sein de l'établissement public de recherche Inria Paris-Rocquencourt [2] sous la direction de María Naya-Plasencia, chargée de recherche au sein de l'équipe-projet SECRET [3].

Dans ce rapport nous présenterons le cadre du stage et ses enjeux puis nous détaillerons les réalisations faites ainsi que les résultats obtenus.

1. Université de Limoges - Faculté des Sciences et Techniques, 123 Avenue Albert Thomas, 87000 Limoges
2. Inria Paris-Rocquencourt Domaine de Voluceau, BP 105, 78153 Le Chesnay
3. SEcurité, CRyptologie Et Transmissions

Remerciements

J'aimerais adresser ici mes remerciements aux personnes qui ont contribué à la réalisation de ce stage.

Tout d'abord, mes remerciement vont à María Naya-Pasencia, chargée de recherche au sein de l'équipe SECRET, qui a accepté de m'encadrer durant ces 6 mois de stage. Je lui suis reconnaissante de s'être toujours rendue disponible et d'avoir trouvé la patience de répondre à mes nombreuses questions. Ses conseils et ses idées ont été très précieux et ont rendu ce stage très enrichissant.

J'adresse également mes remerciement à Anne Canteaut, directrice de recherche et responsable de l'équipe SECRET, pour m'avoir accueillie et intégrée à son équipe et ainsi permis de profiter de la grande expérience du Projet en cryptographie symétrique.

J'aimerais aussi remercier l'équipe pédagogique du Master Cryptis de L'Université de Limoges pour la qualité de l'enseignement dispensé. J'aimerais plus particulièrement exprimer ma gratitude à Thierry Berger, responsable du parcours Mathématiques, Cryptologie, Codage et Applications (MCCA) dont les cours de clef secrète sont pour beaucoup dans mon orientation et qui m'a offert l'opportunité de rencontrer les membres de l'équipe SECRET et par suite d'y effectuer mon stage.

Finalement j'aimerais également remercier tous les membres de l'équipe SECRET pour leur accueil et leur gentillesse et les discussions intéressantes que nous avons eues.

Table des matières

1 Introduction

1.1 Environnement du Stage

1.1.1 Présentation d'Inria

Inria est un établissement public de recherche à caractère scientifique et technologique créé en 1967. Ses domaines de recherches sont l'informatique et les mathématiques appliqués aux sciences du numérique. Réparti en 8 centres de recherche dans toute la France, il compte pas moins de 179 équipes-projets traitant des thèmes touchant à la robotique, à l'automatique ou encore aux réseaux.

Les 3 150 scientifiques d'Inria se partagent en petites structures d'une vingtaine de personnes appelées Équipes-Projets Inria (EPI) réunissant autours d'un leader scientifique un ensemble de chercheurs, de doctorants et d'ingénieurs. Cette entité possède un objectif scientifique précis et a pour devoir de communiquer largement ses résultats par le biais de ses publications et de ses participations à des colloques mais aussi de contribuer au transfert des connaissances et des technologies vers l'industrie par des partenariats industriels ou par la création de brevets.

1.1.2 Présentation de l'EPI SECRET

L'EPI SECRET (**SE**curité, **CR**yptologie **E**t **T**ransmissions) fait suite à l'équipe de recherche CODES depuis 2008. Dirigée par Anne Canteaut, elle a comme thème de recherche la conception et l'analyse de la sécurité d'algorithmes cryptographiques. Ses 2 principaux axes de recherche se situent en cryptographie symétrique et en codes correcteurs d'erreurs. Concernant la cryptographie symétrique qui est l'axe suivi par ce stage, ses objectifs consistent à éprouver la sécurité des systèmes existants en testant la robustesse des algorithmes, mais aussi à concevoir de nouveaux systèmes. On peut à ce sujet évoquer sa participation au projet européen eSTREAM de conception de chiffrements à flots et au concours international SHA-3 de conception de fonctions de hachage.

La section suivante présente le cadre théorique des travaux de ce stage et introduit les notions et les notations nécessaires à la compréhension de la suite de ce rapport.

1.2 La Cryptographie Symétrique

La cryptographie ou science du secret, est une science ayant pour objectif de garantir la confidentialité, l'intégrité et l'authenticité de communications échangées sur un canal non sûr.

Ces objectifs sont atteints grâce à des algorithmes (dits de *chiffrement*) transformant le message de départ (*clair*) en un autre message (*chiffré*) inintelligible pour un éventuel espion qui écouterait le canal. Seul le destinataire est en mesure de retrouver le message original (*déchiffrer*) grâce à des informations secrètes que lui seul connaît (*la clef*).

On distingue 2 familles d'algorithmes selon le type d'informations secrètes utilisées lors des procédures de chiffrement et de déchiffrement : les algorithmes à clef secrète et à clef publique. Les algorithmes à clef secrète sont basés sur le partage d'une même clef par l'expéditeur et le destinataire. On parle aussi de cryptographie symétrique pour souligner le fait que la même clef est utilisée pour chiffrer et déchiffrer.

A l'inverse, les algorithmes à clef publique, appelés aussi asymétriques, utilisent 2 clefs lors des échanges ; la première est *publique*, et sert à l'expéditeur à chiffrer son message

alors que la seconde est *privée* et connue uniquement du destinataire qui s'en sert pour déchiffrer.

Lors de ce stage, je me suis concentrée uniquement sur la cryptographie à clef secrète qui a notamment l'avantage d'être plus rapide et plus adaptée à des environnements contraints.

La cryptographie à clef secrète comprend elle-même 3 grandes familles d'algorithmes aux finalités variées :

- Les ***chiffrements à flot*** chiffrent les messages en combinant le message clair avec une *suite chiffrante* produite par un générateur pseudo-aléatoire initialisé grâce à la clef secrète. Typiquement, cette combinaison se fait à l'aide d'un OU exclusif (*XOR*) bit à bit.
- Les ***chiffrements par bloc*** ont la particularité de découper le message d'entrée en blocs de taille fixe (64, 128 ou 256 bits) pour leur appliquer ensuite la même fonction, paramétrée par la clef secrète. Le plus célèbre et le plus utilisé actuellement est l'algorithme *Rijndael* choisi en 2000 par le NIST pour être le nouveau standard international [AES01].
- Les ***fonctions de hachage*** ont de nombreuses applications et servent par exemple à assurer l'intégrité d'un message ou d'un fichier en fournissant une *empreinte* représentant le message de façon compacte avec un nombre fixé de bits. Associée au fichier, l'empreinte permet au destinataire de vérifier que le message n'a pas été modifié puisqu'il a la possibilité de calculer lui même l'empreinte à partir du message et de la fonction de hachage publique. Ces algorithmes n'utilisent pas de clef mais font partie de la cryptographie à clef secrète à cause de la similitude des primitives utilisées (leur construction est très proche de celle des chiffrements par bloc).

1.2.1 Les Chiffrements Itératifs par Bloc

On introduit ici quelques notations et notions essentielles à la compréhension de la suite de l'exposé.

Définition 1. *Chiffrement par bloc :*
Un chiffrement par bloc est une fonction E prenant en entrée 2 arguments : un bloc M de m bits et une clef secrète K de k bits et ayant comme sortie un bloc de m bits C.

$$\begin{aligned} E : \{0,1\}^k \times \{0,1\}^m &\to \{0,1\}^m \\ (K \quad , \quad M \quad) &\mapsto \quad c := E(K,M) = E_K(M) \end{aligned}$$

Comme le chiffré doit permettre de retrouver le clair pour le récepteur qui possède la clef, la fonction $E_K(M)$ (désignant le chiffrement à clef K fixée) est une permutation. Cette caractéristique implique aussi que la taille de l'entrée et de la sortie sont identiques.

La majorité des chiffrements par bloc actuels sont itératifs ; c'est à dire qu'ils sont formés par r répétitions d'une même fonction F appelée *fonction de tour* et paramétrée par une information dérivée de la clef : les *clefs de tour* ou *sous clefs* $K_1 \cdots K_r$.

$$E_K = F_{K_r} \circ F_{K_{r-1}} \circ \cdots \circ F_{K_1}$$

Les clefs de tour sont calculées à partir de la clef secrète de base K (appelée aussi *clef maître*) avec l'*algorithme de cadencement de clef.*

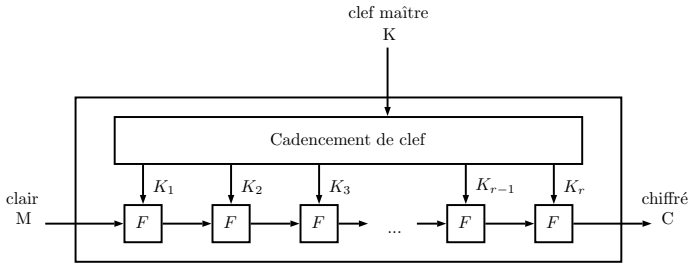


FIGURE 1 – Chiffrement itératif

La fonction interne est elle-même composée de plusieurs opérations aux propriétés définies et bien maîtrisées qui permettent de répondre aux 2 principes fondamentaux de diffusion et de confusions introduits par Claude Shannon en 1949 [Sha49]. La confusion doit rendre les liens entre la clef et le chiffré aussi complexe que possible et d'autre part la diffusion doit dissiper la redondance du message clair dans le texte chiffré (un éventuel biais d'entrée ne doit pas se retrouver en sortie).

La partie confusion est la seule qui soit non linéaire. Elle correspond généralement à l'application parallèle d'une même ou de différentes tables de substitution agissant sur des mots de l'état de 4 ou 8 bits : ces tables offrant de la non linéarité sont appelées *boîtes S*.

Les deux principaux types de constructions utilisés pour les fonctions de tour F sont les schémas de Feistel et les réseaux de Substitution-Permutation (SPN).

Définition 2. *Chiffrement de Feistel :*
Un chiffrement de Feistel est un système de chiffrement itératif agissant sur des blocs de taille paire ($m = 2n$) avec une fonction de tour F paramétrée par une clef de tour K_i agissant sur la moitié de gauche (L) et la moitié de droite (R) de la façon suivante :

$$\begin{aligned} F_{K_i} : \ \mathbb{F}_2^{2n} &\to \mathbb{F}_2^{2n} \\ (L, R) &\mapsto (R, L \oplus f_{K_i}(R)) \end{aligned}$$

où f_{K_i} est une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^n$ paramétrée par la clef du tour i K_i

On peut noter que quelle que soit la fonction f l'inverse de la fonction de tour existera toujours et pourra être calculé par la fonction :

$$\begin{aligned} F_{K_i}^{-1} : \ \mathbb{F}_2^{2n} &\to \mathbb{F}_2^{2n} \\ (L, R) &\mapsto (R \oplus f_{K_i}(L), L) \end{aligned}$$

Ce système de chiffrement a notamment été utilisé dans le DES "Data Encryption Standard" [Sta99], qui fut de 1977 à 1999 le standard de chiffrement des données et qui agit sur des blocs de 64 bits et nécessite une clef de 56 bits.

Définition 3. *Chiffrement SPN :*
Un chiffrement de type Substitution-Permutation possède une fonction de tour pouvant se décomposer en 3 opérations qui sont l'ajout de clef, une substitution non linéaire et une fonction de diffusion linéaire.

Contrairement au schéma de Feistel, chaque opération se doit d'être inversible pour pouvoir assurer le déchiffrement. Un exemple de tel système est l'AES [AES01] (Advanced Encryption Standard) choisi en 2000 comme standard de chiffrement et toujours utilisé à ce jour.

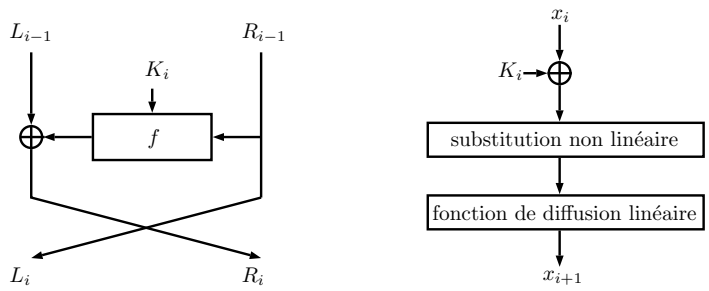


FIGURE 2 – Fonction de tour d'un schéma de Feistel (à gauche) et d'un réseau de Substitutions-Permutations (à droite)

1.2.2 La Cryptographie à Bas Coût

Les dernières années ont vu apparaître de nouveaux besoins cryptographiques nés de la nécessité de sécuriser les nouveaux supports de plus en plus miniaturisés tels que les puces RFID [4] et les réseaux de capteurs. Les algorithmes existants, notamment AES, ne sont pas adaptés à de tels environnements et nécessitent trop d'énergie et trop de mémoire, d'où la nécessité de créer de nouveaux algorithmes spécialement adaptés qui ont comme objectifs à la fois de prendre peu de place, de consommer peu d'énergie, d'être rapides et évidemment de répondre aux besoins de sécurité. On parle alors de cryptographie à bas coût (en anglais *lightweight cryptography*).

On peut distinguer deux sous-familles de tels algorithmes selon leurs applications ; ceux crées pour une utilisation hardware et ceux orientés software. D'autres spécialisations viennent ensuite, notamment la distinction entre les algorithmes répondant le mieux à la contrainte de taille, évaluée en *Gate Equivalent* [5] (GE), ceux se concentrant sur le débit, évalué en Kb/s, (central notamment pour des applications nécessitant de la sécurité en temps réel) ou encore ceux nécessitant très peu de courant.

De nombreuses propositions ont été faites ces dernières années, dont certaines sont spécifiées dans la table 1.

CLEFIA, développé par SONY Corporation, est un chiffrement par bloc de type Feistel et a été désigné en 2012 comme un des standards de cryptographie lightweight (ISO/IEC 29192 International standards in lightweight cryptography) au même titre que PRESENT développé conjointement par Orange Labs, l'Université de la Ruhr à Bochum et l'Université Technique du Danemark.

4. pour Radio-Frequency IDentification, technologie d'identification à distance par radiofréquence
5. unité de mesure exprimée en nombre de portes nécessaires pour câbler l'algorithme sur un circuit

Chiffrement	Taille de clef	Taille de Blocs	Gate Equivalent	Débit
PRESENT-80 [BKL+07]	80	64	1570	200
PRESENT-128	128	64	1886	200
CLEFIA-128 [SSA+07]	128	128	5979	711
KATAN-64 [CDK09]	80	64	1054	25
LBlock [WZ11]	64	80	1320	200
KLEIN-64 [GNL11]	64	64	1220	N/A
KLEIN-80	80	64	1478	N/A
KLEIN-96	96	64	1528	N/A

TABLE 1 – Spécification et performances de quelques chiffrements à bas coût (chiffres donnés dans [GPPR11], [CDK09] et [BKL+07]). Les tailles de bloc et de clefs sont exprimées en bits, le débit en Kb/s à 100KHz

1.2.3 Sécurité d'un Chiffrement par Bloc

L'objectif de sécurité visé par un système de chiffrement est de rendre toute recherche de la clef secrète au moins aussi lente que la recherche exhaustive. Si ce n'est pas le cas, le système sera considéré comme *cassé*. Plusieurs hypothèses peuvent être faites sur l'information à disposition de l'attaquant : on parlera d'attaque à clair connu (l'attaquant possède des couples (clair,chiffré) aléatoires), à clair choisi (il peut choisir les messages clairs et obtenir les chiffrés correspondants) ou encore à chiffré choisi (il peut demander le chiffrement et le déchiffrement des messages qu'il désire). Les attaques détaillées dans ce rapport se placent en situation de clair choisi.

Contrairement aux cryptosystèmes à clef publique ou basés sur les codes, il est rarement possible de réaliser une réduction de sécurité pour les algorithmes itératifs par bloc. La sécurité du système est donc définie par rapport à la meilleure attaque connue. C'est pourquoi la cryptanalyse joue un rôle de premier plan en cryptographie symétrique : un cryptosystème n'apparaîtra fiable que s'il à été étudié sans succès par plusieurs (bons) cryptanalystes indépendants. Au contraire, il est tout aussi important de repérer rapidement les systèmes non sûrs pour ne pas les utiliser.

Au delà de son rôle d'évaluation de la sécurité des systèmes, la cryptanalyse et ses avancées permanentes permettent d'aboutir à de nouveaux principes et critères pour construire de nouveaux systèmes plus résistants.

Pour rendre compte de la faisabilité d'une attaque, on mesure généralement sa difficulté avec 3 quantités qui sont :

- La ***complexité en données***, correspondant au nombre de couples (clair, chiffré) nécessaires pour réaliser l'attaque,
- La ***complexité en temps***, correspondant au nombre d'opérations de chiffrement réalisées au cours de l'attaque,
- La ***complexité en mémoire***, correspondant à la taille de stockage nécessaire.

On distingue 2 principaux types d'attaques s'appliquant aux chiffrements par bloc :

- Les ***attaques algébriques*** consistent de façon simplifiée à traduire le système de chiffrement par un système d'équations d'inconnue la clef et ensuite à résoudre le sys-

tème. Le plus souvent le degré très élevé des équations obtenues rend ces attaques très difficiles.
- Les ***attaques statistiques*** reposent sur l'observation de comportements du système qui s'éloignent de ceux attendus d'une permutation aléatoire pour retrouver de l'information sur la clef.

C'est ce dernier type d'attaque qui sera développé ici. La notion centrale associée est celle de ***distingueur*** :

Définition 4. *Distingueur :*
Un distingueur est un algorithme qui par un jeu de questions/réponses à une permutation arrive à décider avec probabilité supérieure à 1/2 s'il s'agit d'une permutation aléatoire ou s'il s'agit d'un chiffrement particulier.

La construction d'un distingueur de la permutation constituée de toutes les fonctions de tour du système de chiffrement exceptée la dernière avec une clef fixée et secrète permet d'attaquer le système par une ***attaque statistique sur le dernier tour***. Celle-ci permet de retrouver la sous clef du dernier tour et procède en 3 étapes :
- L'attaquant obtient les chiffrés (par la fonction de chiffrement complète sur r tours) correspondant à N messages qu'il sait distinguer sur $r-1$ tours.
- Il déchiffre partiellement sur le dernier tour l'ensemble des chiffrés avec chacune des hypothèses de sous clef candidate et obtient un ensemble de valeurs d'états. Si son hypothèse de clef est correcte, ceux-ci correspondent aux images des chiffrés après $r-1$ tours.
- Pour chacun de ces ensembles, il utilise le distingueur qui réagira si la sous clef testée est la bonne, alors qu'une mauvaise sous clef est supposée donner un résultat proche de celui d'une permutation aléatoire (c'est l'hypothèse de répartition aléatoire par fausse clef).

1.3 Objectifs, Motivations et Réalisations du Stage

Le cadre de ce stage se situe en cryptographie symétrique et concerne plus particulièrement la sécurité des chiffrement à bloc. Dans ce contexte et comme énoncé dans la section précédente, la cryptanalyse est essentielle pour évaluer la sécurité d'un cryptosystème.

C'est pourquoi l'objectif de ce stage a été d'étudier en détail la cryptanalyse différentielle qui est un type particulier d'attaque statistique. Au delà de la compréhension de ses principes et de l'étude de cas concrets d'applications, ce stage a consisté à comprendre plus spécifiquement les attaques réalisées sur le chiffrement à bas coût *KLEIN* et à chercher à améliorer les résultats des précédentes cryptanalyses.

Cette recherche menée par María Naya-Plasencia et moi-même a abouti à une proposition d'attaque sur la version complète (comportant tous les tours) de *KLEIN*-64. Nous avons réussi à étendre la portée de l'attaque de 8 à 12 tours, c'est à dire à casser la version complète du cryptosystème *KLEIN*. Suite à cette découverte, nous avons rédigé nos résultats en vue de leur soumission à la conférence FSE 2014 (International Workshop on Fast Software Encryption). En parallèle, une partie importante du stage a été consacrée à la programmation de l'attaque puis à l'optimisation du programme. Cette étape non négligeable a permis de valider nos calculs théoriques.

Dans les sections suivantes nous exposerons les principes de la cryptanalyse différentielle et de quelques unes de ses variantes, puis nous verrons le fonctionnement du chif-

frement à bloc *KLEIN* pour finalement exposer en détail notre proposition d'attaque et les résultats de nos expérimentations pratiques.

2 Cryptanalyse Différentielle

Le type d'attaque auquel je me suis intéressé lors de ce stage est la cryptanalyse différentielle. Cette section présente ses principes et certaines de ses variantes. Une étude plus approfondie peut être trouvée dans [Blo11].

2.1 Définition et Description de l'Attaque de Base

La cryptanalyse différentielle est une famille d'attaques statistiques s'appliquant aux chiffrements à clef secrète et surtout appliquée aux chiffrements par bloc et et aux fonctions de hachage. C'est avec la cryptanalyse linéaire une des attaques statistiques les plus efficaces si bien que chaque nouveau système prend généralement en compte ces attaques (dans leur forme la plus basique) dans son design. Elle a été introduite en 1991 par Eli Biham et Adi Shamir dans l'article fondateur *Differential cryptanalysis of DES like cryptosystems* [BS91].

Son idée principale est d'étudier la propagation des différences [6] présentes dans les messages d'entrée tout au long des étapes du chiffrement pour en déduire un biais dans la distribution des différences dans les chiffrés en sortie. Ce biais pourra ensuite être exploité pour construire un distingueur et retrouver des informations sur la clef, typiquement en réalisant une attaque sur le dernier tour. Cette attaque nécessite de pouvoir obtenir le chiffrement de paires de messages suivant un certain motif et fait donc généralement partie des attaques *à clair choisi*.

Pour réaliser une cryptanalyse différentielle sur un schéma de chiffrement par bloc itératif, la première étape consiste à trouver une *différentielle* significativement plus fréquente que la moyenne :

Définition 5. *Différentielle :*
Une différentielle sur t tours d'un système de chiffrement est un couple $(\delta_{in}, \delta_{out}) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$ constitué d'une différence en entrée et d'une différence en sortie après t tours.

Il existe le plus souvent plusieurs façons d'obtenir δ_{out} à partir de δ_{in} : chaque manière spécifique constitue un *chemin différentiel* :

Définition 6. *Chemin différentiel :*
Un chemin différentiel sur t tours d'un système de chiffrement itératif est un $(t+1)-$uplet $(\delta_{in} = \delta_0, \delta_2, \cdots, \delta_t = \delta_{out}) \in (\mathbb{F}_2^m)^{(t+1)}$ de différences intermédiaires à chaque sortie de tour.

À clef secrète fixée, on parlera de *bonne paire* (ou de paire conforme) pour désigner deux messages d'entrée dont les différences générées sont celles données par le chemin différentiel et de *mauvaise paire* dans le cas contraire.

Concrètement, le biais se traduit par le fait que si on chiffre 2 messages m_1 et m_2 sur t tours vérifiant $m_1 \oplus m_2 = \delta_{in}$, on observera avec forte probabilité que les états après t tours vérifient $s_1 \oplus s_2 = \delta_{out}$. Pour assurer de grandes chances de réussite à l'attaque, la probabilité de cette différentielle doit être la plus éloignée possible des probabilités d'apparition des autres différences de sortie. Cette probabilité est le plus souvent assez difficile à évaluer puisqu'elle doit être calculée en moyenne sur tous les messages d'entrée et sur toutes les

6. le terme 'différence' correspondra dans la grande majorité des cas à l'addition dans $\mathbb{F}_2^m$ puisque cela permet une meilleure maîtrise de l'influence de la clef qui est le plus souvent additionnée à l'état

clefs possibles.

Pour pouvoir calculer une probabilité théorique, on émet l'hypothèse que le chiffrement est *de Markov* :

Définition 7. *Chiffrement de Markov relativement à la cryptanalyse différentielle : Un système de chiffrement est dit de Markov relativement à la cryptanalyse différentielle si la probabilité de la différence en sortie connaissant la différence en entrée est indépendante de la clef utilisée pour chiffrer.*

En admettant aussi que les clefs de tour sont indépendantes et uniformément distribuées dans l'espace des clefs, on démontre que la probabilité d'un chemin différentiel vaut le produit des probabilités des différentielles sur un tour qui le composent. On peut ensuite calculer le biais total d'une différentielle sur t tours en remarquant que la probabilité d'une différentielle est la somme des probabilités des chemins qui la composent.

Comme les étapes réalisant la diffusion dans un schéma de chiffrement sont linéaires et propagent une différence avec probabilité 1, l'attaquant doit se concentrer sur la partie réalisant la confusion du chiffrement pour trouver un chemin différentiel biaisé.

Il doit donc étudier en détail la propagation des différences dans les boîtes S en étudiant les probabilités qu'une différence en donne une autre après avoir traversé la boîte S. Notons s la taille (nombre de bits d'entrée et de sortie) d'une boîte S bijective. L'attaquant calcule pour tout couple (δ_a, δ_b) de différences en entrée et en sortie de boîte S les valeurs de :

$$T_{\delta_a,\delta_b} = {}^{\#}\{x \in \mathbb{F}_2^s | S(x) \oplus S(x \oplus \delta_a) = \delta_b\}$$

On résume souvent ces calculs dans un tableau à deux entrées dont les lignes correspondent aux différences d'entrée δ_a et les colonnes aux différences de sortie δ_b et où on renseigne à l'intersection des deux T_{δ_a,δ_b}. Cette table est appelée *table de différences* (voir par exemple celle présentée en annexe A).

Ces calculs permettent de déduire la probabilité d'une différentielle pour une boîte S, donnée par $\frac{T_{\delta_a,\delta_b}}{2^s}$ puis d'obtenir les différentielles de tour les plus fréquentes. En en composant plusieurs, on obtient des chemins de forte probabilité puis des différentielles sur le nombre de tours voulu.

Une fois que le chemin différentiel est défini, l'attaquant peut l'utiliser pour créer un distingueur qui lui permettra de monter une attaque sur le dernier tour : il demande en premier lieu le chiffrement de N paires de messages de différence d'entrée δ_{in}, puis procède à leur inversion partielle avec ses différentes sous clefs candidates. Pour décider quels sont les candidats les plus probables, il crée un compteur par candidat et l'incrémente à chaque fois que l'inversion partielle d'une paire le mène à la différence attendue par son chemin différentiel.

Si son chemin différentiel est de probabilité p, la bonne sous clef aura un compteur de $p \times N$ alors que l'hypothèse de répartition aléatoire par fausse clef implique qu'un mauvais candidat sera compté environ $\frac{1}{2^m-1} \times N$ fois.

Il classe ensuite les sous clefs en fonction de leur compteur et conserve les meilleures. Pour juger laquelle est correcte, il peut par exemple remonter à la clef maître avec l'algorithme de cadencement de clef et effectuer le chiffrement d'une paire pour s'assurer qu'elle concorde avec les valeurs obtenues en début d'attaque. Au besoin il devra effectuer une recherche exhaustive sur des informations manquantes nécessaires pour obtenir les autres

sous clefs de tour. Une autre possibilité consiste à effectuer une attaque différentielle avec un distingueur sur $r-2$ tours.

Dans certains cas, comme par exemple si la clef maître a la même taille que chaque clef de tour, La recherche de clef ne se fera pas sur la sous clef totale, puisqu'alors l'attaque serait moins performante que la recherche exhaustive, mais seulement sur les bits de clefs intervenant dans le calcul de la différence à laquelle on s'intéresse.

2.2 Variantes des Attaques Différentielles

2.2.1 Cryptanalyse Différentielle Tronquée

La cryptanalyse différentielle tronquée a été introduite en 1994 par Lars Knudsen dans [Knu94] à propos des schémas de Feistel. C'est une généralisation des attaques différentielles où les différences intervenant sont seulement partiellement déterminées.

Définition 8. *Différentielle tronquée :*
Une différentielle tronquée d'un chiffrement itératif par bloc de r tours est un couple formé par un ***ensemble de différences en entrée*** $\Delta_{in} \subset \mathbb{F}_2^m$ *et un* ***ensemble de différences en sortie*** *après r tours* $\Delta_{out} \subset \mathbb{F}_2^m$.

On a l'analogue de la notion de chemin différentiel dans ce cas ci :

Définition 9. *Chemin différentiel tronqué :*
Un chemin différentiel tronqué relatif à Δ_{in} et Δ_{out} tels que définis précédemment est un ensemble constitué d'ensembles de différences pour chaque tour : $(\Delta_{in} = \Delta_0, \Delta_1, \cdots \Delta_r = \Delta_{out})$.

La probabilité d'une différentielle tronquée sera alors définie tout naturellement comme la probabilité d'obtenir une différence dans l'ensemble d'arrivé voulu étant donné une différence dans l'ensemble de départ :

$$P[\Delta_{in} \rightarrow \Delta_{out}] = P_{X,K}[E_K(X) \oplus E_K(X \oplus a) \in \Delta_{out} | a \in \Delta_{in}]$$

Ce type d'attaque permet de construire des chemins ne dépendant pas des boîtes S ; en effet on pourra par exemple s'intéresser uniquement à distinguer si une boîte S est active [7] ou inactive [8] en construisant un chemin formé d'ensembles contenant toutes les valeurs de différences possibles pour une boîte S. Ce type de construction permet de tirer parti de la mauvaise diffusion du système.

C'est ce type d'attaque qu'on développera dans la section 4, notamment en construisant des ensembles de différences se résumant à s'intéresser à l'activité ou à l'inactivité des boîtes S.

7. On dit qu'un ensemble de bits ou qu'une boîte S est active si des différences associées au chemin différentiel peuvent y apparaître

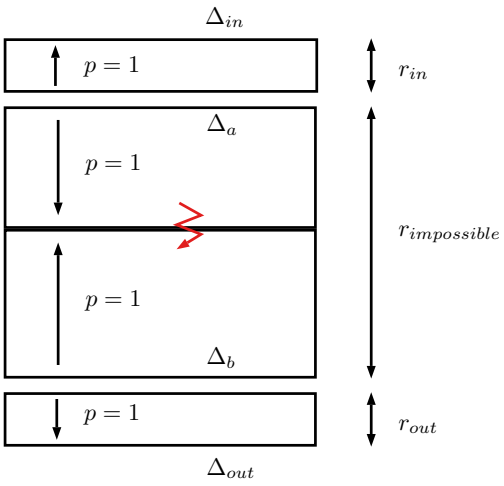
8. On dit qu'un ensemble de bits ou qu'une boîte S est inactive si une bonne paire n'a pas de différences en ces positions

2.2.2 Cryptanalyse Différentielle Impossible

Une autre variante de cryptanalyse différentielle se base sur des chemins différentiels de probabilité nulle pour obtenir de l'information sur la clef : c'est la cryptanalyse différentielle impossible.

L'attaque de ce type la plus connue porte sur l'AES (voir par exemple [BK00]) et utilise un motif de différences de 2 tours ne pouvant jamais mener à un autre motif de 2 tours.

Dans le cas général, l'attaque commence par la construction d'un chemin différentiel impossible par exemple avec une technique du type *miss in the middle* (l'attaquant choisi 2 différences (tronquées) Δ_a et Δ_b et les dérive - l'une en chiffrant, l'autre en déchiffrant - avec probabilité 1 jusqu'à obtenir une contradiction entre les différences obtenues).



Sans rentrer dans les détails, une cryptanalyse différentielle impossible utilise un chemin tel que représenté en figure ci-contre où p représente les probabilités de diffusion des différences. Si pour une paire de messages donnés avec une différence Δ_{in} en entrée et Δ_{out} en sortie l'attaquant réalise des hypothèses sur les sous clefs intervenant dans les r_{in} premiers tours et les r_{out} derniers tours et observe Δ_a et en même temps Δ_b, il peut déduire avec certitude que ses hypothèses ne sont pas les bonnes et peut éliminer ces candidats. Les éliminations successives lui permettent de réduire l'ensemble des clefs possibles.

Une différence majeure entre cette variante et la cryptanalyse différentielle de base réside dans la gestion des clefs candidates : cette méthode-ci permet d'éliminer des sous clefs avec certitude alors que dans l'attaque de base on essaye de savoir lesquelles sont les plus probables.

2.2.3 Cryptanalyse Différentielle d'Ordre Supérieur

Remarquons ici que la notion de différentielle est liée à celle de dérivée :

Définition 10. *Dérivée d'ordre 1 :*
La dérivée d'une fonction F de $\mathbb{F}_2^p$ à $\mathbb{F}_2^q$ en un point $a \in \mathbb{F}_2^n$ est définie par :

$$D_a F(x) = F(x \oplus a) \oplus F(x)$$

En prenant pour F une partie du chiffrement (de $\mathbb{F}_2^m$ à $\mathbb{F}_2^m$) et pour a le vecteur de différence entre les états d'entrée de F, on voit que la notion de dérivée se confond avec celle de différentielle comme précédemment définie.

L'idée de la cryptanalyse différentielle d'ordre supérieur consiste alors à établir l'analogue d'un chemin différentiel reposant sur la notion de dérivée d'ordre supérieur :

Définition 11. *Dérivée d'ordre supérieur :*
La k-ième dérivée d'une fonction F de $\mathbb{F}_2^p$ à $\mathbb{F}_2^q$ en un sous-espace V de $\mathbb{F}_2^p$ est définie par :

$$D_V F(x) = D_{a_1} D_{a_2} \cdots D_{a_k} F$$

où $(a_1, a_2, \cdots a_k)$ est une base quelconque de V. De plus, pour tout $x \in \mathbb{F}_2^p$ on a :

$$D_V F(x) = \bigoplus_{v \in V} F(x \oplus v)$$

L'objectif est alors de construire une attaque sur le dernier tour en étudiant les propriétés de la fonction G correspondant au chiffrement itératif réduit d'un tour : $G = F_{K_{r-1}} \circ \cdots \circ F_{K_1}$.

Si on suppose qu'il existe un sous espace $V \in \mathbb{F}_2^m$ de dimension k et une constante c ne dépendant pas des clefs de tour $K_1 \cdots K_{r-1}$ tels que :

$$D_V G(x) = c \quad \forall x \in \mathbb{F}_2^m$$

alors quelles que soient les clef de tours on aura la relation suivante :

$$\forall x \in \mathbb{F}_2^m, D_V G(x) = \bigoplus_{v \in V} G(x \oplus v) = c$$

Ce qui permet de réaliser l'attaque sur le dernier tour suivante :

- Choisir un message quelconque $x_0 \in \mathbb{F}_2^m$ et demander les chiffrés correspondant à l'ensemble des messages $\{x_0 \oplus v, v \in V\}$
- Calculer
$$c := \bigoplus_{v \in V} G(x_0 \oplus v)$$
avec des clefs de tour quelconques (par exemple toutes nulles)
- Déchiffrer sur le dernier tour avec chacune des clefs k_r candidates et calculer

$$a_{k_r} = \bigoplus_{v \in V} F_{k_r}^{-1}(E_K(x_0 \oplus v))$$

- Le bon candidat est k_r^* tel que $a_{k_r^*} = c$. Si plusieurs candidats sont possibles, choisir un autre vecteur de base x_0

2.3 Techniques d'Amélioration des Attaques Différentielles

Dans cette partie nous allons voir différentes méthodes possibles pour améliorer les performances des attaques différentielles.

2.3.1 Bits Neutres

Le terme *neutral bits* (bits neutres) à été introduit par Biham et Chen au sujet de la recherche de *presque-collisions* de la fonction de hachage SHA0 [BC04]. La définition donnée est la suivante :

Définition 12. *Bit neutre [BC04] :*
Soit (M,M') une paire de messages se conformant à un chemin différentiel. On dira que le ième bit des messages est un bit neutre pour M et M' si la paire de messages obtenue en modifiant simultanément la valeur du ième bit dans M et M' suit aussi le chemin différentiel.

Cette technique permet de dériver des paires de messages conformes à partir d'une seule paire et avec probabilité 1.

Le plus souvent les bits neutres ne couvriront qu'une partie du chemin différentiel ce qui permet tout de même d'accélérer la recherche de paires conformes puisque les paires dérivées seront bonnes avec probabilité plus élevée qu'une paire aléatoire. Cette technique a été utilisée avec succès dans une attaque sur les permutations de la fonction de hachage Luffa [KNPRS10] et sur le chiffrement à bloc *KLEIN* dans [ANPS11] que nous détaillerons dans la section 3.2.

2.3.2 Filtre du Dernier Tour

Pour limiter le nombre de candidats (messages, chiffrés) à étudier pour chaque hypothèse de sous clef, il est possible de construire un crible sur le dernier tour. En effet, connaissant la différence en sortie du chemin différentiel visé, on peut déduire l'ensemble des différences possibles en fin de chiffrement. Si la différence observée dans les chiffrés n'appartient pas à cet ensemble, on la rejette immédiatement. Cette opération permet de diminuer la complexité de l'attaque puisque c'est autant de paires de messages qu'on n'étudiera pas pour chacune des hypothèses de clef.

2.3.3 Défaire Plusieurs Tours

Une autre possibilité pour améliorer les performances de la cryptanalyse différentielle et notamment le nombre de tours attaqués consiste à élargir le nombre de tours partiellement déchiffrés. Un exemple de telle cryptanalyse a été réalisé sur le chiffrement par bloc PRESENT par Meiqin Wang dans [Wan08]. L'attaquant nécessite pour ce type d'attaque d'effectuer des hypothèses sur des portions des sous clefs des t derniers tours.

3 Le Chiffrement à Bas Coût *KLEIN*

Dans cette section je présente en détail l'apport de ce stage : il s'agit du travail réalisé par María Naya-Plasencia et moi-même au sujet du chiffrement symétrique *KLEIN*. Nous avons réussi à casser *KLEIN*-64, et ainsi à montrer que ce n'est pas un chiffrement fiable puisque la sécurité effective qu'il offre est inférieure aux 64 bits annoncés. Les concepteurs de *KLEIN* ont reconnu la validité de nos attaques.

Dans la première partie je vais introduire le chiffrement *KLEIN*, sa procédure de chiffrement et les différentes versions existantes, puis j'expliquerai une des précédentes attaques ; il s'agit de celle présentée à IndoCrypt 2011 par Aumasson et al [ANPS11]. Finalement, je détaillerai la nouvelle attaque que nous avons trouvée lors de ce stage.

3.1 Présentation de *KLEIN*

KLEIN est une famille de chiffrements par bloc proposée pour répondre aux nouveaux besoins définis par la cryptographie à bas coût. Présentée en 2011 à la conférence RFIDsec par Gong, Nikova et Law [GNL11], elle a été construite pour une application software mais possède aussi de très bonnes performances hardware.

La procédure de chiffrement proposée est un réseau de Substitution-Permutation qui agit sur des blocs de message de 64 bits. La famille comporte 3 variantes : *KLEIN*-64 qui réalise 12 tours et utilise une clef de 64 bits, *KLEIN*-80 qui itère 16 tours et nécessite une clef de 80 bits et finalement *KLEIN*-96 qui fait 20 tours avec une clef de 96 bits.

Le déroulement du chiffrement est très proche de celui de l'AES puisque dans 1 tour on y retrouve 4 opérations d'esprit similaire appelées *AddRoundKey*, *SubNibbles*, *RotateNibbles* et *MixNibbles*. La grande différence réside dans le fait que l'AES considère des messages de 128 bits et des boîtes S de 8 bits alors que KLEIN opère sur des plus petits blocs (64 bits) et des boîtes S de 4 bits.

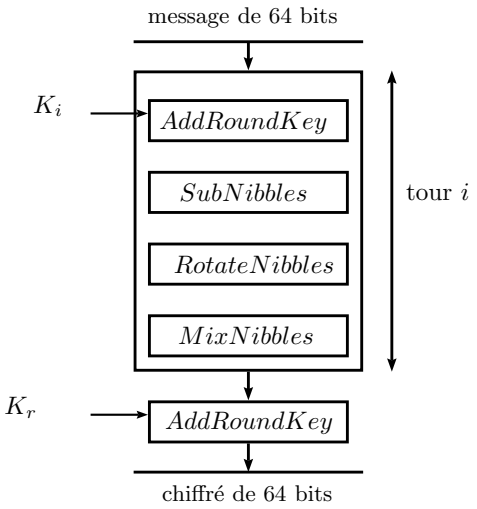


FIGURE 3 – Procédure de chiffrement de *KLEIN*. L'indice i va de 0 à $r-1$, où r vaut 12, 16 ou 20 selon la version

Le message clair, le chiffré et l'état intermédiaire font 64 bits et sont représentés sous forme d'un tableau en ligne. Chaque tour i commence par l'opération *AddRoundKey* qui consiste simplement en l'addition bit à bit de la clef de tour K_i avec l'état courant. Les clefs de tour sont déduites de la clef maître par l'algorithme de cadencement de clef qui sera explicité plus loin.

L'opération suivante est *SubNibbles*, qui applique la même boîte S à chaque regroupement de 4 bits de l'état que l'on nommera *quartet* (en anglais nibble, d'où le nom des opérations). Chaque quartet est traité indépendamment par la boîte S suivante :

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[x]	7	4	a	9	1	f	b	0	c	3	2	6	8	e	d	5

FIGURE 4 – Boîte S de *KLEIN*, notée en hexadécimal : la colonne *x=1, S[x]=4* signifie que le quartet 0001 est transformé en 0100 par la boîte S

On peut remarquer que la permutation définie par cette boîte S est une involution, ce qui permet d'économiser la mémoire nécessaire au stockage de son inverse nécessaire pour le déchiffrement.

L'opération suivante, *RotateNibbles*, réalise une rotation de l'état de 16 bits vers la gauche (2 octets). Son inverse est trivialement une rotation vers la droite de 16 bits.

Finalement, *MixNibbles* applique à chaque demi-état de 32 bits l'opération *MixColumn* de *Rijndael*. On rappelle que celle-ci est réalisée en considérant chaque octet comme un élément de $GF(2^8) = GF(2)/(x^8 + x^4 + x^3 + x + 1)$ et en multipliant le vecteur colonne composé des 4 octets par la matrice suivante :

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

Il est utile de remarquer que multiplier x par '02' revient à décaler son écriture binaire de 1 position vers la gauche si son MSB vaut 0 et que dans le cas contraire le résultat sera la somme de l'écriture décalée avec '1b'.

Une fois que l'ensemble des tours de la fonction de chiffrement a été effectué, un dernier ajout de clef est réalisé, donc au total la procédure nécessitera 1 clef de plus que le nombre de tours.

On peut faire ici une première remarque concernant les opérations de tours :

Propriété 1.
Toutes les opérations de tour sauf MixNibbles agissent indépendamment sur les quartets. De plus, les quartets bas restent en position basse et les quartets hauts restent en position haute.

Démonstration.
L'indépendance des quartets dans les opérations *AddRoundKey*, *SubNibbles* et *RotateNibbles* est triviale étant donnée la définition de ces opérations.

La deuxième partie de la propriété tient au fait que le nombre de bits de décalage effectué par *RotateNibbles* est multiple de 8 : cela implique qu'un octet prend la place d'un autre et que quartets hauts et bas conservent leur place relative. □

Algorithme de cadencement de clef

L'algorithme de cadencement de clef permet de calculer les clefs de tours à partir de la clef maître initiale de 64, 80 ou 96 bits. Cette dernière est aussi utilisée comme clef du premier tour K_0. Le cadencement de clef se fait en suivant un schéma de Feistel prenant en entrée la clef maître. Chaque moitié de la clef subit d'abord une rotation d'un octet vers la gauche, puis la partie de droite est XORée à celle de gauche pour former la nouvelle moitié de droite. Celle-ci traverse 4 boîtes S (correspondant aux octets aux positions 2 et 3). La partie de gauche est formée par la moitié de droite décalée à laquelle on somme bit à bit un compteur sur le 3ème octet. Ce compteur est l'indice du tour (commençant à 1). Ce procédé est représenté figure 5 pour le cas de *KLEIN*-64.

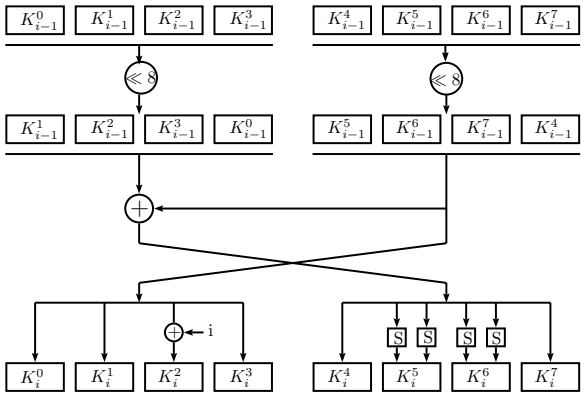


FIGURE 5 – Principe du calcul de la clef K_i à partir de la clef K_{i-1} de *KLEIN*-64. i varie de 1 à r et K_0 prend la valeur de la clef maître

Dans le cas où la clef maître fait 80 ou 96 bits, le cadencement de clef opère sur les 80 ou 96 bits mais seuls les 64 premiers bits de clef seront conservés pour être utilisés dans l'opération *AddRoundKey*.

Le pseudo code du cadencement de clef est détaillé dans l'*algorithme 1*, dans lequel t désigne la taille en octets de la clef maître (8, 10 ou 12 selon la version) et S désigne la boîte S explicitée figure 4. p représente la position dans l'état et l'indice i correspond à l'indice de la clef de tour : par exemple $K_i^{3,0}$ sera le quartet haut de l'octet en position trois de la ième clef de tour et $K_i^{3,1}$ son quartet bas).

Algorithm 1 Cadencement de clefs de *KLEIN*

Cadencement (Clef Maître MK)

$K_0 \leftarrow MK$
for i=1 to r **do**
  for p=0 to $\frac{t}{2} - 1$ **do**
    $K_i^p \leftarrow K_{i-1}^{((p+1) \mod \frac{t}{2})+\frac{t}{2}}$
    $K_i^{p+\frac{t}{2}} \leftarrow K_{i-1}^{((p+1) \mod \frac{t}{2})+\frac{t}{2}} \oplus K_{i-1}^{((p+1) \mod \frac{t}{2})}$
  end for
  $K_i^2 \leftarrow K_i^2 \oplus (i)$
  $K_i^{\frac{t}{2}+1,0} \leftarrow S[K_i^{\frac{t}{2}+1,0}]$
  $K_i^{\frac{t}{2}+1,1} \leftarrow S[K_i^{\frac{t}{2}+1,1}]$
  $K_i^{\frac{t}{2}+2,0} \leftarrow S[K_i^{\frac{t}{2}+2,0}]$
  $K_i^{\frac{t}{2}+2,1} \leftarrow S[K_i^{\frac{t}{2}+2,1}]$
end for

3.2 Cryptanalyses Précédentes

Plusieurs attaques sur des versions réduites ont été proposées suite à la parution de *KLEIN*, principalement sur la version avec 64 bits de clef. Récemment une cryptanalyse biclique atteignant les 12 tours de *KLEIN*-64 a été postée en Eprint ([ASA13]). Sa validité n'a pas été accordée par les concepteurs de *KLEIN* et de plus cette analyse ne peut pas vraiment être considérée comme une attaque puisque qu'elle se résume à une recherche exhaustive accélérée de la clef.

Les complexités de certaines des précédentes attaques sont récapitulées dans le tableau suivant :

version de *KLEIN*	Source	Nombre de tours	C. Données	C. Temps	C. Mémoire	Type d'attaque
KLEIN-64	[YWLZ11]	7	$2^{34.3}$	$2^{45.5}$	2^{32}	intégrale
	[YWLZ11]	8	2^{32}	$2^{46.8}$	2^{16}	tronquée
	[ANPS11]	8	2^{35}	2^{35}	-	tronquée
	[ASA13]	12	2^{39}	$2^{62.84}$	$2^{4.5}$	biclique
KLEIN-80	[YWLZ11]	8	$2^{34.3}$	$2^{77.5}$	2^{32}	intégrale

TABLE 2 – Récapitulatif des résultats des précédentes cryptanalyses

3.2.1 Observations et Propriétés Utilisées

Les attaques différentielles décrites dans [ANPS11, YWLZ11] et [Jha11] reposent sur des observations précises de la procédure de chiffrement utilisée dans *KLEIN*. Les cryptanalyses [ANPS11] et [YWLZ11] introduisent la propriété d'indépendance des quartets dans 3 opérations de tour sur 4 (propriété 1) ainsi que la propriété suivante concernant la quatrième étape *MixNibbles* :

Propriété 2.
Si la différence en entrée de MixColumn est de la forme '0X0X0X0X' où X désigne un

quartet de valeur quelconque et "0" désigne le quartet nul (0000) *alors la différence de sortie est de la même forme avec probabilité* 2^{-3}.

Il est commode d'introduire ici deux notations : on parlera de *quartet bas* pour désigner les 4 bits de poids faible d'un octet et de *quartets hauts* pour désigner ceux de poids fort. La propriété précédente peut donc se reformuler comme le fait que la différence reste dans les quartets bas après l'opération *MixColumn* avec probabilité 2^{-3}.

Démonstration.

Comme *MixColumn* est linéaire, le problème revient à s'intéresser à la probabilité d'obtenir un état avec des quartets hauts nuls étant donné un état de la même forme en entrée.

Développons le calcul du produit effectué dans *MixColumn*, en notant $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ le développement binaire d'un octet a (on fait le choix de placer les bits de poids fort à gauche) :

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} (0,0,0,0,a_4,a_5,a_6,a_7) \\ (0,0,0,0,b_4,b_5,b_6,b_7) \\ (0,0,0,0,c_4,c_5,c_6,c_7) \\ (0,0,0,0,d_4,d_5,d_6,d_7) \end{pmatrix} =$$

$$\begin{pmatrix} (0,0,0,a_4+b_4, a_5+b_4+b_5+c_4+d_4, a_6+b_5+b_6+c_5+d_5, a_7+b_6+b_7+c_6+d_6, b_7+c_7+d_7) \\ (0,0,0,b_4+c_4, a_4+b_5+c_4+c_5+d_4, a_5+b_6+c_5+c_6+d_5, a_6+b_7+c_6+c_7+d_6, a_7+c_7+d_7) \\ (0,0,0,c_4+d_4, a_4+b_4+c_5+d_4+d_5, a_5+b_5+c_6+d_5+d_6, a_6+b_6+c_7+d_6+d_7, a_7+b_7+d_7) \\ (0,0,0,a_4+d_4, a_4+a_5+b_4+c_4+d_5, a_5+a_6+b_5+c_5+d_6, a_6+a_7+b_6+c_6+d_7, a_7+b_7+c_7) \end{pmatrix} \quad (1)$$

On voit que les 3 bits de poids fort sont toujours nuls dans les 4 octets de sortie mais que pour que les 4 quartets hauts soient nuls il faut de plus que :

$$\begin{cases} a_4 + b_4 &= 0 \\ b_4 + c_4 &= 0 \\ c_4 + d_4 &= 0 \\ a_4 + d_4 &= 0 \end{cases}$$

Et comme la dernière équation est la somme des 3 premières on doit en fait satisfaire seulement 3 équations équivalentes à l'égalité entre a_4, b_4, c_4 et d_4 ce qui arrive avec probabilité 2^{-3}, soit une probabilité de 2^{-6} pour l'opération *MixNibbles* complète.

□

L'idée des précédentes cryptanalyses est de combiner cet événement de forte probabilité avec l'indépendance des quartets hauts et bas pour construire un chemin différentiel : en considérant en entrée de tour une différence portant uniquement sur les quartets bas, la forme de celle-ci sera conservée jusqu'à l'entrée de *MixNibbles* avec probabilité 1 (conséquence de la propriété 1) et passera l'étape *MixNibbles* avec probabilité 2^{-6}. On peut remarquer de plus que ce chemin a l'avantage d'être *itératif* : c'est à dire que la forme de sa différence d'entrée est la même que celle de sa sortie, ce qui permet de construire un chemin différentiel en enchaînant plusieurs fois cette propriété sur 1 tour.

Une autre propriété présentée dans [ANPS11, YWLZ11] concerne l'algorithme de cadencement de clef :

Propriété 3.
Lors du cadencement de clefs, les quartets bas et hauts ne sont pas mélangés : il est possible de calculer les quartets bas d'une clef de tour en possédant uniquement les quartets bas de la clef maître (ou dans le cas de KLEIN-64 les quartets bas d'une autre clef de tour).

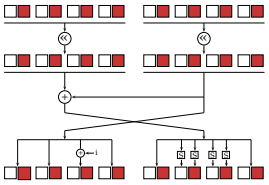


FIGURE 6 – Indépendance des quartets hauts et bas dans l'algorithme de cadencement de clef

3.2.2 Description de l'Attaque Réalisée dans [ANPS11]

L'attaque proposée en 2011 par Aumasson et al. est une attaque différentielle sur *KLEIN*-64. Nous allons expliquer ici le principe de l'attaque sur 7 tours et ensuite nous verrons comment les auteurs l'ont étendue à 8 tours.

Description du chemin différentiel utilisé
Le chemin différentiel tronqué utilisé est schématisé sur la figure 7 sur laquelle on représente un quartet par un carré ; les états internes sont composés de 16 carrés (pour 64 bits) tout comme le message clair (représenté tout en haut du chemin) et le chiffré (en bas de la figure). Les carrés rouges représentent les positions des différences. Les états représentés en face des abréviations RN, SN et MN sont respectivement les états en sortie des opérations *RotateNibbles*, *SubNibbles* et *MixNibbles*. L'opération *AddRoundKey* n'est pas représentée puisqu'elle n'influence pas le chemin différentiel.

Les quantités inscrites à droite du schéma représentent la probabilité que la différence se propage comme voulu sur le tour considéré. Les probabilités associées aux tours itératifs sont différentes de celles présentées dans l'article original [ANPS11] car nous les avons corrigées. En effet, lors de l'étude des précédentes attaques réalisées sur le système de chiffrement *KLEIN* nous avons pu constater une divergence sur le calcul des probabilités des chemins différentiels. L'article de Yu et al évalue la probabilité d'un tour itératif à 2^{-6} alors qu'elle est annoncée à $2^{-5.82}$ dans l'article d'Aumasson et al.

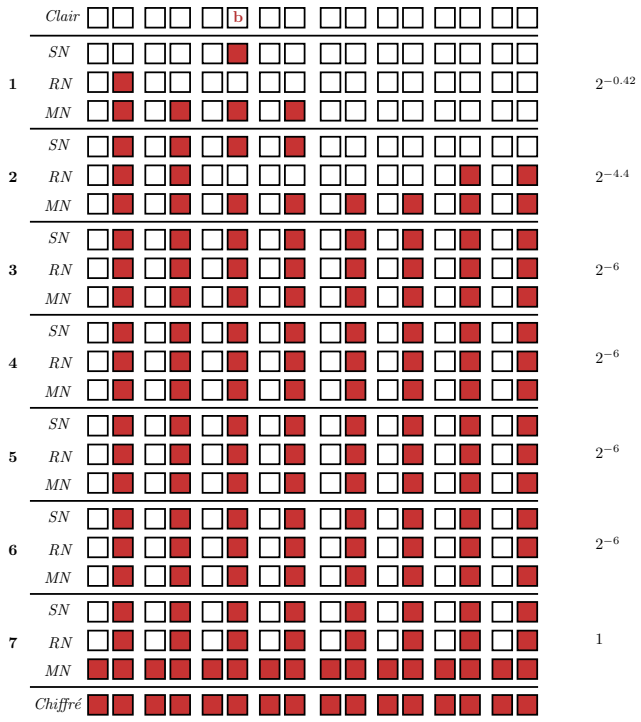


FIGURE 7 – Schéma différentiel utilisé dans [ANPS11]

Nous avons calculé et implémenté une série de tests qui nous ont amenées à la conclusion que la probabilité d'un tour itératif vaut 2^{-6} et que le fait que les différences d'entrée pouvaient être nulles avait un impact négligeable : nous considérons donc dans la suite que la probabilité d'un tour itératif est 2^{-6}.

Les calculs des probabilités de chaque tour se font comme suit :

- Pour que le premier tour soit traversé avec succès, il faut que la différence en entrée de *MixNibbles* vérifie la condition $a_4 = b_4 = c_4 = d_4$ vue dans la démonstration de la propriété 2. Cette propriété signifie que les 4 bits de poids fort des quartets bas de la différence doivent être égaux mais comme dans notre cas on a déjà $b_4 = c_4 = d_4 = 0$, cela se résume à obtenir $a_4 = 0$ soit une différence en sortie de *SubNibbles* inférieure strictement à $0x8$. C'est justement pour s'assurer une probabilité élevée de passer ce tour que les auteurs ont choisi d'imposer une différence d'entrée de valeur $0xb$: si on se réfère à la distribution des différences de la boîte S de *KLEIN* (cf annexe A) on remarque que la probabilité que la différence de sortie soit inférieure strictement à $0x8$ vaut $3/4 = 2^{-0.42}$.
- Comme le *branch number*[9] de *MixColumn* vaut 5, on est assuré d'avoir 4 quartets actifs en début de tour 2. Si on se concentre sur le *MixColumn* de droite, on remarque qu'on a $c_4 = d_4 = 0$ donc il suffit de s'assurer que les 2 premiers quartets bas de différence sont inférieurs strictement à $0x8$ pour passer. En se référant à la table des

9. Le branch number est une quantité décrivant la propagation des différences dans une opération. Dire que *MixColumn* possède un branch number de 5 signifie que deux messages différents auront au minimum 5 octets de différence en sommant sur l'entrée et la sortie : si une différence porte sur 1 octet en entrée on observera 4 octets différents en sortie.

différences de la boîte S, on voit que cet événement arrive dans 7 cas sur 15 pour chaque quartet soit une probabilité de $2^{-2.2}$ de passer le *MixColumn* de gauche et par symétrie une probabilité de $2^{-4.4}$ de passer le tour complet.
- Comme nous l'avons vérifié, la probabilité de passer un tour itératif (ici les tours 3,4,5 et 6) vaut 2^{-6}.
- Le dernier tour est laissé libre c'est-à-dire qu'aucune condition n'est imposé sur les différences. Cependant, comme les 3 premières opérations de tour agissent indépendamment sur chaque quartet, on peut remarquer que si une paire suit le chemin différentiel alors avec certitude la différence en entrée du dernier *MixNibbles* portera uniquement sur les quartets bas. En revanche, la différence après l'opération *MixColumn* peut porter sur l'ensemble des quartets.

Au final, la probabilité du chemin différentiel vaut donc $2^{-28.82}$.

L'objectif de l'attaque est de trouver une paire de messages qui suit le chemin différentiel pour en déduire de l'information. Si on teste assez de paires en entrée, il est assez facile de distinguer une bonne paire puisque :
- Une paire qui suit le chemin différentiel possède avec certitude une différence portant uniquement sur les quartets bas en entrée du dernier *MixNibbles*. La probabilité d'obtenir la différence voulue par ce chemin est donc $2^{-28.82}$.
- Si une paire ne suit pas le chemin différentiel alors la différence en entrée du dernier *MixNibbles* sera aléatoire et uniformément distribuée, donc on observera la différence attendue (qui comporte obligatoirement 32 positions à 0) avec probabilité $(\frac{1}{2})^{32} = 2^{-32}$.

Si on demande le chiffrement de $2^{28.82}$ paires on obtiendra avec bonne probabilité une paire conforme et celle-ci conduira à l'observation de la différence cherchée avant le dernier *MixNibbles*.

Puisque l'opération *AddRoundKey* n'influence pas sur la différence, l'attaquant peut accéder à la différence voulue en calculant pour chaque couple la valeur de l'inverse de la différence des chiffrés par *MixNibbles*. Il vérifie alors que la différence obtenue porte uniquement sur les quartets bas pour savoir si une paire est conforme ou non.

Ainsi, si l'attaquant demande le chiffrement de $2^{28.82}$ paires de messages il s'attend à en obtenir 1 qui suit le chemin différentiel et il peut la distinguer avec certitude en inversant par *MixNibbles* la différence des chiffrés.

Avant de continuer plus loin dans la description de l'attaque, faisons une remarque :

L'addition de clef finale κ peut être déplacée après la dernière opération *SubNibbles* avec l'addition de la clef correspondante. En effet puisque les opérations *MixNibbles* et *SubNibbles* sont linéaires, on peut poser
$\tilde{\kappa} = \mathit{RotateNibbles}^{-1}(\mathit{MixNibbles}^{-1}(\kappa))$
et on aura bien (avec les notations de la figure 8) :

$$\begin{aligned}
\mathit{MixNibbles}(\mathit{RotateNibbles}(E \oplus \tilde{\kappa})) &= \mathit{MixNibbles}(\mathit{RotateNibbles}(E \oplus \mathit{RotateNibbles}^{-1}(\mathit{MixNibbles}^{-1}(\kappa)))) \\
&= \mathit{MixNibbles}(\mathit{RotateNibbles}(E)) \oplus \mathit{MixNibbles}(\mathit{RotateNibbles}(\mathit{RotateNibbles}^{-1}(\mathit{MixNibbles}^{-1}(\kappa)))) \\
&= \mathit{MixNibbles}(\mathit{RotateNibbles}(E)) \oplus (\kappa) \\
&= \mathit{AddRoundKey}(\mathit{MixNibbles}(\mathit{RotateNibbles}(E)), \kappa) \\
&= C
\end{aligned}$$

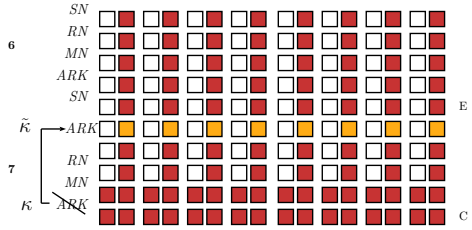


FIGURE 8 – Déplacement de l'addition finale

Déroulement

L'attaque proposée dans [ANPS11] procède de la façon suivante :

1. L'attaquant demande le chiffrement de $N = 2^{28.82}$ paires de messages (m_i^1, m_i^2), $i \in [\![1 \cdots N]\!]$vérifiant $m_i^1 \oplus m_i^2 = 0x00\ 00\ 0b\ 00\ \ 00\ 00\ 00\ 00$. On note (c_i^1, c_i^2) les chiffrés correspondants.

2. Il trouve ensuite la paire se conformant au chemin différentiel en calculant les valeurs de $MixNibbles^{-1}(c_i^1 \oplus c_i^2)$.

3. L'attaquant travaille ensuite uniquement avec la bonne paire qui lui sert à tester les 2^{32} valeurs possibles des quartets bas de $\tilde{\kappa}$. Si son hypothèse est correcte, lorsqu'il inversera en différence l'opération *MixNibbles* du tour 6 la différence obtenue se conformera au chemin différentiel (quartets hauts inactifs). Cependant, la probabilité d'inverser *MixNibbles* de sorte à obtenir une différence sur les quartets bas uniquement est 2^{-6} (voir annexe B.2) donc des mauvaises valeurs de clefs passeront aussi le test.

4. Pour réduire l'ensemble des clefs possibles, l'attaquant réalise les étapes précédentes avec plusieurs paires conformes au chemin différentiel. Chacune lui permettra d'éliminer des clefs qui passaient par chance et au final avec 6 paires ($2^{-36} \times 2^{32}$) il ne lui restera que la bonne clef.

5. Finalement, lorsque les quartets bas de $\tilde{\kappa}$ sont déterminés de façon sûre, l'attaquant parcours les 32 bits restant de $\tilde{\kappa}$ et en déduit la valeur correspondante de κ. Avec l'algorithme de cadencement de clef il obtient la valeur de toutes les clefs de tour et peut ainsi valider son hypothèse de clef en comparant un chiffré avec ses calculs.

La complexité totale en temps de cette attaque sur 7 tours est d'environ 2^{33} chiffrements et sa complexité en données est de $2^{28.82}$.

Il est possible d'appliquer cette attaque à 8 tours de *KLEIN*-64 en rajoutant un tour itératif au chemin différentiel vu précédemment. Cela fait passer la probabilité du chemin de $2^{-28.82}$ à $2^{-28.82-6} = 2^{-34.82}$. Comme cette quantité est inférieure à 2^{-32}, si on teste

comme précédemment l'inactivité des quartets bas en inversant *MixNibbles* des paires non conformes au chemin aléatoire vont réagir : c'est ce qu'on appelle des *fausses alarmes*.

Pour contourner ce problème, les auteurs proposent d'utiliser la technique des bits neutres introduite dans la section 2.3.1. En effet, ils font la remarque que les 2 premiers et les 2 derniers octets d'une paire de messages sont neutres dans les 2 premiers tours.

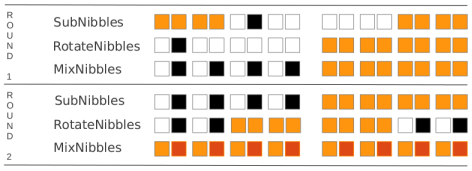


FIGURE 9 – Représentation de l'influence des bits neutres

Comme représenté en figure 9, lors du premier tour les 2 premiers et 2 derniers octets modifient les valeurs de la moitié d'état de droite. Comme celle-ci ne possède pas de différence, le chemin différentiel reste suivi après cette modification. Dans le second tour les modifications de valeur se propagent aux deux demi-états pour finalement modifier la valeurs des quartets de sorties du tour 2. Comme ceux-ci passent ensuite dans l'étape SubNibble qui n'est pas linéaire, on ne peut pas être assuré que les différences de sortie seront les mêmes ou seront propices au chemin différentiel à partir du 3ème tour.

Ainsi on a 32 bits neutres pour 2 tours ce qui permet de construire 2^{32} paires dérivées d'une paire conforme. Chacune conformera avec probabilité 1 le chemin différentiel dans les 2 premiers tours mais le reste du chemin ne sera assuré qu'avec probabilité 2^{-30} donc environ 4 paires sur les 2^{32} dérivées seront conformes.

Dans le cadre de cette attaque, les bits neutres permettent de décider si une paire initialement trouvée avec le filtre est conforme ou non : si en faisant varier les bits neutres on obtient aucune ou 1 nouvelle paire qui passe le filtre, on peut déduire que la paire était une fausse alarme car ce chiffre correspond à ce qui est attendu dans le cas générique. Si au contraire on obtient environ 4 nouvelles paires, on en déduit que la paire est bonne et ces 4 nouvelles paires permettent de plus de retrouver la clef plus vite. Une fois qu'on est assuré qu'une paire n'est pas une fausse alarme, on procède à la même attaque que précédemment. La complexité totale de l'attaque avoisine les 2^{35} chiffrements.

4 Notre Nouvelle Attaque Améliorée sur *KLEIN*

L'objectif de la nouvelle proposition d'attaque était bien évidemment de pouvoir atteindre plus de tours et idéalement les 12 tours que compte *KLEIN*-64. Pour cela, l'idée de base était de réaliser une attaque différentielle avec suffisamment de filtres pour pouvoir en dépit de la diminution de la probabilité du chemin différentiel distinguer univoquement une bonne paire et grâce à elle la bonne clef.

Il faut cependant garder à l'esprit que si on attaque plus de tours on aura inévitablement une diminution de la probabilité qu'une paire suive le chemin différentiel : si par exemple on décide de rajouter des tours itératifs en fin du précédent chemin différentiel, la probabilité sera multipliée par 2^{-6} à chaque tour supplémentaire. Cela aura comme conséquence de demander plus de données pour que l'attaquant puisse être certain d'avoir une paire conforme et d'autre part il faudra gérer plus de fausses alarmes.

Plusieurs options peuvent être envisagées pour faire face à ces problèmes : on peut modifier le chemin différentiel, modifier la configuration du dernier tour pour obtenir un filtre final plus grand ou encore utiliser comme précédemment une technique du type *bits neutres*.

Nous avons exploré toutes ces pistes et finalement nous avons réussi à monter une famille d'attaques sur les 12 tours de *KLEIN*-64 ; prouvant que cette primitive n'est pas sûre. L'attaque sur *KLEIN*-64 est décrite dans les sections suivantes. Nous verrons ensuite comment étendre ces attaques à des versions réduites de *KLEIN-80* et de *KLEIN-96*.

Ajoutons ici que nous avons envoyé notre proposition d'attaque aux concepteurs de *KLEIN* et que ceux-ci ont reconnu sa validité.

4.1 Idée Centrale

L'idée centrale de notre nouvelle attaque est d'utiliser les tours précédents pour filtrer les candidats ; l'attaque développée par Aumasson et al. proposait d'utiliser l'avant dernière opération MixNibble pour décider si une clef était possible ou non et d'utiliser plusieurs paires conformes pour filtrer suffisamment. Ici, comme notre chemin sera de très faible probabilité, trouver plusieurs paires conformes sera très difficile et impliquera une trop grande complexité en données. C'est pourquoi l'idée retenue est de filtrer les clefs candidates avec 1 seule paire et d'utiliser le filtre des étapes *MixNibbles* précédentes.

Nous allons à présent montrer comment inverser les tours pour connaître les différences nécessaires à l'application du filtre.

Comment Inverser un Tour pour Parvenir à un Autre Filtre

On rappelle que l'opération de filtre consiste à observer la valeur de la différence en sortie de *MixNibbles* pour déduire la valeur de celle-ci en entrée de l'opération : si la différence d'entrée porte uniquement sur les quartets bas, la clef est considérée comme probable, sinon, elle est rejetée.

Considérons que la situation de départ est celle illustrée dans la figure 10 : on possède les quartets bas d'une clef candidate associée à une paire d'états dont on connaît la valeur au point A et celle-ci a passé le filtre du *MixNibbles* représenté tout en bas de la figure 10. On souhaite connaître la valeur de la différence au point E pour pouvoir appliquer le filtre.

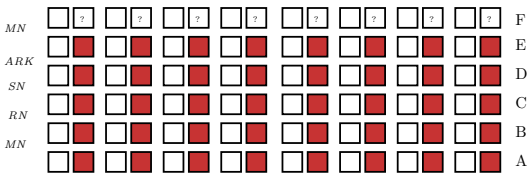


FIGURE 10 – Configuration de départ

Pour cela il va nous falloir inverser les opérations *MixNibbles*, *RotateNibbles*, *SubNibbles* et *AddRoundKey* en valeur pour les quartets bas. En effet l'opération *SubNibbles* (non linéaire) nous empêche de déduire la différence voulue de celle détenue et il nous est donc nécessaire de disposer des valeurs des états au point C.

On peut remarquer cependant que la différence cherchée porte uniquement sur les quartets bas par le chemin différentiel, ce qui implique que seules les valeurs des quartets bas nous intéressent au point C. Pour connaître celles-ci il faut connaître les valeurs des quartets bas en entrée du *MixNibbles* au point B.

Au minimum il semble donc que l'attaquant doive connaître les valeurs des quartets bas en sortie de l'opération *SubNibbles* : il peut ensuite en déduire la différence en sortie de *MixNibbles* et utiliser le filtre. Cependant il faut garder à l'esprit que l'objectif est d'utiliser ce procédé plusieurs fois dans l'optique de filtrer un maximum. Cela signifie qu'une fois ce tour vérifié on souhaite vérifier le précédent avec le même procédé. Cette condition impose de connaître à chaque étape la valeur des quartets bas donc de pouvoir inverser *AddRoundKey*. Heureusement les propriétés 3 et 1 nous assurent que si on possède une hypothèse des quartets bas d'une clef cela se fait sans problème.

De même l'étape *RotateNibbles* est inversée trivialement puisqu'elle opère un décalage multiple de 8 bits donc assure que les quartets bas d'un état deviennent des quartets bas après rotation. La seule opération délicate est l'inversion de *MixNibbles* qui n'agit pas indépendamment sur les quartets. Cependant, nous avons prouvé la propriété suivante, reliée à la propriété 2 :

Propriété 4.

Pour obtenir les quartets bas résultant de l'inversion de l'opération MixColumn connaissant les quartets bas de l'entrée (e, f, g, h), il est nécessaire de connaître 3 valeurs dépendant des quartets hauts inconnus :

$$\begin{cases} e_1 + e_2 + f_2 + g_0 + g_1 + g_2 + h_0 + h_2 \\ e_1 + f_0 + f_1 + g_1 + h_0 + h_1 \\ e_0 + e_1 + e_2 + f_0 + f_2 + g_1 + g_2 + h_2, \end{cases}$$

où on dénote le développement binaire de l'octet a par $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ *et le bit de poids fort par* a_0.

Démonstration. Voir annexe B.3. □

Cette proposition nous assure qu'on peut inverser un tour complet pour obtenir les quartets bas en entrée avec uniquement la donnée des quartets bas de l'état de sortie, des 32 bits

de quartets bas de la clef et de 6 bits sans nécessiter la connaissance des 64 bits de l'état entier. Au prix de 6 bits d'hypothèse par tour on peut donc inverser et filtrer chaque tour.

Comme prouvé en annexe B.1, il existe de même 3 bits permettant de calculer les quartets bas en sortie de *MixColumn*. Cela implique que cette méthode consistant à réaliser une hypothèse de 6 bits peut être utilisée indifféremment soit en remontant soit en descendant dans le schéma de chiffrement.

Il faut cependant rester prudent sur l'intérêt pratique de cette propriété : la valeur des 6 bits n'étant pas connue l'attaquant devra envisager toutes les possibilités, soit 2^6 valeurs possibles pour les quartets bas des états après inversion de MN. Chaque tour nous permettant d'obtenir un filtre de probabilité 2^{-6} en utilisant l'opération *MixNibbles*, le nombre de candidats restera le même. Le fait de remonter des tours n'augmente donc pas le nombre de candidats possibles. Nous verrons dans la prochaine section comment réduire le nombre de candidats.

4.2 Déroulement

La nouvelle attaque différentielle proposée repose sur le chemin différentiel tronqué représenté en figure 11 utilisant les propriétés 1, 2 et 3 d'indépendance entre quartets qui permettent notamment d'obtenir une probabilité de suivre le chemin différentiel sur les 12 tours qui est raisonnable ($2^{-69.5}$). Nous verrons plus loin qu'il est possible de construire d'autres chemins différentiels selon l'objectif visé (rapidité de l'attaque, limitation du nombre de données ...).

Sur ce chemin différentiel, on impose de prendre 2 messages d'entrée possédant une différence non nulle positionnée dans les 2 premiers quartets bas et les 2 derniers donc de la forme $0X\ 0X\ 00\ 00\ \ 00\ 00\ 0X\ 0X$ où X est un mot de 4 bits pouvant prendre toutes les valeurs possibles. Ce choix à été fait pour que, lors du passage dans la première opération *RotateNibbles*, les 4 quartets de différence entrent dans un unique *MixColumn* (celui de droite). On impose ensuite que la différence après *MixColumn* porte sur 1 seul quartet en position 3 ce qui arrive en fait avec une seule valeur de différence parmi les 2^{16} possibles. Cette différence correspond à $0x0d\ 0b\ 0e\ 09$ qui mène à la différence $0x00\ 00\ 01\ 00$.

Étape 1 - Chiffrements initiaux

En premier lieu, l'attaquant doit obtenir les chiffrés correspondant à $2^{69.5}$ paires de messages avec la différence voulue en entrée. Pour limiter la quantité de données nécessaire, il peut utiliser l'astuce déjà évoquée dans [BS91] consistant à utiliser 1 même message pour construire des paires différentes : puisqu'en effet toutes les différences d'entrée portant sur les 2 premiers et 2 derniers quartets sont autorisées, un message a peut servir pour construire toutes les paires allant de $(a, a \oplus 00\ 00\ 00\ 00\ \ 00\ 00\ 00\ 01)$ à $(a, a \oplus 0f\ 0f\ 00\ 00\ \ 00\ 00\ 0f\ 0f)$ soit $2^{16} - 1$ paires. Une autre façon de voir est qu'avec n messages tous différents en les 4 quartets actifs en entrée et égaux ailleurs on peut former $\frac{n \times (n-1)}{2}$ paires de messages puisqu'il suffit de choisir le premier message (n choix possibles) puis le second qui lui est différent ($n - 1$ choix) mais en prenant en compte que les paires (a, b) et (b, a) sont identiques (d'où la division par 2).

Au maximum, puisqu'il faut que 2 messages ne différent que sur les 4 quartets, on peut

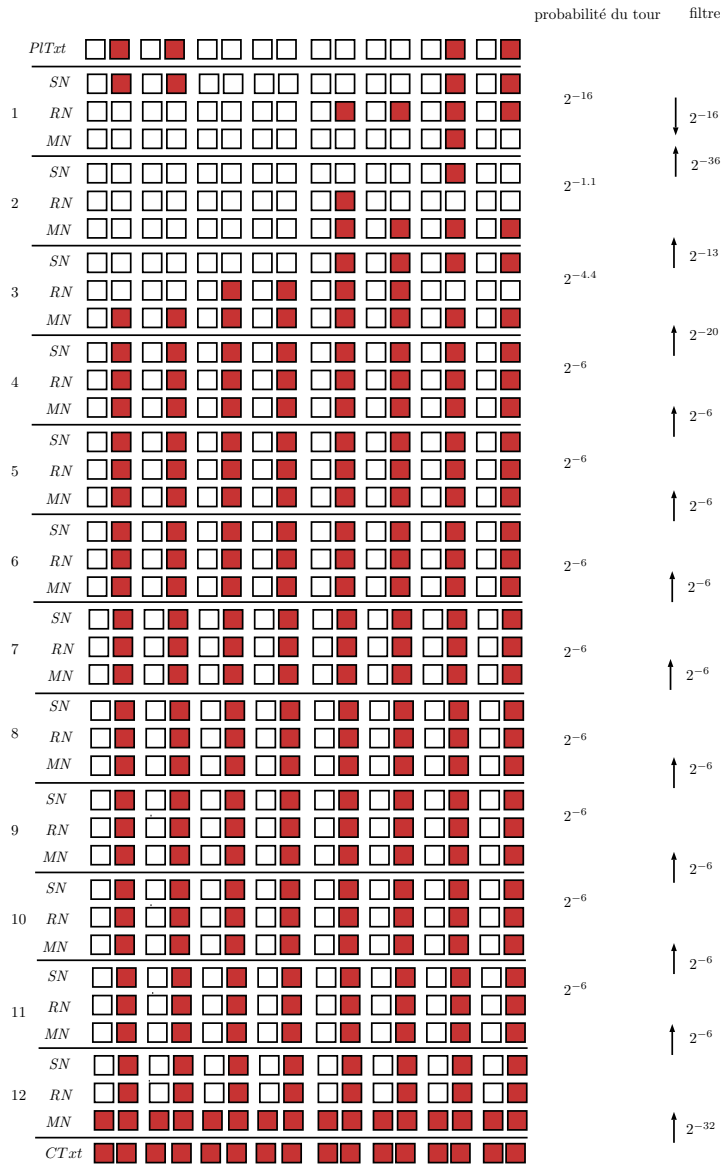


FIGURE 11 – Chemin différentiel (même notations et conventions qu'en figure 7)

construire autant de messages qu'il y a de places pouvant contenir des différences soit ici 2^{16}. Les 48 autres positions, doivent être fixées et communes. Cette notion est connue sous le nom de *structure*.

Propriété 5.
Si le nombre de positions de différences est $|\Delta_{in}|$, une structure contient $2^{|\Delta_{in}|}$ messages et permet de construire environ $2^{2|\Delta_{in}|-1}$ paires

Dans notre cas, une structure permettra de construire 2^{31} paires en ne nécessitant que 2^{16} chiffrements. Comme nous avons besoin de $2^{69.5}$ paires, il va falloir construire $\frac{2^{69.5}}{2^{31}} = 2^{38.5}$ structures (C'est possible puisque au maximum on peut en former $2^{64-16} = 2^{48}$). Au total cette opération nécessitera $2^{38.5} \times 2^{16} = 2^{54.5}$ chiffrements complets.

Une fois que les données ont été générées, (ou plutôt au fur et à mesure que les structures sont générées et jusqu'à ce que la clef soit trouvée ce qui nécessitera en moyenne $2^{38.5}$ structures), on va utiliser le dernier tour pour effectuer un premier filtrage. Comme remarqué dans [ANPS11, YWLZ11], si une paire de messages vérifie le chemin différentiel alors la différence entrant le *MixNibbles* du dernier tour du schéma (laissé libre) aura ses quartets hauts inactifs. Ce test se fait comme dans la précédente attaque en inversant *MixNibbles* à partir des chiffrés. Comme on vérifie l'inactivité de 32 bits, la probabilité qu'une paire passe le test est 2^{-32}, donc on conservera en moyenne $2^{69.5} \times 2^{-32} = 2^{37.5}$ paires candidates (soit 1 pour 2 structures analysées).

Étape 2 - Test des hypothèses de clef avec les conditions du premier tour

Les prochaines étapes de l'attaque reposent sur l'utilisation d'autres étapes *MixNibbles* et nécessitent d'inverser ou de chiffrer en quartets bas des tours de l'algorithme et en particulier l'opération *AddRoundKey*. Avant de continuer l'attaque il faut donc réaliser ici une hypothèse de clef.

L'attaquant a deux options : soit il réalise une hypothèse sur K_0 pour calculer le 1er tour et utiliser le filtre de l'opération *MixNibbles* du 1er tour, soit il devine K_{12} pour inverser le dernier tour et utiliser le filtre associé à l'avant dernier *MixNibbles*. Le coût des deux options est le même mais les valeurs des filtres associés différent : les différences du premier tour sont conformes au chemin différentiel avec probabilité 2^{-16} alors que la conformité de l'avant dernier tour a une probabilité de 2^{-6} donc il est plus avantageux d'effectuer en premier une hypothèse sur K_0. Ce choix permet de diminuer le coût des opérations suivantes puisqu'on aura moins de candidats à étudier dans la suite, donc moins d'opérations et une complexité en temps moindre.

L'attaquant effectue donc pour chaque paire une hypothèse sur les quartets bas de K_0, calcule *AddRoundKey*, *SubNibbles* puis *RotateNibbles* à partir des messages clairs et en déduit la valeur de la différence en entrée de *MixNibbles*.

Remarque
*À partir de ce moment les **candidats** examinés sont en fait des ensembles constitués de la paire de messages clairs combinée aux hypothèses effectuées durant l'attaque : les quartets bas de K_0 mais aussi les valeurs des 6 bits utilisés pour inverser ou chiffrer les tours en quartets bas.*

Comme on possède à ce stade $2^{37.5}$ paires ayant passé le filtre du dernier tour et que pour chaque paire toutes les valeurs de quartets bas de K_0 sont possibles, l'attaquant possède $2^{37.5+32} = 2^{69.5}$ candidats formés des 2 messages et des quartets bas de clef devinés.

Il calcule ensuite la valeur de la différence sortant du premier *MixNibbles* et obtient la différence voulue avec probabilité 2^{-16} : il lui reste donc $2^{69.5-16} = 2^{53.5}$ candidats.

Cette opération nécessiterait à priori $2^{69.5} \times 2$ chiffrements de tour (puisqu'il faut calculer le tour pour chacun des 2 messages de chaque candidat), ce qui dépasse la limite des $2^{64} * 12$ chiffrements nécessaires à une recherche exhaustive de clef. Nous verrons dans la partie suivante la méthode que nous avons trouvée pour contourner ce problème.

Ces premières étapes ne sont pas encore suffisantes pour déterminer univoquement la paire conforme et la clef associée donc l'attaquant doit continuer à utiliser les filtres fournis par les étapes *MixNibles*.

Comme précédemment il a le choix entre continuer à descendre dans le schéma et remonter. S'il décide de descendre, il aura un filtre de $2^{-1.1}$ alors que s'il remonte le filtre sera de 2^{-6} : il est préférable de déchiffrer.

Étape 3 - Déchiffrement des tours 12 à 5

En application de la propriété 5, on peut déduire directement les valeurs des quartets bas de K_{12} et de toutes les autres sous-clefs à partir de l'hypothèse sur K_0. Les déductions sur K_{12} permettent de remonter la dernière opération *AddRoundKey* et d'obtenir la valeur des quartets bas en sortie du dernier *MixNibbles*.

L'attaquant inverse en quartets bas le dernier tour avec la procédure décrite en 4.1 qui nécessite une hypothèse supplémentaire de 6 bits si bien que chaque ensemble candidat contient à présent la paire de messages, l'hypothèse de clef et les valeurs des 6 bits de conditions. Comme on doit envisager les 2^6 possibilités pour chaque candidat, leur nombre passe à $2^{53.5+6} = 2^{59.5}$. Pour chacun il faut évaluer la différence en entrée de l'étape *MixNibbles* ce qui demande d'inverser un tour complet 2 fois : cette opération nécessite $2^{59.5} \times 2$ chiffrements de tour. Là encore nous avons trouvé une façon de diminuer le coût. Ces optimisations serons présentés dans la section suivante.

L'hypothèse faite sur les 6 bits de condition augmente le nombre de candidats à étudier mais étant donné que le filtre utilisé juste ensuite a comme probabilité 2^{-6} on est assuré que le nombre de candidats avant et après cette étape est environ le même.

L'attaquant continue à inverser les tours de la même façon pour les $2^{53.5}$ quadruplets restants, en faisant une hypothèse de 6 bits pour inverser en quartets bas le *MixNibbles* du tour i qu'il filtre avec le *MixNibbles* du tour $i + 1$. Ce procédé est répété pour tous les tours itératifs c'est à dire pour i de 12 à 5 (cf figure 11) et aura un coût total de $2^{59.5} \times 2 \times 7$ chiffrements de tour.

Étape 4 et 5 - Déchiffrement du tour 4 et du tour 3

Les tours suivants ne sont plus itératifs et offrent des filtres plus puissants comme décrits en figure 11 : une fois qu'on a inversé en quartets bas l'opération *MixNibbles* du tour 4, la différence à vérifier au tour 3 avant *MixNibbles* possède les quartets hauts nuls mais aussi 4 quartets bas nuls. Comme prouvé en annexe B.4, la probabilité d'obtenir cette forme de différence est 2^{-20} donc il nous reste $2^{53.5+6-20} = 2^{39.5}$ candidats au prix non optimisé de $2^{59.5} \times 2$ chiffrements de tour.

Le tour suivant offre un filtre de 2^{-13} et coûte $2^{45.5} \times 2$ chiffrements de tour et il reste $2^{39.5+6-13} = 2^{32.5}$ candidats.

Étapes 6 et 7 - Déchiffrement du tour 2 et chiffrement du tour 1

Avec une hypothèse de 6 bits supplémentaires et $2^{38.5} \times 2$ chiffrements on obtient les valeurs possibles des quartets bas de l'état avant l'opération *MixNibbles* du tour 2. On peut ensuite inverser les opérations *RotateNibbles*, *SubNibbles* et *AddRoundKey* et obtenir les quartets bas des états en entrée de tour 2. Ces valeurs sont à confronter à celles que l'on obtient en descendant le schéma de chiffrement (avec aussi $2^{38.5} \times 2$ chiffrements) et on teste donc une égalité sur 36 bits (32 bits de valeur et 4 bits de différence). Après avoir utilisé l'ensemble des 12 opérations *MixNibbles*, il reste $2^{32.5+6+6-36} = 2^{8.5}$ candidats.

Étape 8 - Recherche des quartets hauts

Pour départager les quelques 362 candidats restants, on fait intervenir des hypothèses sur les quartets hauts de clef : ceux-ci permettent d'obtenir des filtres supplémentaires.

Une première technique possible consiste a effectuer une hypothèse sur les 2^{32} quartets hauts de la clef, à chiffrer la paire et à comparer les 2 valeurs obtenues avec celles des chiffrés. Seule une paire conforme associée à la bonne clef passera ce test. Cela nécessite $2^{32} \times 2^{8.5}$ chiffrements mais encore une fois nous verrons une façon d'en diminuer la complexité.

L'attaque totale dans cette version nécessite $2^{70.51}$ chiffrements de tour soit l'équivalent de 2^{67} chiffrements complets. C'est beaucoup trop, d'où la nécessité des optimisations expliquées dans le paragraphe suivant.

4.3 Complexité et Optimisation

Nous avons proposé de nombreuses optimisations pour améliorer la complexité de l'attaque de base. On a déjà vu qu'on pouvait diminuer la complexité en données en utilisant des structures, mais il est aussi possible de limiter le nombre de calculs effectués au cours de l'attaque de sorte à obtenir une complexité totale en dessous de celle de la recherche exhaustive. Cela nous a permis de monter une attaque valide et de casser *KLEIN*-64.

Optimisation 1

La complexité de l'attaque présentée plus haut est dominée par le coût de l'étape 2 qui consiste à parcourir les 2^{32} valeurs de quartets bas possibles pour la clef maître et cela pour chacun des $2^{37.5}$ candidats. Elle nécessite $2^{70.5}$ chiffrements de tour et rend l'attaque moins performante que la recherche exhaustive. C'est donc celle-ci qu'il faut optimiser en priorité.

L'idée de l'optimisation que nous proposons est de tirer profit du fait que les étapes *AddRoundKey*ey, *SubNibbles* et *RotateNibbles* agissent indépendamment sur les quartets des états et que *MixNibbles* agit indépendamment sur chaque moitié d'état pour pouvoir tester des morceaux de clefs plus courts que 32 bits.

Plus en détail, on a vu que le premier filtrage des clefs se faisait en fonction de la conformité de la différence au premier tour et plus particulièrement à l'étape *MixColumn* traversée par le demi-état de droite. Au contraire, aucune restriction n'est faite sur les différences en sortie du *MixColumn* de gauche. L'idée est donc de considérer uniquement les bits de clef intervenant dans le *MixColumn* de droite (les 2 premiers et 2 derniers quartets bas) et ainsi de calculer uniquement 2^{16} fois sur *un demi-état* le premier tour pour 1 paire donnée au lieu de tester 2^{32} clefs avec autant de chiffrements de tour complet.

Une fois que les 2^{16} valeurs des 4 quartets des extrémités de la clef ont étés testés et filtrés avec la condition du premier tour (2^{-16}), on forme l'hypothèse de clef complète en

ajoutant les quartets bas de clefs restant qui eux peuvent prendre l'ensemble des 2^{16} valeurs possibles.

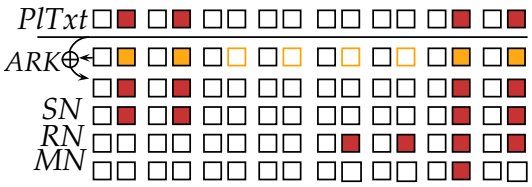


FIGURE 12 – Situation du premier tour

Cette méthode permet de n'effectuer que $2^{16} \times 2^{37.5} = 2^{55.5}$ chiffrements de tour au lieu des $2^{32} \times 2^{37.5} \times 2$ initiaux et ainsi d'avoir une complexité inférieure à celle de la recherche exhaustive ($2^{64} \times 12$). Les valeurs des filtres n'ont pas changé donc on obtient comme précédemment $2^{53.5}$ candidats possibles à la fin de cette étape.

Optimisation 2

Une autre optimisation possible consiste à diminuer le coût du parcours des 2^6 possibilités de valeur des bits de conditions. La version naïve consiste à parcourir les 2^6 valeurs possibles, à en déduire les valeurs des quartets bas, à inverser *RotateNibbles*, *SubNibbles* et *AddRoundKey* pour finalement tester la différence en inversant *MixNibbles*.

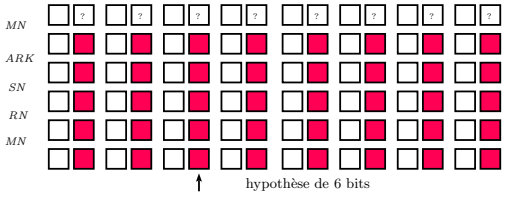


FIGURE 13 – Version naïve

Pour limiter les calculs, l'idée est de réaliser les 2^3 hypothèses possibles pour l'opération *MixColumn* de gauche et indépendamment les 2^3 hypothèses possibles pour celle de droite et ensuite de les combiner pour inverser l'opération *MixNibbles* du tour précédent. Une telle procédure permet d'inverser seulement $2 \times 2^3 \times 2$ fois les opérations *MixColumn* alors que la version naïve demande $2^6 \times 2$ inversions de tour : on gagne un facteur 8 en introduisant le stockage de 2^3 valeurs de quartets bas de demi-états (soit $2^3 \times$ 16 bits).

Pour éviter de faire 2^6 recombinaisons puis inversions de l'opération *MixNibbles*, on peut remarquer qu'inverser cette opération de sorte à obtenir aucune différence dans les quartets hauts de la différence revient à vérifier un système dépendant de certains bits de a, b, c et d de la différence d'entrée de $MixColumn^{-1}$ (figure 14) :

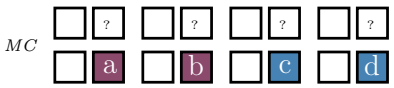


FIGURE 14 – La conformité de la différence obtenue après inversion de *MixColumn* dépend de certains bits des octets a, b, c et d de la différence

Comme prouvé en annexe B.2, les équations à vérifier sont les suivantes :

$$\begin{cases} a_4 + b_4 + c_4 + d_4 = 0 \\ a_4 + a_5 + b_5 + c_4 + c_5 + d_5 = 0 \\ a_4 + a_5 + a_6 + b_4 + b_6 + c_5 + c_6 + d_6 = 0 \end{cases}$$

L'idée est alors de calculer ces relations au lieu d'inverser *MixNibbles* pour rendre le filtre plus rapide.

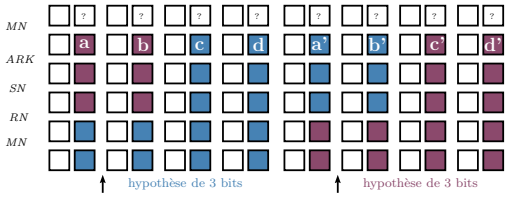


FIGURE 15 – Version optimisée

Comme représenté figure 15, les 3 bits d'hypothèse nécessaires à l'inversion du *MixColumn* de gauche permettent de calculer les valeurs de c, d, a' et b', donc de calculer les valeurs suivantes :

$$\begin{aligned} a'_4 + b'_4 \\ a'_4 + a'_5 + b'_5 \\ a'_4 + a'_5 + a'_6 + b'_4 + b'_6 \end{aligned} \qquad \begin{aligned} c_4 + d_4 \\ c_4 + c_5 + d_5 \\ c_5 + c_6 + d_6 \end{aligned}$$

On effectue ensuite les 2^3 hypothèses possibles pour le *MixColumn* de droite et on calcule au fur et à mesure les 6 bits :

$$\begin{aligned} c'_4 + d'_4 \\ c'_4 + c'_5 + d'_5 \\ c'_5 + c'_6 + d'_6 \end{aligned} \qquad \begin{aligned} a_4 + b_4 \\ a_4 + a_5 + b_5 \\ a_4 + a_5 + a_6 + b_4 + b_6 \end{aligned}$$

On chercher ensuite à égaler ces 6 bits dans cet ordre avec les 6 de la précédente liste pour chercher à savoir si le système de conditions nécessaires à l'inversion est vérifié, ce qui arrive avec une probabilité de 2^{-6}. Au final on aura effectué 2^6 comparaisons au lieu de 2^6 inversions et la complexité en temps de cette étape passera de $2^{53.5} \times 2^6 \times 2$ à $2^{53.5} \times 2^4$.

Optimisation 3

Finalement, une méthode plus efficace et moins coûteuse pour trouver les quartets hauts consiste à utiliser les valeurs des bits de conditions liés à un candidat : une fois qu'on a réalisé une hypothèse sur les quartets hauts, on chiffre la paire de messages jusqu'à atteindre l'état en sortie de *MixNibbles* sur lequel nous avons réalisé les 3 bits d'hypothèse lors de la recherche des quartets hauts. On compare alors les bits d'hypothèse à l'état obtenu ici ce qui nous permet d'avoir un filtre de 6 bits à chaque tour.

À partir des $2^{8.5}$ quadruplets possibles on construit $2^{8.5+32} = 2^{40.5}$ candidats cette fois composés de la paire de messages, des quartets bas de clef, des bits de condition et des quartets hauts de clef. On dispose de 12 tours donc d'un filtre total de $2^{-6*12} = 2^{-72}$ ce qui assure qu'on ne conservera que le bon candidat.

On peut remarquer que la complexité de cette procédure est dominée par le coût du premier tour étant donné qu'on y teste le plus grand nombre de candidats.

L'indépendance des mots de 32 bits de *MixNibbles* va encore une fois nous permettre d'optimiser cette étape, et de la même manière que précédemment on pourra tester indépendamment 16 quartets hauts à la fois, correspondant aux bits de clef intervenant dans un même *MixColumn*. La complexité du premier tour passera de $2^{8.5} \times (2^{32} \times 2) = 2^{41.5}$ chiffrements du premier tour à $2^{8.5} \times ((2^{16} + 2^{16})) = 2^{25.5}$.

La procédure totale aura un coût de : $2^{8.5} \times 2 \times (2^{16} * 2 + 2^{26} + 2^{20} + 2^{14} + 2^8 + 2^2) \simeq 2^{8.5} \times 2 \times 2^{26.0} \simeq 2^{35.8}$ (soit $2^{22,4}$ chiffrements complets).

Cet ensemble d'améliorations permet de faire passer la complexité de l'attaque de $2^{70.51}$ à $2^{60.9}$ chiffrements de tour (soit $2^{57,31}$ chiffrements complets) comme décrit dans le tableau suivant. La sécurité de *KLEIN*-64 s'en trouve réduite de 64 bits à 57,31.

Étape	chiffrements de tour	hypothèses faites	Filtre utilisé	candidats restants
1. chiffrements initiaux	$2^{54.5} \times 12$	0	2^{-32}	$2^{69.5-32} = 2^{37.5}$
2. test clefs 1er tour	$2^{53.5}$	32	2^{-16}	$2^{53.5}$
3. déchiffrement tours 12 à 5	$2^{53.5} \times 2^4 \times 8$	6×8	2^{-6} (8 fois)	$2^{53.5}$
4. déchiffrement tour 4	$2^{53.5} \times 2^4$	6	2^{-20}	$2^{39.5}$
5. déchiffrement tour 3	$2^{39.5} \times 2^4$	6	2^{-13}	$2^{32.5}$
6. déchiffrement tour 2	$2^{32.5} \times 2^4$	6	0	$2^{32.5}$
7. chiffrement tour 1	$2^{32.5} \times 2^4$	6	2^{-36}	$2^{8.5}$
8. rechch des quartets hauts	$2^{35.8}$	32	2^{-6} / tour	1
attaque totale	$2^{60.90}$			

FIGURE 16 – Complexité finale de l'attaque. La complexité totale équivaut à $2^{57.31}$ chiffrements par *KLEIN*-64, à comparer aux 2^{64} de la recherche exhaustive.

4.4 Autres Compromis Possibles

Il est possible d'obtenir d'autres compromis entre les complexités en donnée, en temps et en mémoire. Pour cela, on peut notamment modifier les conditions imposées sur les 4 premiers tours du chemin différentiel.

Les éléments à prendre en compte sont les suivants :

– Modifier le chemin différentiel modifie la probabilité de trouver une paire conforme, donc la quantité de données nécessaires en début d'attaque
– Si le nombre de quartets actifs change, la valeur des filtres sera modifiée donc le nombre de paires que l'on peut éliminer à chaque inversion sera différent et avec elle la rapidité de l'attaque
– La taille de la différence d'entrée impacte la taille des structures donc la quantité de données ainsi que la mémoire nécessaires.

Les 4 chemins que nous avons retenus sont ceux représentés figure 17 (le cas 1 est celui qui a été présenté en détail précédemment).

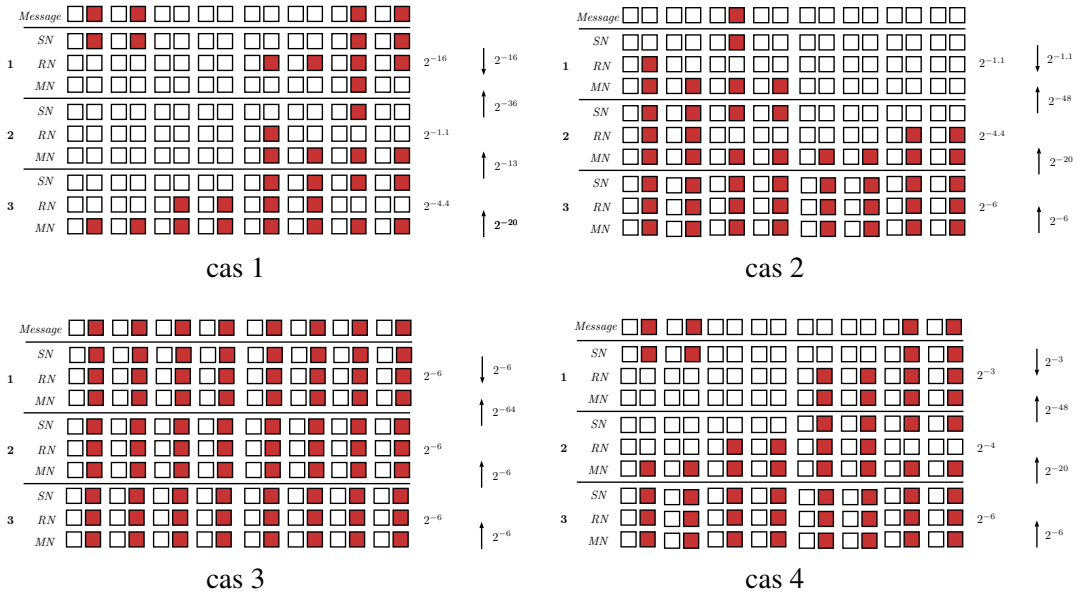


FIGURE 17 – Débuts possibles pour le chemin différentiel

Et les complexités associées sont les suivantes :

Cas	Données	Temps	Mémoire
1	$2^{54.5}$	2^{57}	2^{16}
2	$2^{56.5}$	2^{62}	2^{4}
3	2^{35}	$2^{63.8}$	2^{32}
4	2^{46}	2^{62}	2^{16}

FIGURE 18 – Complexités associées aux précédents chemins

Adaptation de l'Attaque à *KLEIN*-80 et *KLEIN*-96

L'attaque que nous venons de décrire sur *KLEIN*-64 est adaptable aux 2 autres versions du chiffrement, *KLEIN*-80 et *KLEIN*-96. Ces versions utilisent des clefs plus grandes et réalisent plus de tours. Comme la procédure de chiffrement est strictement la même, nous pouvons réutiliser les précédents chemins différentiels et le même processus de filtrage.

La difficulté de l'adaptation de l'ancienne attaque à ces nouvelles versions tient au nombre plus important de tours - donc la probabilité de suivre le chemin différentiel en

est d'autant diminuée - alors que le nombre maximal de messages à disposition reste le même que précédemment (2^{64}) puisque la taille des messages d'entrée est la même. En contrepartie, la borne de complexité en temps de l'attaque est plus large puisque l'objectif est de rester en dessous de la complexité de la recherche exhaustive de la clef (2^{80} ou 2^{96} chiffrements).

La propriété 3 est toujours valable dans ces 2 versions. Le nombre de quartets bas à deviner devient respectivement 40 et 48 mais on peut remarquer qu'il n'est pas nécessaire de deviner l'ensemble des quartets bas dès le début de l'attaque. En effet, il est plus efficace de deviner uniquement les bits de clef intervenant dans le premier tour, de les filtrer et ensuite seulement de deviner les bits restants.

Si on prend l'exemple du chemin différentiel vu en figure 11 que l'on adapte par exemple à *KLEIN*-80, l'idée est de réaliser une hypothèse sur les quartets bas de la sous clef de 64 bits utilisée au tour 1, ce qui nécessite 2^{16} chiffrements de tour pour chaque paire candidate comme pour la précédente version, puis après avoir filtré ceux-ci de compléter toutes les hypothèses des quartets bas de clef candidates avec les 2^8 valeurs possibles pour les quartets bas restants.

Cette opération permet de réaliser 2^{16} chiffrements de tour au lieu de 2^{40} pour chaque paire candidate.

Version	Cas	Tours	Données	Temps	Mémoire
80	I	12	$2^{54.49}$	2^{65}	2^{16}
80	I	13	$2^{60.49}$	$2^{71.1}$	2^{16}
80	II	13	$2^{62.49}$	2^{76}	2^{4}
80	III	13	2^{41}	2^{78}	2^{32}
80	IV	13	2^{52}	$2^{71.3}$	2^{16}
96	III	14	2^{47}	2^{92}	2^{32}
96	IV	14	2^{58}	$2^{89.2}$	2^{16}

TABLE 3 – Complexité des attaques appliquées à *KLEIN*-80 et *KLEIN*-96

4.5 Programmation de l'Attaque

Pour valider nos propositions d'attaques, nous avons réalisé les implémentations des versions 1 et 3 décrite figure 17 sur la version de *KLEIN* avec 64 bits de clef. Le langage choisi a été le langage C pour sa performance. Les programmes complets peuvent être obtenus sur demande à *virginie.lallemand@inria.fr*.

Des extraits de nos programmes sont présentés en annexe C. Il faut noter que les complexités trop élevées des attaques sur 12 tours nous ont imposé de tester nos programmes sur des versions réduites de l'algorithme comportant moins de 10 tours. Ces versions nous ont permis de valider nos calculs de complexités et de réaliser la première attaque pratique sur 9 tours de *KLEIN*-64.

La section suivante présente les grandes lignes de nos décisions de programmations. Nous verrons ensuite en détail les résultats obtenus et leur interprétation.

4.5.1 Choix de Programmation

Comme aucune implémentation officielle n'existe, nous avons nous-même implémenté l'algorithme de chiffrement *KLEIN*. Pour rendre le chiffrement plus rapide, nous avons utilisé des tables pour les opérations de multiplication par 2, 3, 9, 11, 13 et 14 utilisées pour *MixColumn* et son inverse ainsi que des tables de correspondance par octet pour l'opération *SubNibbles*.

Une des particularités de notre implémentation est la gestion du filtre du dernier tour. Dans cette étape, il faut sélectionner les paires de messages ayant une différence inactive en quartets hauts dans l'état précédant le dernier *MixNibbles*. Pour cela, au lieu d'étudier une à une toutes les paires possibles associées à une structure, on réalise une liste chaînée triée selon la valeur des quartets hauts de l'état obtenu en inversant *MixNibbles* à partir des chiffrés. Cette liste permet de repérer très facilement les égalités de valeur des quartets hauts. Une telle égalité, que l'on appellera dans la suite *collision*, implique que la paire formée par les 2 messages correspondants passent le premier filtre de 2^{-32} puisque leur différence est nulle en quartets hauts.

Le reste de l'attaque suit strictement la version optimisée expliquée dans les sections précédentes si ce n'est que l'attaque implémentée pour le case 3 est légèrement différente du cas 1. L'optimisation que nous avons trouvée repose sur la remarque que le chemin entièrement itératif nous permet d'envisager l'attaque avec les mêmes valeurs de filtre en descendant ou en remontant dans la procédure de chiffrement. Néanmoins, si on choisit de descendre dans la procédure de chiffrement, le tri des hypothèses de bits de condition peut se faire plus efficacement.

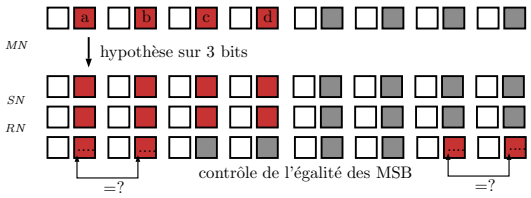


FIGURE 19 – Accélération du filtre des hypothèses

Comme démontré en annexe B.1, il suffit de réaliser une hypothèse sur 3 bits pour calculer les quartets bas en sortie de *MixColumn*, étant donnés ceux d'entrée. De plus, pour obtenir un état avec des quartets hauts nuls en différence, il faut vérifier que les 4 bits de poids fort des quartets bas sont égaux (c'est la propriété 2). L'idée est donc de réaliser une hypothèse sur 3 bits au niveau d'un des 2 *MixColumn* (disons celui de gauche) pour obtenir une valeur possible de ses quartets bas en sortie et pouvoir traverser l'opération *SubNibbles*. On vérifie alors sur le demi état obtenu l'égalité des MSB des quartets extrêmes (voir figure 19) ce qui permet de ne conserver que $2^{3-2} = 2$ hypothèses pour les valeurs des 3 bits.

On effectue alors la même opération pour le *MixColumn* de droite. L'hypothèse de 6 bits correcte est celle menant à 4 MSB de quartets bas égaux en entrée d'un même *MixColumn*. Comme on a une chance sur 2 pour chaque *MixColumn*, 1 seule hypothèse sur les 2^6 initiales est retenue. Cette différence avec le cas 1 est vraiment minime et permet juste d'accélérer légèrement le tri des bits de conditions.

Le reste de l'attaque suit la procédure expliquée sans particularité spéciale.

4.5.2 Résultats Obtenus et Interprétation

Cette section présente les résultats expérimentaux obtenus sur *KLEIN*-64 sur des versions réduites. Les premiers résultats, présentés dans le tableau suivant, concernent la version réduite à 5 tours attaquée avec le chemin différentiel du cas 1 :

Messages	33545	24580	29010	29689	19862	32580	29774	17336	18172	25094	19711	18 581	17862	16579	19173
Structures	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
collisions	1092	606	848	837	392	1014	906	319	341	595	416	311	319	284	375
Durée(sec)	262.6	139.6	208.7	242.2	75.45	241.0	185.1	58.6	63.5	71.7	61.6	30	44.2	34.2	28.5

FIGURE 20 – Résultats obtenus sur la version réduite à 5 tours, case 1

En moyennant sur les 15 mesures prises, on obtient une moyenne de 23 436 messages chiffrés par exécution (soit $2^{14.5}$), construits avec une seule structure et produisant 577 collisions ($2^{9.2}$) sur les valeurs de $MixNibbles^{-1}$(chiffré). La clef est retrouvée après moins de 2 minutes (environ 116 secondes).

On peut comparer ces résultats pratiques à la théorie exposée en section 4 : en se référant aux probabilités du chemin différentiel du cas 1, on constate qu'on a besoin de $2^{27.5}$ paires de messages pour trouver 1 bonne paire. Nos observations nous indiquent que $2^{14.5}$ messages ont été chiffrés en moyenne donc $2^{14.5*2-1} = 2^{28}$ paires ont été observées, on est donc très proche de la théorie.

Le nombre de collisions est à relier à l'observation suivante : une paire de différence d'entrée portant sur les quartets bas des 2 premiers et 2 derniers octets peut mener à une collision (aboutir à la différence cherchée en fin de quatrième tour) avec la configuration décrite en figure 21, et avec probabilité 2^{-19}.

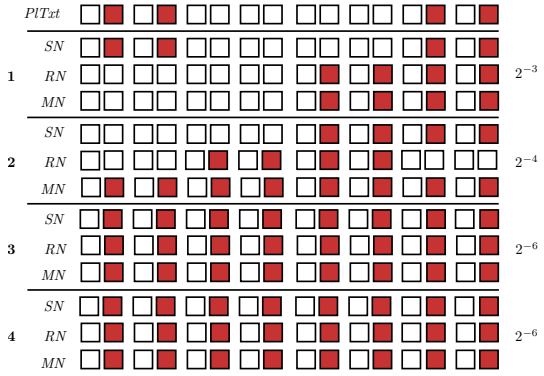


FIGURE 21 – Configuration de probabilité 2^{-19}

Cela implique que si on a 2^{28} paires formées comme dans nos observations, on s'attend à avoir $2^{28-19} = 2^9$ collisions. C'est très proche de la quantité observée expérimentalement qui est $2^{9.2}$.

Remarquons que ces collisions sont écartées dès le premier test réalisé par le programme.

Les mêmes tests avec le cas 3 (tout itératif) mènent aux observations suivantes :

En moyenne on a donc 4 325 messages clairs ($2^{12.08}$), soit 2^{23} paires formées avec une unique structure et menant immédiatement à la paire conforme par le test d'inversion de

Messages	8067	4949	3246	6825	1870	3388	2065	2421	3494	5596	6470	5688	560	6717	3521
Structures	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
collisions	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Durée(sec)	163.2	27.6	65.0	60.9	41.1	116.2	153.2	92.3	16.8	60.6	47.7	58.8	120.3	70.7	33.1

FIGURE 22 – Résultats obtenus sur la version réduite à 5 tours, cas 3

MixNibbles.

Là encore on est très proche de ce qui était attendu en théorie puisqu'on avait évalué la probabilité du chemin à 2^{-24}. Aucune configuration ne permet d'obtenir une paire conforme en fin de tour 4 avec une probabilité inférieure et le cas générique a une probabilité trop faible pour apparaître donc on n'observe pas de collision contrairement au cas précédent.

Pour 6 tours nous avons réalisé le même type de tests et obtenu encore une fois des résultats proches de ceux attendu :

Messages	280472	458061	28452	224876	92246	681153	232221	48664	766727	358971	1996632	186899	615246	770034	158648
Structures	5	7	1	4	2	11	4	1	12	6	31	3	10	12	3
collisions	271	475	10	208	78	698	196	32	804	339	2005	176	607	720	152
Durée(sec)	134.5	218.2	18.3	83.8	39.3	333.1	92.9	24.4	328.1	143.4	841.3	65.4	249.8	31.4	54.1

FIGURE 23 – Résultats obtenus sur la version réduite à 6 tours, cas 1

Théoriquement, on a besoin de $2^{33.5}$ paires donc $2^{18.5}$ messages, et on attend $2^{33.5-25.4} = 2^{8.1} \simeq 274$ collisions (à cause d'un chemin similaire à celui représenté en figure 21 mais pour 6 tours et de probabilité $2^{25.4}$). Les mesures ci-dessus ont une moyenne de $2^{18.81}$ messages obtenus avec 451 collisions ($2^{8.8}$) en 177.2 secondes

Messages	21242	42584	4379	79635	17643	64662	38769	25035	66087	37847	71700	41930	5864	26313	34318
Structures	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
collisions	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Durée(sec)	118.6	124.4	66.0	100.6	146.7	106.8	71.5	84.9	132.6	85.6	123.9	107.1	127.5	56.	79.8

FIGURE 24 – Résultats obtenus sur la version réduite à 6 tours, cas 3

Pour le cas 3 on obtient une moyenne de 38 607 messages $(2^{15.23})$, 1 collision et 101 secondes. La théorie indiquait 2^{30} paires soit $2^{15.5}$ messages, et 1 structure nécessaire.

On réalise ensuite des observations pour 7 tours :

Messages	11190710	16637764	11028635	47120384	24576000	10732377	6734642	15577730
Structures	171	254	169	719	375	164	103	238
collisions	255	376	255	1123	550	263	152	374
Durée(sec)	6148.4	10850.5	5655.3	30417.0	17073.7	5663.9	3951.9	9251.2

FIGURE 25 – Résultats obtenus sur la version réduite à 7 tours, cas 1

La moyenne observée est de $2^{24.1}$ messages, 418 collisions ($2^{8.7}$) et 11 126.5 sec.
Les quantités attendues étaient de $2^{24.5}$ messages et de $2^{8.8}$ collisions. Ce nombre de collisions provient du chemin différentiel de la figure 21 ($2^{8.1}$ collisions) additionné au cas générique de probabilité 2^{-32} créant $2^{7.5}$ collisions.

Messages	149574	545480	3955778	569517	568068	84653	324193	144977	97842	324193	258508
Structures	1	1	1	1	1	1	1	1	1	1	1
collisions	4	46	26	50	27	2	11	2	3	11	10
Durée(sec)	1593.7	20283.2	7298.9	15293.7	11610.2	365.7	5158.8	938.0	1156.35	5158.85	2724.42

FIGURE 26 – Résultats obtenus sur la version réduite à 7 tours, cas 3

Pour le cas 3 sur 7 tours on obtient une moyenne de $2^{19,28}$ messages ($2^{18.5}$ étaient attendus) et $2^{4,1}$ collisions contre 2^4 attendues.

Les résultats sur la version réduite à 8 tours sont plus long à obtenir (plus de 3 jours pour le cas 1) :

Messages	$2^{30.44}$	2^{26}	$2^{27.8}$	$2^{31.25}$	$2^{30.01}$
Structures	22212	1947	3588	38967	16542
collisions	11251	1018	1852	20269	8720
Durée(sec)	531156.1	36396.4	68665.3	913639.0	345526.2

FIGURE 27 – Résultats obtenus sur la version réduite à 8 tours, cas 1

On obtient une moyenne de $2^{13.07}$ collisions avec $2^{30.01}$ messages. La théorie annonçait $2^{30.5}$ messages et $2^{13.5}$ collisions. Les collisions sont dues au cas générique de probabilité 2^{-32} alors que le chemin de la figure 21 devient négligeable en comparaison ($2^{8.1}$ collisions) à partir de ce nombre de tours.

Des expérimentations sur 8 et 9 tours ont aussi été lancés pour le case 3, demandant près de 9 jours d'exécution. Quelques un de ces résultats sont consignés ci-dessous :

```
NB rounds: 8
MasterKeyToFound:     ed 80 15 0b  7a 40 43 f7
structure:            10 c0 30 60  50 30 20 d0
Plaintext 1:          1c c7 3d 6d  53 3a 23 dd
Plaintext 2:          16 cb 35 6a  58 38 27 d7
lower nibbles found:  0d 00 05 0b  0a 00 03 07
Complete Key:         ed 80 15 0b  7a 40 43 f7
false alarms:         173
time elapsed:         65967.610000 sec
```

```
NB rounds: 9
MasterKeyToFound:     ed 80 15 0b  7a 40 43 f7
structure:            80 c0 30 10  60 80 f0 50
Plaintext 1:          84 c5 3c 1a  60 8d f0 56
Plaintext 2:          8f cb 3a 1c  6d 8e f4 59
lower nibbles found:  0d 00 05 0b  0a 00 03 07
Complete Key:         ed 80 15 0b  7a 40 43 f7
false alarms:         1838
time elapsed:         792416.740000 sec
```

```
NB rounds: 9
MasterKeyToFound:     90 f4 cd 77  2a ef e8 ba
structure:            a0 a0 d0 20  40 80 e0 10
Plaintext 1:          a3 ac d6 25  42 88 e5 18
Plaintext 2:          a9 a0 d6 29  49 88 e7 1b
lower nibbles found:  00 04 0d 07  0a 0f 08 0a
Complete Key:         90 f4 cd 77  2a ef e8 ba
false alarms:         1007
time elapsed:         583708.960000 sec
```

```
NB rounds: 9
MasterKeyToFound:     20 8b 01 38  e9 62 a0 b1
structure:            c0 00 a0 90  60 10 d0 90
Plaintext 1:          ca 06 a9 9d  66 1f d5 99
Plaintext 2:          c9 03 a5 92  6a 15 d5 99
lower nibbles found:  00 0b 01 08  09 02 00 01
Complete Key:         20 8b 01 38  e9 62 a0 b1
false alarms:         1919
time elapsed:         829888.290000 sec
```

5 Conclusion

Lors de ces 6 mois de stage, mon travail s'est porté sur les chiffrements itératifs par blocs et plus particulièrement sur leur sécurité. Le type d'attaque étudié a été la cryptanalyse différentielle qui est un type particulier d'attaque statistique, type d'attaque qui est souvent voire toujours plus performant que l'autre type majeur d'attaque existant en cryptographie symétrique qui est l'attaque algébrique.

En collaboration avec ma maître de stage María Naya-Plasencia, nous avons étudié plus précisément la sécurité du chiffrement à bas coût *KLEIN* et amélioré les résultats précédemment publiés. Notre proposition d'attaque démontre que *KLEIN*-64 n'est pas un chiffrement sûr puisque sa sécurité se réduit à 57,3 bits au lieu des 64 annoncés. Notre attaque casse donc ce système. Notre proposition a été approuvée par les auteurs de *KLEIN* et de plus nous l'avons validée expérimentalement sur des versions réduites allant jusqu'à 9 tours, réalisant ainsi la première attaque pratique sur 9 tours.

Les perspectives de continuation de ce stage, qui seront notamment traitées lors de la thèse, sont d'explorer les différentes façons d'améliorer les performances des attaques différentielles parmi lesquelles la technique des bits neutres vue ici et de continuer à analyser la sécurité des nouvelles primitives à bas coût.

Références

[AES01] NIST AES. Advanced Encryption Standard. *Federal Information Processing Standard, FIPS-197*, page 12, 2001.

[ANPS11] Jean-Philippe Aumasson, María Naya-Plasencia, and Markku-Juhani O. Saarinen. Practical attack on 8 rounds of the lightweight block cipher KLEIN. In *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2011.

[ASA13] Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Biclique cryptanalysis of the full-round KLEIN block cipher. Cryptology ePrint Archive, Report 2013/097, 2013.

[BC04] Eli Biham and Rafi Chen. Near-collisions of SHA-0. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004.

[BK00] Eli Biham and Nathan Keller. Cryptanalysis of reduced variants of Rijndael. 2000.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT : An ultra-lightweight block cipher. In *CHES*, pages 450–466, 2007.

[Blo11] Céline Blondeau. *La cryptanalyse différentielle et ses généralisations*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2011.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *J. Cryptology*, 4(1) :3–72, 1991.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In *CHES*, pages 272–288, 2009.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN : A new family of lightweight block ciphers. In *RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[Jha11] Vikash Kumar Jha. Cryptanalysis of lightweight block ciphers. Aalto University Master's Thesis, 2011.

[KNPRS10] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *luffa* v2 components. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, 2010.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4) :656–715, 1949.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher clefia (extended abstract). In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[Sta99] NF Standard. Data Encryption Standard. *Federal Information Processing Standards Publication*, 1999.

[Wan08] Meiqin Wang. Differential cryptanalysis of reduced-round PRESENT. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2008.

[WZ11] Wenling Wu and Lei Zhang. LBlock : A Lightweight Block Cipher. In *ACNS 2011*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2011.

[YWLZ11] Xiaoli Yu, Wenling Wu, Yanjun Li, and Lei Zhang. Cryptanalysis of reduced-round KLEIN block cipher. In *Inscrypt*, volume 7537 of *Lecture Notes in Computer Science*. Springer, 2011.

A Table des Différences de la Boîte S de *KLEIN*

Le tableau suivant est la table des différences associée à la boîte S de *KLEIN*. Les lignes représentent les différences d'entrée Δ_I et les colonnes les différences de sortie Δ_O : à l'intersection d'une ligne et d'une colonne données on trouve le nombre de valeurs $x \in \{0, \cdots f\}$ vérifiant $S[x] \oplus S[x \oplus \Delta_I] = \Delta_O$

	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	0	0	4	2	0	2	0	2	0	0	2	0	0	2	2
2	0	0	0	0	4	0	0	0	0	2	2	0	4	2	2
3	4	0	2	2	0	0	0	0	0	2	0	0	2	4	0
4	2	0	2	2	0	2	0	0	2	0	2	0	2	0	2
5	0	4	0	0	2	0	2	2	0	2	4	0	0	0	0
6	2	0	0	2	0	4	0	2	0	2	2	2	0	0	0
7	0	0	0	0	2	0	2	2	2	0	0	2	0	4	2
8	2	0	0	0	2	2	2	2	2	0	2	0	0	0	2
9	0	0	0	2	0	0	2	2	0	0	2	2	2	2	2
a	0	2	2	0	2	2	0	0	0	4	0	2	0	2	0
b	2	2	0	2	4	2	0	2	2	0	0	0	0	0	0
c	0	0	0	0	0	2	2	0	2	2	0	4	2	0	2
d	0	4	2	2	0	0	0	0	2	0	0	2	2	0	2
e	2	2	4	0	0	0	4	0	2	2	0	0	0	0	0
f	2	2	0	2	0	0	2	2	2	0	0	2	2	0	0

FIGURE 28 – Table des différences associée à la boîte S 4x4 de *KLEIN*.

On peut noter que la table est totalement symétrique puisque la boîte S définit une involution.

B Preuves des Affirmations Concernant *MixColumn*

B.1 Calculs des Quartets Bas en Sortie de $MixColumn$ Étant Donnés les Quartets Bas d'Entrée

Comme précédemment, notons le développement binaire d'un octet a par :
$a = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$. On suppose qu'on connaît la valeurs des quartets bas de l'état (a, b, c, d) en entrée de *MixColumn* (donc les bits d'indice 4 à 7 de chaque octet) et on souhaite en déduire les quartets bas de la sortie. L'équation de départ est la suivante :

$$\begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \tag{2}$$

Si on développe l'expression des valeurs des octets de sortie e, f, g et h on obtient :

$$
\begin{aligned}
e_0 &= a_1 + b_0 + b_1 + c_0 + d_0 \\
e_1 &= a_2 + b_1 + b_2 + c_1 + d_1 \\
e_2 &= a_3 + b_2 + b_3 + c_2 + d_2 \\
e_3 &= a_0 + a_4 + b_0 + b_3 + b_4 + c_3 + d_3 \\
e_4 &= a_0 + a_5 + b_0 + b_5 + b_4 + c_4 + d_4 \\
e_5 &= a_6 + b_5 + b_6 + c_5 + d_5 \\
e_6 &= a_0 + a_7 + b_0 + b_6 + b_7 + c_6 + d_6 \\
e_7 &= a_0 + b_0 + b_7 + c_7 + d_7
\end{aligned}
\qquad
\begin{aligned}
f_0 &= a_0 + b_1 + c_0 + c_1 + d_0 \\
f_1 &= a_1 + b_2 + c_1 + c_2 + d_1 \\
f_2 &= a_2 + b_3 + c_2 + c_3 + d_2 \\
f_3 &= a_3 + b_0 + b_4 + c_0 + c_3 + c_4 + d_3 \\
f_4 &= a_4 + b_5 + b_0 + c_0 + c_4 + c_5 + d_4 \\
f_5 &= a_5 + b_6 + c_5 + c_6 + d_5 \\
f_6 &= a_6 + b_0 + b_7 + c_0 + c_6 + c_7 + d_6 \\
f_7 &= a_7 + b_0 + c_0 + c_7 + d_7
\end{aligned}
$$

$$
\begin{aligned}
g_0 &= a_0 + b_0 + c_1 + d_0 + d_1 \\
g_1 &= a_1 + b_1 + c_2 + d_1 + d_2 \\
g_2 &= a_2 + b_2 + c_3 + d_2 + d_3 \\
g_3 &= a_3 + b_3 + c_0 + c_4 + d_0 + d_3 + d_4 \\
g_4 &= a_4 + b_4 + c_0 + c_5 + d_0 + d_4 + d_5 \\
g_5 &= a_5 + b_5 + c_6 + d_5 + d_6 \\
g_6 &= a_6 + b_6 + c_0 + c_7 + d_0 + d_6 + d_7 \\
g_7 &= a_7 + b_7 + c_0 + d_0 + d_7
\end{aligned}
\qquad
\begin{aligned}
h_0 &= a_0 + a_1 + b_0 + c_0 + d_1 \\
h_1 &= a_1 + a_2 + a_1 + b_1 + d_2 \\
h_2 &= a_2 + a_3 + b_2 + c_2 + d_3 \\
h_3 &= a_0 + a_3 + a_4 + b_3 + c_3 + d_0 + d_4 \\
h_4 &= a_0 + a_4 + a_5 + b_4 + c_4 + d_0 + d_5 \\
h_5 &= a_5 + a_6 + b_5 + c_5 + d_6 \\
h_6 &= a_0 + a_6 + a_7 + b_6 + c_6 + d_0 + d_7 \\
h_7 &= a_0 + a_7 + b_7 + c_7 + d_0
\end{aligned}
$$

En considérant uniquement les valeurs des quartets bas on remarques que les quantités inconnues sont

- $a_0 + b_0$ pour e
- $b_0 + c_0$ pour f
- $c_0 + d_0$ pour g
- $d_0 + a_0$ pour h (et c'est la somme des 3 valeurs précédentes)

donc seulement 3 valeurs nous manquent pour pouvoir calculer les valeurs des quartets bas

B.2 Probabilité et Équations de l'Événement "Obtenir un État Inactif en Quartets Hauts Après $MixColumn^{-1}$ Étant Donnée une Entrée du Même Type"

L'opération $MixColumn^{-1}$ est la suivante :

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \times \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} \tag{3}$$

Après développement on obtient les formules suivantes pour a :

$$\begin{aligned}
a_0 &= e_1 + e_2 + e_3 + f_0 + f_1 + f_3 + g_0 + g_2 + g_3 \\
a_1 &= e_0 + e_2 + e_3 + e_4 + f_0 + f_1 + f_2 + f_4 + g_0 + g_1 + g_3 + g_4 + h_0 + h_1 + h_4 \\
a_2 &= e_1 + e_3 + e_4 + e_5 + f_0 + f_1 + f_2 + f_3 + f_5 + g_1 + g_2 + g_4 + g_5 + h_0 + h_1 + h_2 + h_5 \\
a_3 &= e_2 + e_4 + e_5 + e_6 + f_0 + f_1 + f_2 + f_3 + f_4 + f_6 + g_0 + g_1 + g_2 + g_3 + g_5 + g_6 + h_1 + h_2 + h_3 + h_6 \\
a_4 &= e_1 + e_2 + e_5 + e_6 + e_7 + f_2 + f_4 + f_5 + f_7 + g_0 + g_1 + g_2 + g_4 + g_6 + g_7 + h_0 + h_2 + h_4 + h_7 \\
a_5 &= e_1 + e_6 + e_7 + f_0 + f_1 + f_5 + f_6 + g_1 + g_5 + g_7 + h_0 + h_1 + h_5 \\
a_6 &= e_2 + e_7 + f_0 + f_1 + f_2 + f_6 + f_7 + g_0 + g_2 + g_6 + h_1 + h_2 + h_6 \\
a_7 &= e_0 + e_1 + e_2 + f_0 + f_2 + f_7 + g_1 + g_2 + g_7 + h_2 + h_7
\end{aligned}$$

et de même pour b, c et d :

$$\begin{aligned}
b_0 &= e_0 + e_3 + f_1 + f_2 + f_3 + g_0 + g_1 + g_3 + h_0 + h_2 + h_3 \\
b_1 &= e_0 + e_1 + e_4 + f_0 + f_2 + f_3 + f_4 + g_0 + g_1 + g_2 + g_4 + h_0 + h_1 + h_3 + h_4 \\
b_2 &= e_0 + e_1 + e_2 + e_5 + f_1 + f_3 + f_4 + f_5 + g_0 + g_1 + g_2 + g_3 + g_5 + h_1 + h_2 + h_4 + h_5 \\
b_3 &= e_1 + e_2 + e_3 + e_6 + f_2 + f_4 + f_5 + f_6 + g_0 + g_1 + g_2 + g_3 + g_4 + g_6 + h_0 + h_1 + h_2 + h_3 + h_5 + h_6 \\
b_4 &= e_0 + e_2 + e_4 + e_7 + f_1 + f_2 + f_5 + f_6 + f_7 + g_2 + g_4 + g_5 + g_7 + h_0 + h_1 + h_2 + h_4 + h_6 + h_7 \\
b_5 &= e_0 + e_1 + e_5 + f_1 + f_6 + f_7 + g_0 + g_1 + g_5 + g_6 + h_1 + h_5 + h_7 \\
b_6 &= e_1 + e_2 + e_6 + f_2 + f_7 + g_0 + g_1 + g_2 + g_6 + g_7 + h_0 + h_2 + h_6 \\
b_7 &= e_2 + e_7 + f_0 + f_1 + f_2 + g_0 + g_2 + g_7 + h_1 + h_2 + h_7
\end{aligned}$$

$$\begin{aligned}
c_0 &= e_0 + e_2 + e_3 + f_0 + f_3 + g_1 + g_2 + g_3 + h_0 + h_1 + h_3 \\
c_1 &= e_0 + e_1 + e_3 + e_4 + f_0 + f_1 + f_4 + g_0 + g_2 + g_3 + g_4 + h_0 + h_1 + h_2 + h_4 \\
c_2 &= e_1 + e_2 + e_4 + e_5 + f_0 + f_1 + f_2 + f_5 + g_1 + g_3 + g_4 + g_5 + h_0 + h_1 + h_2 + h_3 + h_5 \\
c_3 &= e_0 + e_1 + e_2 + e_3 + e_5 + e_6 + f_1 + f_2 + f_3 + f_6 + g_2 + g_4 + g_5 + g_6 + h_0 + h_1 + h_2 + h_3 + h_4 + h_6 \\
c_4 &= e_0 + e_1 + e_2 + e_4 + e_6 + e_7 + f_0 + f_2 + f_4 + f_7 + g_1 + g_2 + g_5 + g_6 + g_7 + h_2 + h_4 + h_5 + h_7 \\
c_5 &= e_1 + e_5 + e_7 + f_0 + f_1 + f_5 + g_1 + g_6 + g_7 + h_0 + h_1 + h_5 + h_6 \\
c_6 &= e_0 + e_2 + e_6 + f_1 + f_2 + f_6 + g_2 + g_7 + h_0 + h_1 + h_2 + h_6 + h_7 \\
c_7 &= e_1 + e_2 + e_7 + f_2 + f_7 + g_0 + g_1 + g_2 + h_0 + h_2 + h_7
\end{aligned}$$

$$
\begin{aligned}
d_0 &= e_0 + e_1 + e_3 + f_0 + f_2 + f_3 + g_0 + g_3 + h_1 + h_2 + h_3 \\
d_1 &= e_0 + e_1 + e_2 + e_4 + f_0 + f_1 + f_3 + f_4 + g_0 + g_1 + g_4 + h_0 + h_2 + h_3 + h_4 \\
d_2 &= e_0 + e_1 + e_2 + e_3 + e_5 + f_1 + f_2 + f_4 + f_5 + g_0 + g_1 + g_2 + g_5 + h_1 + h_3 + h_4 + h_5 \\
d_3 &= e_0 + e_1 + e_2 + e_3 + e_4 + e_6 + f_0 + f_1 + f_2 + f_3 + f_5 + f_6 + g_1 + g_2 + g_3 + g_6 + h_2 + h_4 + h_5 + h_6 \\
d_4 &= e_2 + e_4 + e_5 + e_7 + f_0 + f_1 + f_2 + f_4 + f_6 + f_7 + g_0 + g_2 + g_4 + g_7 + h_1 + h_2 + h_5 + h_6 + h_7 \\
d_5 &= e_0 + e_1 + e_5 + e_6 + f_1 + f_5 + f_7 + g_0 + g_1 + g_5 + h_1 + h_6 + h_7 \\
d_6 &= e_0 + e_1 + e_2 + e_6 + e_7 + f_0 + f_2 + f_6 + g_1 + g_2 + g_6 + h_2 + h_7 \\
d_7 &= e_0 + e_2 + e_7 + f_1 + f_2 + f_7 + g_2 + g_7 + h_0 + h_1 + h_2
\end{aligned}
$$

Si on suppose que les quartets hauts de e, f, g et h sont nuls, obtenir le quartet haut de a nul revient à annuler a_0, a_1 , a_2 et a_3 donc à vérifier les équations suivantes :

$$e_4 + f_4 + g_4 + h_4 = 0 \quad (4)$$
$$e_4 + e_5 + f_5 + g_4 + g_5 + h_5 = 0 \quad (5)$$
$$e_4 + e_5 + e_6 + f_4 + f_6 + g_5 + g_6 + h_6 = 0 \quad (6)$$

De la même façon pour que les quartets hauts de b soient nuls on doit vérifier :

$$e_4 + f_4 + g_4 + h_4 = 0 \quad (7)$$
$$e_5 + f_4 + f_5 + g_5 + h_4 + h_5 = 0 \quad (8)$$
$$e_6 + f_4 + f_5 + f_6 + g_4 + g_6 + h_5 + h_6 = 0 \quad (9)$$

or on peut remarquer que (7) = (4), $(8) = (4) \oplus (5)$ et $(9) = (5) \oplus (6)$.
Les conditions sur c imposent les équations :

$$e_4 + f_4 + g_4 + h_4 = 0 \quad (10)$$
$$e_4 + e_5 + f_5 + g_5 + g_4 + h_5 = 0 \quad (11)$$
$$e_5 + e_6 + f_6 + g_4 + g_5 + g_6 + h_4 + h_6 = 0 \quad (12)$$

Ici encore on obtient des équations équivalentes à celles obtenues pour a puisque $(10) = (4)$, $(11) = (5)$ et $(12) = (4) \oplus (6)$.
Finalement d nous donne :

$$e_4 + f_4 + g_4 + h_4 = 0 \quad (13)$$
$$e_5 + f_4 + f_5 + g_5 + h_4 + h_5 = 0 \quad (14)$$
$$e_4 + e_6 + f_5 + f_6 + g_6 + h_4 + h_5 + h_6 = 0 \quad (15)$$

Les conditions sont les mêmes que pour a puisque $(13) = (4)$, $(14) = (4) \oplus (5)$ et $(15) = (4) \oplus (5) \oplus (6)$

Au final on a uniquement 3 conditions à vérifier donc la probabilité d'obtenir un état inactif en quartets hauts est 2^{-3}

B.3 Calcul des Quartets Bas en Sortie de $MixColumn^{-1}$ Étant Donnés les Quartets Bas d'Entrée

En utilisant les expressions du précédent paragraphe et en supposant que les valeurs des quartets bas de e, f, g et h sont connues, on obtient que les valeurs de a_4, a_5, a_6 et a_7 dépendent de :

$$e_1 + e_2 + f_2 + g_0 + g_1 + g_2 + h_0 + h_2 \quad (16)$$
$$e_1 + f_0 + f_1 + g_1 + h_0 + h_1 \quad (17)$$
$$e_2 + f_0 + f_1 + f_2 + g_0 + g_2 + h_1 + h_2 \quad (18)$$
$$e_0 + e_1 + e_2 + f_0 + f_2 + g_1 + g_2 + h_2 \quad (19)$$

Ce qui se résume en fait aux 3 quantités (16), (17) et (19) puisque (18)=(16)+(17).
De façon similaire, lorsqu'on exprime les quantités inconnues pour b on obtient :

$$e_0 + e_2 + f_1 + f_2 + g_2 + h_0 + h_1 + h_2 \quad (20)$$
$$e_0 + e_1 + f_1 + g_0 + g_1 + h_1 \quad (21)$$
$$e_1 + e_2 + f_2 + g_0 + g_1 + g_2 + h_0 + h_2 \quad (22)$$
$$e_2 + f_0 + f_1 + f_2 + g_0 + g_2 + h_1 + h_2 \quad (23)$$

Il n'est pas difficile de voir que (22) = (20) + (21) et que les autres quantités s'expriment en fonctions de celles intervenant pour a : (20)=(17)+(19), (21)=(16)+(17)+(19), (22)=(16), (23)=(16)+(17).
Pour c on a :

$$e_0 + e_1 + e_2 + f_0 + f_2 + g_1 + g_2 + h_2 \quad (24)$$
$$e_1 + f_0 + f_1 + g_1 + h_0 + h_1 \quad (25)$$
$$e_0 + e_2 + f_1 + f_2 + g_2 + h_0 + h_1 + h_2 \quad (26)$$
$$e_1 + e_2 + f_2 + g_0 + g_1 + g_2 + h_0 + h_2 \quad (27)$$

encore une fois on remarque qu'il y a seulement 3 quantités différentes puisque (26) = (24) + (25) et que les autres quantités s'expriment en fonction de (16), (17) et (19) : (24)=(19), (25)=(17) et (27)=(16).
Finalement pour d les quantités sont :

$$e_2 + f_0 + f_1 + f_2 + g_0 + g_2 + h_1 + h_2 \quad (28)$$
$$e_0 + e_1 + f_1 + g_0 + g_1 + h_1 \quad (29)$$
$$e_0 + e_1 + e_2 + f_0 + f_2 + g_1 + g_2 + h_2 \quad (30)$$
$$e_0 + e_2 + f_1 + f_2 + g_2 + h_0 + h_1 + h_2 \quad (31)$$

Et (28)=(16)+(17), (29)=(16)+(17), (31)=(17)+(19) et encore une fois on a la relation (30) = (28) + (29)

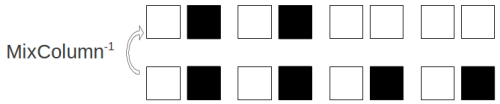
En fait, on peut montrer que la propriété selon laquelle les quantités intervenant dans z_6 est la somme de celles intervenant dans z_4 et z_5 est vraie pour z=a,b,c et d. On s'intéresse à l'influence des quartets hauts de e, f, g et h sur les quartets bas d'un octet z.

D'après la remarque concernant *MixColumn* faite en 3.1, lorsqu'on multiplie un octet x par **02**, on obtient $(x_1, x_2, x_3, x_4 + x_0, x_5 + x_0, x_6, x_7 + x_0, x_0)$ donc le vecteur d'influence des quartets hauts est $(0, 0, 0, x_0, x_0, 0, x_0, x_0)$ soit $(x_0, 0, x_0, x_0)$ en considérant uniquement le quartet bas. Si on multiplie par **04**, l'influence est $(x_1, 0, x_0 + x_1, x_1)$. En effet multiplier par **04** se fait en multipliant x par **02** deux fois donc en décalant l'écriture obtenue précédemment d'une position vers la gauche puis en ajoutant $0x1b \times a_1$. Sur le même principe, la multiplication par **08** ajoutera $(x_2 + x_0, x_0 + x_1, x_2 + x_1, x_2)$.

Dans ces 3 vecteurs d'influence, le 3ème bit est la somme des 2 premiers. Si on considère un octet z=a,b,c ou d obtenu par inversion de *MixColumn*, on remarque que puisque les vecteurs d'influence de **09**, **0b**, **0d** et **0e** s'expriment comme combinaison de ceux de **02**, **04** et **08**, les vecteurs d'influence des quartets bas de $a = \mathbf{0e} \times e + \mathbf{0b} \times f + \mathbf{0d} \times g + \mathbf{09} \times h$ (et de même pour b, c et d) vérifient la propriété et uniquement 3 valeurs différentes provenant des quartets hauts interviennent.

B.4 Probabilité et Équations de l'Événement "Obtenir Uniquement 2 Quartets Bas Actifs Après $MixColumn^{-1}$ Étant Donné un État Inactif en Quartets Hauts en Entrée"

Le schéma de la situation étudiée est la suivante :



Pour atteindre cette situation, les conditions à satisfaire sont les 3 précédentes ((4),(5) et (6)) auxquelles on rajoute celles concernant les quartets bas de c et d :

$$\begin{aligned} e_4 + f_4 + g_4 + h_4 &= 0 \\ e_5 + f_4 + f_5 + g_5 + h_4 + h_5 &= 0 \\ e_4 + e_6 + f_5 + f_6 + g_6 + h_4 + h_5 + h_6 &= 0 \end{aligned} \qquad \begin{aligned} g_5 + g_6 + g_7 + h_4 + h_5 + h_7 + e_4 + e_6 + e_7 + f_4 + f_7 &= 0 \\ e_5 + e_7 + f_5 + g_6 + g_7 + h_5 + h_6 &= 0 \\ g_7 + h_6 + h_7 + e_6 + f_6 &= 0 \\ h_7 + e_7 + f_7 &= 0 \end{aligned}$$

$$\begin{aligned} h_5 + h_6 + h_7 + e_4 + e_5 + e_7 + f_4 + f_6 + f_7 + g_4 + g_7 &= 0 \\ f_5 + f_7 + g_5 + h_6 + h_7 + e_5 + e_6 &= 0 \\ h_7 + e_6 + e_7 + f_6 + g_6 &= 0 \\ e_7 + f_7 + g_7 &= 0 \end{aligned}$$

Ce qui matriciellement donne :

$$\begin{pmatrix} 1000100010001000 \\ 1100010011000100 \\ 1110101001100010 \\ 0000000100010001 \\ 0001001100100010 \\ 0011011001010100 \\ 0111110110111001 \\ 1001011111011011 \\ 0100001101100101 \\ 0110010001000010 \\ 0001000000010001 \end{pmatrix}$$ Qui donne après mise sous forme échelonnée : $$\begin{pmatrix} 1000000000000000 \\ 0100000000111101 \\ 0010000000010010 \\ 0001000000010001 \\ 0000100000000100 \\ 0000010000011110 \\ 0000001000100010 \\ 0000000100010001 \\ 0000000010001100 \\ 0000000001101011 \\ \mathbf{0000\ 0000\ 0000\ 0000} \end{pmatrix}$$

Ce qui signifie qu'on dispose de 10 équations indépendantes soit une probabilité de 2^{-10} d'obtenir l'état désiré.

C Extraits de l'Implémentation de Notre Nouvelle Attaque

Cette section présente quelques extraits choisis du programme C de notre nouvelle attaque (cas 1).

La première fonction présentée est `GenerateData` : celle-ci est appelée en début d'attaque à chaque nouvelle structure pour générer et chiffrer les 2^{16} messages associés et rechercher des collisions sur les quartets hauts des états obtenus lors de l'inversion de *MixNibbles* à partir des chiffrés. Comme expliqué précédemment, la recherche de collisions est accélérée grâce à l'utilisation d'une liste chaînée.

```
unsigned char GenerateData()
{
unsigned char keyFound=0x00;
// value recording if the lower nibbles of the key are found (0x01) or not (0x00)
unsigned int a, nbCol =0;

State cipher1,b;
State state1 = {{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}} ;//null plaintext
unsigned int i;

cell_et tmp_pointer=(cell_et)malloc(sizeof(cellule));
cell_et new_element ;
cell_et tmp;

cell_et liste = (cell_et)malloc(sizeof(cellule));
//list storing the lower nibbbles plaintexts
//and the corresponding higher nibbles of MixNibbles ^-1 (ciphertext)
liste = NULL;

  for( i=0;i<65536 && keyFound==0x00;i++)
  // encrypts the 2^16 possible plaintexts with the considered structure
{
state1.T[0]=(i&0xf);
state1.T[1]=((i>>4)&0xf);
state1.T[2]=0x00;
state1.T[3]=0x00;

state1.T[4]=0x00;
state1.T[5]=0x00;
state1.T[6]=((i>>8)&0xf);
state1.T[7]=((i>>12)&0xf);

NB_pltxt=NB_pltxt+1;

state1 = AddRoundKey(state1,ActualStruct);

// the plaintext to encrypt is made of the addition of state1
// (lower nibbles in position 0,1,6,7; varying)
// with the structure (fixed values)

cipher1 = KLEIN_64(NB_ROUNDS, state1) ; //encryption

a = IndiceHigh(MixNibbles_inverse( cipher1 ));
//  higher nibbles of MixNibbles ^-1 (ciphertext)
b = state1;

new_element = malloc(sizeof(cellule)); // construction of the cellule
new_element->MN=a;
new_element->Ptx=b;
new_element->Suiv = NULL;

tmp = NULL;
tmp_pointer=liste;
```

```
// once a new plaintext is chosen, the corresponding 'cellule' is created and
// put into the list 'liste'the list is ordened by the value of the
//  higher nibbles of MixNibbles^-1(ciphertext) seen as an integer
while(tmp_pointer!=NULL && (tmp_pointer->MN)<=a )
// sorts the list
{
if( (tmp_pointer->MN)==a  )
// if there is a collision on the higher nibbles of MixNibbles^-1(ciphertext)
{
nbCol++; // there is one more collision for this structure
NB_falseAlarms++; // sum of all the collisions (all structures)

state1_current = b; // update of the global value state1_current
printf("colliding plaintext 1: \t"); PrintState(b);

state2_current = tmp_pointer->Ptx ; // update of the global value state2_current

cipher1_current = cipher1;
// update of the global value cipher1_current
cipher2_current = KLEIN_64(NB_ROUNDS, state2_current);
// update of the global value cipher2_current

keyFound = FirstRoundLowRight(); // beginning of the filtering steps
}
tmp = tmp_pointer;
tmp_pointer = tmp_pointer->Suiv;
}
new_element->Suiv = tmp_pointer;
if(tmp != NULL)
{
tmp->Suiv=new_element;
}
else
{
liste = new_element;
}
}
// at the end of the function (1 structure totally used), free the memory
cell_et p;
while(liste->Suiv!= NULL)
{
p=liste->Suiv;
free(liste);
liste = p;
}
return keyFound;
}
```

La fonction suivante correspond à l'étape 3 de l'attaque décrite à la section 4. Elle parcourt les 2^3 possibilités de bits de condition nécessaires à l'inversion du *MixNibbles* de droite puis cherche à combiner ses résultats avec ceux de la fonction homologue s'occupant du *MixColumn* de gauche.

```
unsigned char UndoRoundWithGuessOn3bitsHalfRight(State AfterMN1,State AfterMN2,unsigned char round)
{
unsigned char condition, u,v,w, cpt ;
// u,v,w the 3 information bits, cpt the counter of the number of halves treated
unsigned char keyFound=0x00;
// value recording if the lower nibbles of the key are found (0x01) or not (0x00)
HalfState infoBitsIncidence,a,b,delta;
// infoBitsIncidence: halfState getting the information bits influence
// a,b,delta: HalfStates
State c1,c2; // lower nibbles of the states computed with the information bits
unsigned char t1,t2,t3,t4,t5,t6;

// computes the halfStates that fulfill the conditions on the left
```

```
candidatsLeft downHalves=UndoRoundWithGuessOn3bitsHalfLeft(Down(AfterMN1),Down(AfterMN2),round);

//halfStates outputting right MixColumn
HalfState MixCo1 =  MixColumn_inverse(Up(AfterMN1));
HalfState MixCo2 =  MixColumn_inverse(Up(AfterMN2));

if( downHalves.nbC!=0x00 ) // if they are left states that have been found treat them
{
for(condition = 0; (condition<=0x07)&& (keyFound==0); condition ++ )
// look over all the possible values for the 3 information bits
{
u = condition%2;
v = (condition>>1)%2;
w = (condition>>2)%2;

// creation of the halfstate to xor to the halfState outputing MixColumn
// to obtain the good lower nibbles values with the influence of the condition bits

infoBitsIncidence.T[0] = (u<<3) + (v<<2) + ((u^v)<<1) + (w) ;
infoBitsIncidence.T[1] = ((v^w)<<3) + ((u^v^w)<<2) + ((u)<<1) + (u^v);
infoBitsIncidence.T[2] = ((w)<<3) + ((v)<<2) + ((v^w)<<1) + (u);
infoBitsIncidence.T[3] = ((v^u)<<3) + ((u^v^w)<<2) + ((w)<<1) + (w^v);

// computation of the corresponding halfStates
a=SubNibblesHalf( AddHalf(MixCo1 ,infoBitsIncidence) );
b=SubNibblesHalf( AddHalf(MixCo2 ,infoBitsIncidence) );

// difference between the two halfStates
delta = AddHalf(a,b) ;

// computation of the values that appear in the equations that must be fulfill to ensure
// MN is inverted with lower nibbles only
t4= (((delta.T[0])&0x08)>>3)^ (((delta.T[0])&0x04)>>2) ^ (((delta.T[0])&0x02)>>1) ^
  (((delta.T[1])&0x08)>>3) ^ (((delta.T[1])&0x02)>>1) ;
t5= (((delta.T[0])&0x04)>>2) ^ (((delta.T[1])&0x08)>>3) ^ (((delta.T[1])&0x04)>>2) ;
t6= (((delta.T[0])&0x04)>>2) ^ (((delta.T[0])&0x02)>>1) ^ (((delta.T[1])&0x02)>>1) ;

t1= (((delta.T[2])&0x04)>>2) ^ (((delta.T[2])&0x02)>>1) ^ (((delta.T[3])&0x02)>>1) ;
t2= (((delta.T[2])&0x04)>>2) ^ (((delta.T[3])&0x08)>>3) ^ (((delta.T[3])&0x04)>>2) ;
t3= (((delta.T[2])&0x08)>>3) ^ (((delta.T[2])&0x04)>>2) ^ (((delta.T[2])&0x02)>>1) ^
  (((delta.T[3])&0x08)>>3) ^ (((delta.T[3])&0x02)>>1) ;

cpt=0;
while( (cpt<downHalves.nbC) &&  (keyFound==0))
// study all the left parts until the key is found or all have been tried
{
// see if it's possible to have the 2 halfstates together with the 6 information bits guessed
if(  downHalves.tabLowNib[cpt] ==( t1 ^ t2<<1 ^ t3<<2 ^t4<<3 ^t5<<4 ^t6<<5)  )
{
// compute the corresponding states outputting next MixNibbles
c1 = AddRoundKey(FusionOfHalves(downHalves.tabA[cpt],a),LowerAllKEYS.KEY[round-1]);
c2 = AddRoundKey(FusionOfHalves(AddHalf(downHalves.tabDelta[cpt],downHalves.tabA[cpt]),b),
     LowerAllKEYS.KEY[round-1]);
// go on with the attack, with a different treatment if round=5
if( round == 5 )
{
keyFound =  ThirdHalfRight(LNO_Full(c1) ,LNO_Full(c2) );
if(keyFound == 0x01) // if the key has been found, the condition bits were the right ones
{ // so we store them into TabBdC
TabBdC[round-1] = condition ^ downHalves.tabConditionsDown[cpt] ;
}
}
else
{
// do an other round with the same process
keyFound = UndoRoundWithGuessOn3bitsHalfRight( LNO_Full(c1) ,LNO_Full(c2),  round-1);
if(keyFound == 0x01) // if the key has been found, the condition bits were the right ones
```

```
{ // so we store them into TabBdC
TabBdC[round-1] = condition ^ downHalves.tabConditionsDown[cpt] ;
}
}
}
cpt++;
}
}
}
return  keyFound;
}
```

Les autres fonctions sont très proches de celles-ci ou sans particularités spécifiques nous ne les présentons donc pas ici.

Nos programmes ont été lancés avec les options de compilations suivantes :

```
-Wall -O3 -m64 -flto -march=native -fomit-frame-pointer -funroll-loops
```

Celles-ci permettent d'optimiser la vitesse (-03, -funroll-loops qui déroule les boucles), la mémoire (-fomit-frame-pointer), ou encore d'optimiser la compilation selon les paramètres de la machine (-m64, -flto, -march=native).

Nous avons utilisé un pc de bureau possédant 8Go de RAM ainsi qu'un CPU Intel(R) Xeon(R) W3670 à 3.20GHz (12Mo de cache).