Cryptanalyse d'un *blockcipher* minimaliste

Olivier Levillain

1 Introduction

L'objectif de ce TP est d'étudier une primitive de chiffrement par bloc (*blockcipher*) minimaliste, MiniCipher.

Cette primitive est en effet minimaliste car elle manipule des blocs de 16 bits avec des clés de 80 bits. Cependant, il est possible d'étendre MiniCipher pour ressembler à un vrai *blockcipher*.

L'attaque par cryptanalyse différentielle que nous vous proposons de réaliser rend assez bien compte des techniques de base que l'on peut imaginer pour casser des cryptosystèmes réels.

Le TP se décompose en trois grandes parties :

— la première consiste en l'écriture d'une version correcte du chiffrement et du déchiffrement d'un message par la primitive MiniCipher ;

— la seconde a pour but de mettre en place une attaque par cryptanalyse différentielle, et permet de retrouver la dernière sous-clé ;

— enfin, la troisième décrira les attaques par cryptanalyse différentielle permettant de récupérer les autres bits de la clé.

Il existe deux implémentations du TP : une en C, l'autre en Python. Il s'agit d'implémentations à trous, pour vous faire gagner du temps.

Ce TP est inspiré du document *A Tutorial on Linear and Differential Cryptanalysis*, écrit par Howard M. Heys, qui est disponible sur le site du cours.

En cas de problème de compréhension des questions, n'hésitez pas à demander des éclaircissements aux responsables de ce TP.

2 Présentation de MiniCipher

2.1 Notations

Dans tout le texte, nous noterons les bits à partir de zéro. Ainsi, un mot de 16 bits sera composé des bits 0 (le bit de poids le plus faible) à 15 (le bit de poids le plus fort).

Le XOR (*eXclusive OR*, *ou exclusif* en français) est noté par un $\oplus$, et correspond à l'addition bit à bit modulo 2 : $6 \oplus 5 = 110b \oplus 101b = 011b = 3$.

2.2 Conception

Le schéma de MiniCipher est assez classique, et repose sur une succession de 4 étapes. Les trois premières étapes consistent en l'application d'une sous-clé d'étape, d'une application des 4 *S-boxes*, et d'une permutation. La quatrième étape est identique aux précédentes, la permutation en moins. Enfin, la primitive se termine par l'ajout d'une cinquième et dernière sous-clé.

La figure 1 récapitule ce schéma.

2.3 Utilisation

Pour faire fonctionner cette fonction de *blockcipher*, il faut 80 bits de clés (pour obtenir 5 sous-clés d'étapes de 16 bits). Après, chaque application de la primitive, en chiffrement ou en déchiffrement, prend en entrée 16 bits et sort 16 bits.

Si l'on souhaite chiffrer plus de deux octets, il suffit d'employer un mode de chiffrement, comme *CBC* (*Cipher-Block Chaining*) afin d'éclater l'entrée à chiffrer en blocs de 2 octets.

3 Première partie : chiffrement et déchiffrement

Récupération de l'archive

Commencez par télécharger le texte à trou du TP correspondant à la première partie, qui se trouve dans le répertoire `part1`.

Les fichiers `minicipher.c` et `minicipher.py` sont les textes à trous, en C et en Python, concernant l'écriture de MiniCipher.

Les fichiers `minicipher.h` et `minicipher-main.c` complètent les sources en C. Le fichier `minicipher-main.py` complète les sources en Python. Le `Makefile` permet de compiler vos programmes. Si vous choisissez le langage C, il vous faudra taper `make c`, qui produira un exécutable `minicipher`. À l'inverse, si vous choisissez Python, `make python`, `minicipher` sera un lien symbolique vers le programme Python.

En plus des fichiers ci-dessus, vous trouverez trois scripts, `test-encryption.sh`, `test-decryption.sh`, `test-identity.sh`. Ils prennent en paramètre un label.

Il existe un label, `0`, pour lequel les résultats des scripts vous sont fournis. Il s'agit des fichiers `test-encryption.0.out`, `test-decryption.0.out`, `test-identity.0.out`. Ils vous permettent de tester vos programme, en comparant la sortie de `./test-XXX.sh 0` avec le fichier `test-XXX.0.out`.

Les fichiers sources proposent une fonction `main` qui, une fois les trous comblés, permet de chiffrer et de déchiffrer des documents avec MiniCipher selon la ligne de commande suivante :

`./minicipher [-b] [-e|-d] [-1|-M] [-k` ***key***`] [-i` ***iv***`]`

où l'on peut choisir le sens de fonctionnement (`-e` pour le chiffrement, et `-d` pour le déchiffrement), le mode et la clé (avec `-k` ***key*** où ***key*** correspond à 20 caractères hexadécimaux représentant les 80 bits de clé à utiliser). Pour le mode, il faut choisir entre chiffrer un unique bloc (16 bits) avec `-1`, ou une entrée quelconque en utilisant le mode CBC avec remplissage (*padding*), avec `-M`. De plus, l'option `-b` permet de considérer des contenus binaires et pas seulement des chaînes de caractères héxadécimaux. Enfin, lorsque le mode CBC est utilisé, l'IV peut être précisé avec `-i`.

Description de MiniCipher

Dans le schéma de la figure 1, voici les détails qui permettent de compléter la description :

— la clé utilisée par le *blockcipher* compte 80 bits, c'est-à-dire 5 sous-clés de 16 bits ;

— toutes les boîtes marquées S sur le schéma correspondent toutes à la même *S-Box*, décrite à la figure 3 ;

— la permutation qui intervient dans les trois premiers tours, et qui mélange les "fils" (correspondant aux 16 bits du mot manipulé), est décrite à la figure 4 ; cette permutation apparaît explicitement sur le schéma de la figure 2.

Au travail

Question 1 À l'aide des briques de bases qui vous sont données au début du fichier (définitions de la *S-Box* `s` et de la permutation `perm`), écrivez la fonction `encrypt_round` qui calcule une étape (un *round*) du *blockcipher*.

On rappelle qu'une étape de chiffrement consiste en la succession des étapes suivantes :

— *ajout (au sens XOR) de la sous-clé d'étape (donnée par le paramètre* ***key*** *) ;*

— *application des S-Boxes sur chacun des quatre quartets du mot de 16 bits ;*

— *s'il ne s'agit pas du dernier tour, application de la permutation.*

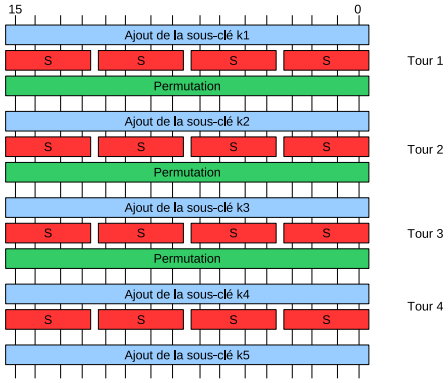

FIGURE 1 – Schéma décrivant MiniCipher

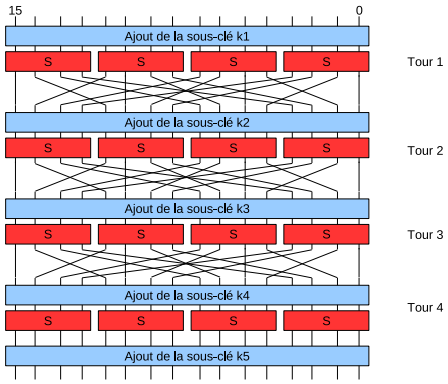

FIGURE 2 – Schéma décrivant MiniCipher, avec la permutation explicitée

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7

FIGURE 3 – Description de la *S-Box* : il s'agit d'une permutation de [0..15] : x correspond à la valeur encodée par les 4 bits en entrée, et $S(x)$ la valeur de sortie, qui sera représentée par les 4 bits de sortie.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(x)$	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

FIGURE 4 – Description de la permutation des 16 bits à chaque tour : à travers la boîte P, les lignes sont réarrangées. Ainsi, par exemple, $P(2) = 8$ indique que le bit 2 de sortie de la première *S-Box* (tout à gauche) du premier tour devient le bit 0 d'entrée de la troisième *S-Box* au deuxième tour.

Question 2 Ecrivez la fonction `encrypt` [1] qui réalise le calcul complet du chiffrement par MiniCipher.

Le chiffrement complet d'un bloc se déroule en 4 étapes (la dernière ne faisant pas intervenir de permutation), suivi de l'ajout (au sens XOR) de la dernière sous-clé.

Si vous souhaitez ici vérifier votre implémentation à l'aide de scripts, vous pouvez appeler le script de test avec le paramètre 0, à l'aide de la commande `./test-encryption.sh 0`. Vous pourrez alors comparer votre résultat au fichier `test-encryption.0.out`.

Question 3 Ecrivez la fonction `decrypt` [2] qui réalise le calcul complet du déchiffrement. Pour cela, vous réaliserez les fonctions suivantes :

— `init_inverse_ops` qui remplit les tableaux décrivant les inverses des permutations et S-Boxes ;

— `decrypt_round` qui code le déchiffrement d'un round ;

— `decrypt` qui code le déchiffrement complet.

De la même manière que pour le chiffrement, vous pourrez tester votre implémentation en comparant la sortie du script `./test-decryption.sh 0` avec le fichier `test-decryption.0.out`.

Enfin, vous pouvez faire des tests de chiffrement / déchiffrement à l'aide de `./test-identity.sh 0`.

Question 4 Lorsque vous serez convaincu que votre implémentation est correcte, vous pourrez valider la première partie du TP en lançant les trois scripts avec le label correspondant au nom du module, et en envoyant le résultat aux responsables du TP. Vous préparerez une archive contenant toutes les choses utiles à l'aide des commandes suivantes :

```
./test-encryption.sh <label> > test-encryption.out
./test-decryption.sh <label> > test-decryption.out
./test-identity.sh <label> > test-identity.out
tar cvzf partie-1.tgz <fichiers sources> *.out
```

4 Seconde partie : cryptanalyse de la dernière sous-clé

L'objectif de cette partie est de mettre en évidence un chemin différentiel.

Étude statistique des *S-Boxes*

La première chose à faire pour trouver un chemin différentiel est de s'intéresser aux propriétés de la *S-Box*.

Soit $x \in [0..15]$. On s'intéresse à la probabilité qu'une différence (au sens XOR) δ_{in} sur x implique une différence δ_{out} sur $S(x)$, i.e. $P(\delta_{in} \Rightarrow \delta_{out}) = P_{x \in [0..15]}\left[S(x) \oplus S(x \oplus \delta_{in}) = \delta_{out}\right]$.

Question 5 Etablissez le tableau des probabilités $P(\delta_{in} \Rightarrow \delta_{out})$ pour toutes les valeurs de δ_{in} et δ_{out}.

Il est bien entendu conseillé de calculer ce tableau avec quelques lignes de code. Pour cela, vous pourrez utiliser les squelettes donnés dans le répertoire `part2`. Vous fournirez votre résultat sous forme d'un fichier texte.

Question 6 En déduire les probabilités les plus intéressantes pour un chemin différentiel, c'est-à-dire les plus grandes.

1. Pour des raisons de conflit de nommage avec certaines bibliothèques standard C, la fonction s'appelle `minicipher_encrypt` dans le squelette en C.

2. Même remarque pour le code C que précédemment. On utilise ici le nom de fonction `minicipher_decrypt`.

Etablissement d'un chemin différentiel

On propose de s'intéresser au chemin différentiel suivant :
- soit M un mot de 16 bits en entrée du *blockcipher* ;
- on considère $M' = M \oplus 000d$;
- on note cette différence $Delta_1^- = 000d$ car il s'agit de la différence avant la première rangée de *S-boxes* ;
- à la sortie de la première ligne de *S-boxes*, la différence la plus probable[3] entre le chiffrement de M et le chiffrement de M' est $Delta_1^+ = 0001$
- cette différence reste inchangée à travers la permutation et on a donc $Delta_2^- = 0001$ avant le deuxième tour ;
- on s'intéresse ensuite au cas $Delta_2^+ = 000a$, là encore le plus intéressant en terme de probabilités ;
- A travers la permutation, $Delta_2^+$ devient $Delta_3^- = 1010$;
- Avec une bonne probabilité, $Delta_3^+$ vaut alors a0a0, qui reste inchangé par la permutation et on a donc $Delta_4^- = a0a0$.

Ce chemin différentiel est décrit à la figure 5. On s'intéresse donc à l'exécution de MiniCipher en parallèle sur M et $M+000d$. Tant que l'on reste sur les « fils » ou que l'on ajoute les sous-clés, le comportement est linéaire, et la probabilité que tout se passe comme décrit sur la figure est 1. En revanche, il faut utiliser les probabilités calculées à la question précédente pour connaître le comportement de ce chemin différentiel à travers les *S-boxes*.

Question 7 En utilisant les probabilités calculées à la question précédente, donnez la probabilité, pour un couple de messages clairs $(M, M' = M \oplus 000d)$ que $Delta_4^-$ vaille effectivement $a0a0$.

Question 8 De la même manière, quelle est la probabilité du chemin différentiel de la figure 6 ($Delta_1^- = 0b00 \Rightarrow Delta_4^- = 0606$) ?

Récupération du second texte à trous, et des couples de chiffrés

À présent, nous allons utiliser les chemins différentiels précédents pour récupérer la dernière sous-clé k_5.

Prenons par exemple le second chemin différentiel, $Delta_1^- = 0b00 \Rightarrow Delta_4^- = 0606$. Supposons que nous disposions de N couples (C, C') tels que $C = E_K(M)$ et $C' = E_K(M \oplus 0b00)$. Alors, avec une bonne probabilité (p calculée plus haut), nous savons que la différence entre le chiffrement de C et de C' vaut, juste avant la 4ème rangée de *S-Boxes*, $Delta_4^- = 0606$. Il s'agit d'un distingueur sur le dernier tour.

Un premier algorithme pour retrouver k_5 serait le suivant :

```
Pour chaque couple (C, C'),
  Pour chaque sous-clé possible k,
    Inverser l'ajout de k pour C et C'
    On obtient T et T'
    Inverser la 4ème rangée de S-Boxes
    On obtient U et U'

    Si U XOR U' = 0606,
    Alors on ajoute un point pour k
  Fin Pour
Fin Pour
```

3. En réalité, il s'agit d'une des différences les plus probables. Nous avons choisi celle-ci car c'est celle qui donne le meilleur chemin différentiel ensuite.

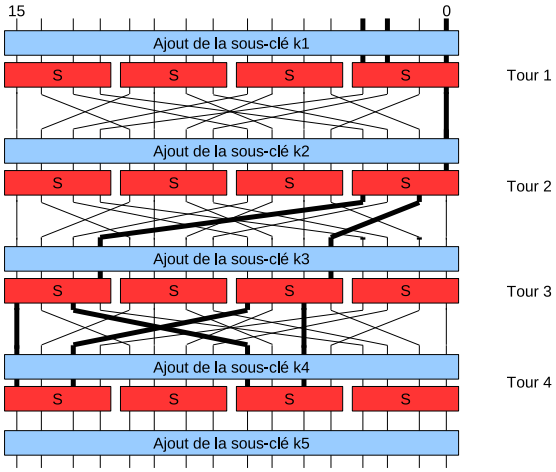

FIGURE 5 – Schéma décrivant le chemin différentiel $Delta_1^- = 000d \Rightarrow Delta_4^- = a0a0$

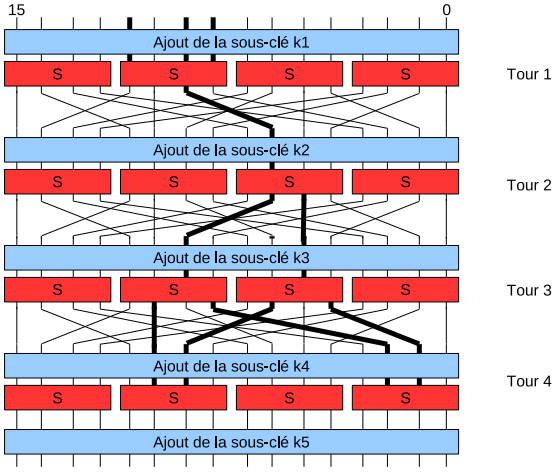

FIGURE 6 – Schéma décrivant le chemin différentiel $Delta_1^- = 0b00 \Rightarrow Delta_4^- = 0606$

A la fin, la clé k_5 valide devrait être celle pour laquelle le rapport $\frac{\text{nombre de points}}{\text{nombre de paires } (C,C')}$ est le plus proche de la probabilité p. Cet algorithme est illustré par la figure 8.

En pratique, la bonne sous-clé est celle pour laquelle le nombre de points obtenus est le plus grand, car la probabilité du chemin est assez grande pour que cela revienne au même.

Le seul souci de cette méthode, c'est qu'il faut réaliser deux boucles imbriquées, une sur les paires de chiffrés, l'autre sur les clés possibles. Afin de réduire cette complexité, voici une alternative à l'algorithme ci-dessus, moins précise mais qui fonctionne dans notre cas, si l'on a suffisamment de couples de chiffrés.

Au lieu d'inverser l'ajout de k_5 et le dernier tour de *S-Boxes* en entier, on ne va s'intéresser qu'à 2 quartets du mot à la fois : pour le chemin différentiel $0b00 \Rightarrow 0606$, on ne s'intéressera qu'aux bits 0 à 3 et 8 à 11 (masque $0f0f$), et pour le chemin différentiel $000d \Rightarrow a0a0$, on ne s'intéressera qu'aux bits 4 à 7 et 12 à 15 (masque $f0f0$).

L'algorithme devient donc, pour $0b00 \Rightarrow 0606$, avec un masque $0f0f$:

```
Pour chaque sous-clé possible k,
  Si k AND NOT mask != 0,
  Alors on passe au k suivant

  Pour chaque couple (C, C'),
    Inverser l'ajout de k pour C et C'
    On obtient T et T'
    Inverser la 4ème rangée de S-Boxes
    On obtient U et U'

    Si (U XOR U') AND mask = 0606,
    Alors on ajoute un point pour k
  Fin Pour

Fin Pour
```

On remarque que l'on a inversé les deux boucles, pour des raisons de performances (en effet, le test pour savoir si la sous-clé k est conforme au masque recherché n'est ainsi effectué qu'une fois pour chaque k).

La figure 8 décrit l'attaque.

Mise en place de l'attaque

Vous trouverez le matériel cryptographique dans le répertoire `crypto-material`. Il contient des fichiers comme `pairs-k5_0b00_0606.txt` qui contient un certain nombre de lignes, qui correspondent aux paires de chiffrés (C, C') tels que $C = E_K(M)$ et $C' = E_K(M \oplus 0b00)$. De même, le fichier `pairs-k5_000d_a0a0.txt` contient des paires de chiffrés (C, C') tels que $C = E_K(M)$ et $C' = E_K(M \oplus 000d)$ [4].

En appliquant le deuxième algorithme au masque $0f0f$ de la sous-clé dans un premier temps, et au masque $f0f0$ dans un second temps, il est possible de récupérer la sous-clé k_5.

Question 9 Ecrivez le code implémentant l'attaque pour le chemin différentiel $0b00 \Rightarrow 0606$, et déduisez-en 8 bits de la sous-clé k_5.

Afin de vérifier votre travail, des paires supplémentaires ont été générées pour la clé nulle, et sont fournis dans le répertoire `crypto-material-null-key`.

4. Il existe aussi une version binaire de ces paires, où C et C' sont encodés en *big endian* (fichiers `.bin`), qui peuvent être plus simples à manipuler en C.

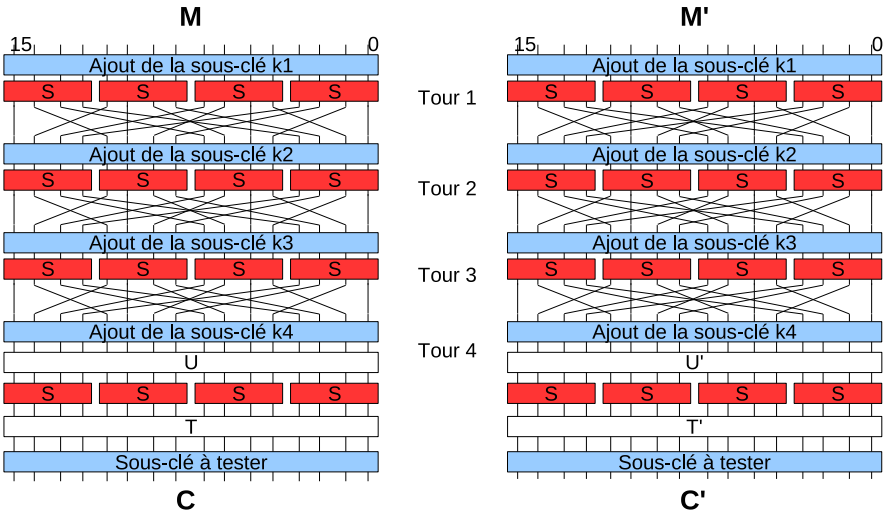

FIGURE 7 – Schéma décrivant le premier algorithme pour trouver k_5

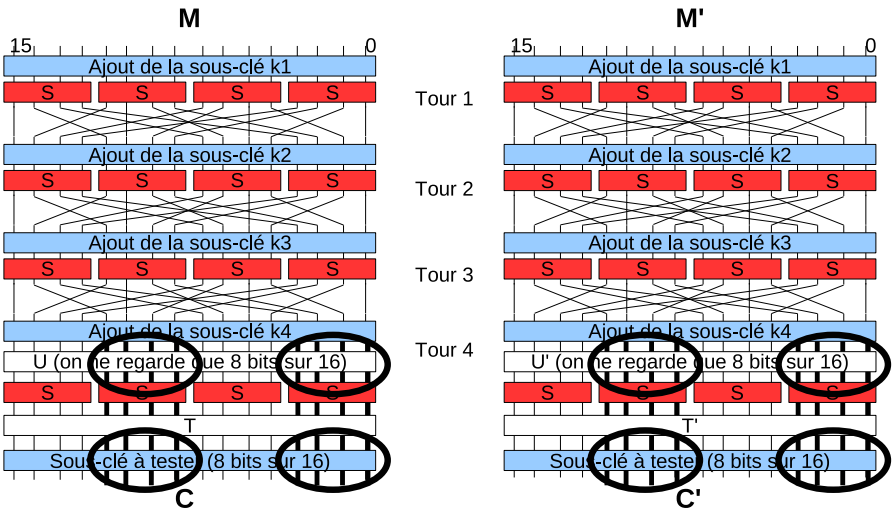

FIGURE 8 – Schéma décrivant le second algorithme pour trouver 8 bits k_5 : on utilise un masque $0f0f$.

Question 10 Recopiez le code précédent et adaptez-le pour le chemin différentiel $000d \Rightarrow a0a0$, puis déduisez-en 8 bits de la sous-clé k_5.

Question 11 Lorsque vous aurez reconstitué la sous-clé k_5, vous pourrez valider la seconde partie du TP en inscrivant votre sous-clé dans un fichier texte, et en envoyant vos sources et ce fichier aux responsables du TP. Vous préparerez une archive contenant toutes les choses utiles à l'aide des commandes suivantes :

```
echo <Votre clé K5> > K5
tar cvzf partie-2.tgz K5 <fichiers sources>
```

5 Troisième partie : cryptanalyse de la clé complète

Cette partie donne les pistes permettant de retrouver toute la clé correspondant à votre groupe. N'hésitez pas à demander conseil par courrier électronique en cas de souci.

Il s'agit à présent de retrouver, dans l'ordre, les sous-clés k_4 et k_3, à l'aide du même algorithme que ci-dessus. Ensuite, on récupèrera k_2 et k_1 par des méthodes plus simples.

Récupération de la sous-clé 4 et de la sous-clé 3

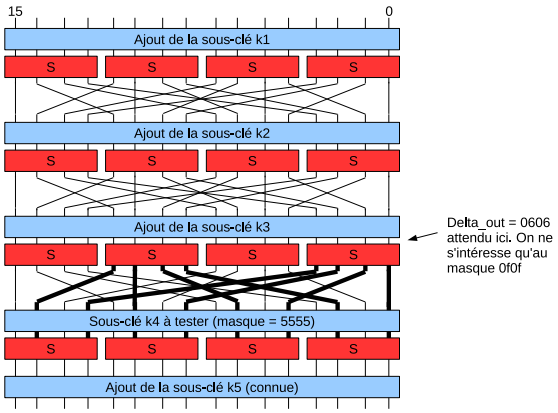

FIGURE 9 – Schéma décrivant le second algorithme pour trouver 8 bits k_4 : on considère seulement les sorties des S-Boxes 0 et 2 (i.e. un masque $0f0f$.), qui se transforme en un masque 5555 sur les bits de la clé k_4 à tester.

La figure 5 décrit l'adaptation du second algorithme à la recherche de 8 bits (masque 5555) de la sous-clé k_4. En effet, on considère une différentielle de la forme $Delta_1^- = 0040 \Rightarrow Delta_3^- = 0606$ et la sortie des *S-Boxes* 0 et 2 se reflète sur les bits du masque 5555 de la sous-clé k_4.

Question 12 Détailler le chemin différentiel $Delta_1^- = 0040 \Rightarrow Delta_3^- = 0606$ et calculer sa probabilité.

Question 13 Détailler de même le chemin différentiel $Delta_1^- = 0005 \Rightarrow Delta_3^- = a0a0$ et calculer sa probabilité.

Question 14 Suite à la validation de la partie 2, vous recevrez le fichier `TP-Partie3-GroupeG.tar.gz`. Celui-ci contient le corrigé de la partie précédente, ainsi que du code permettant d'exploiter les chemins décrits ci-dessus. Utilisez le code pour retrouver k_4.

Le répertoire `crypto-material` contient en particulier des fichiers `pairs-*` contenant le contenu nécessaire aux attaques de cette partie.

Question 15 Sachant que la sous-clé k_3 peut se trouver avec des différentielles similaires à celles utilisées ci-dessous, trouver les 2 chemins différentiels utiles de la forme $Delta_1^- = 0220 \Rightarrow Delta_2^- = ?$ et $Delta_1^- = 1010 \Rightarrow Delta_2^- = ?$, ainsi que les probabilités de ces chemins.

Question 16 À l'aide des chemins différentiels trouvés et des variables intitulées `pairs_k3_delta_in_0220` et `pairs_k3_delta_in_1010`, adaptez le code de calcul de la sous-clé k_4 pour retrouver k_3.

Récupération des sous-clés 2 et 1

Question 17 Achevez l'attaque pour trouver k_2 et k_1.

Pour k_2, n'importe quel chemin différentiel a une probabilité de 1. En considérant k_2 par morceaux de 8 bits, comme précédemment, il est facile de retrouver la sous-clé.

Enfin, k_1 se retrouve à l'aide d'un simple couple clair/chiffré, puisqu'on peut remonter tout le reste de la fonction de déchiffrement.

Déchiffrement du message donné à votre groupe

Question 18 Une fois que vous avez retrouvé toute la clé, déchiffrez le message `message.xyz` présent dans les ressources en mode CBC.

A Schéma de MiniCipher pour décrire un chemin différentiel