Attaque différentielle

Paul Rozière

janvier 15, 2020

Le texte de cette feuille `tpdiff.html` et les fichiers référencés sont accessibles à partir du répertoire `tpdiff/` .

1 Objectif

Le but de la séance est de mettre en oeuvre une attaque différentielle sur le chiffrement SPN simplifié qui est présenté dans le tutoriel de Howard Heys : `http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf`.
Il s'agit d'un chiffrement simplifié à but pédagogique : la taille de bloc est très petite (16 bits), il n'y a pas de procédure de diversification de clef (la clef de chiffrement est utilisée directement par segments comme clef de tour). Mais sa structure générale est celle d'un SPN (Substitution-Permutation-Network), dans le style de celle de l'AES (voir aussi en annexe ./heys.pdf).

2 Description du chiffrement

Le chiffrement agit en 4 tours sur des blocs de 16 bits.
`typedef uint16_t block_t;`

2.1 Les substitutions et permutations

Il utilise une substitution : `block_t heys_subst(block_t, sbox_t);`
obtenue en juxtaposant 4 fois la même substitution sur 4 bits, l'image de l'entier de 4 bits i est `sbox[i]` :
`sbox_t sbox = {0xe, 0x4, 0xd, 0x1, 0x2, 0xf, 0xb, 0x8, 0x3, 0xa, 0x6, 0xc, 0x5, 0x9, 0x0, 0x7};`
une permutation sur 16 bits, `block_t heys_perm(block_t);` décrite ci-dessous (le i-ème bit se trouve en position `pbox[i]` : `pbox = {0x0, 0x4, 0x8, 0xc, 0x 1, 0x 5, 0x 9, 0xd, 0x2, 0x6, 0xa, 0xe, 0x 3, 0x 7, 0xb, 0xf}`
et 5 clefs de tour de chacune 16 bits. `k[0], k[1], k[2], k[3], k[4]`
Le schéma du chiffrement est le suivant (le dernier tour n'utilise pas de permutation) :
`block_t m; // clair` `block_t r; // chiffré` `r = m ^ key[0]`

- 1er tour `r = heys_subst(r)`
  - `r = heys_perm(r)`
  - `r = r ^ key[1]`
- 2nd tour
  - `r = heys_subst(r)`
  - `r = heys_perm(r)`
  - `r = r ^ key[2]`
- 3ème tour
  - `r = heys_subst(r)`
  - `r = heys_perm(r)`
  - `r = r ^ key[3]`
- 4ème tour
  - `r = heys_subst(r)`
  - `r = r ^ key[4]`

2.2 La clef

Pour simplifier l'attaque on va utiliser une clef de 40 bits (cf. Stinson, Cryptography Theory and Practice), soit `key`. On note `key[i]` le i+1-ème bit de `key` (on commence donc à 0), compté à partir du poids fort. Les clefs de tour sont engendrées de la façon suivante :

— k[0] = key[0] ... key[15]
— k[1] = key[6] ... key[21]
— k[2] = key[12] ... key[27]
— k[3] = key[18] ... key[33]
— k[4] = key[24] ... key[39]

Ni le chiffrement ni la longueur de la clef ne sont très réalistes, mais le principe est à peu près celui de l'AES (en dehors de la procédure pour engendrer les clefs de tour, la clef de chiffrement n'est pas utilisée directement comme ici, les clefs de tour sont engendrées par des transformations non linéaires).
Vous disposez des fichiers nécessaires dans le répertoire `tpdiff/` .

3 Attaque différentielle

On rappelle que l'attaque différentielle repose sur la recherche d'une corrélation entre une certaine différence (`xor`) entre deux textes clairs en entrée et une différence en sortie après n-1 tours, quand le chiffrement est à n tours, donc dans le cas de ce chiffrement après 3 tours.

3.1 Table de distribution des différences

La première étape consiste à fabriquer une table des différences pour la S-box du chiffrement (cf. cours et tutoriel de Heys). Cette table des différences fait apparaître des probabilités plus importantes pour certains couples de différences entrée/sortie.
Exercice 1. Produire la table de différence de la S-box du chiffrement de Heys (sur 4 bits).

3.2 Caractéristique différentielle

On construit ensuite une *caractéristique différentielle* en composant par le biais des permutations des couples de différences entrée/sortie qui ont une probabilité importante d'apparition. La différence de sortie d'une ou plusieurs S-boxes au tour t doit être envoyée au tour t+1 en entrées d'une ou plusieurs S-boxes, de façon à que ce soit la différence d'entrée d'un couple de différences intéressant (cf. cours et tutoriel de Heys).
Pour ce chiffrement on s'aperçoit que la différence en entrée :
`De = [0000 1011 0000 0000]`
produit au 3ème tour une différence de sortie de :
`Ds = [0000 0110 0000 0110]`
avec une probabilité de ~ 2,6 %
Exercice 2. Donner les écritures en hexadécimal des deux différences.
Exercice 3. Vérifier la caractéristique (utiliser le schéma donné en cours) et justifier le résultat ci-dessus.

3.3 Décryptage

L'attaque est une *attaque à texte clair choisi*. Vous disposez donc d'un binaire qui prend en entrée un fichier de taille paire en octets, et renvoie le texte chiffré correspondant (binaire!) sur la sortie standard. Le binaire utilise le chiffrement de Heys avec une certaine clef de chiffrement (40 bits), qu'il s'agit de déterminer.
Le binaire se trouve sous gitlab (répertoire `tp5/`), au format elf 64 bits (linux).

3.3.1 Détermination d'une partie de la clef par étude des différences

1. On construit un échantillon statistiquement significatif de couples de messages (de la taille d'un bloc) ayant une différence de `De`, et les chiffrés correspondants.
Exercice 4. Programmer une fonction qui construit un échantillon pseudo-aléatoire de couples de blocs de 16 bits avec une différence fixée. La fonction prend par exemple en entrée la différence, le nombre de couples. La sortie est envoyée sur `stdout` (rediriger ensuite dans un fichier), les blocs de 16bits sont simplement juxtaposés.
Une façon de produire des données pseudo aléatoires est de lire le fichier spécial `/dev/urandom`.
Sous la plupart des unix (linux, bsd, ...), les fichiers spéciaux (devices) `/dev/urandom` et `/dev/random` (voir `man 4 random`) fournissent chacun une interface avec un générateur aléatoire ou pseudo-aléatoire. L'aléa est produit à partir de l'activité générale de la machine et de fonctions pseudo-aléatoires. La lecture sur le device `/dev/random` est bloquée tant qu'il n'y a pas suffisamment de "bruit" (entropie) pour fournir une donnée vraiment aléatoire. La lecture sur `/dev/urandom` est non bloquante : l'aléa produit est de moins bonne qualité. Dans ce cas particulier il est complètement inutile d'avoir des données vraiment aléatoires (par contre c'est important pour la génération de clefs dans le protocoles

cryptographiques). On peut donc utiliser `/dev/urandom`. On pourrait tout aussi bien utiliser directement une fonction pseudo-aléatoire (`man 3 random`).
Pour lire et écrire un fichier : `man 2 open`, `man 2 read`, `man 2 write` ou `man fopen`, `man fread`, `man fwrite`. Le descripteur de fichier de la sortie standard est `1` (`0` pour l'entrée standard, `2` pour la sortie erreur standard).
2. On peut ainsi retrouver les bits de la 5ième clef de tour correspondant au second et au 4ième segment de 4 bits pour cette clef, en essayant les 256 sous-clefs correspondantes pour déchiffrer les couples de messages chiffrés obtenus à la première étape. On calcule les différences obtenues et on compte pour chaque clef la fréquence d'apparition de la différence `Ds`. La clef partielle correcte donne une fréquence qui est maximale (et proche de celle calculée ˜ 2,6 %).
Exercice 5. Déterminer par la méthode indiquée une partie de la clef utilisée par le binaire fourni.

3.3.2 Détermination du reste de la clef par force brute

On déchiffre ensuite le reste de la clef par recherche exhaustive sur quelques couples de clairs chiffrés.
Exercice 6. Construire l'attaque, déterminer la clef utilisée par le binaire fourni.