Devoir #1

Sécurité informatique - IFT 3275 / IFT 6271

30 janvier 2020

Remise du devoir

- La date de remise est le 20 février à 12:30 (heure de l'examen intra).
- Aucun retard ne sera permis.
- Le devoir est à faire en **équipe de deux**.
- Le rapport contenant les réponses à la partie théorique doit être effectué en LaTeX et remis en format PDF.
- La partie pratique doit être remise en trois fichiers `.java` nommés `Bruijn.java`, `Decrypt.java` et `Differential.java` qui seront votre version modifée des fichiers squelettes, ainsi que le fichier texte `result.txt`.
- Vos noms et matricules doivent être inscrits au début de chaque fichier.

Partie Théorique

1. (5 Points) - Démontrez que pour une distribution **uniforme** de fréquences f_i de n lettres d'un language, nous avons :

$$\sum_{i=0}^{n-1} f_i^2 = f_i$$

2. (10 Points) - Soit un réseau de Feistel composé de deux "rounds" utilisant les fonctions de "rounds" f_1 et f_2. Démontrez que :

$$\textbf{Feistel}_{f_1,f_2}(L_0, R_0) = (L_2, R_2) \implies \textbf{Feistel}_{f_2,f_1}(R_2, L_2) = (R_0, L_0)$$

3. (10 Points) - Démontrez la propriété de complémentarité de DES, c'est-à-dire que :

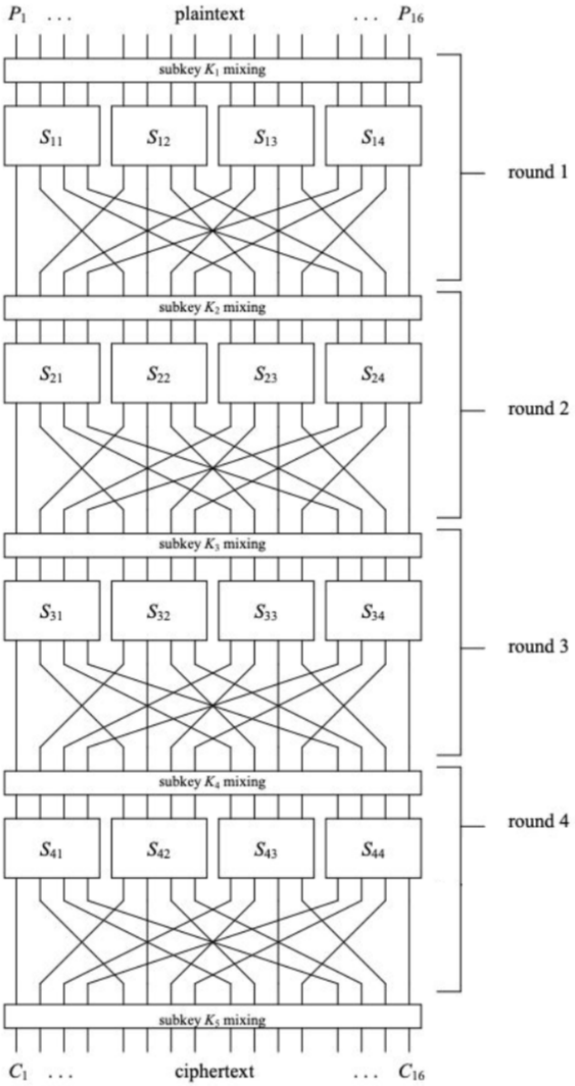
$$DES_k(m) = \overline{DES_{\overline{k}}(\overline{m})}$$

pour toute clef k et message m (où $\overline{x}$ représente la négation logique bit à bit de x).

4. (30 Points) - Soit le réseau de permutation-substitution ci-dessous composé des boîtes à substitutions et des permutations suivantes :

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	B	5	4	C	6	3	9	A	D	F	1	0	E	8	7	2

input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16



Effectuez une cryptanalyse différentielle de ce SPN en incluant dans votre rapport :

- La table démontrant les fréquences des différences de sortie Δ_y (colonnes) pour chaque différence d'entrée Δ_x (lignes).
- Le schéma du SPN démontrant une différentielle caractéristique. Tracez des lignes sur le schéma précédent et indiquez votre différence d'entrée ΔP ainsi que celle de votre intermédiaire ΔI.

Partie Pratique

4. (Suite) - Une fois ceci fait, vous devez retrouver la clef maître k **de votre équipe** à partir de votre sous-clef partielle k_5^* qui est maintenant à votre portée en faisant une *attaque à texte choisi* utilisant votre différentielle caractéristique.

   Le fichier squelette `Differential.java` contient les fonctions qui vous permettront de suivre la procédure qui a été décrite durant la troisième démonstration. **Voir le fichier PDF sur le site du cours**. Vous n'avez qu'à ajouter du code où il est inscrit `//TO DO`. Vous devez inscrire votre numéro d'équipe à l'affectation de la varible `teamNumber` au début du fichier. Pour connaître votre numéro d'équipe, veuillez inscire vos noms dans ce tableau.

   Le fichier `Differential.java` dépend de `SPNServer.java` qui se charge de transmettre vos messages clairs à un serveur qui retourne vos chiffres qui ont été encryptés avec une clef maître qui correspond à votre numéro d'équipe. L'utilisation du serveur est expliquée par le besoin évident de vous cacher la valeur de votre clef.

   Une fois votre sous-clef partielle k_5^* découverte, vous pourrez en déduire une clef maître partielle k^* en suivant **la même préparation des sous-clefs que celle de la troisième démonstration**. Vous aurez donc 8 des 20 bits de votre clef maître.

   Il suffit ensuite de faire une **fouille exhaustive** pour découvrir les valeurs manquantes de votre clef. Pour ce faire, vous devez chiffrer un texte clair avec le serveur et ensuite vérifier si vous obtenez le même chiffre en chiffrant vous-même le même texte clair en essayant les 2^{12} valeurs possibles de k.

Une fois que vous avez une correspondance, il est important de **vérifier que vous avez bien la bonne clef** (et qu'il ne s'agisse pas d'un coup de chance) avec un autre texte clair pour la même clef candidate.

Utiliser `teamNumber = 0` fait en sorte que le serveur chiffre les messages à l'aide de l'exemple de la troisième démonstration. Ceci est **très utile** pour le débogage puisque vous connaissez déjà la valeur de la clef maître.

5. (25 Points) - Performez une *attaque à texte chiffré seulement* sur le fichier `cipher.txt` sachant que le chiffrement de Vigenère a été utilisé sur un texte écrit en anglais. Vous devez modifier le fichier squelette `Decrypt.java`.

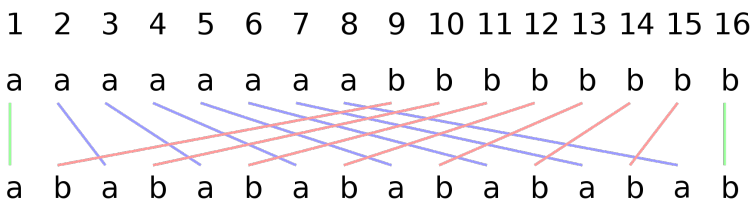
   Vous devez trouver la longueur de la clef et ensuite trouver le décalage pour chaque courant (cipher stream) dans le texte chiffré. La procédure à suivre a été expliquée en détail lors de la première démonstration. **Voir le fichier PDF sur le site du cours**.

6. (20 Points) - *Mise en situation.* Vous travaillez pour une firme de sécurité informatique qui offre parmi sa gamme de produits une sorte de clef passe-partout électronique pour les portes de garage.

   Ce produit ouvre n'importe quelle porte de garage en émettant en ondes radio toutes les combinaisons possibles de 4 chiffres allant de 0 à 9. La porte de garage s'ouvre donc une fois sa clef transmise par fouille exhaustive.

   En regardant le code Java de l'ingénieur qui a implémenté cette attaque, vous remarquez qu'une série de 4×10^4 chiffres est utilisée pour cette fouille, soient 4 chiffres pour chacune des 10^4 combinaisons. En vous remémorant la première démonstration de votre cours de sécurité informatique, vous vous rendez compte qu'utiliser **un cycle de de Bruijn** qui contient toutes ces combinaisons serait beaucoup plus intelligent puisque le système de portes de garage ne considère que les 4 derniers chiffres envoyés.

   Vous implémentez donc un algorithme qui génère ce cycle en concaténant en ordre lexicographique tous les **mots de Lyndon** dont la longueur divise $n = 4$ en utilisant l'inverse de la **transformation de Burrows-Wheeler**. En voici un exemple dans le cas de $B(2, 4)$:



(a) Écrire une répétition de l'alphabet pour former un mot de longueur k^n, soit $2^4 = 16$ dans notre exemple.

(b) Placer au dessus du mot en (a) une copie triée.

(c) Établir les correspondances (lignes dans le dessin) entre le $i^{ème}$ symbole dans le mot du haut et celui du bas. Par exemple, il y a une ligne entre le `a` à l'index 2 du mot du haut et le `a` à l'index 3 du mot du bas puisqu'il s'agit du deuxième `a` dans chaque mot.

(d) Nous créons ensuite des cycles en partant du symbole à l'index 1 du mot du haut. Le premier cycle est trivial: nous partons du `a` à l'index 1 qui nous mène au `a` à l'index 1 du bas. Une fois en bas, nous regardons si nous sommes à l'indice où nous avons débuté dans le mot du haut. Si oui, le cycle prend fin. Si non, nous suivons la prochaine correspondance du symbole du haut vers le bas et ainsi de suite. Pour le premier cycle, nous voyons que nous sommes où nous avons débuté. Nous avons donc (1) comme cycle. Nous partons ensuite du `a` à la position 2 du haut. Ce dernier nous mène vers le `a` à l'index 5 du mot du bas. Puisque nous avons commencé à 2 et non à 5, nous suivons la correspondance entre le a à la position 5 du mot du haut et le `a` à la position 9 du mot du bas. Nous allons ensuite du `b` à la position 9 du mot du haut vers le `b` à la position 2 du mot du bas. Nous sommes où nous avons débuté et le cycle prends fin. Nous avons donc le cycle (2359). Les autres cycles sont détectés de manière similaire.

(e) Nous écrivons tous les cycles présents en ordre croissant :

$$(1)(2\ 3\ 5\ 9)(4\ 7\ 13\ 10)(6\ 11)(8\ 15\ 14\ 12)(16)$$

(f) Nous substituons chaque chiffre pour le symbole du mot du haut à cet index, ce qui nous donne :

$$B(2,4) = aaaabaabbababbbb$$

Vous devez modifier le fichier squelette `Bruijn.java` pour implémenter cet algorithme qui devra générer $B(10,4)$ de sorte qu'il agisse **exactement comme la série de 40 000 chiffres qu'il remplacera**. Voyez les notes de la démonstration #1 pour plus de détails sur les cycles de de Bruijn. Cet exercice est inspiré d'une vraie attaque sur les systèmes de portes de garage. Si ça vous intéresse, regardez ce vidéo.