DM: Cryptographie Symétrique

christina.boura@uvsq.fr

19 mars 2020

Modalités pratiques : Le projet doit être envoyé par mail à l'adresse `christina.boura@uvsq.fr`. La date limite pour le rendre est le **mardi 7 avril à 20h**. Un point de pénalité sera attribué à chaque heure de retard. Vous pouvez travailler seuls ou en groupe de 2 ou 3 personnes. Dans le cas des groupes, il est impératif d'indiquer clairement qui a fait quoi dans le projet. Le projet doit être rendu sous forme d'archive `.zip` et doit contenir :
- un rapport sous forme `.pdf` avec les réponses aux questions ;
- le code demandé (un ou plusieurs fichiers) ;
- un fichier `README.txt` indiquant comment compiler, interpréter et utiliser votre code ;
- un fichier `organisation.txt` indiquant clairement qui a fait quoi dans le projet (uniquement pour les projets à 2 ou 3 étudiants).

Des points bonus seront attribués à la présentation et la clarté du rapport.

1 Générateur de type Geffe pour le chiffrement à flot

On considère le générateur de nombres pseudo-aléatoires pour le chiffrement à flot décrit dans la figure 1. Ce générateur qui suit le même principe que le générateur de Geffe, emploie trois LFSR L_0, L_1 et L_2 de taille 16 bits chacun. Ces trois LFSR sont initialisés avec une clé $K = (k_0, k_1, k_2)$ de 48 bits. Plus précisément, le contenu du registre L_0 est initialisé avec la clé k_0, le contenu du registre L_1 est initialisé avec la clé k_1 et finalement le contenu du registre L_2 est initialisé avec la clé k_2. À chaque fois, le bit de poids faible de chaque clé initialise le bit le plus à droite de chaque registre. Les coefficients de retroaction $(c_{15}, c_{14}, \ldots, c_1, c_0)$ pour chacun des trois registres sont :

L_0 : $(c_{15}, c_{14}, \ldots, c_1, c_0) = (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1)$

L_1 : $(c_{15}, c_{14}, \ldots, c_1, c_0) = (0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1)$

L_2 : $(c_{15}, c_{14}, \ldots, c_1, c_0) = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1)$

À chaque coup d'horloge, les bits de sortie des trois registres sont combinés à l'aide d'une fonction de filtrage F afin de produire un bit de la suite chiffrante s_i. Plus précisément, un bit s_i de la suite chiffrante est calculé comme $s_i = F(x_0, x_1, x_2)$, où x_i est le bit de sortie du registre L_i, $i = 0, 1, 2$.

La fonction de filtrage $F : \{0,1\}^3 \to \{0,1\}$ prend trois bits en entrée et donne un bit en sortie. On la décrit par sa représentation en table (on donne la valeur de sortie pour chacune des 8 valeurs possibles en entrée) :

$x_0x_1x_2$	000	100	010	110	001	101	011	111
$F(x_0x_1x_2)$	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7

Chaque f_i, $i = 0 \ldots, 7$ est une valeur binaire (0 ou 1). Pour l'instant on laisse la description de F générique et on la spécifiera plus tard.

1. Implémenter en `C` ce générateur. Votre programme doit prendre en entrée les valeurs $f_0, f_1, \ldots, f_7$ définissant la fonction de filtrage F, la clé $K = (k_0, k_1, k_2)$ et un entier n. Il doit produire et afficher les n premiers bits de la suite chiffrante.
2. Expliquer comment calculer théoriquement la correlation entre la sortie du générateur s_i et la sortie de chaque LFSR. Calculer ensuite ces correlations pour toutes les fonctions de filtrages $F = (f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7)$ possibles.

   On suppose à partir de maintenant que $(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7) = (1, 0, 0, 0, 1, 1, 1, 0)$.

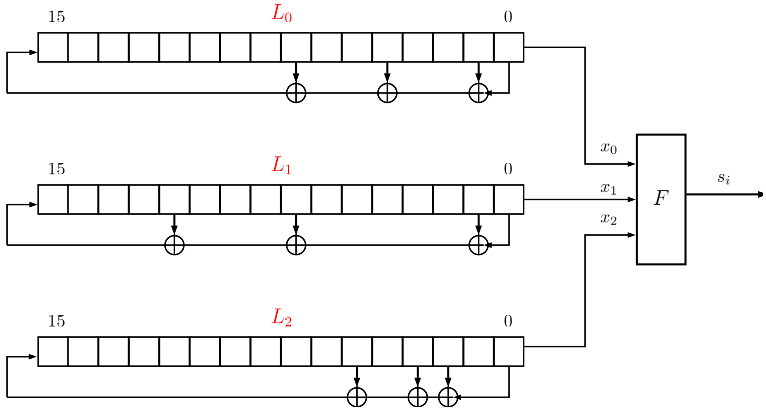

Figure 1 – Générateur de nombres aléatoires

3. Utiliser la réponse à la question précédente pour monter une attaque de type *diviser pour régner* contre ce générateur afin de récupérer la clé secrète $K = (k_0, k_1, k_2)$ qui a servi pour l'initialisation des trois LFSR.
4. Donner une estimation du nombre de bits de la suite chiffrante que l'attaquant doit connaître pour mettre l'attaque en œuvre, la complexité en temps et en mémoire de cette attaque. Comparez la complexité en temps par rapport à la complexité de la recherche exhaustive de la clé.
5. Implémenter l'attaque en `C` afin de récupérer la clé (k_0, k_1, k_2).
6. Donner un exemple de fonction F qui rend l'attaque contre ce générateur la plus difficile possible.

2 Un chiffrement par bloc faible

On définit un chiffrement par bloc E_{k_0,k_1}. Ce chiffrement est de type Feistel, il opère sur des blocs de 64 bits et utilise deux clés k_0 et k_1 de 32 bits chacune. La fonction de tour de ce chiffrement est représentée à la figure 2, où x_r^L et x_r^R sont deux registres (mots) de 32 bits chacun stockant l'état interne de la fonction au tour r. (x_0^L, x_0^R) est le texte clair et (x_{12}^L, x_{12}^R) est le texte chiffré.

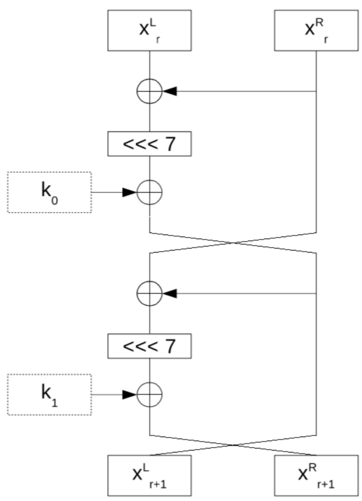

Figure 2 – Le chiffrement E_{k_0,k_1}.

L'opération $\lll 7$ applique une rotation de 7 bits vers la gauche sur la branche gauche. Par exemple

$$(1111000001111000000101010101010011) \lll 7 = (0111100000010101010101001111111000)$$

1. On réduit pour le moment le chiffrement à un tour. Calculer le résultat (x_1^L, x_1^R) du chiffrement après un tour, si on suppose que

$$(x_0^L, x_0^R) = (\texttt{0x45019824}, \texttt{0x51023321}),\ k_0 = \texttt{0x01020304} \text{ et } k_1 = \texttt{0x98765432}.$$

2. On suppose toujours que le chiffrement ne fait qu'un tour. Les clés k_0 et k_1 sont inconnues. On suppose que l'attaquant peut choisir de chiffrer autant de textes clairs (x_0^L, x_0^R) de son choix et obtenir les chiffrés (x_1^L, x_1^R) correspondants. Écrire le chiffrement sous forme de système d'équations, où les bits des clés k_0 et k_1 sont les inconnues du système. Montrer comment résoudre le système. Implémentez cette approche. Votre programme doit prendre en entrée autant de textes (claires/chiffrés) choisis nécessaires et retourner la clé secrète (k_0, k_1).
3. Généraliser votre cryptanalyse au chiffrement complet (12 tours).
4. Implémenter l'attaque dans le langage de programmation de votre choix. Votre programme doit prendre en entrée le nombre nécessaire de couples (clair, chiffré) et de retourner la clé secrète (k_0, k_1).
5. Est-ce que ajouter plus de tours rendra le chiffrement plus solide ? Justifier votre réponse.
6. Proposer une amélioration du chiffrement E_{k_0,k_1} de façon que votre attaque ne s'applique plus.