DM: Attaque contre le chiffrement à flot CSS

christina.boura@uvsq.fr

5 avril 2024

1 Généralités

Le but de ce devoir-maison d'implanter en `Python` une attaque pratique contre CSS, un chiffrement à flot utilisé pour sécuriser le contenu des disques DVDs.

Modalités pratiques Vous pouvez travailler seuls ou en groupe de 2 personnes au plus. Bien évidement, chaque équipe devra travailler indépendamment des autres. La date limite pour rendre votre projet est le **vendredi 26 avril** à **23 heures**. Aucun retard ne sera toléré et les projets rendus tardivement ne seront pas notés. Les projets doivent être envoyés par mail à l'adresse `christina.boura@uvsq.fr` sous la forme d'une archive compressée ayant comme nom votre nom de famille (ou les deux noms si vous travaillez en binôme). Cette archive doit contenir :

- Un seul fichier `.py` contenant tout le code. Le code doit être propre et bien commenté. Il doit être clair à quelle partie du DM correspond chaque morceau de code et comment chaque partie peut être testée.
- un fichier `README` expliquant comment tester votre code.
- un fichier supplémentaire sous forme `.pdf`, contenant les réponses aux questions théoriques. Les réponses doivent être claires et bien développées. D'autres précisions et remarques peuvent y être présentes.

2 Description du projet

2.1 LFSR

Un registre à décalage à rétroaction linéaire (LFSR : Linear Feedback Shift Register) permet de construire de générateurs de nombres pseudo-aléatoires (PRNG) très rapides en matériel. Plusieurs produits électroniques utilisent des chiffrements à flot basés sur des LFSRs comme les lecteurs de DVD. Un LFSR consiste en un registre $\mathtt{s} = (\mathtt{s}[\mathtt{n}-1], \ldots, \mathtt{s}[0])$ de longueur n bits. À chaque coup d'horloge le bit le plus à droite, c.-à-d. $\mathtt{s}[0]$ est extrait pour former une suite binaire $\mathtt{y}$ de longueur ℓ. Ensuite on calcule une valeur binaire $\mathtt{b}$ comme le XOR de certains bits du registre $\mathtt{s}$. Les bits du registre qui feront partie de cet XOR sont donnés par les coefficients de retroaction $(\mathtt{c}_{n-1}, \ldots, \mathtt{c}_1, \mathtt{c}_0)$, avec $\mathtt{c}_i \in \{0,1\}$, $i = 0, \ldots, n-1$. En effet $\mathtt{b}$ est calculé comme

$$\mathtt{b} = \mathtt{c}_{n-1}\mathtt{s}[\mathtt{n}-1] \oplus \cdots \oplus \mathtt{c}_1\mathtt{s}[1] \oplus \mathtt{c}_0\mathtt{s}[0].$$

Enfin tous les bits du registre sont décalés vers la droite et on recopie $\mathtt{b}$ dans $\mathtt{s}[\mathtt{n}-1]$. Un exemple d'un LFSR de taille $n = 8$ est donné ci-dessus. Cette procédure est décrite par l'algorithme 1.

Exemple Le LFSR ci-dessous a une taille de $n = 8$ bits et ses coefficients de retroaction sont $(\mathtt{c}_7, \ldots, \mathtt{c}_1, \mathtt{c}_0) = (0, 0, 0, 1, 1, 1, 0, 1)$.

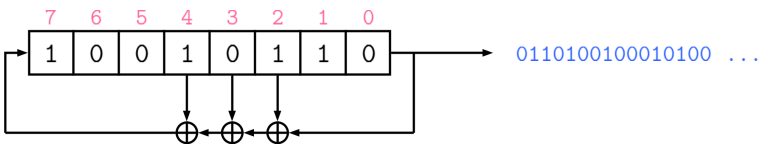

Figure 1 – LFSR de taille 7

Algorithme 1 : Fonctionnement d'un LFSR avec coefficients de rétroaction $(\mathtt{c}_{n-1}, \ldots, \mathtt{c}_1, \mathtt{c}_0)$

Données : $\mathtt{s} = (\mathtt{s}[n-1], \ldots, \mathtt{s}[0]) \in \{0,1\}^n$, $\mathtt{s} \neq 0^n$ et $\ell > n$

Résultat : $\mathtt{y} \in \{0,1\}^\ell$

pour $i = 1, \ldots, \ell$ **faire**

  Renvoyer $\mathtt{s}[0]$; `/* renvoyer un bit */`

  $\mathtt{b} \leftarrow \mathtt{c}_{n-1}\mathtt{s}[n-1] \oplus \cdots \oplus \mathtt{c}_1\mathtt{s}[1] \oplus \mathtt{c}_0\mathtt{s}[0]$; `/* calculer la valeur de retroaction b */`

  $\mathtt{s} \leftarrow (\mathtt{b}, \mathtt{s}[n-1], \ldots, \mathtt{s}[1])$; `/* décaler d'une position vers la droite */`

2.2 CSS

Le Content Scrambling System (CSS) est un chiffrement à flot utilisé pour sécuriser le contenu des disques DVDs en chiffrant le contenu des films. CSS a été conçu dans les années 1980, époque où la clé secrète des chiffrements exportables était limitée à 40 bits. En conséquence, CSS utilise une clé de 40 bits seulement. Ce chiffrement est aujourd'hui connu d'avoir plusieurs faiblesses et le but de ce devoir maison est d'implémenter une attaque pratique contre ce système.

Le chiffrement CSS est construit à partir de deux LFSRs s_1 et s_2 de tailles respectives de 17 et 25 bits. Le fonctionnement de ce chiffrement est détaillé dans l'algorithme 2 et peut être illustré par la figure 2.

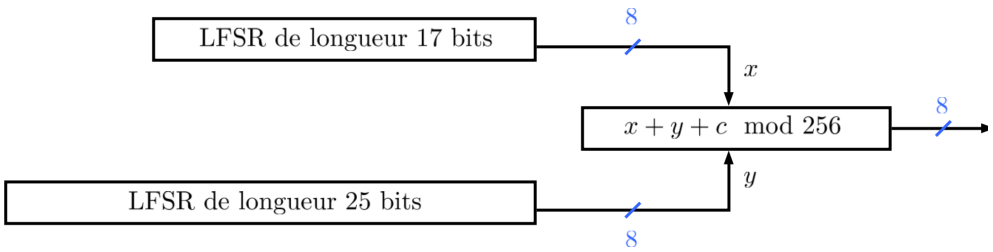

FIGURE 2 – Le Content Scrambling System (CSS)

Algorithme 2 : CSS

Données : Une valeur initiale $s \in \{0,1\}^{40}$

Écrire $s = s_1 \| s_2$ où $s_1 \in \{0,1\}^{16}$ et $s_2 \in \{0,1\}^{24}$;

Charger $1\|s_1$ dans le LFSR de 17 bits;

Charger $1\|s_2$ dans le LFSR de 25 bits;

$c \leftarrow 0$; `/* carry bit */`

tant que *forever* **faire**

  Exécuter les deux LFSRs pendant 8 cycles et obtenir $x, y \in \{0,1\}^8$;

  Traiter x et y comme des entiers dans $\{0 \ldots 255\}$;

  return $z = x + y + c \bmod 256$;

  Si $x + y > 255$, alors $c \leftarrow 1$, sinon $c \leftarrow 0$; `/* carry bit */`

CSS renvoie un octet à chaque itération. Le premier bit sorti de chaque LFSR formera le bit de poids faible du mot x et y respectivement. Les coefficients de retroaction pour les deux LFSRs sont fixes. Le LFSR de 17 bits a comme seuls coefficients non-nuls les coefficients $\{14, 0\}$ et celui de 25 bits les coefficients $\{12, 4, 3, 0\}$. L'ajout du bit à 1 pour s_1 et s_2 dans la position à poids fort ($\mathtt{s}_1[16]$ et ($\mathtt{s}_2[24]$, garantit que les LFSRs ne seront pas initialisés à 0. Les autres bits (à part le bit de poids fort de l'un et le bit de poids fort de l'autre) des deux LFSRs sont initialisés avec les 40 (16 + 24) bits de la clé secrète.

2.3 À vos machines

1. Programmer en `Python` le premier LFSR de 17 bits où l'état sera représenté par une seule variable. Implementer une fonction de test qui vérifie que l'état prend bien les $2^{17} - 1$ valeurs différentes pour une initialisation quelconque non-nulle du registre $s_1 = \{0,1\}^{17}$.
2. Programmer le second LFSR de 25 bits.
3. Programmer l'opération de chiffrement et de déchiffrement d'un texte avec CSS. Chiffrer le message $m =$ `0xfffffffffff` avec CSS initialisé à $s =$ `0x0` $\in \{0,1\}^{40}$. Vérifier que le chiffré correspondant est $c =$ `0xffffb66c39`. Vérifier que le déchiffrement se passe correctement.
4. Soit x_1, x_2, x_3 les 3 premiers octets de sortie du LFSR de 17 bits. Montrer que l'état initial s_2 du second LFSR peut être obtenu en fonction de (z_1, z_2, z_3) et (x_1, x_2, x_3).
5. On suppose que l'on connaît les 6 premiers octets $z_1, z_2, \ldots, z_6$. Décrire (dans le document `pdf`) une attaque de complexité 2^{16} qui permet de récupérer l'initialisation du générateur, en exploitant le point 4 ci-dessus.
6. Programmer l'attaque contre ce générateur. Pour cela, initialiser le générateur avec une valeur aléatoire $s \in \{0,1\}^{40}$ et générer par la suite 6 octets $z_1, z_2, \ldots, z_6$. Vérifier que votre attaque permet de bien retrouver l'état initial.