Cryptographie – Feuille d'exercices 4

Attaques génériques

M1 Informatique – 2014-2015

1 Exercice 1 : Chiffrement double : l'attaque "par le milieu" ou "meet-in-the-middle"

Supposons que l'on essaye d'améliorer un système de chiffrement $(\mathcal{E}_K)_{K\in\mathcal{K}}$ dont les clés trop courtes n'interdisent pas la recherche exhaustive (on peut penser au DES) en chiffrant deux fois :

$$M \mapsto C = \mathcal{E}_{K_2}(\mathcal{E}_{K_1}(M)).$$

On double donc la longueur de la clé secrète. On considère l'attaque à clair connu suivante : Charlie connaît M et $C = \mathcal{E}_{K_2}(\mathcal{E}_{K_1}(M))$ et cherche K_1 et K_2. Il crée deux listes

$$\mathcal{L}_M = (\mathcal{E}_K(M))_{K\in\mathcal{K}} \qquad \text{et} \qquad \mathcal{L}_C = (\mathcal{E}_K^{-1}(C))_{K\in\mathcal{K}}.$$

Question 1 :
Expliciter une attaque pour retrouver la clé (K_1, K_2). On commencera par chercher un élément commun aux deux listes.

Question 2 :
Vérifier que le nombre de comparaisons nécessaire à la découverte d'un élément commun aux deux listes est de l'ordre de

$$(\#\mathcal{K}) \cdot \ln(\#\mathcal{K}).$$

Question 3 :
De combien de mémoire faut-il disposer pour pouvoir réaliser l'attaque ?

2 Exercice 2 : Compromis Temps Mémoire

Soit E une primitive de chiffrement symétrique parfaite, c'est-à-dire :

$$E_{K_1}(M) = E_{K_2}(M) \Rightarrow K_1 = K_2.$$

Dans la suite, on suppose que la taille de clé utilisée par cette primitive est n bits.
On va faire une cryptanalyse générique de cette primitive via une attaque à clair connu.

Question 1 :
Rappeler la définition de "attaque à clair connu" .

Question 2 :
Donner le coût algorithmique moyen d'une recherche exhaustive de la clé.

Dans la suite, on suppose que l'on est capable de faire des "précalculs", i.e., on peut effectuer des calculs avant de mener l'attaque. L'idée étant de précalculer l'ensemble des chiffrements du message clair M connu. L'attaque de recouvrement de la clé nécessitera simplement l'identification de la clé parmis l'ensemble des valeurs précalculées.

Question 3 :
Donner la meilleure structure de données permettant un stockage et une recherche efficace de la clé dans ces conditions.
Donner en fonction de n l'espace de stockage nécessaire pour effectuer le stockage de tous les chiffrés possibles. Pour une valeur de $n = 256$ (cas de l'AES), donner une valeur approchée de l'espace nécessaire au stockage.

Question 4 : D'une manière générale, on néglige les précalculs à condition que le temps et l'espace nécessaires à ceux-ci soient "raisonnables". Vous paraît-il raisonnable d'envisager l'attaque décrite à la question 3 ?

Afin d'attaquer la primitive E, on se propose de procéder comme décrit ci-dessous.
On choisit une longueur t et un nombre aléatoire noté K_1^1 de n bits.
On fixe $SP_1 = K_1^1$, puis on calcule :

$$\begin{array}{ccccccccc}
SP_1 = K_1^1 & \rightarrow & K_2^1 = E_{K_1^1}(M) & \rightarrow & K_3^1 = E_{K_2^1}(M) & \rightarrow & \cdots & \rightarrow & E_{K_{t-1}^1}(M) = EP_1 \\
SP_2 = EP_1 = K_1^2 & \rightarrow & K_2^2 = E_{K_1^2}(M) & \rightarrow & K_3^2 = E_{K_2^2}(M) & \rightarrow & \cdots & \rightarrow & E_{K_{t-1}^2}(M) = EP_2 \\
\vdots & & & & & & & & \vdots \\
SP_m = EP_{m-1} = K_1^m & \rightarrow & K_2^m = E_{K_1^m}(M) & \rightarrow & K_3^m = E_{K_2^m}(M) & \rightarrow & \cdots & \rightarrow & E_{K_{t-1}^m}(M) = EP_m
\end{array}$$

Et on stocke les couples (SP_i, EP_i).

Question 5 :

1. Expliquer comment retrouver la clé à la suite de ces précalculs ;
2. Cette attaque fonctionne-t-elle toujours ?

Question 6 :
La technique précédente n'est pas toujours réalisable. Expliquer pourquoi et proposer une modification permettant de la rendre pratique.

D'une manière générale, dans cette cryptanalyse, on utilise la dernière valeur de la chaîne précédente pour commencer la chaîne suivante. En supposant que l'on ne souhaite plus être en mesure de retrouver toutes les clés, on se propose de prendre une nouvelle valeur aléatoire pour chacune des valeurs de début de chaîne :

$$\begin{array}{ccccccccc}
SP_1 =_r r_0^1 & \rightarrow & r_1^1 & \rightarrow & r_2^1 & \rightarrow & \cdots & \rightarrow & r_t^1 = EP_1 \\
SP_2 =_r r_0^2 & \rightarrow & r_1^2 & \rightarrow & r_2^2 & \rightarrow & \cdots & \rightarrow & r_t^2 = EP_2 \\
SP_3 =_r r_0^3 & \rightarrow & r_1^3 & \rightarrow & r_2^3 & \rightarrow & \cdots & \rightarrow & r_t^3 = EP_3 \\
\vdots & & & & & & & & \vdots \\
SP_m =_r r_0^m & \rightarrow & r_1^m & \rightarrow & r_2^m & \rightarrow & \cdots & \rightarrow & r_t^m = EP_m
\end{array}$$

Cette attaque a été étudiée par Martin Hellman en 1980, dans l'article :

Martin E. Hellman. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory*, 26(4) :401–406, 1980. Available at `http://www-ee.stanford.edu/~hellman/publications/36.pdf`.

Il y montre notamment une borne inférieure sur la probabilité Pr de succès d'une telle attaque en fonction de la longueur des chaînes et de leur nombre :

$$Pr \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left[1 - \frac{it}{N}\right]^{j+1}.$$

Question 7 :
- Quelle est l'influence de la longueur des chaînes sur la quantité de calculs nécessaire à l'attaque ?
- Quelle est l'influence du nombre de chaînes sur la quantité de mémoire nécessaire à l'attaque ?

Question 8 :
Quel problème peut maintenant survenir lors de la constitution de la table ?
Proposer une solution à ce problème.

Dans la pratique, les tables contenant les valeurs stockées sont trop grandes pour être stockées en RAM. Ainsi la conduite d'une attaque requiert un nombre très important d'accés au disque dur. Sachant que le temps d'un accès disque est 1000 fois plus important qu'un accès en RAM, et qu'un accès registre est environ 100 fois plus rapide qu'un accès RAM, il est intéressant de limiter au maximum les accès au disque dur, i.e., de limiter le nombre de recherches dans la table. Pour atteindre cet objectif, Ronald Rivest a suggéré l'utilisation de *"points distingués"*.
Par *point distingué* on désigne des éléments de chaîne ayant une propriété particulière. Par exemple, on pourra s'intéresser aux chiffrés terminés par 10 bits à 0. L'idée est de ne plus stocker des chaînes de taille fixe, on s'arrête au moment où on tombe sur un point distingué.

$$\begin{array}{ll}
SP_1 =_r r_0^1 \to r_1^1 \to r_2^1 \to \cdots & \to r_{t_1}^1 = EP_1 \in \{0,1\}^{n-10}0000000000 \\
SP_2 =_r r_0^2 \to r_1^2 \to r_2^2 & \to r_{t_2}^2 = EP_2 \in \{0,1\}^{n-10}0000000000 \\
SP_3 =_r r_0^3 \to r_1^3 \to r_2^3 \to \cdots \to \cdots & \to r_t^3 = EP_3 \in \{0,1\}^{n-10}0000000000 \\
\vdots & \vdots \\
SP_m =_r r_0^m \to r_1^m \to r_2^m \to \cdots\cdots\cdots & \to r_{t_m}^m = EP_m \in \{0,1\}^{n-10}0000000000
\end{array}$$

Question 9 :
En supposant que l'on choisisse comme point distingué les chiffrés terminés par 10010010011, quelle est la longueur moyenne d'une chaîne ?

Il est clair que certaines chaînes peuvent être très longues dans ces conditions : il faudra un très grand nombre d'itérations avant d'arriver sur un point distingué. D'autres chaînes peuvent même boucler.

Question 10 :
Comment contourner le problème des chaînes qui bouclent ou qui ont une longueur trop importante ?

Question 11 :
Proposer une implémentation en pseudo-code de l'attaque par compromis temps mémoire générique et comparer-là avec une implémentation utilisant l'amélioration des points distingués en précisant où sont stockées les données.

Actuellement, la version la plus populaire des attaques par compromis temps-mémoire repose sur l'utilisation des « *tables arc-en-ciel* »proposées par Philippe Oechslin en 2003 dans l'article

> Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Dan Boneh*, editor. *Advances in Cryptology - CRYPTO 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003. Pages 617–630.

De nombreuses implémentations ce cette attaque sont disponibles sur le web. Elle est aujourd'hui largement utilisée pour ~~le crack~~ la récupération de mots de passes de certains systèmes d'exploitation. Elle a également permis le crack d'A5/1, le standard de chiffrement de la voix dans le protocole GSM.