TD1 - Premières Cryptanalyses

Responsable : M. Bombar

1 Compromis Temps-Mémoire

On considère dans cet exercice un chiffrement par blocs E_k opérant avec des clés de n bits. On note b la taille (en bits) des blocs. En d'autres terems, pour chaque $k \in \mathbb{F}_2^n$, E_k est une permutation de $\mathbb{F}_2^b$. On note D_k le déchiffrement correspondant.

Mise en situation Vous êtes responsables de la sécurité dans une entreprise. Afin d'améliorer la confiance que vos clients pourraient avoir, vous souhaitez augmenter la taille des clés. Deux situations s'offrent alors à vous :

1. Soit vous changez de chiffrement pour une primitive supportant des tailles de clés plus grosses, ce qui nécessitera du temps et de l'argent pour effectuer la transition.
2. Soit vous pensez être très malin et envisagez de chiffrer plusieurs fois les messages avec E et des clés différentes.

1.1 Cas du Double-Chiffrement

Afin de faire votre choix, vous étudiez la sécurité du double chiffrement. Dans la suite, on suppose que l'on dispose de r couples clair-chiffré (P_i, C_i) où pour $i \in \{1, \dots, r\}$,

$$C_i = \widetilde{E}_k(P_i) = E_{k_2}(E_{k_1}(P_i)).$$

(Q1) Décrire l'algorithme de recherche exhaustive sur $k = (k_1, k_2) \in \mathbb{F}_2^{2n}$. Quel est son coût, en temps et en mémoire ? On considèrera que l'évaluation du chiffrement est une opération élémentaire.

On se propose de faire mieux en utilisant le paradigme « Meet-in-the-Middle » (MiTM).

1. **Précalcul** Pour toutes les valeurs de $k_2 \in \mathbb{F}_2^n$, on calcule $X = D_{k_2}(C_1)$ et on stocke dans une table de hachage $X \mapsto k_2$.
2. **Recherche** On initialise une liste vide $\mathcal{L}$ de clés candidates.
   Pour chaque valeur de k_1, on calcule $Y = E_{k_1}(P_1)$. Si $Y \in T$, on ajoute toutes les paires (k_1, k_2) correspondantes à la liste $\mathcal{L}$.
3. **Filtrage** On utilise les couples (P_i, C_i) pour $i = 2, \dots, r$ afin de filtrer les faux-positifs et ne garder que la bonne clé.

En pratique, il faut faire attention au fait que même à C fixé, l'application $k \mapsto D_k(C)$ n'est pas injective : Plusieurs clés peuvent donner le même chiffré !

(Q2) En ignorant le filtrage des faux-positifs, quel est le coût de cette attaque (en temps et en mémoire) ? A-t-on amélioré la sécurité ?

On se propose maintenant d'analyser la phase de vérification, et de choisir le nombre r de paires clair-chiffré dont on a besoin pour déterminer la véritable clé utilisée avec une probabilité extrêment proche de 1. Pour cela, on va utiliser la modélisation suivante :

— La véritable clé $k^\star \stackrel{\text{def}}{=} (k_1^\star, k_2^\star)$ utilisée a été choisie uniformément dans $\mathbb{F}_2^{2n}$.

— Puisque l'on suppose que la primitive est a priori sécurisée, lorsque la clé est incorrecte, les valeurs de $E_{k_1}(P)$ et $D_{k_2}(C)$ devraient être impredictibles. On va donc modéliser E_{k_1} et D_{k_2} comme des fonctions aléatoires de $\mathbb{F}_2^b$ dans lui même, et indépendantes.

Remarque 1. *Notons que le premier modèle est totalement réaliste. En revanche, le second est a priori faux puisque si $k_1 = k_2$ par exemple, E_{k_1} et D_{k_2} sont inverses l'une de l'autre. En particulier, elles ne sont pas indépendantes. Cependant,*

$$\mathbb{P}(k_1^\star = k_2^\star) = 2^{-n}$$

qu'on va pouvoir considérer comme négligeable.

Soit $k = (k_1, k_2) \in \mathbb{F}_2^{2n}$ une clé candidate et (P, C) un couple clair chiffré tel que $E_{k_1}(P) = D_{k_2}(C)$. On s'intéresse à la probabilité p suivante

$$p \stackrel{\text{def}}{=} \mathbb{P}\Big[k^\star = k \mid E_{k_1}(P) = D_{k_2}(C)\Big].$$

Remarque 2. *Le fait suivant va souvent apparaître dans ce cours, donc c'est quelque chose qu'il faut bien comprendre. Lorsqu'un événement se produit avec une certaine probabilité p, alors le nombre de tests à effectuer avant d'observer effectivement cet évènement est de l'ordre de $1/p$.*

En effet, étant donnée une clé candidate k, on peut considérer la variable aléatoire de Bernoulli qui vaut 1 si $k = k^\star$, i.e., si k est effectivement la bonne clé, et 0 sinon. Autrement dit, c'est une variable aléatoire de Bernoulli du paramètre p ci-dessus. Par ailleurs, chaque clé candidate que l'on teste est indépendante de toutes les autres. Ainsi, le nombre de clés à tester peut-être modélisé par une loi géométrique de paramètre p, dont l'espérance est $1/p$.

(Q3) Montrer que

$$p = \mathbb{P}(k^\star = k)\frac{\mathbb{P}\Big[E_{k_1}(P) = D_{k_2}(C) \mid k^\star = k\Big]}{\mathbb{P}\Big[E_{k_1}(P) = D_{k_2}(C)\Big]}$$

(Q4) Si $k^\star = k$, que dire de l'événement $\Big[E_{k_1}(P) = D_{k_2}(C)\Big]$?

(Q5) On suppose que $k^\star \neq k$. D'après nos hypothèses, on modélise alors $E_{k_1}(P)$ et $D_{k_2}(C)$ comme des variables aléatoires indépendantes et uniformément distribuées dans $\mathbb{F}_2^b$. Que vaut la probabilité de l'événement $\Big[E_{k_1}(P) = D_{k_2}(C)\Big]$ dans ce cas ?

(Q6) En déduire $\mathbb{P}\Big[E_{k_1}(P) = D_{k_2}(C)\Big]$.

(Q7) Montrer que

$$p = \frac{2^b}{2^b + 2^{2n} - 1}.$$

On distingue alors 3 régimes :

— Lorsque $b \ll 2n$, $p \approx 2^{b-2n}$.

— Lorsque $b \approx 2n$, alors $p \approx 1/2$

— Lorsque $b \gg 2n$, alors $p \approx 1$.

Remarque 3. *Ainsi, lorsque $b \ll 2n$, le nombre de faux-positifs à filtrer va être de l'ordre de 2^{2n-b}, qui peut être énorme. Cependant, dans le cas où $b \gg 2n$, alors on peut typiquement considérer que la première clé trouvée va être la bonne !*

(Q8) Justifier que la phase de filtrage revient à considérer des chiffrements sur des blocs de taille $r \cdot b$ dans la probabilité précédente.

(Q9) Comment alors choisir r pour obtenir la bonne clé avec une bonne probabilité ?

1.2 Sécurité du Triple-Chiffrement

Puisque le double chiffrement n'offre pas une amélioration satisfaisante de la sécurité, on a alors proposé de considérer un triple-chiffrement. Ce fut notamment le cas pour l'algorithme `DES` (*Data Encryption Standard*).

DES Ce fut l'un des premiers standards internationnaux de chiffrement. Il a été proposé par la NSA en 1976 à partir d'un design initialement développé par IBM, et avait vocation à être utilisé par toutes les entreprises. Il s'agit d'un schéma de Feistel à 16 tours, opérant sur des blocs de 64 bits, avec des clés de 56 bits, dont la sécurité en pratique a rapidement chuté sous le niveau exigé [1].

Pour pallier ce problème, il a alors été proposé d'itérer les chiffrements `DES`. Puisque le double chiffrement n'offrait que 57 bits de sécurité (voir le début de cet exercice), c'est la variante `3DES` utilisant un triple chiffrement qui a été considérée. Elle utilise un triplet de clés $k = (k_1, k_2, k_3) \in \mathbb{F}_2^{168}$ et le chiffrement d'un bloc P opère de la façon suivante :

$$\widetilde{E}_k(P) = E_{k_3}(D_{k_2}(E_{k_1}(P))).$$

On a noté $\widetilde{E}$ le chiffrement de `3DES`, et E_k et D_k les chiffrements et déchiffrement du `DES` simple.

Remarque 4. *L'utilisation du déchiffrement D_{k_2} est uniquement pour une raison d'optimisation. D'un point de vue de la sécurité, ceci est équivalent à considérer la composition de trois évaluation du `DES` avec 3 clés différentes.*

(Q10) Montrer qu'il est possible de construire une attaque de type « Meet-in-the-Middle » pour `3DES` faisant 2^{112} évaluations du chiffrement ou déchiffrement `DES`.

Indication : On pourra construire deux tables de hachage dans la phase de précalcul : une table gauche $E_{k_1}(P) \mapsto k_1$, et une table droite $D_{k_3}(C) \mapsto k_3$.

(Q11) Quelle est la complexité en mémoire de cette attaque ?

2 Challenge de Cryptanalyse

Remarque 5. *Ce challenge est adapté du France Cybersecurity Challenge organisé par l'ANSSI.*

Lors d'un audit de sécurité, vous vous rendez compte que l'entreprise que vous évaluez a laissé à l'air libre la base de données des mots de passe de ses utilisateurs.

Affolé, vous vous empressez de le signaler aux ingénieurs responsables, mais ceux-ci ne semblent pas réagir. Ils estiment en effet que puisqu'ils ne stockent que les mots de passe

1. En Janvier 1999, il a été démontré que l'on pouvait récupérer en pratique une clé `DES` en 22 heures et 15 minutes avec du matériel de l'époque.

en version hashée, et qu'ils utilisent les meilleurs algorithmes de chiffrement comme l'`AES`, il n'y a aucun risque. Pour vous le prouver, ils vous ont chiffré un fichier `flag.txt` à l'aide d'un mot de passe (qui vous est inconnu), et vous ont exposé les mêmes informations sur ce mot de passe que ce qui se trouve dans le fichier que vous avez expertisé. Tout ce mécanisme se trouve dans `challenge.py` et la sortie est dans `output.txt`. Ils vous ont même précisé que `flag.txt` contenait une chaîne de caractères de la forme suivante :

`CSI{f580cdaf5ab54da35b600414508fc5eab8ad30ebc1256f1563067a8935876e5e}`

Attention, pour executer le fichier, il vous faudra installer la bibliothèque `PyCryptodome`. Celle-ci est disponible librement sur Github : https://github.com/Legrandin/pycryptodome/tree/master et activement maintenue. Vous trouverez la documentation de cette bibliothèque sur https://www.pycryptodome.org/.

Au CREMI Sur les machines du CREMI, `PyCryptodome` est normalement déjà installé. Vous pouvez le vérifier en ouvrant un shell `Python` ou `iPython` et en exécutant la commande suivante. Si elle fonctionne sans erreur, c'est que la bibliothèque est bien installée.

```
mbombar@hagrid $ ipython3
Python 3.11.2 (main, May  2 2024, 11:59:08) [GCC 12.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from Cryptodome.Cipher import AES
```

Installation en locale Si vous souhaitez l'installer sur votre machine personnelle, vous pouvez installer la version de votre distribution, ou bien l'installer avec le gestionnaire de paquet Python `pip`. Dans ce dernier cas, je vous incite très fortement à utiliser un environnement virtuel afin de bien séparer les dépendances liées à ce TP. Par exemple sur ma machine, en exécutant les commandes suivantes, je crée un dossier venv qui va contenir les dépendances de ce TP sans affecter le reste de mon système. Je vous renvoie à la documentation pour plus d'information https://docs.python.org/3/library/venv.html.

```
  python3 -m venv venv
  source venv/bin/activate
  pip install pycryptodome
```

Attention, si vous l'installez avec pip, cette bibliothèque s'appelle alors `Crypto` :

```
Python 3.12.6 (main, Sep  7 2024, 14:20:15) [GCC 14.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.20.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from Crypto.Cipher import AES
```

(Q12) Retrouvez le flag. Vous fournirez un fichier `NOM_solver.py` ainsi que `NOM_flag.txt` contenant uniquement le flag.

2.1 Bonus : Implémentation : Cryptanalyse de 3-SimplifiedDES

`SDES` est une version simplifiée de l'algorithme de chiffrement DES. Elle a conservé la même structure, mais opère avec des tailles nettement plus petites (et n'est donc absolument pas sécurisée). On va l'utiliser pour des raisons pédagogiques.

`SDES` opère sur des blocs de 8 bits, et utilise des clés de 10 bits. Sage implémente nativement ce chiffrement sous la forme d'une classe `SimplifiedDES`.

```
from sage.crypto.block_cipher.sdes import SimplifiedDES
```

Dans ce challenge, on considère le chiffrement `3SDES` obtenu de manière similaire à `3DES`, c'est à dire en composant un chiffrement, un déchiffrement et un nouveau chiffrement de `SDES`, avec 3 clés différentes.

— Le fichier `bonus_challenge.sage` contient une liste `PC` de couples clair-chiffrés obtenus via le chiffrement `3SDES` et 3 clés inconnues (K_1, K_2, K_3).

— Le fichier `sdes.sage` contient quelques exemples d'utilisation de `SDES` en Sage. Je vous invite à consulter la documentation pour plus d'informations.

Pour les charger, vous pouvez utiliser

```
sage: load("bonus_challenge.sage")
sage: load("sdes.sage")
```

(Q13) Retrouvez les trois clés (K_1, K_2, K_3) que j'ai utilisées.