UNIVERSITÉ LAVAL

Travail pratique N^o 1

Cours : GLO-3100 Cryptographie et sécurité informatique

Objectif

Le but de ce travail est de faire la cryptanalyse de chiffrement de Vigenère et de se familiariser avec les systèmes de chiffrement symétriques ainsi que leurs modes de fonctionnement.

Remarques

— Pour l'exercice 1, vous avez besoin du programme Vigenere (voir le fichier Vigenere.jar sur le site web du cours) qui vous aide à faire vos calculs.
— Pour l'exercice 2 et 3 vous avez besoin de la machine virtuelle Kali (une vidéo qui montre la procédure d'installation et de configuration de cette machine est disponible sur le site web du cours).

1 Exercice 1 : Briser Vigenère (2pts)

Le chiffrement d'un texte m écrit en anglais en utilisant Vigenère a donné le résultat suivant :

```
JEXCGQDUILIXXDIMXVNGXIBVPGLSMIUVVLZHPCQINXTOGFPWXMIFTXDVRWMCEKMIIAWZGBDWIYIJ
MSRVWMJJRFRPZVIZRAJVSZRUMCXEJIMQPKMIIGPCPYYTARMHOIMKMHOSJEMHOIACMADFAVXYSX
RRPFZHRZTBZVIVBNYIRICJOMDEMMOLTIIPZVHVMHJXWVVQJVSJQIQMCXJLJQIYIOIMCKIFGMVZFFZ
GXGLYMXTOXVVGZKSJGEXEXYSXPTMJCIGZWUKEXISZVPVFVCOLBJXBVXRIIUOIIYIYIGGPTNDSCRRX
OLTIIPZVHZRAYIRICJOMDEXBZHTKECGISFTYMEIZSHJJPTMJCIGZWWJRIISFGISSSNCFNKLYVPVFVCO
LBRRXDRTRGBDRHKEHXIQPEEZCIYMMDWPJIWMIIZHYVPAPOHJACFRFTXDKLYXSBDYHDGPEXMPWJ
RPFTEHYSLOWIIMHBSUTLUMERKILNAWZGBDWCVIXZHIFHYXVNGXNCIRZTBZVIVBNVGGPTNJWNJXY
HMHKLYJVSVVYYPXJXIAIAVQYIXHFJZDRXKIJJWHZFFZTARMHOIMKWZDRXKIJJWHZFFZGNGLYMXT
OXMAMCZXYKSHJMVGIZVCMVRSKLYZRRICJOMDEEHYHTTVSKXXFRUGKDIMNCQHNLCXLRFVLZWEF
RXOSTRGBFINBISNEGVMGKSGKEHOEHTMJCIGJACOLDLXPVVXRFFZOTPWWVRQVXLDZXRPFTFGFOYI
AXKLIIPNKLYFRDNPYYKTFJNCIRZTBZVJJIXVRSRVYOLTIIZJVTLWYGIHJSLZZTEGIPRIVVJMSSLGNDZT
WSLHSHKTOMTDJIMCMHKSLDGPCPSXMEYILNATIIIAXTEYMZHSZVYXXAPJIMICTVSKXXFRIMHTTVSK
XXFRQDXWFYNVHSZXCJRPCTLJGTUYLZWHLGBVWPLXBZRIZGUOMDESLDRIVKLDXNTLYXOH
```

Pour les questions suivantes, on vous demande de **bien justifier vos réponses en donnant tout le calcul intermédiaire qui vous amène à la conclusion**. À noter que des réponses sans justifications ne vous donnent pas des points.

1. **(0.5pt)** Utiliser le test de Friedman pour estimer la taille de la clé sachant qu'elle est inférieure ou égale à 9.
2. **(1.5pts)** Utiliser l'indice de coïncidence mutuel pour trouver la clé et décrypter le message. L'indice de coïncidence mutuel devrait être utilisé pour trouver le décalage des colonnes par rapport à la colonne 0. Par la suite, en se basant sur la lettre la plus fréquente dans la colonne 0 et celle dans la langue d'origine, on calcule la valeur K_0. À partir de K_0 et des décalages, on calcule les autres valeurs de la clé et on décrypte le message.

2 Exercice 2 : Modes de chiffrement (4pts)

Dans cet exercice, on vous demande d'implanter les modes de chiffrement ECB, CBC, CFB, OFB et CTR avec le système cryptographique symétrique par bloc. C'est un simple chiffrement par *xor* travaillant sur un bloc de 7 bits comme suit :

$$E_{(k_0.k_1.k_2.k_3.k_4.k_5.k_6)}(b_0.b_1.b_2.b_3.b_4.b_5) = (b_0 \oplus k_0).(b_1 \oplus k_1).(b_2 \oplus k_2).(b_3 \oplus k_3).(b_4 \oplus k_4).(b_5 \oplus k_5).(b_6 \oplus k_6)$$

Votre programme doit s'appeler *ex2* et offrir les options suivantes (dans n'importe quel ordre) :
-msg suivie d'un message m qui est une suite binaire ayant une taille multiple de 7 et contenant 70 bits au maximum,
-key key suivie d'une suite binaire de taille 7.
-op suivie de *enc* ou *dec* pour dire si l'on veut chiffrer ou déchiffrer le message.

-mode suivie d'un mode de chiffrement qui est un texte dans ECB, CBC, CFB, OFB, CTR.
-iv (optionnel) suivie d'une suite binaire aléatoire de taille 7. Si le *iv* n'a pas été donné par l'utilisateur, alors le programme le choisi aléatoirement.
-r (seulement pour les modes CFB et OFB) suivie d'un nombre r avec $1 \leq r \leq 7$.

Ensuite, il retourne le résultat de l'opération (chiffrement, déchiffrement), du message m par E_k selon le mode choisi et en utilisant iv comme vecteur d'initialisation.

Exemples de possibilté d'appels de votre programme :

```
ex2 -msg 11101110111111010111 -key 1001111 -op enc -mode CBC -iv 0011111
ex2 -msg 11101110111111010111 -key 1001111 -op dec -mode CBC
ex2 -msg 11101110111111010111 -key 100111 -op enc -mode CFB -iv 1011111 -r 3
ex2 -key 1001111 -mode CBC -msg 11001110111111010111 -op enc
```

Il est important de bien commenter le code et de s'assurer qu'on peut facilement le compiler et l'exécuter sur Kali. La note sera répartie comme suit : (ECB : 0.5 pt, CBC : 0.75 CFB : 0.75, OFB : 0.75, CTR : 0.75, commentaires : 0.5pt)

3 Exercice 3 : OpenSSL (2pts)

Pour cet exercice, nous vous recommandons d'utiliser la machine virtuelle Kali. Elle contient déjà OpenSSL et d'autres outils intéressants. Le site site officiel de OpenSSL est : www.openssl.org/. Plus de documentation sur OpenSSL est disponible ici.

Voici quelques commandes de OpenSSL :

— Générer une clé privée RSA de "*size*" bits (512, 1024, etc.)
$openssl genrsa -out <fichierRsa.priv> <size>
— Création d'un clé publique associée à la clé privé "*fichierrRsa.priv*"
$openssl rsa -in <fichierrRsa.priv> -pubout -out <fichierRsa.pub>
— Chiffrer une clé privée avec l'algorithme DES3 ou autre.
$openssl rsa -in <fichierRsa.priv> -des3 -out <fichierOut.>
— Chiffrer le "*fichier.txt*" avec l'algorithme "*algo*" en utilisant la clé qui se trouve dans la première ligne du fichier "*key*".
$openssl enc <-algo> -in <fichier.txt> -out <fichier.enc> -kfile <key>
— Déchiffrer le "*fichier.enc*" avec l'algorithme "*algo*".
$openssl enc <-algo> -in <fichier.enc> -d -out <fichier.txt> -kfile <key>
— Hacher un fichier avec "*algo*" (sha1, md5, rmd160, etc.)
$openssl dgst <-algo> <entree> -out <sortie>
— Générer un nombre aléatoire sur "*nbits*" et mettre le résultat dans "*file.key*" (utiliser l'option "*base64*" pour la lisibilité) $openssl rand -out <file.key> <nbits>

Questions : Écrire votre nom et votre prénom dans un fichier nommé plaintext.txt et utiliser OpenSSL pour faire les opérations suivantes :

1. **(0.25 pts)** Chiffrer plaintext.txt avec AES256 et le mode CBC en utilisant un mot de passe de votre choix et mettre le résultat en format base64 dans le fichier ciphertext.enc. Prendre une capture d'écran montrant, le contenu de fichier plaintext.txt, la commande que vous avez tapée pour le chiffrer ainsi que le contenu du fichier ciphertext.enc
2. **(0.25 pts)** Comparer la taille des deux fichiers (pliantext.txt et ciphertext.enc) et expliquer la différence.
3. **(0.25 pts)** Utiliser triple DES_3EDE (Encryption-Decryption-Encryption avec trois clés différentes) avec le mode *ofb* pour chiffrer, puis déchiffrer le fichier plaintext.txt en utilisant un mot de passe de votre choix À noter que OpenSSL utilise une fonction KGF (Key Generation Function) pour générer des clés à partir d'un mot de passe et d'un *salt* (une valeur générée aléatoirement). Prendre une capture d'écran montrant les commandes utilisées et les résultats obtenus. Vos copies d'écran doivent montrer aussi le contenu du fichier en clair, son correspondant chiffré et le résultat de son déchiffrement.
4. **(0.25 pts)** Mettre le fichier plaintext.txt dans un répertoire portant votre nom, le compresser avec la commande "tar" et le chiffrer avec DESX en mode *cbc* et en utilisant un mot de passe de votre choix. Demander aussi à OpenSSL de vous afficher la clé, le *salt* et le *IV* qu'il a utilisés. Prendre une capture d'écran montrant les commandes utilisées et les résultats obtenus.
5. **(0.25 pts)** Pour chiffrer un fichier avec une clé explicite, il faut utiliser les options *-K* (clé en hexadécimal) et *-iv* (vecteur d'initialisation en hexadécimal). Générer une clé de 128 bits et un *iv* de 128 bits et utiliser les pour chiffrer, puis déchiffrer le fichier plaintext.txt avec camellia128 en utilisant le mode *cbc*. Prendre une capture d'écran montrant les commandes utilisées et les résultats obtenus. Vos copies d'écran doivent montrer aussi le contenu du fichier en clair, son correspondant chiffré et le résultat de son déchiffrement.
6. **(0.5 pts)** Chiffrer le contenu du fichier plaintext.txt en mode binaire avec blowfich (bf) en mode *cbc* et le déchiffrer avec l'option *-nopad* (permet de préserver le bourrage lors de déchiffrement) puis visualiser le contenu du fichier déchiffré avec la commande *xxd* (visualiser le contenu en hexadécimal). Répétez l'expérience 3 fois, mais à chaque fois vous enlevez un caractère du contenu de plaintext.txt et observez l'effet sur le bourrage introduit par OpenSSl. Donner vos constatations accompagnées d'une ou plusieurs captures d'écran montrant les commandes utilisées et les résultats obtenus.
7. **(0.25 pts)** Avec la commande `openssl speed`, comparer la rapidité de AES par rapport à DES et DES avec RSA. Prendre des captures d'écran montrant les commandes et les résultats obtenus tout en commentant les résultats.

4 Remarques

1. Le travail est individuel.
2. Les langages C, C++ ou Java sont permis.
3. Le barème est donné à titre indicatif.
4. Attention au plagiat ! Faites vos TPs par vous-même.

5 À remettre

Utiliser le site web du cours pour remettre un seul fichier ".zip" (de taille maximale 40 Mb) qui porte votre nom au complet et qui contient un répertoire par exercice (ne m'envoyez pas vos TPs par courriels s.v.p.). Quand il s'agit d'un exercice de programmation, le répertoire en question doit contenir l'exécutable aussi bien que le code source **bien commenté**. Assurez-vous aussi que votre exécutable n'aura besoin d'aucun autre fichier externe pour pouvoir fonctionner sur Kali. Retourner un fichier ".pdf" ou ".doc" pour l'exercice 1 et un fichier ".pdf" ou ".doc" pour l'exercice 3. Les réponses doivent garder les mêmes numéros que les questions et les captures d'écrans doivent être bien lisibles.

6 Échéancier

Le 14 octobre 2020 avant 14h00. Une pénalité de 0,0028% de la note sera appliquée à chaque minute de retard (l'équivalent de 0.166% par heure), et ce, pour un maximum de 48 heures. Après 48 heures de retard, la note sera zéro. Par exemple, pour un étudiant qui a eu 8 points, mais avec 5 heures de retard, sa note sera 8 - 8*0.166=6,68.