UNIVERSITÉ LAVAL

Travail pratique N^o 2

Cours : GLO-3100 Cryptographie et sécurité informatique

© M. Mejri, 2021

Objectif

Le but de ce travail est de se familiariser avec des librairies cryptographiques fournies avec des langages de programmation comme C++ ou Java. En particulier, nous vous demandons d'implanter un outil de piratage (Ransomware) et une trousse sécurisée de mots de passe. Dans ce TP, nous ne vous demandons pas d'implanter des systèmes cryptographiques comme AES-CBC-128 ou AES-CTR-128, mais plutôt d'utiliser la librairie cryptographique offerte par votre langage de programmation. Un petit exemple en C++ et un autre en Java montrant comment manipuler ces librairies sont disponibles sur le site web du cours. Nous utilisons la machine virtuelle Kali du premier TP.

1 Exercice 1 : Piratage informatique : *Ransomware* (5.5pts)

Nous vous demandons d'écrire un programme nommé *ex1* qui prend les paramètres suivants :

- un répertoire (spécifié via l'option *-d* ; par défaut, c'est le répertoire courant) ;
- une liste de types de fichiers dont chacun est spécifié avec l'option *-f*. Pour cet exercice, nous nous limitons aux types *xls*, *doc*, *pdf*, *png*, *mp3*, *avi* et *txt*) ; et
- une opération (spécifiée via l'option `-op`) indiquant deux valeurs possibles : `enc` (pour un chiffrement) et `dec` (pour un déchiffrement).

Quand la valeur indiquée par `op` est `enc`, le programme chiffre tous les fichiers ayant les types indiqués par l'option `-f` du répertoire spécifié par `-d` avec le système cryptographique AES-CTR-128. La même opération se fait aussi récursivement pour tous les sous-répertoires du répertoire indiqué.

Une fois le travail terminé, le message suivant est affiché à l'utilisateur : "cet ordinateur est piraté, plusieurs fichiers ont été chiffrés, une rançon de 5000$ doit être payée sur le compte PayPal hacker@gmail.com pour pouvoir récupérer vos données".

Par ailleurs, toutes les informations nécessaires au chiffrement (clé, iv, etc.) sont générées aléatoirement par le programme. Pour la plupart des *ransomware*, toutes ces informations sont envoyées au pirate. Mais, pour simplifier l'exercice, nous nous contentons de les enregistrer dans un fichier nommé *pirate.txt* et de le mettre dans le répertoire fourni avec l'option -d. Par ailleurs, les types de fichiers chiffrés sont également stockés de préférence dans le fichier *pirate.json*. En quelque sorte, tous les paramètres nécessaires à remettre le répertoire dans son état initial seront enregistrés dans le fichier *pirate.json*. Sur le site web de cours, nous vous donnons un exemple de code (en C++ et en Java) de manipulation des fichiers de type "json". Ce code vous serez également utile aussi pour l'exercice 2.

Exemples d'appels de votre programme.

```
ex1 -f doc -f pdf -f png -op enc
ex1 -d c: -f doc -f pdf -f png -op enc
```

Une fois un répertoire est chiffré, il est possible de le déchiffrer en faisant appel à *ex1* en donnant son nom (avec l'option *-d*), avec l'option *-op dec*. Si le répertoire ne contient pas le fichier *pirate.json*, un message d'erreur sera affiché.

Exemple :

```
ex1 -d c: -op dec
```

2 Exercice 2 : Gestionnaire de mots de passe (6.5pts)

Nous vous demandons d'écrire un programme qui vous permet de gérer vos mots de passe liés à plusieurs comptes (Gmail, Facebook, etc.) d'une manière sécuritaire. Votre trousse de mots de passe est protégée par un secret unique (un autre mot de passe qui ne devrait pas être enregistré sur votre ordinateur.). Chaque fois que vous voulez accéder à un de vos comptes, vous déverrouillez les informations requises dans le gestionnaire en fournissant le secret.

Voici plus de détails techniques sur les fonctionnalités de ce gestionnaire.

- Nous voulons avoir un fichier sur notre disque dur contenant des enregistrements ayant les informations suivantes :
  - *URL* : l'adresse URL du service.
  - *user* : le nom d'utilisateur lié au compte.
  - *pwd* : le mot de passe lié au compte associé à l'URL.
- Nous vous recommandons d'utiliser le format *Json* et d'écrire les enregistrements en question dans un fichier "trousse.json", mais vous êtes libres de faire d'autres choix.
- Une fois sur le disque dur, les informations liées à ces champs devraient être chiffrées séparément avec AES-CBC-128 bits. La clé de protection est générée à partir d'un secret (un mot de passe) donné par l'utilisateur. Il s'agit du seul secret que l'utilisateur a besoin de s'en souvenir. Chaque champ aura son propre iv (nécessaire pour le mode CBC) généré aléatoirement et enregistré à côté du champ en question pour faciliter son déchiffrement plus tard.

Nous supposons que le programme s'appelle `ex2` et que nous pouvons le manipuler en mode ligne de commandes avec les options suivantes :

- l'option -a permet d'ajouter un nouvel enregistrement à la trousse de clé. Elle doit être suivie par le mot de passe de la trousse. Dans ce cas, l'utilisateur doit spécifier les champs de l'enregistrement avec les options suivantes :
  `-url` : donne l'URL.
  `-user` : donne le nom d'utilisateur (username).
  `-pwd` : donne le mot de passe permettant d'accéder au service.

  Exemple :
  ```
  ex2 -a 123pwd -url www.facebook.com -user alice@gmail.com -pwd 12t3r
  ```
- l'option -l suivie du mot de passe de la trousse permet des lister les enregistrements. L'affichage commence par une ligne indiquant le contenu de chaque colonne. Chaque enregistrement est précédé par un numéro de ligne. Tout doit apparaitre en clair sauf les noms d'utilisateurs et leurs mots de passe qui restent chiffrés et nous affichons "*****" à leurs places.

  Exemple de commande :
  ```
  ex2 -l 123pwd
  ```

  Exemple de résultats.

  ```
  ligne             url                          user             pwd
  1                  www.facebook.co             *****             *****
  2                  www.gmail.com               *****             *****
  ```

- l'option `-d` suivie du mot de passe de la trousse permet d'afficher une ligne particulière tout en déchiffrant certains champs de plus. Le numéro de la ligne est spécifié avec l'option `-i`. Si nous voulons déchiffrer le nom, nous le spécifions avec le champ `-user` et si nous voulons déchiffrer le mot de passe, nous le spécifions avec l'option `-pwd`. Autrement, ces informations restent chiffrées.

  L'exemple suivant permet d'afficher le contenu de toute la ligne numéro 1 en clair :
  ```
  ex2 -d 123pwd -i 1 -user -pwd
  ```

  Exemple de résultats.

  ```
  ligne            url                user             pwd
    1              www.facebook.com   alice@gmail.com  12t3r
  ```

  Un autre exemple `ex2 -d 123pwd -i 2 -pwd`

  ```
  ligne        url                 user             pwd
  1           www.facebook.com      *****           5ewr1
  ```

  À noter que votre programme ne devrait enregistrer nulle part le mot de passe qui protège la trousse de clés. Il n'a pas à savoir aussi si l'utilisateur a fourni le bon mot de passe ou non. Tout ce qu'il fait, c'est de déchiffrer l'information demandée avec le mot de passe fourni. Si l'utilisateur fournit un mauvais mot de passe, le message affiché n'aurait pas de sens.

3 OpenSSL (3 points)

Dans cet exercice, nous allons découvrir d'autres utilisations de OpenSSL. Voici quelques commandes de `openssl`, mais vous devrez les compléter par vous-même pour répondre aux questions demandées.

— 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 privée "`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.pub>
```
— 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`"
```
$openssl rand -out <file.key> <nbits>
```

On vous demande de faire les opérations suivantes et de prendre des captures d'écran montrant les commandes utilisées et les résultats obtenus :

1. (0.25pts) Générer une clé privée RSA pour Alice tout en la chiffrant avec triple DES (`des3`) en utilisant un mot de passe comme clé symétrique. La clé RSA générée devrait être mise dans le fichier `alice-privatekey.pem`.
2. (0.25pts) Extraire la clé publique de Alice et la mettre dans le fichier `alice-publickey.pem`
3. Faire la même chose pour Bob pour lui générer une paire de clés et les mettre dans `bob-privatekey.pem` et `bob-publickey.pem`.
4. Créer un fichier `message.txt` dans lequel vous écrivez votre nom et votre prénom.
5. (0.5pts) Montrer, via une capture d'écran, les opérations `openssl` qu'Alice fait pour envoyer `message.txt` haché avec SHA1 et signé avec sa clé privée.
6. (0.5pts) Montrer, via une capture d'écran, les opérations *openssl* que Bob fait pour vérifier la signature.
7. Pour qu'Alice envoie `message.txt` d'une manière confidentielle à Bob, il exécute les étapes suivantes :
   a) (0.25pts) Alice génère une clé aléatoire et mets le résultat binaire dans le fichier `key.bin`.
   b) (0.25pts) Alice chiffre `message.txt` avec `key.bin` en utilisant AES-128-CBC et met le résultat dans `protected-message.txt` codé en base64.
   c) (0.5pts) Alice chiffre, en utilisant `rsautl`, la clé `key.bin` avec la clé publique de bob et met le résultat dans `protected-key.bin`
8. (0.5pts) Aider Bob à retrouver le contenu de `message.txt` à partir de `protected-key.bin` et `protected-message.txt`. Montrer les étapes de calculs, via des captures d'écran `openssl`, ainsi que le message.txt obtenu une fois le calcul terminé.

4 Remarques

1. Le travail est individuel.
2. Seuls les langages C, C++ et Java sont permis.
3. Le barème est à titre indicatif et 10% des points seront attribués aux commentaires.
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.). Le répertoire en question doit contenir l'exécutable (.jar, etc.) aussi bien que le code source **bien commenté**. Assurez-vous que votre exécutable n'aura besoin d'aucun autre fichier externe pour pouvoir fonctionner sur Kali.

6 Échéancier

Le 5 décembre 2021 avant 14h00. Une pénalité de 0,0347% de la note sera appliquée à chaque minute de retard (l'équivalent de 2,08% points 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 10 points, mais avec 6 heures de retard, sa note sera 10 - 10*0,0208*6=8,75.