**Université de Caen** **Année universitaire 2009-2010**
**U.F.R Sciences** **Master 2 - DNR2I**

**Sécurité**

Travaux pratiques n°1

---

# 1 Méthode de César

**Exercice 1.** Aller à l'adresse

`http://info.unicaen.fr/~jfroment/`

rubrique enseignement et sécurité et télécharger les fichiers

`code.cpp, decode.cpp, stat.cpp et article.txt`

Compiler les fichier `*.cpp` en utilisant la commande:

```
g++ fichier.cpp -o fichier
```

Vous devez alors avoirs trois fichiers exécutables: `code, decode, stat`. Editer les sources et expliquer quel est leur fonctionnement.

**Exercice 2.** Chiffre le fichier `article.txt` (ou un autre de votre choix) à l'aide d'une clé que vous gardez secrète. Envoyer le chiffre à votre voisin. Lorsque vous recevrez le fichier de votre voisin, essayer de le déchiffrer.

# 2 Cryptographie RSA avec openSSL

Cette partie est tirée de la page

`http://www2.lifl.fr/~wegrzyno/portail/PAC/Doc/TP5/tp-certif003.html`

## 2.1 Génération de clés

On peut générer une paire de clés RSA avec la commande `genrsa` de `openSSL`:

```
openssl genrsa -out <fichier> <taille>
```

où fichier est un nom de fichier de sauvegarde de la clé, et *taille* est la taille souhaitée (en bits). La commande `rsa` permet de visualiser un fichier PEM contenant une paire de clés RSA:

```
openssl rsa -in <fichier> -text -noout
```

**Exercice 3.** Générer une paire de clés de $1024$ bits, stockée dans le fichier `maCle.pem`. Visualiser le contenu du fichier produis.

Il n'est pas prudent de laisser une paire de clé en clair (surtout la partie privée). Avec la commande `rsa`, il est possible de chiffrer une paire de clés . Pour cela trois options sont possibles qui précisent l'algorithme de chiffrement symétrique à utiliser : `-des`, `-des3` et `-idea`:

```
openssl rsa -in <fichier> -des3 -out <fichier>
```

**Exercice 4.** Chiffrer votre paires de clés stockée dans `maCle.pem` à l'aide de *triple des*. Utiliser de nouveau la commande `rsa` pour visualiser le contenu du fichier.

La partie publique d'une paire de clés RSA est publique, et à ce titre peut être communiquée à n'importe qui. Le fichier `.pem` contient la partie privée de la clé, et ne peut donc pas être communiqué tel quel (même s'il est chiffré). Avec l'option `-pubout` on peut exporter la partie publique d'une clé:

```
openssl rsa -in <fichier> -pubout -out <fichier>
```

**Exercice 5.** Créer un fichier `maClePub.pem` contenant la clé publique de la paire de clés du fichier `maCle.pem`. Ouvrer ce fichier avec un éditeur de texte. Envoyer votre clé publique à votre voisin que l'enregistrera dans le fichier `clePubVoisin.pem`.

### 2.2 Chiffrement déchiffrement

On peut chiffrer des données avec une clé RSA. Pour cela on utilise la commande `rsautl`:

```
openssl rsautl -encrypt -in <clair> -inkey <clePub> -pubin -out <chiffre>
```

On peut déchiffrer des données avec une clé RSA avec :

```
openssl rsautl -decrypt -in <clair> -inkey <cle> -out <chiffre>
```

**Exercice 6.** Créer un fichier texte de moins d'une dizaine de caractère. Crypter le avec la clé publique de votre voisin, puis transmetter lui le fichier chiffré. Quand vous recevrez le chiffré de votre voisin, déchiffrer le à l'aide de votre clé secrète.

## 3 Signatures

**Exercice 7.** Télécharger la dernière version de Gimp sur

`http://www.gimp.org/downloads/`

Vérifier que le fichier a été copier sans problème à l'aide de la commande `md5sum`.

**Exercice 8.** `Enigmail` est un plugin de signature numérique et de chiffrement se greffant sur des clients de courrier électronique tels que `Thunderbird` Il utilise le logiciel `GnuPG` qui lui fournit les fonctions cryptographiques utilisées lors du chiffrement ou de la signature. Le logiciel `GnuPG` fait lui-même partie de la suite d'algorithmes de chiffrement asymétrique PGP.

Afin d'utiliser `Enigmail` sur `Thunderbird`, suiver le guide se trouvant à l'adresse

http://enigmail.mozdev.org/documentation/quickstart.php

## 4 Authentification SSH par clé publique

Lors de l'établissement d'une connexion SSH, le processus suivant s'engage :

## 4.1 Authentification au serveur

1. Le client contacte le serveur.
2. Ils s'échangent ensuite des informations relatives à la version de SSH utilisée ; la communication continue si les deux versions sont compatibles.
3. Le client et le serveur basculent vers un protocole par paquets.
4. Le serveur s'identifie auprès du client en lui faisant connaitre :
   - Sa clé d'hôte, une clé publique, identifiant le serveur.
   - Sa clé de "serveur", une clé publique supplémentaire.
5. Le client procède alors à la recherche de la clé d'hôte dans sa base de données des hotes connus. Si le serveur distant y figure et que la clé fournie est la même que celle stockée, le serveur est authentifié, et le processus continue. Sinon, il est possible que la clé d'hôte du serveur ait été modifiée, ou que le client soit victime d'une tentative d'attaque "Man in the middle".
6. Le client génère une clé privée (**bulk key**) pour le chiffrement de la session, qui sera connue du client comme du serveur (méthode de chiffrement symétrique). Pour la transmettre au serveur, il la chiffre grâce à la clé publique du serveur.
7. Le serveur est à même de déchiffrer car il dispose de sa propre clé privée d'hôte, et c'est le seul à pouvoir le faire. Le serveur est ainsi authentifié de manière implicite. Le serveur envoie un message de confirmation, chiffré avec la **bulk key**.
8. Le client s'aperçoit que le serveur est bien celui qu'il prétend être, puisqu'il est parvenu à récupérer la **bulk key** et a envoyé un message de confirmation. La communication est maintenant sécurisée, le client et le serveur connaissent la **bulk key**.

Il s'agit ensuite d'authentifier le client.

## 4.2 Authentification du client

L'authentification par mot de passe est, à ce point, la méthode la plus basique. Elle a certains inconvénients, comme celui de devoir retaper son mot de passe 'a chaque connexion SSH, sans compter que le mot de passe sort (même si crypté) de votre machine. Nous allons développer ici la méthode d'authentification par clé publique.

1. Commencez par générer votre couple (clé publique / clé privée). Le programme `ssh-keygen` devrait vous y aider. Ces clés seront vraisemblablement stockées dans un fichier du répertoire `/.ssh/`
2. Identifiez le fichier comportant votre clé publique.
3. Vous devez alors porter à la connaissance de tous les hôtes distants que vous voulez joindre avec `ssh` votre clé publique.
4. Copiez avec votre fichier clé publique vers un serveur de votre choix, disons `mike`. Vous enregistrez temporairement ce fichier sous le nom suivant : `cle.tmp` dans le `répertoire .ssh`.
5. Si le fichier de clés `authorized_keys` n'existe pas, créez-le et stockez-y le contenu de votre clé publique (dans `cle.tmp`). Sinon ajoutez à `authorized_keys` la clé manquante en fin de fichier.

## 4.3 Utilisation d'un agent

L'utilisation d'un agent permet de ne pas taper à chaque connexion la phrase de passe usuelle. Un agent est un programme qui garde les clés privées en mémoire et qui fournit les services d'authentification de SSH, si bien qu'une fois lancé, ils ne vous demande plus vos phrases d'authentification.

- Pour en lancer un, taper `ssh-agent $SHELL` où `$SHELL` peut être remplacé par le `shell` que vous souhaitez utiliser.
- Vous pouvez ensuite ajouter votre (ou vos) clés privées grâce au programme `ssh-add`. Une fois que vous avez entré votre phrase d'authentification (pour la dernière fois), `ssh-add` la déchiffre et la stocke dans l'agent. Il ne faut pas laisser votre poste sans surveillance, car quiconque aurait accès à ces clés pourrait se connecter à n'importe quel site distant accessible avec ces clés, voire même pourrait les dérober. Utiliser les options `ssh-add -d` et `ssh-add -D` pour retirer une ou toutes les clés stockée(s) dans l'agent.
- Tentez de vous connecter via `ssh` sur un compte distant et constatez la différence.