Compte rendu du TP de cryptographie

par HARTMANN Matthias et NOILOU Quentin

Création de clé de chiffrement sur Gn

Génération d'une clé

Commande utilisée:

```
openssl genrsa -out nlu-hrt_key 2048
```

Résultat obtenu:

```
Generating RSA private key, 2048 bit long modules (2 primes)
```

Visualisation de la clé

Extrait de la clé obtenue (le fichier):

```
-----BEGIN RSA PRIVATE KEY-----
118nbV/TgqsgdphPQxJT5nZXzISU8+L5vSME+2xWCIHRWuFE9ww3u7IqjVqueOYc
6wgDXVlJ2cdHeLhlt15Qt1VamahCHgUdu+0oKajs6f3NgUIZDiwTONdP2BwUZX1y
31RbRAXISvpNB8/Aljf28+iK+iQoY9Y/c9REfR1U6KbKJPcUQ3q+sT5MwHOintJJ
vHf76rEEiwiqIJz/sYHr6FESyz3y9JzaPAbyMMYHcAAJABYu0tLStM8ryRMxQzdl
Ygm15bzuQDk0FXIIBmn9E5Ny5nJzhyyY7GaZTqFirjX3FcB6s+JMeL44QMnIj8Ke
i+RwWpqvmGEcAoDedsd5xD13N5QgjHzx4qOkoiF9YDT/RwWqFKOHPgsVogqu3Chv
hi+SdNGWZfnvc8WzvD9TpBGzsznCKUCVuhx3BNhIe2g2Y4TvRo0Zgeg9DGgMReMK
/pjovNtJE9mcGu4rgRKxN8pAqk/FSYRDKezff9BHrsbW+7RlQWqvQVaaWv2DkHJL
mhGZr0wP1bBdeO2q+Ayqyo5mRvZN4NgUgo+BF9dObKaIYBBUAItYyw2CYmniTDg1
9BTkcdsk7fO7DhxBCyXUL8kwLFn08pLjduj51gJnb7VwGDlXu0dQY4iXMrhm61dL
et/o9h4OeaurVIVADwtVB0sUcDUBPYliSK5TYCE7K6ZFXNLiQBLXYLB5xvWy31yu
4buwr7NYCa0sJFPQwVatdz03M5Qvu67s8AUOF7tsIgPGF3rj3RqlM5p9NjQwyxXw
BBjBJ3FUWk0CoUk9bicSptQj1JkiD8iZs1gHQLtdHKg=
-----END RSA PRIVATE KEY-----
```

Nous remarquons que le contenu est codé en base64, il est donc inintelligible sans conversion.

Après utilisation de la commande

```
openssl rsa -in nlu-hrt_key -text -noout
```

On obtient l'extrait suivant:

```
RSA Private-Key: (2048 bit, 2 primes)
modulus:
  00:cb:d7:c3.....
  ..................
  ..................
```

On remarque que la clé est décomposée en plusieurs parties :

- Modulus
- privateExponent
- publicExponent
- prime number 1
- prime number 2
- exponent 1
- exponent 2
- coefficient

Cela correspond aux variables utilisé lors du calcul mathématique des clés RSA.

Cette clé est très critique car elle contient l'ensemble des informations permettant de chiffrer et déchiffrer les messages selon la norme RSA. Cette clé ne doit absolument pas être partagée. Cette clé n'est pas signée. Son contenu est codé en base64 et la conversion permet d'obtenir les grands chiffres (codé en hexadécimal) utilisés dans le chiffrement et le déchiffrement.

Protection de votre clé

Commande utilisée :

```
openssl rsa -in nlu-hrt_key -des3 -out cle1.pem
```

- Il faut absolument supprimer le fichier `nlu-hrt_key` car il représente la clé non chiffrée donc très sensible.
- Lorsqu'on fait à nouveau la manipulation et qu'on regarde les différences entre les fichiers (`diff a b`) on remarque que les fichiers sont complètement différent.
- Les fichiers sont des fichiers textes, la notion de protection est dans le fait que la valeur binaire a été passée par un algorithme de chiffrement (DES3) puis recodé en base64 dans le fichier. Pour déchiffrer la clé, il faut absolument connaître la `passphrase` utilisée.
- Cette fois-ci la commande demande la `passphrase` utilisée par le fichier `cle1.pem` avant de nous afficher le contenu de la clé.
- La clé est la même alors que les fichiers sont différents. On remarque cela car cle1.pem (déchiffré) est égale à cle2.pem (déchiffré)
  On comprend donc que l'algorithme de chiffrement prend en compte autre chose que simplement la `passphrase`.

Génération de la clé publique

Commande utilisée :

```
openssl rsa -in cle1.pem -pubout -out ma-cle-publique
```

Cette clé n'a pas besoin d'être chiffrée car son but est d'être partagée à *tout le monde*. Cette clé permet uniquement de chiffrer les messages sans pouvoir les déchiffrer.

Lorsqu'on refait la manipulation avec la clé 2 on obtient bien la même clé publique.

Réalisation d'un échange chiffré entre Gn et Dn

Rappel de système

Génération de vos clés

On génère une clé de RSA :

```
openssl genrsa -out gn_key 2048
```

Puis on la chiffre et on supprime la clé non chiffrée:

```
openssl rsa -in gn_key -des3 -out private.pem
rm gn_key
```

On génère la clé publique :

```
openssl rsa -in private.pem -pubout -out publique.pem
```

On envoie à Dn la clé publique :

```
scp ./publique.pem root@10.24.0.3:/home/echange/
```

Utilisation de la clé pour un premier échange de

Création d'un message sur Dn et échange du message

On créé un fichier de moins 116 octets :

```
echo "Coucou c'est qno et mha" > test.txt
```

Puis on chiffre ce message :

```
openssl rsautl -encrypt -in test.txt -pubin -inkey publique.pem -out sortie
```

On communique ce message à Gn :

```
scp ./sortie root@10.24.0.1:/home/echange/
```

Lecture du message sur Gn

On essaye de lire le message sur Gn :

```
openssl rsautl -decrypt -in test.txt -pubin -inkey private.pem -out
fichier_decrypte
```

Si ce même message était envoyé à tous, seul Gn serait capable de déchiffrer puis lire ce message car c'est le seul qui possède la clé privée.
Les autres étudiants auraient juste un fichier chiffré illisible.

Cette méthode est le chiffrement et permet de sécuriser la confidentialité d'un message ou d'un fichier. C'est du chiffrement asymétrique car il y a 2 clé, une clé publique utilisée pour chiffré le message, puis une clé privée utilisée pour déchiffrer le message.

Conclusion

On peut observer plusieurs vulnérabilités avec le système mis en place plus tôt :

- Bien qu'on sécurise la confidentialité du message, on ne le fait qu'en étant sûr du destinataire.
  car nous n'avons aucun moyen de vérifier que la clé publique est bien celle de Gn, il faut alors sécuriser l'échange de ces clés.
- Il faut faire attention lors de la génération de la clé publique et privé, car un ordinateur mal sécurisé pourrait être atteint par une attaque de proximité.
  La génération de ces clés doit se faire dans le secret le plus absolu.
- La vulnérabilité principale est alors le partage des clés.

Utilisation de la clé pour un second échange de message

Création d'un message sur Gn et échange du message

On créé un fichier de moins 116 octets :

```
echo "Coucou c'est qno et mha 2" > test.txt
```

Puis on chiffre ce message :

```
openssl rsautl -encrypt -in test.txt -pubin -inkey private.pem -out fichier_code
```

On communique ce message à G=Dn :

```
scp ./fichier_code root@10.24.0.3:/home/echange/
```

Lecture du message sur Dn

On essaye de lire le message sur Dn :

```
openssl rsautl -decrypt -in test.txt -pubin -inkey publique.pem -out
fichier_decrypte
```

Si ce même message était envoyé à tous, tous les postes seraient capable de déchiffrer puis lire ce message car tous les possesseurs de la clé publique peuvent déchiffrer le message.

Ce processus est aussi le chiffrement asymétrique, partant de la clé privé source vers tout le monde (la destination)

Conclusion

On ne retrouve pas les vulnérabilités de la manipulation précédente car :

- L'auteur du message est connu car il est le seul à avoir la clé privée, si le message n'est pas decodable avec la clé publique de X alors X n'est pas l'auteur
- Le partage de la clé publique ne représente plus une menace à la sécurité de l'échange car l'auteur X est la source du message

Néanmoins ce message n'est plus confidentiel car le fichier est chiffré à l'aide la clé privée.

On met juste un place un processus d'authentification de l'auteur du message.

Sécurisation d'un serveur SSH par double authentification

Introduction

Pré requis

Test du fonctionnement du server ssh de Sn :

```
ssh root@127.0.0.1
```

Après connexion on se déconnecte et on met à jour la date :

```
ntpdate -b 10.254.0.254
```

Configuration de Linux

Après avoir installé la librairie Google Authentificator :

```
apt install libpam-google-authenticator
```

On lance le service et on vérifie que tout fonctionne correctement:

```
google-authenticator
```

On obtient les codes de scratch :

```
Your new secret key is: SPQQN7MXZMBJPCMF4LGNE6J6DU
Enter code from app (-1 to skip): 739871
Code confirmed
Your emergency scratch codes are:
  70161429
  25276468
  18575756
  84574451
  69112653
```

On lance un smartphone et on scanne le qr code, cela nous donne des informations supplémentaires et des codes de secours :

```
SPQQN7MXZMBJPCMF4LGNE6J6DU
" RATE_LIMIT 3 30
" WINDOW_SIZE 17
" DISALLOW_REUSE
" TOTP_AUTH
70161429
25276468
18575756
84574451
69112653
```

On édite le fichier `/etc/pam.d/sshd` et on rajoute la ligne suivante :
`auth required pam_google_authenticator.so`

Puis on change le fichier `/etc/ssh/sshd_config` de la façon suivante :
`ChallengeResponseAuthentication yes`

On redémarre le service ssh avec la commande :

```
systemctl restart ssh
```

On vérifie l'accès au ssh de Sn à l'aide la commande suivante depuis Dn :

```
ssh root@10.24.0.2
```

On obtient ce résultat :

```
root@S24:/home/test# ssh root@10.24.0.2
Password:
Verification code:
Linux S24.tp241.iut 5.10.0-19-amd64 #1 SMP Debian 5.10.149-2 (2022-10-21) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Mar  6 15:02:11 2023 from 10.24.0.3
```

Utilisation de GPG entre Gn et Dn

Prise en main de GPG

Pour générer une nouvelle paire de clé il faut utiliser la commande suivante :

```
gpg --generate-key
```

Pour lister les clés il faut utiliser la commande suivante :

```
gpg --list-keys
```

Création des clés sous Dn et Gn

On créé les clés de la façon suivante :

```
gpg --generate-key
```

Puis on a remplie les différents prompts

```
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Remarque : Utilisez « gpg --full-generate-key » pour une fenêtre de dialogue de
génération de clef complète.

GnuPG doit construire une identité pour identifier la clef.

Nom réel : uti_gaston
Adresse électronique : uti_gaston@mail.fr
Vous avez sélectionné cette identité :
    « uti_gaston <uti_gaston@mail.fr> »

Changer le (N)om, l'(A)dresse électronique ou (O)ui/(Q)uitter ? o
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
gpg: clef 2BCB3DB853ADBB21 marquée de confiance ultime.
gpg: revocation certificate stored as '/root/.gnupg/openpgp-
revocs.d/4651B8FB1A39AEB5A2C508272BCB3DB853ADBB21.rev'
les clefs publique et secrète ont été créées et signées.

pub   rsa3072 2023-03-06 [SC] [expire : 2025-03-05]
      4651B8FB1A39AEB5A2C508272BCB3DB853ADBB21
uid                      uti_gaston <uti_gaston@mail.fr>
sub   rsa3072 2023-03-06 [E] [expire : 2025-03-05]
```

Enfin on liste la création des clés de chaque poste :

Sur Gn:

```
/root/.gnupg/pubring.kbx
------------------------
pub   rsa3072 2023-03-06 [SC] [expire : 2025-03-05]
```

```
      4651B8FB1A39AEB5A2C508272BCB3DB853ADBB21
uid          [  ultime ] uti_gaston <uti_gaston@mail.fr>
sub   rsa3072 2023-03-06 [E] [expire : 2025-03-05]
```

Sur Dn:

```
gpg: vérification de la base de confiance
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: profondeur : 0  valables :   1  signées :   0
     confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
gpg: la prochaine vérification de la base de confiance aura lieu le 2025-03-05
/root/.gnupg/pubring.kbx
------------------------
pub   rsa3072 2023-03-06 [SC] [expire : 2025-03-05]
      4BC83A24C403BCEEC59D2FB6E8EA35C44D815F70
uid          [  ultime ] uti_didier <uti_didier@mail.fr>
sub   rsa3072 2023-03-06 [E] [expire : 2025-03-05]
```

Manipulation d'une clé

On exporte la partie secrète et la partie publique de la clé de Dn:

```
gpg --export-secret-key uti_didier > uti_didier.key
gpg --export uti_didier > uti_didier.pub
```

On supprime la clé uti_didier :

Il faudra supprimer avant la clé privée et la clé publique qui avaient été générées

```
gpg --delete-secret-and-public-keys uti_didier
```

Lors de cette suppression, on doit confirmer la suppression de chaque clé.
Les fichiers précédemment générés ne sont pas supprimés

Pour importer la clé il faut importer la clé privée:

```
gpg --import uti_didier.key
```

car c'est la clé qui comporte les informations sensibles du chiffrement, il suffit d'importer une seule clé, la clé privée. (Le calcul de la clé publique est possible depuis la clé privée)

Échange des parties publiques des clés

On récupère sur chaque poste la clé publique de l'autre :

```
scp /home/echange/uti_nom.pub @root@destination:/home/echange/
```

Puis on les importe avec la commande `gpg --import key`

Échange des messages

Premier échange

On saisie un premier message puis on le chiffre avec gpg :

```
gpg -e -r uti_didier message.txt
```

- -e permet de chiffrer le message
- -r permet d'utiliser la clé publique du destinataire (ici uti_didier)

La commande ne demande rien.

Cette commande chiffre le message et crée un fichier `nom_fichier.extension.gpg` contenant le message chiffré.

Un message d'avertissement : `La clé n'appartient pas forcément à la personne nommée dans l'identité` car la clé publique ne permet pas de vérifier l'authenticité du destinataire.

Après avoir transféré le message sur le poste Dn on le déchiffre :

```
gpg -d message.txt.gpg
```

Ici la commande nous demande le mot de passe de la clé privée avant de déchiffrer le message afin de vérifier que nous sommes le bon destinataire.
Mais il n'y a pas de message d'avertissement.

Pour conclure, dans cet échange le message est chiffré car seul le destinataire est capable de le lire néanmoins l'échange n'est pas signé car on ne peut pas être sûr de l'identité du destinataire ni de l'émetteur.

Second échange

On utilise la commande:

```
gpg -s -u uti_gaston message2.txt
```

Les options sont:

- -s : Signe le message avec la clé privée
- -u : L'identité de l'auteur du message

La commande demande le mot de passe de l'auteur (uti_gaston) car il y a utilisation de la clé privée.
Au final cette commande chiffre le message puis l'exporte dans le fichier `message2.txt.gpg`
Cette commande n'a pas affiché d'avertissement.

Après transfère sur la machine Dn, on déchiffre le message à l'aide de la clé publique de uti_gaston :

```
gpg -d message2.txt.gpg
```

La commande ne nous demande rien et nous affiche le message déchiffré.
Cependant elle nous affiche un message d'avertissement indiquant que le message est bien signé avec l'utilisateur `uti_gaston` mais que la clé n'est pas certifiée par une signature de confiance.
Rien n'indique que la signature appartient à son propriétaire.

En conclusion cette échange est signé car on sait que le propriétaire du message est `uti_gaston` mais il n'est pas confidentiel car n'importe qui peut le déchiffrer à l'aide de la clé publique. Le destinataire peut le savoir car seul une message de X est déchiffrable avec la clé publique de X.

Dernier échange

On chiffre et on signe le message à l'aide de la commande:

```
gpg -e -r uti_didier -s -u uti-gaston
```

Cette commande nous affiche l'avertissement comme quoi la clé du destinataire n'est peut être pas la bonne puis chiffre et signe le message
La commande ne nous a rien demandé.
La commande exporte le message chiffré et signé sous la forme : `nom_fichier.extension.gpg`

Après avoir transféré le message on le déchiffre avec la commande:

```
gpg -d message3.txt.gpg
```

La commande nous demande le mot de passe de uti_didier puis déchiffre le message.

Il nous indique que le message est à la fois chiffré et signé mais comme avant que la signature ne peut pas être certifiée.

En conclusion l'échange est chiffré (avec la clé publique uti_didier) et signé (avec la clé privée uti_gaston).

Conclusion

Conclusion la dernière vulnérabilité est le partage des clés car un attaquant peut se positionner en `man in the middle` et usurper l'identitée du destinataire.

Révoquer une clé

On génère un certificat de révocation:

```
gpg --gen-revoke uti_gaston
```

On nous propose la création d'un certificat puis l'ajout de la cause de la révocation, une description et enfin la confirmation de la création.
Cela génère une clé protégée qu'il faut envoyer à Dn

On le transfère à Dn et on l'importe.

```
gpg --import revoke.gpg
```

Cela a importé le certificat de révocation et a placé la clé de uti_gaston dans le registre des clés révoquées.

On chiffre un message pour Gaston

```
gpg -e -r uti_gaston message_gaston.txt
```

La commande nous indique qu'il n'y a plus de clé publique associée à Gaston. (Car elle a été révoquée)

On créer un message en tant que gaston et on le signe puis on le donne à Didier.

```
gpg -s -u uti_gaston message_didier.txt
```

On déchiffre sur Dn:

```
gpg -d message_didier.txt
```

La commande nous indique plusieurs choses:

- Le message est signé par `uti_gaston`
- La clé a été révoqué par le propriétaire, avec la cause et la description
- La clé n'est pas certifiée avec une signature de confiance.

Pour que GPG indique à Gaston que sa clé est révoqué il faut que lui aussi importe le certificat de révocation.

```
gpg --import revoke.gpg
```

> signing failed: Unusable secret key