Introduction

Ce TP a pour objectif de se familiariser avec les notions liées à la cryptographie. Il s'articule autour de deux parties : Dans la première partie, vous serez amenés à utiliser la méthode de chiffrement symétrique des messages et vérifier la rapidité de cette méthode ainsi que ses inconvénients. La deuxième partie traitera l'utilisation des chiffrements asymétriques, la lourdeur de cette méthode ainsi que ses avantages. Le logiciel GPG (gardien de la vie privée) `https://gnupg.org` est la version GNU de PGP (Pretty Good Privacy) `https://openpgp.org` crée par Philip Zimmermann, c'est un logiciel de cryptographie renforcé.

Exercice 1

1. **Chiffrement symétrique**

   (a) En utilisant la commande "`man gpg`", essayez de parcourir l'aide de notre logiciel.

   (b) Dans un répertoire TP1, créer un fichier RT4.txt contenant le message suivant : "Bonjour les gars !". Exécuter les commandes suivantes :
   - "`gpg --symmetric RT4.txt`" :
   - "`gpg --symmetric --armor RT4.txt`" :
   Vérifiez le output de ces deux commandes, Que remarquez vous ?

   (c) Décrypter le fichier chiffré en utilisant la commande "`gpg --decrypt nomFichier-généré`". Quel est le résultat de l'exécution ? Quel algorithme de chiffrement a été utilisé ?

   (d) Chiffrer le fichier RT4.txt avec l'algorithme AES.

   (e) Pour garantir l'intégrité des messages, vous allez utiliser une fonction de hachage H() de votre choix. Pour ce faire :
      i. Échanger ce tuple (H(RT4.txt),RT4.txt.asc,le nom de la fonction de hachage utilisée, un Random) en utilisant Secure Shell avec votre camarade.

      ii. Si vous avez la clef de déchiffrement, procédez au déchiffrement et vérifiez l'intégrité de votre fichier.

      iii. Pour prouver à votre expéditeur la bonne réception du message, envoyez lui une incrémentation de votre random, plus votre identité (votre nom par exemple).

      iv. Changez légèrement le déchiffré du fichier et vérifiez la détection de la fraude.

      v. Énumérez les inconvénients du chiffrement symétrique, et comment y remédier ?

2. **Chiffrement asymétrique**

   (a) Exécuter la commande suivante :"`gpg --gen-key`". Relever les choix introduits par défaut. GPG a besoin de générer des octets aléatoires pour la création des clefs. Alors pour accélérer le processus, vous pouvez générer un shell avec une boucle infinie ou bien $ls - R/$. Ou bien $cat/dev/urandom$. On nomme cette paire de clefs $CLEF\_A$.

   (b) Pour afficher la liste des clefs, utilisez la commande suivante : "`gpg --list-keys`". Donner le résultat d'exécution de cette commande.

(c) Quel est le contenu du réprtoire "∼/.gnupg" ? Quel sont les rôles des fichiers "pubring.gpg" et "secring.gpg" ?

(d) Générer un certificat de révocation de votre paire de clefs *Rev_A*.

(e) Générer une autre paire de clefs dans une machine distante ou bien dans une autre VM. On nomme cette paire de clefs *CLEF_B*.

(f) Générer un certificat de révocation de votre paire de clefs *Rev_B*.

(g) Échangez la clef publique des deux machines. *CLEF_A_Pub* vers la machine B et la *CLEF_B_Pub* vers la machine A. Pour ce faire, vous utilisez la commande export et import de notre logiciel GPG. Et pour le transfert, utiliser *python −m SimpleHTTPServer* dans la machine par exemple A et dans la machine B, utiliser $wget\ http://@ip_A:8000/CLEF\_A\_Pub$

(h) En tant que machine A, chiffrez le fichier "RT4.txt" avec la clef publique de B en utilisant l'option *armor*, puis l'envoyer à B d'une manière sécurisée.

(i) B Déchiffre le fichier reçu et envoie un message de confirmation de réception à A.

(j) En utilisant le serveur de clefs pgp.mit.edu (ou bien http ://wwwkeys.pgp.net/), uploader vos clefs publiques sur ce serveur. Puis, les importer sur les machines adéquates.

(k) Générer un nombre aléatoire, puis signer le avec votre clef privée. Envoyez à B le fichier "RT4.txt" avec le Random signé, le tout chiffré par *CLEF_B_Pub*.

(l) B déchiffre le fichier reçu, vérifie l'authenticité de la signature. Cependant, comme la signature de A n'est pas certifiée, B obtient un avertissement. Pour valider la signature de A, B utilise "−−edit-key" et l'option "sign" pour signer et certifier la signature de A. Ou bien B utilise cette commande "−−sign-key id-de-A". L'envoie de cette clef signée au serveur est convié. Pour voir la liste des clefs signées, vous tapez cette commande : "gpg −−list-sigs" et pour vérifier les différentes signatures que A possède tapez : "gpg −−check-sigs id-de-A". Le sig ? correspond à des signatures qui n'ont pas été vérifiées (on ne possède pas la clef publique pour vérifier), alors que sig ! indique que la signature correspond à la clef publique que l'on a sur notre trousseau.

(m) Pour remédier à la perte du temps dédiée à la récupération des signatures des nouvelles clefs, vous pouvez écrire un script shell et le paramétrer dans votre crontab à la fin de la journée.

```
#!/bin/sh
echo "Récupération des nouvelles signatures des clefs
 hebergé dans le serveur qui font partie de notre trousseau"
for i in $(/usr/bin/gpg --list-keys | grep '^pub' | cut -c 13-20); \
    do /usr/bin/gpg --keyserver pgp.mit.edu --recv-key $i; done
```

(n) Revenons à notre sénario, B vérifie à nouveau l'authenticité de la signature de A, en utilisant la commande "check", elle ne devrait plus avoir un avertissement.

(o) B incrémente le Random de 1, puis, envoie à A le Random incrémenté signé par *Priv_B*. A certifie de même la signature de B.

(p) Utilisez les différentes versions de vérification dans vos messages échangés, c'est à dire "−−detach-sign" (Signatures détachées), "−−clearsign" (Les fichiers signés en clair) et "−−sign". Déduisez la différence.

(q) Pour vérifier la signature, utilisez l'option "-verify". Pour vérifier la signature et extraire le fichier, utilisez l'option "-decrypt". Le fichier et la signature détachée sont tous deux nécessaires pour vérifier la signature du fichier.

(r) Avant de passer à la partie graphe de confiance, prenez du temps à bien gérer votre trousseau de clefs. Un attaquant peut usurper une clef publique d'un utilisateur à son insu ou falsifier son trousseau de clefs. S'attarder sur les options de la commande "−−`edit-key`" est recommandée. Vous pouvez rajouter des sous-clefs et des identifiants d'utilisateur à votre paire de clefs une fois qu'elle a été créée avec l'option "`adduid`" et "`addkey`". Vous pouvez aussi révoquer une sous-clef avec la commande "`revkey`" en lui ajoutant une auto-signature de révocation, contrairement à "−−`gen-revoke`" que son effet est immédiat. Vous pouvez aussi révoquer la signature d'un identifiant en utilisant "`revsig`". Vous pouvez mettre à jour la date d'expiration d'une clef avec "`expire`". Sans oublié la commande "`trust`" pour définir le niveau de confiance que vous avez dans le propriétaire d'une clef.

(s) Dans cette partie, vous devez comprendre la notion de confiance en GPG en créant un graphe de confiance entre les différentes clefs du trousseau.

i. Dans un graphe de reconnaissance de signature, une arête entre A et B indique que A a signé la clef publique de B. Dans ce cas, A fait confiance à B. Nous n'allons pas vérifier clef par clef qui à signé qui. Nous nous appuyons sur un graphe pour en déduire la confiance.

ii. Vous devez tout d'abord installer les paquets graphviz et sig2dot.

iii. Vous êtes amenés à créer un graphe qui illustre la confiance suivante : Une confiance bilatérale entre A et B : A $\Longleftrightarrow$ B, A fait confiance à C : A $\Longrightarrow$ C, B fait confiance à C : B $\Longrightarrow$ C.

A. Indication : "`gpg −−list-sigs | sig2dot | dot -Tpng > path/Graphe.png`"

Exercice 2

Protéger votre vie privée est cruciale. L'email fait partie. Pour les utilisateur de linux cette tâche est actuellement simple en s'appuyant sur des outils open source : Mozilla Thunderbird, Enigmail, and GNU PGP(GPG). L'internaute peut utiliser ces outils afin de recevoir et envoyer des messages chiffrés et signés. Le but de cet exercice est de reprendre ce que nous avons fait dans l'exercice 1 partie 2 tout en utilisant le mode graphique de Thunderbird avec son plugin Enigmail.

♣ S.Y. ♣
Bon travail