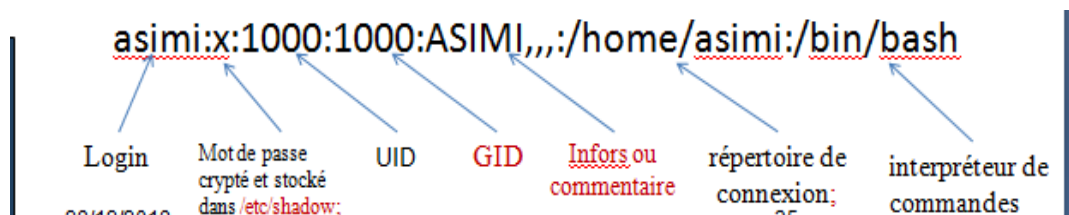


- I. Pour assurer la sécurité des mots de passé, Unix a changé la structure de stockage des informations propres à chaque utilisateur. Ici, nous citons un exemple d'une ligne de stockage dans le fichier `/etc/passwd`:

asimi:x:1000:1000:ASIMI,,,:/home/asimi:/bin/bash

☞ donner la signification de chaque champ de cette ligne en précisant son impact sur la sécurité des mots de passe sous Unix.



La seule personne qui a le pouvoir de changer les informations propres à un utilisateur dans le fichier `/etc/passwd` est le **superutilisateur (root)**:

- **Login**: nom du compte de l'utilisateur ;
- **X**: mot de passe de l'utilisateur (codé bien sûr) ;
- **UID**: identifie l'utilisateur pour le système d'exploitation (UID=User ID) ;
- **GID**: identifie le groupe de l'utilisateur (GID=Group ID) ;
- **Commentaire**: donne des informations sur l'utilisateur ou bien son nom réel;
- **Répertoire de connexion**: après s'authentifier, l'utilisateur se trouve dans ce répertoire `:/home/asimi`;
- **Interpréteur de commandes**: est l'interpréteur des commandes par défaut après connexion au système : `/bin/bash`;

- II. Le résultat de cryptage d'un mot de passe dans le fichier `/etc/shadow` ressemble à:

\$1\$s1f5m5j4\$0LyI.z6g6/.Fx4x0y6nxD0

- ☞ Déterminer les deux parties composant ce mot de passe crypté ;
- ☞ Citer les deux fonctions utilisées pour régénérer ce mot de passe en précisant leurs impacts sur la sécurité de ce mot de passe régénéré ;
- ☞ Trouver le mot de passe original saisi par l'utilisateur ;

Mot de passe crypté: le résultat de cryptage d'un mot de passe crypté dans /etc/shadow ressemble à :

`1S1f5m5j4$0Ly1.z6g6/.Fx4x0y6nxD0`

Ce mot de passe crypté est composé de deux parties:

- **Première partie : `1S1f5m5j4$` correspond au salt:**
 - Elle commence toujours par **`1`** et finit par **`$`**;
 - Le "salt" est une chaîne aléatoirement, qui n'est pas secrète, et qui sert à perturber le cryptage;
 - Le "salt" devrait être régénérer par **régénérateur aléatoire**;
- **Deuxième partie : `0Ly1.z6g6/.Fx4x0y6nxD0` est le mot de passe crypté :**
 - Il est crypté à l'aide de la fonction de hachage MD5
 - **Cette fonction de hachage est montré cassable;**

III. Gestion des groupes et des utilisateurs

Pour protéger son environnement de travail, le système d'exploitation Linux scinde les utilisateurs en trois classes, également, ses droits d'accès:

- III.1. Donner ces trois classes des utilisateurs ;
- III.2. Créer un nouveau compte utilisateur portant votre nom ;
- III.3. Verrouiller le compte utilisateur créé ;
- III.4. Changer le nom et l'emplacement de ce compte utilisateur créé.
- III.5. Créer un nouveau groupe supplémentaire portant votre prénom (GID : 2021);
- III.6. Ajouter ce compte utilisateur à ce groupe ;
- III.7. Modifier le nom de ce groupe des utilisateurs ;
- III.8. Lister les groupes d'utilisateur ;
- III.9. Supprimer l'utilisateur et le groupe créés ;

Solution

- 1) Donner ces trois classes des utilisateurs ; (0.75)
 - User, Group, Others
- 2) Créer un nouveau compte utilisateur portant votre nom ;
 - `Sudo useradd asimi`
- 3) Verrouiller le compte utilisateur créé ;
 - `#passwd -l asimi`
- 4) Changer le nom et l'emplacement de ce compte utilisateur créé.
 - `# usermod login nouvel_identifiant --home nouvel_emplacement_du_dossier_personnel movehome identifiant_actuel`

Prof. Y. ASIMI

- 5) Créer un nouvel groupe supplémentaire portant votre prénom (GID : 2021);
 - `#groupadd -G 2021 younes`
 - 6) Ajouter ce compte utilisateur à ce groupe ;
 - `#useradd -G younes asimi`
 - 7) Modifier le nom de ce groupe des utilisateurs ;
 - `#groupmod -n kaka younes`
 - 8) Lister les groupes d'utilisateur ;
 - `#groups ASIMI`
 - 9) Supprimer l'utilisateur et le groupe créés ;
 - `[root@root]# userdel -r ASIMI`
 - `#groupdel kaka`
 - 10) **EXERCICE :**
 - 1) Créer deux groupes groupG1 (GID : 2020) et groupG2 (GID : 2021);
 - a. `#groupadd -G 2020 groupG1`
 - b. `#groupadd -G 2021 groupG2`
 - 2) Créer quatre utilisateurs user1, user2, user3 et user4 ;
 - a. `#useradd user1`
 - b. `#useradd user2`
 - c. `#useradd user3`
 - d. `#useradd user4`
 - 3) Rendre les utilisateurs dans les groupes créés :
 - a. Les premier et deuxième utilisateurs sont membres du premier groupe ;
 - b. Les troisième et quatrième utilisateurs sont membres du second groupe ;
 - c. Le deuxième utilisateur est aussi membre du second groupe ;
 - d. Le quatrième utilisateur est aussi membre du premier groupe ;
- On peut utiliser : `usermod -g groupname username` → Groupe principal
Ou bien : `usermod -a -G groupname username` → Groupe supplémentaire
- i. `#usermod -g group1 user1`
 - ii. `#usermod -g group1 user2`
 - iii. `#usermod -g group2 user3`
 - iv. `#usermod -g group2 user4`
 - v. `#usermod -a -G group2 user1`
 - vi. `#usermod -a -G group1 user4`
- 4) Vérifier les membres du groupe groupG2 ;

Less ou bien More : Permettent d'afficher page par page à l'écran le contenu d'un fichier texte.

 - a. `less /etc/passwd`
 - b. `less /etc/group`
 - 5) Créer deux répertoires rep1, rep2 et rep3 en seul ligne ;

- a. **\$ mkdir rep1 rep2 rep3**
- 6) Créer dans rep1 un fichier nommé fich11 et dans rep2 un répertoire nommé rep21 ;
 - a. **\$ cd rep1**
 - b. **\$ touch fich11**
 - c. **\$ cd ..**
 - d. **\$ cd rep2**
 - e. **\$ mkdir rep21**
- 7) Déplacez-vous au répertoire rep21 ;
 - a. **\$ cd rep21**
- 8) Copier le rep1 et son contenu dans le répertoire courant ;
 - a. **\$ cp -r .././rep1 .**
- 9) Copier l'arbre rep2 dans le répertoire rp3 ;
 - a. **\$ cp -r .././rep2 .././rep3**
- 10) Visualiser le contenu de rep3 de façon détaillée ;
 - a. **\$ ls -l rep3**
- 11) Supprimer l'arbre rep3 ;
 - a. **\$ rm -r rep3**

Etude de RSA

IV. ETUDE DE RSA :

1- Les fonctions composantes de RSA :

- Calcul de n et $\ell(n)$.
- Génération de e (recherche des nombre premier avec $\ell(n)$).
- Génération de d (calcul d'inverse).
- Cryptage.
- Décryptage.

2- Les opérations utilisées dans RSA :

- Soustraction
- Multiplication
- Puissance
- Modulo
- Division

3- Description du fonctionnement des composantes:

- ✍ Calcul de n et $\ell(n)$: après avoir choisis les deux nombre p et q qui sont deux paramètres d'entrée de RSA, l'algorithme procède au calcul de $n = p*q$ et $\ell(n) = (p-1)(q-1)$.

Prof. Y. ASIMI

- ✍ Génération de e : pour la génération de la clé publique e l'algorithme recherche l'ensemble des nombres qui seront premiers avec $\phi(n)$ et inférieurs à n .
- ✍ Génération de d : la génération de la clé privée consiste à trouver l'inverse de e de tel sorte que : $e * d = 1 \% n$.
- ✍ Cryptage : le chiffrement se fait comme suit : $C = M^e$.
- ✍ Décryptage : $M = C^d$.

4- Les paramètres de RSA sont p et q .

5- Les variables de sécurisation de RSA sont les tailles des nombres p et q choisis.

Vu que la sécurité de RSA dépend seulement de la longueur des nombres p et q choisis, l'algorithme sera facilement casser avec la recherche exhaustive, si ces deux nombres sont petits.

5.1. $q=7$ & $p=5$:

5.1.1. Détermination l'ensemble des nombres premiers et inférieur à 24 et premier avec 24 :

Un nombre premier avec 24 est un nombre ou le PGCD est égal à 1 on trouve alors :

$$24 = 3 \cdot 2^3$$

Ce qui veut dire que tous les nombres à trouver ne doivent pas avoir 3 et 2 comme diviseur on trouve alors c'est l'ensemble :

$$\{5, 7, 11, 13, 17, 19, 23\}$$

5.1.2. On montrer que pour tous nombre premier trouvé admet son propre inversible :

Pour montrer que x accepte son propre inversible il faut montrer que $x * x = 1 \% 24$:

$$\{5, 7, 11, 13, 17, 19, 23\}$$

$$\begin{aligned} 5 \cdot 5 &= 25 \% 24 = 1 \\ 7 \cdot 7 &= 49 \% 24 = 1 \\ 11 \cdot 11 &= 121 \% 24 = 1 \\ 13 \cdot 13 &= 169 \% 24 = 1 \\ 17 \cdot 17 &= 289 \% 24 = 1 \\ 19 \cdot 19 &= 361 \% 24 = 1 \\ 23 \cdot 23 &= 529 \% 24 = 1 \end{aligned}$$

5.1.3. On montrer que RSA est symétrique dans ce cas:

L'ensemble des clés public e trouver dans ce cas accepte leur propre inversible d ce qui veut dire que $e = d$, alors la clé de chiffrement et elle même utilisée pour le déchiffrement du message ce qui est applicable dans le chiffrement symétrique.

5.2. $p=7$ et $q=11$:

5.2.1. On déterminer toutes des clés pour que RSA soit symétrique & asymétrique:

$$n = 7 \cdot 11 = 77$$

$$\varphi(n) = (7 - 1)(11 - 1) = 60$$

$$e \in \{2, \dots, 77\} \setminus PGDC(e, 60) = 1$$

$$e = \{7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 51, 53, 57, 59, 61, 71, 73\}$$

Pour que RSA soit symétrique on vérifie chaque clé si elle accepte son propre inversible :

on trouve l'ensemble :

$$\text{symétrique} : \{11, 19, 29, 31, 41, 49, 59, 61, 71\}$$

$$\text{asymétrique} : \{7, 13, 17, 23, 37, 43, 47, 53, 67, 73\}$$

5.2.2. Cas clé publique $e = 7$:

5.2.2.1. Montrer que RSA dans ce cas est asymétrique :

Dans ce cas RSA est asymétrique puisque la clé 7 se trouve dans l'ensemble asymétrique démontré dans la question précédente.

5.2.2.2. Calcul de la clé privée d :

Pour trouver la clé privée d de la clé 7 on a :

$$d \in \{7, 13, 17, 23, 37, 43, 47, 53, 67, 73\} \setminus d \cdot 7 = 1 \pmod{60}$$

On trouve alors dans cet ensemble que $d = 43$.

5.2.2.3. Chiffrement avec RSA :

Pour chiffrer un message avec RSA on commence tout d'abord par la conversion du message en code ascii de taille 3 le message choisis est « misr » :

$$M = \text{'Misr'} = 109, 105, 115, 114$$

Ensuite il faut représenter le message en blocs de même taille que n qui est égale à 77 dans notre cas ; chaque bloc sera de taille 2 :

$$M = \{10, 91, 05, 11, 51, 14\}$$

Il faut aussi vérifier que les blocs contiennent des nombres inférieurs à n (77) ce qui nous pousse à réduire la taille de chaque bloc on obtient alors :

$$M = \{1, 0, 9, 1, 0, 5, 1, 1, 5, 1, 1, 4\}$$

Maintenant on procède au calcul de C (message chiffré) :

$$C = \{1^7 \pmod{77}, 0^7 \pmod{77}, 9^7 \pmod{77}, 1^7 \pmod{77}, 0^7 \pmod{77}, 5^7 \pmod{77}, 1^7 \pmod{77}, 1^7 \pmod{77}, 5^7 \pmod{77}, 1^7 \pmod{77}, 1^7 \pmod{77}, 4^7 \pmod{77}\}$$

$$C = \{1, 0, 37, 1, 1, 0, 47, 1, 1, 47, 1, 1, 60\}$$

5.2.2.4. Déchiffrement du message :

Pour le déchiffrement du message on aura :

$$M = \{1^{43} \pmod{77}, 0^{43} \pmod{77}, 37^{43} \pmod{77}, 1^{43} \pmod{77}, 1^{43} \pmod{77}, 0^{43} \pmod{77}, 47^{43} \pmod{77}, 1^{43} \pmod{77}, 1^{43} \pmod{77}, 47^{43} \pmod{77}, 1^{43} \pmod{77}, 1^{43} \pmod{77}, 60^{43} \pmod{77}\}$$

$$M = \{1, 0, 9, 1, 0, 5, 1, 1, 5, 1, 1, 4\}$$

Ensuite on reprend la représentation en bloc de code ascii de taille 3 on obtient :

$$M = \{109, 105, 115, 114\} = \text{'Misr'}$$

V. Atelier : RSA ET OPENSSL

Prof. Y. ASIMI

OpenSSL est un outil cryptographique open source particulièrement pratique. Dans la partie consacrée aux références, vous trouverez toutes les informations nécessaires pour le télécharger, mais sachez que la plupart des distributions GNU/ Linux le proposent par défaut. Nous allons donc l'utiliser ici pour configurer un environnement de test dans lequel l'attaque à l'algorithme RSA sera lancée. Pour cela on procède comme suit :

- 1) Créer votre message à crypter avec RSA (nommé plain.tex).
- 2) Exécuter les commandes suivantes puis les interpréter :
 - 2.1) openssl genrsa -out rsa_privkey.pem m (m un nombre à préciser).
 - 2.2) cat rsa_privkey.pem.
 - 2.3) openssl rsa -in rsa_privkey.pem -pubout -out rsa_pubkey.pem.
 - 2.4) cat rsa_pubkey.pem.
 - 2.5) openssl rsautl -encrypt -pubin -inkey rsa_pubkey.pem -in plain.txt -out cipher.txt.
 - 2.6) openssl rsautl -decrypt -inkey rsa_privkey.pem -in cipher.txt.
 - 2.7) openssl rsa -in rsa_pubkey.pem -pubin -text -modulus.

Solution

Le message qu'on veut crypter « asimi younes », pour ce faire, on va créer un fichier sous le nom « plain.txt ».

🔑 Création d'une clef privée

Le premier travail à faire c'est de créer votre clé privée à partir de laquelle on va générer tous les autres paramètres de RSA.

La commande :

```
root@asimi-HP-620:/home/asimi# openssl genrsa -out rsa_privkey.pem 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
....+++++
e is 65537 (0x10001)
```

Cette commande permet de générer une clef privée d'une taille de 1024 en utilisant l'algorithme de chiffrement asymétrique RSA.

Pour afficher cette clef sous le terminal on exécute la commande suivante :

```
root@asimi-HP-620:/home/asimi# cat rsa_privkey.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDwi0DpHQLdqcC70nH+JRdt00JHf0He+KB6ppfXfbmvwCoQdD53
yM2s6/3fKpN5bbidXhC7E2EKQR+ttNZFiMMfIiAULhSVY5nseLf0XL+0VmvFuyt
VqdA18NiZMDfh8ewuDTN4oJpR/Gq4r0xXiGKdD8eVRpL6FQSBzJNH8PgZwIDAQAB
AoGAALiUCLWmw2RncTepj9ysHRuRUfERxW3Sa2jHq1to9hMytAr2vWBat2wH3hQa
GmVodl57PTwy8q9NyUUU9XNVMHEDhFuSHhYLorclWHSI84vNojbrDYHu6g0aP5w+
GE4m4NlwAWKjBpUKpikexn0mDnXbRUTQzWTL38Rct1ihJkECQQDunsHLu/Vi5IN9
kPsA+1KHppMzyyhBp15a2I/GqQaAaYMB0gkTwCef3tFhWM/7eanqPEeCEowBhBMV
QTeb4dMHAKaE5ikH6HmYaQI8fAmaFWJUV3lpbZQ+XYn7cBJUx6jlrkDiRdGks/7
ZQLCSGwygUy33tmF8NBJS9/rC5JcalPoQJARG+rzS6/FfnL7HtkQcfSx6z3Kaae
iBXkdbGmiqd3geTA+givKcb2llldXbIHFGy0WsWnDJsbrwipj/MAa+i8iQJBALbT
oweFB49qildRXUdf15t9xf1DfMEcnY0bcG0JdBHsw8UzarZ5bc4Nc8yELC5nnat1
CROIsPEBtR41yntyleECQQCn3V3LL17gjeU6Sp821UMurRVQv+KjyRbVfv7tjJIG
qf0wK1368jREosKKpVa6QDNblRkXCdMTpnrtxwFWLo46
```

🔑 Génération d'une clef publique

A partir de cette clef privée qui nous venons de créer on va générer la clé publique par la commande suivante :

```
root@asimi-HP-620:/home/asimi# openssl rsa -in rsa_privkey.pem -pubout -out rsa_pubkey.pem
writing RSA key
root@asimi-HP-620:/home/asimi# cat rsa_pubkey.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCsqGSIb3DQEBAQUAA4GNADCBiQKBgQDwi0DpHQLdqcC70nH+JRdt00JH
f0He+KB6ppfXfbmvwCoQdD53yM2s6/3fKpN5bbidXhC7E2EKQR+ttnZFIMMfIiA
UlhSVY5nseLf0XL+OVmvFuytVqdA18NiZMDfh8ewuDTN4oJpR/Gq4r0xXiGKdD8e
VRpL6FQSBzJNH8PgZwIDAQAB
-----END PUBLIC KEY-----
root@asimi-HP-620:/home/asimi#
```

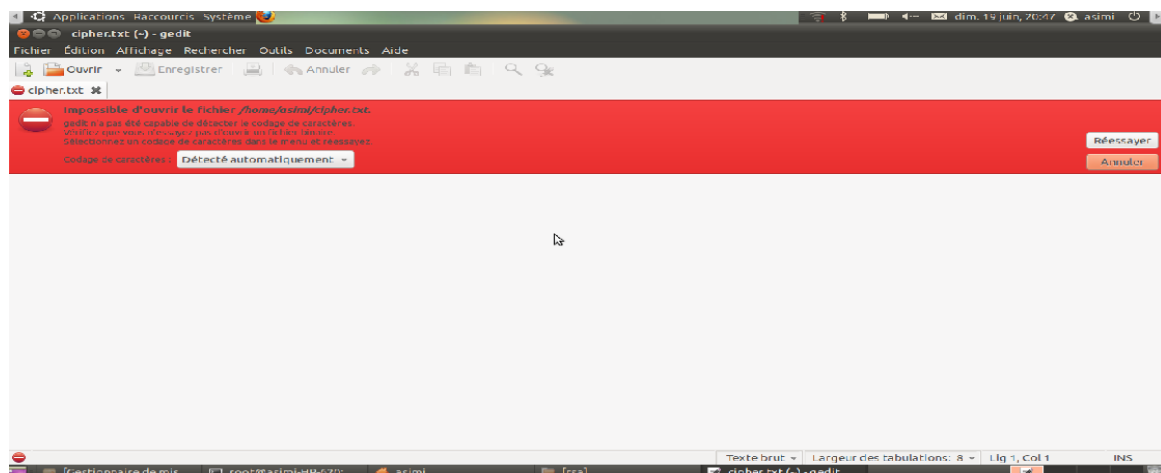
🔗 Chiffrement du message

Pour chiffrer le message on va utiliser la clef publique le résultat de cette commande on va le mettre dans un autre fichier « cipher.txt ».

La commande :

```
root@asimi-HP-620:/home/asimi# openssl rsautl -encrypt -pubin -inkey rsa_pubkey.pem -in plain.txt -out cipher.txt
root@asimi-HP-620:/home/asimi#
```

Le résultat :



On ne peut pas accéder au message car le fichier chiffré.

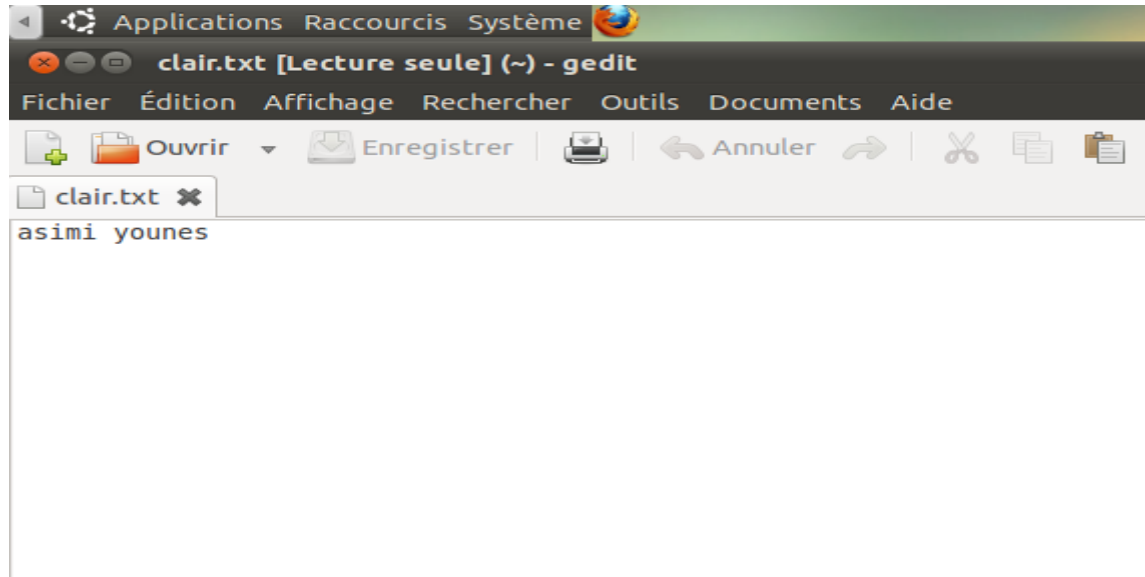
🔗 Déchiffrement du message

Comment RSA est asymétrique on va utiliser la clé publique pour déchiffrer le message crypté. Le résultat on va le mettre dans le fichier « clair.txt ».

La commande :

```
root@asimi-HP-620:/home/asimi# openssl rsautl -decrypt -inkey rsa_privkey.pem -in cipher.txt -out clair.txt
root@asimi-HP-620:/home/asimi#
```

Le résultat :



Le modulus

On sait que le modulo n'autre que le produit scalaire de deux nombre premier p et q pour le voir on va exécuter la commande suivante :

```
root@asimi-HP-620: /home/asimi
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
lus
Modulus (1024 bit):
00:d6:88:e0:e9:1d:02:dd:a9:c0:bb:3a:71:fe:25:
17:6d:3b:42:47:7f:41:de:f8:a0:7a:a6:97:d7:7d:
b9:af:c0:2a:10:74:3e:77:c8:cd:ac:eb:fd:df:2a:
93:79:6d:b8:9d:c5:78:42:ec:4d:84:29:04:7e:b6:
d9:d9:16:23:0c:7c:88:80:52:58:52:55:8e:67:b1:
e2:df:d1:72:fe:39:59:af:16:ec:ad:56:a7:40:d7:
c3:62:64:c0:df:87:c7:b0:b8:34:cd:e2:82:69:47:
f1:aa:e2:b3:b1:5e:21:8a:0d:df:1e:55:1a:4b:e8:
54:12:07:32:4d:1f:c3:e0:67
Exponent: 65537 (0x10001)
Modulus=D688E0E91D02DDA9C0BB3A71FE25176D3B42477F41DEF8A07AA697D77DB9AFC02A10743E
77C8CDACEBFDDF2A93796DB89DC57842EC4D8429047EB6D9D916230C7C8880525852558E67B1E2DF
D172FE3959AF16ECAD56A740D7C36264C0DF87C7B0B834CDE2826947F1AAE2B3B15E218A0DDF1E55
1A4BE8541207324D1FC3E067
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDWi0DpHQLdqcC70nH+JRdt00JH
f0He+KB6ppfxfbmvwCoQdD53yM2s6/3fKpN5bbidXhC7E2EKQR+ttnZFimmfIiA
UlhSVY5nseLf0XL+OVmvFuytVqdA18NiZMdfh8ewJTN4oJpR/Gq4r0xXiGKdd8e
VRpL6FQSBzJNH8PgZwIDAQAB
-----END PUBLIC KEY-----
root@asimi-HP-620:/home/asimi#
```

Conclusion

Comme nous voyons très bien cet algorithme reste le plus simple et le plus rapide dans l'exécution. Mais elle reste le plus efficace et le plus utilisé pour sécurité la communication sur l'internet.

VI. Donner le rôle et un exemple d'utilisation de chacune des commandes suivantes:

🔗 **find, sleep, ps, declare, type, userdel;**

- **find.:** Recherche un fichier à partir du répertoire donné : (0.5)

```
asi ni@asi ni -HP-620 ~$ find -name "*.png"
./Screenshot from 2018-11-29 16:24:39.png
./Screenshot from 2018-11-29 16:24:38.png
./Screenshot from 2018-11-29 16:21:37.png
./Screenshot from 2018-04-20 21:20:15.png
asi ni@asi ni -HP-620 ~$ find *.png
Screenshot from 2018-04-20 21:20:15.png
Screenshot from 2018-11-29 16:24:38.png
Screenshot from 2018-11-29 16:21:37.png
Screenshot from 2018-11-29 16:24:39.png
asi ni@asi ni -HP-620 ~$ find -name asi ni
./asi ni
asi ni@asi ni -HP-620 ~$ find -size +11M
asi ni@asi ni -HP-620 ~$ find -size -11M
```

- **grep:** Recherche, dans un ou plusieurs fichiers, de toutes les lignes contenant une chaîne donnée de caractères : grep bon fl
- **ps,** La commande **ps options** permet d'obtenir des renseignements sur l'état des processus en cours. (1 point)

```
asi ni@asi ni -HP-620 ~$ ps axr
  PID TTY          STAT       TIME COMMAND
    4 ?           R           0:05 [kworker/0:0]
3241 pts/1    R+          0:00 ps axr
asi ni@asi ni -HP-620 ~$ ps u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
asi ni    2270  0.0   0.1  22692  5444 pts/1    Ss   16:18   0:00 bash
```

- **type;** La commande type indique si une commande est interne ou externe : \$type pwd.
- **Declare :** Le Shell possède une commande générale de définition de variable : declare i n1 n2 n3,
- **Userdel:** userdel r asimi \$ suppression du compte asimi;

VII. Expliquer la différence entre:

🔗 La suppression et le verrouillage des comptes utilisateurs sous Linux.

🔗 La commande chmod et umask.

- **chmod :** permet de modifier les droits d'accès des fichiers ou des répertoires déjà créés.
- **umask :** La commande umask permet de définir un masque de protection des fichiers et des répertoires lors de leur création.

VIII. Supposant que je suis connecté en tant que administrateur, par défaut, la création d'un répertoire donne les permissions suivantes : r-- rw- -w-..

🔗 Trouver le masque actuel défini pour tous vos fichiers et vos répertoires.

a. 315

↳ Définir un nouveau masque de protection pour tous vos fichiers et vos répertoires (en mode logique et décimale) : 753.

a. Umask 753// **umask u=rwx,g=rx,o=wx**

↳ Créer un nouveau fichier et un nouveau répertoire.

a. Touch f1 et mkdir// cat dossier ;

↳ Donner les permissions pour un fichier créé selon le nouveau masque de protection ;

a. --- -w- r--.

↳ Interpréter leurs nouvelles permissions.

a. **Utilisateur** : Il ne peut rien faire, pas des permissions pour le propriétaire des fichiers;

b. **Groupe** : Les membres de groupe ont le droit d'écrire dans les fichiers déjà créés ;

c. **Autres** : ils ont le droit de créer des fichiers ;

↳ De point de vu sécurité :

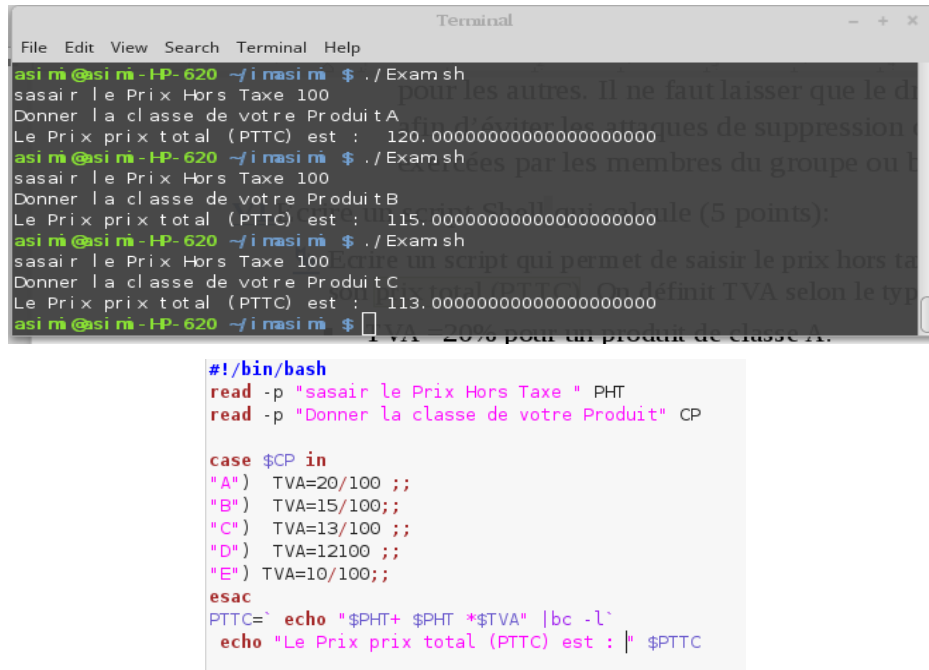
- Quelles sont vos recommandations sur ce masque ?
- Proposer un masque permettant d'éviter les attaques de suppression ou de modification des fichiers exercées par les membres du groupe ou bien les autres ;
- Quels sont les avantages de votre masque ;
 - a. ce masque devrait être changé. Nous devons donner les permissions totales pour les utilisateurs du système, d'ajouter le droit d'exécution pour les membres du groupe et de supprimer le droit de création des fichiers pour les autres. Il ne faut laisser que le droit d'exécution pour les autres afin d'éviter les attaques de suppression ou de modification des fichiers exercées par les membres du groupe ou bien les autres ;

IX. Ecrire un script qui permet de saisir le prix hors taxes (PHT) d'un article et de calculer son prix total (PTTC). On définit TVA selon le type de produit:

- TVA =20% pour un produit de la classe A ;
- TVA =15% pour un produit de la classe B ;
- TVA =13% pour un produit de la classe C ;

- TVA =12% pour un produit de la classe D ;
- TVA =10% pour un produit de la classe E.

Exemple d'exécution de ce script :



```
#!/bin/bash
read -p "sasair le Prix Hors Taxe " PHT
read -p "Donner la classe de votre Produit" CP

case $CP in
"A") TVA=20/100 ;;
"B") TVA=15/100 ;;
"C") TVA=13/100 ;;
"D") TVA=12/100 ;;
"E") TVA=10/100 ;;
esac
PTTC=`echo "$PHT+ $PHT *$TVA" |bc -l`
echo "Le Prix prix total (PTTC) est : |" $PTTC
```

X. Ecrire un script Shell qui calcule:

La somme des carrés de n premiers paramètres non nuls passés en argument à un script :

- Exemple : `$somCarre 1 3 6 0 5 9` → le résultat est : 46.
- `#!/bin/bash`
- `result=0`
- `while [$1 -ne 0] ; do`
- `resul=$(($1*$1))`
- `result=$(($result+$resul))`
- `shift`
- `$1`
- `done`
- `echo "La somme des carrés est : "$result`

Le quotient et le reste de la division euclidienne d'un entier donné x par y (avec x et y sont deux entiers saisis par l'utilisateur).

```
#!/bin/bash
read -p "saisir a: " a
read -p "saisir a: " b
r=$((a%b))
```

```
q=$((($a-$a%$b)/$b))  
echo "Le reste est :" $r  
echo "Le quotient est :" $q
```