

TP3

CONFIGURATION POSTFIX IMAP

Par DEPRADE ELIOTT

SOMMAIRE

But du TP -----	2
Installation et configuration de Postfix en mode "Site Internet" -----	2
Création des boites mails et envoi de mails -----	6
Créer un alias -----	7
Envoi depuis une autre machine locale (Kali) -----	8
Installation Dovecot et accès IMAP sécurisé (SSL/TLS) -----	11
Sécurisation par TLS -----	11
Configuration de Thunderbird (optionnel) -----	13
Accès depuis le WAN et tunnelisation via SSH -----	14
Accès depuis le WAN -----	14
Tunnelisation IMAP+STARTTLS via SSH -----	16
Utilisation de GnuPG pour chiffrer et signer les mails -----	18
Création des clefs (1) -----	18
Export des clefs publics (2) -----	20
Importation des clefs (3) -----	20
Envoi d'un message chiffré et déchiffrement (4) -----	20
Signature de clef (5) -----	23
Renvoi d'un message avec clef signée (6) -----	24
Et si une tierce personne veut lire les messages d'Alice et Bob ? -----	25

But du TP

Mettre en place un serveur de messagerie sécurisé en réseau local, comprenant la transmission, la réception et le chiffrement des emails entre utilisateurs via Postfix, Dovecot et GPG, avec exposition maîtrisée via pfSense.

Installation et configuration de Postfix en mode "Site Internet"

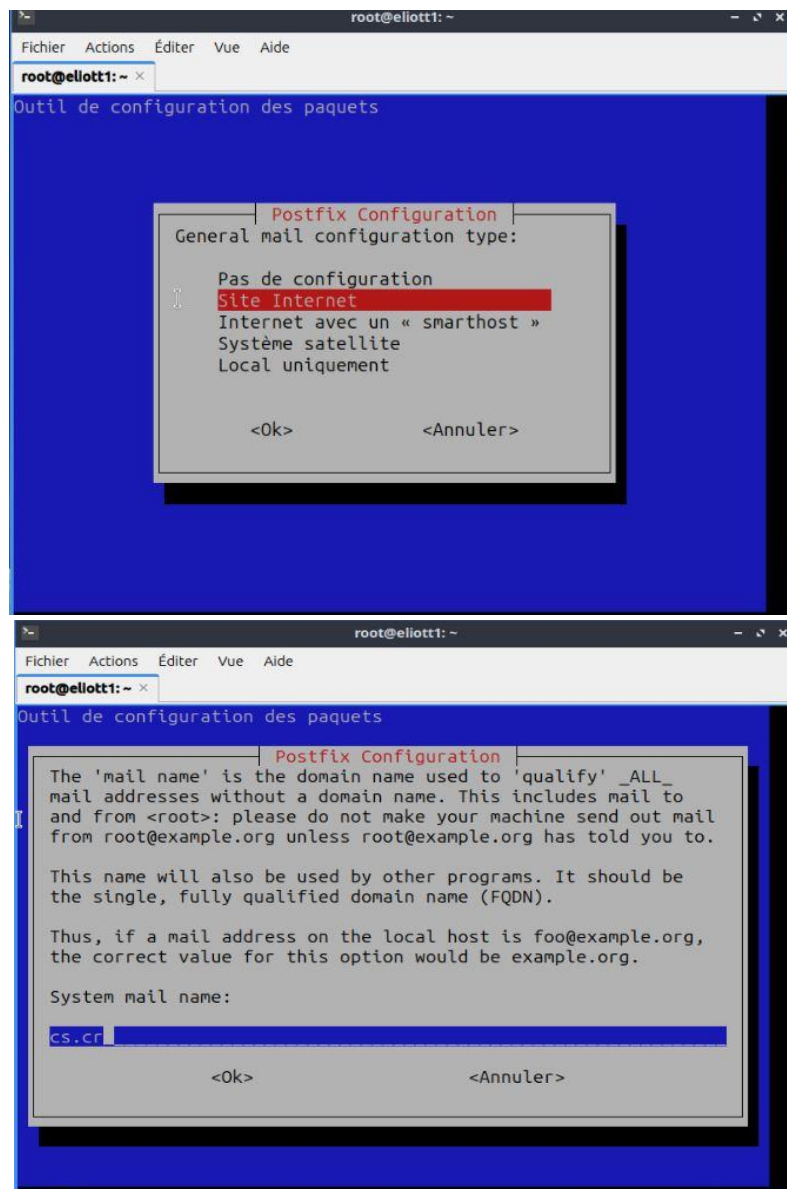
Il est nécessaire de rappeler que les machines virtuelles utilisées ici sont sur un serveur Proxmox et ont été paramétré dans les anciens TP (ex : TP2 pour Pfsense)

Pour mettre en place le serveur de mail, on utilise **Postfix** en configuration "Site Internet". Cela permet au serveur Lubuntu (sur le LAN) d'envoyer et de recevoir des mails pour un domaine donné.

Si vous n'êtes pas en root : `sudo apt install postfix mailutils` (sinon vous pouvez aussi vous logger avec `sudo-i` pour éviter d'utiliser tout le temps `sudo`)

```
root@eliott1:~# sudo apt install postfix mailutils
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Le paquet suivant a été installé automatiquement et n'est
pas :
  libllvm19
Veuillez utiliser « sudo apt autoremove » pour le supprimer.
Les paquets supplémentaires suivants seront installés :
  gssapi-common guile-3.0-libs libgsasl18 libgssglue1
  libgssapi-ldap-2.0-1 libgssapi-nss libgssapi-ocaml libgssapi-openssl
```

On le configure en “Site Internet” avec notre nom de domaine, ici “cs.cr” :



On s’attaque ensuite à la configuration dans le fichier main.cf :

```
root@eliott1:~# nano /etc/postfix/main.cf
```

```
smtp_tls_CApath=/etc/ssl/certs
smtp_tls_security_level=may
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache

smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_unauth_destination
myhostname = eliot1.cs.cr
mydomain = cs.cr
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
mydestination = $myhostname, $mydomain, cs.cr, eliot1, localhost.$mydomain, localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128, 192.168.1.0/24
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
inet_protocols = all
```

^C Aide ^O Écrire ^W Chercher ^K Couper ^T Exécuter ^C Emplacement M-U Annuler
^X Quitter ^R Lire fich. ^_ Remplacer ^U Coller ^J Justifier ^_ Aller ligne M-E Refaire

\$myhostname : le nom d'hôte complet du serveur mail (ex. eliot1.cs.cr ici)

\$mydomain : mon nom de domaine

Les Alias : A ne pas toucher, indique le chemin où l'on peut créer les alias d'un compte utilisateur

\$mynetworks : les réseaux autorisés à relayer des mails via ce serveur, ici le loop (127.0.0.0/8), mon Lan 10.10.10.0 (non affiché sur cette capture) et le réseau "Wan" 192.168.1.0/24

\$inetinterfaces : on laisse en all pour que postfix écoute sur toutes les interfaces.

On enregistre puis on fait un **systemctl restart postfix** pour qu'il prenne en compte les informations.

Systemctl status postfix pour vérifier qu'il est bien **enable**.

Création des boîtes mails et envoi de mails

On va ensuite créer les utilisateurs locaux avec **adduser [NomUser]** pour tester les mails :

```
root@eliott1:~# adduser alice
info: Ajout de l'utilisateur « alice » ...
info: Choix d'un UID/GID dans la plage 1000 à 59999 ...
info: Ajout du nouveau groupe « alice » (1001) ...
info: Ajout du nouvel utilisateur « alice » (1001) avec le groupe « alice » (1001)
info: Création du répertoire personnel « /home/alice » ...
info: Copie des fichiers depuis « /etc/skel » ...
Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mot de passe mis à jour avec succès
Modifier les informations associées à un utilisateur pour alice
Entrer la nouvelle valeur, ou appuyer sur ENTER pour la valeur par défaut
  NOM []: Alice
  Numéro de chambre []: 1
  Téléphone professionnel []:
  Téléphone personnel []:
  Autre []:
```

Les boîtes mails des utilisateurs locaux seront dans le chemin `/var/mail/[NomUser]`

Pour le moment ces boîtes n'existent pas, pour les créer, on va juste utiliser telnet et faire des mails à la main pour les deux utilisateurs créés précédemment : Alice et Bob :

2 Possibilités, soit grâce à mailutils utiliser **mail [NOMUSER]** écrire un message puis **CTRL+D** ou :

```
root@eliott1:~# telnet 127.0.0.1 25
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
220 eliott1.cs.cr ESMTP Postfix (Ubuntu)
```

telnet 127.0.0.1 25 #port 25 pour SMTP

Si la configuration de postfix a bien été effectuée, vous devriez voir : **220 nom de serveur ESMTP Postfix**

Ici on va ecrire ligne par ligne un mail qu'on va envoyer à Alice puis Bob :

EHLO

MAIL FROM:<root@cs.cr>

RCPT TO:<alice@cs.cr>

DATA

Subject: Test

Test

. #Le point sert à terminer le message

QUIT #Permet de sortir de la session SMTP

La boîte mail de vos utilisateurs devrait être créer, en allant dans le dossier
/var/mail/[NOMUSER] vous devriez voir le mail envoyé :

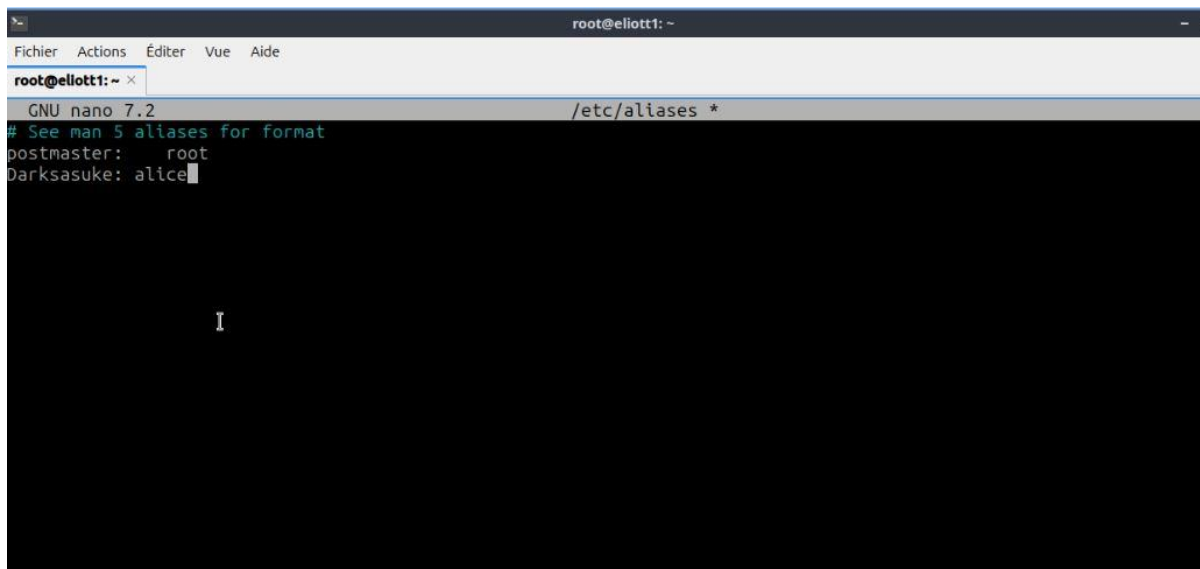
```
root@eliott1:~# cat /var/mail/alice
From root@localhost Thu Dec 4 11:49:47 2025
Return-Path: <root@localhost>
X-Original-To: alice@localhost
Delivered-To: alice@localhost
Received: from localhost (localhost [127.0.0.1])
        by eliott1.cs.cr (Postfix) with ESMTTP id 0ABDB6ABED
        for <alice@localhost>; Thu, 4 Dec 2025 11:45:38 +0100 (CET)
Subject: Test en local
Message-Id: <20251204104807.0ABDB6ABED@eliott1.cs.cr>
Date: Thu, 4 Dec 2025 11:45:38 +0100 (CET)
From: root@localhost

Salut
```

Vous pouvez aussi vous connecter en tant que l'utilisateur **su [NOMUSER]** et taper **mail** pour voir les mail reçus.

Créer un alias

Pour créer un alias il suffit de modifier le fichier qui se trouve dans `/etc/aliases` et de prendre exemple sur le premier (`postmaster: root`) :



```
root@eliott1: ~  
Fichier Actions Éditer Vue Aide  
root@eliott1: ~ x  
GNU nano 7.2 /etc/aliases *  
# See man 5 aliases for format  
postmaster: root  
Darksasuke: alice
```

Et ensuite d'actualiser avec

```
root@eliott1:~# newaliases
```

Ci-dessous un aperçu de `/var/log/mail.log` après avoir envoyer un message à “Darksasuke@cs.cr” :

```
2025-12-04T12:09:21.552057+01:00 eliott1 postfix/local[22094]: 81B116ABF0: to=<alice@cs.cr>, orig_to=<Darksasuke@c  
s.cr>, relay=local, delay=0.03, delays=0.02/0.01/0/0.01, dsn=2.0.0, status=sent (delivered to mailbox)  
2025-12-04T12:09:21.552236+01:00 eliott1 postfix/qmgr[21959]: 81B116ABF0: removed
```

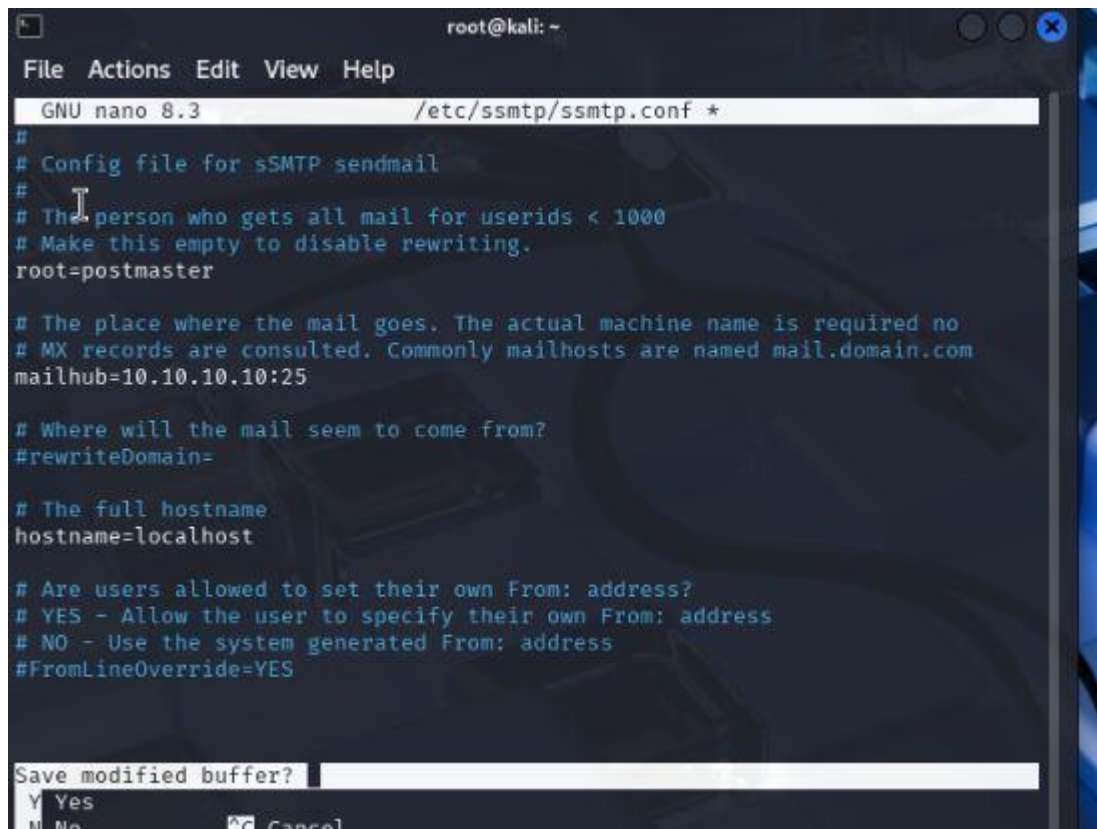
On peut voir qu’il a été envoyé à Alice.

Envoi depuis une autre machine locale (Kali)

A présent on va tenter d’envoyer des mails depuis une autre machine du réseau local, ici une machine kali-linux. Pour cela on va lui installer ssmtp :


```
# apt install ssmtp
Installing:
  ssmtp
```

Puis on va modifier la configuration de la destination des mails pour qu'ils aillent sur le port 25 (SMTP) de notre lubuntu -> ligne mailhub :



```
root@kali: ~
File Actions Edit View Help
GNU nano 8.3 /etc/ssmtp/ssmtp.conf *
#
# Config file for sSMTP sendmail
#
# The person who gets all mail for userids < 1000
# Make this empty to disable rewriting.
root=postmaster

# The place where the mail goes. The actual machine name is required no
# MX records are consulted. Commonly mailhosts are named mail.domain.com
mailhub=10.10.10.10:25

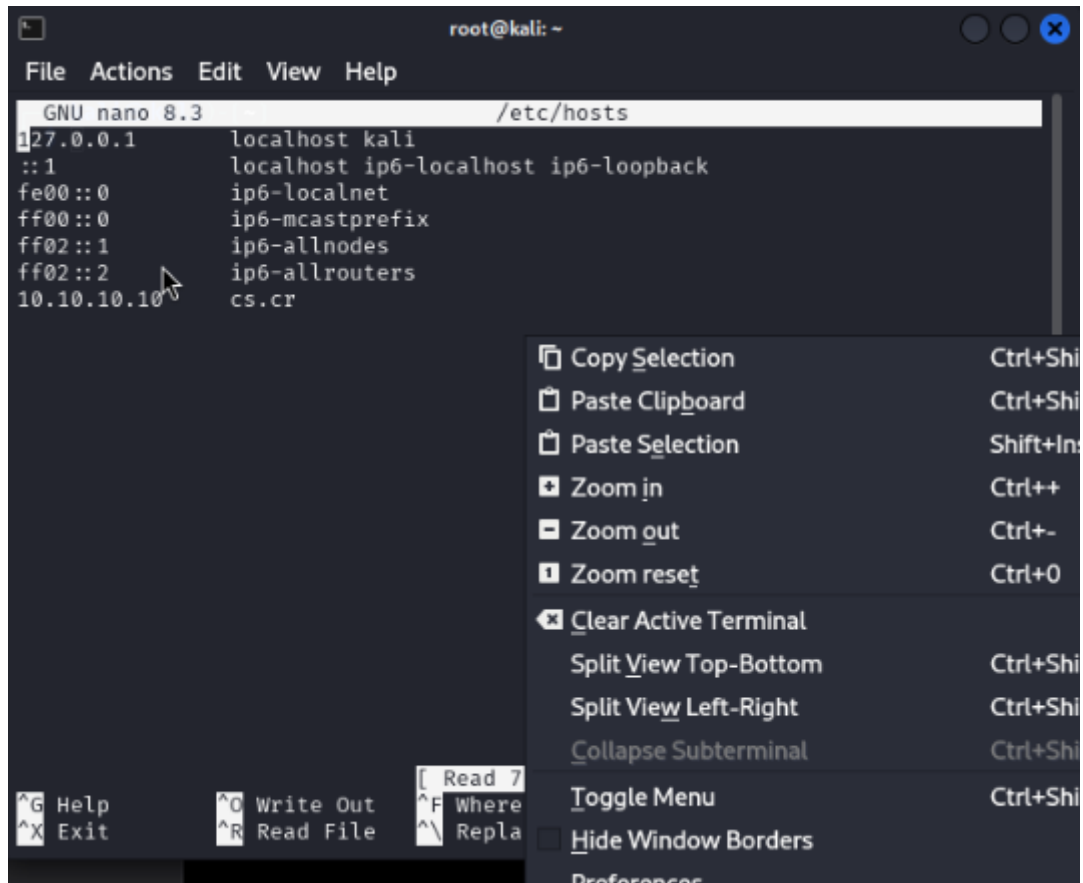
# Where will the mail seem to come from?
#rewriteDomain=

# The full hostname
hostname=localhost

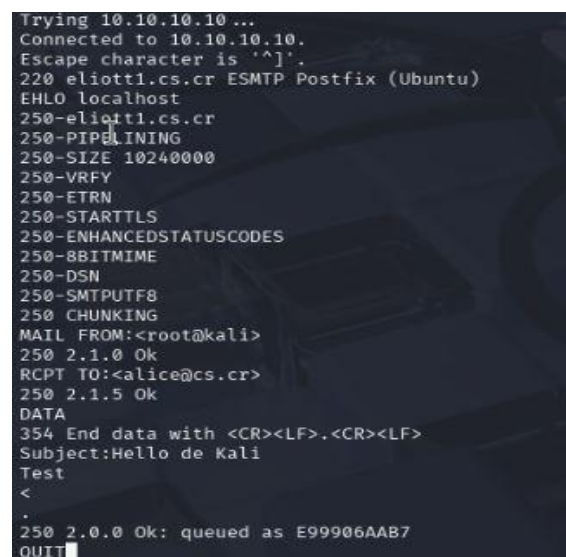
# Are users allowed to set their own From: address?
# YES - Allow the user to specify their own From: address
# NO - Use the system generated From: address
#FromLineOverride=YES

Save modified buffer? [Y] Yes
[ ] No [^C] Cancel
```

Et pour que Kali reconnaisse mieux notre domaine il est sage de rajouter l'IP du domaine dans Hosts :



Ensuite rebelote, on voit si on arrive à se connecter en faisant telnet [IP_serveur] 25 :



On vérifie dans /var/mail/user si le mail à bien été reçu :

```
2025-12-04T13:52:24.871978+01:00 eliott1 postfix/smtpd[22642]: disconnect from kali.cs.cr[10.10.10.50] ehlo=1 mail=1 rcpt=1 data=1 quit=1 commands=5
```

```
From root@kali Thu Dec 4 13:51:43 2025
Return-Path: <root@kali>
X-Original-To: alice@cs.cr
Delivered-To: alice@cs.cr
Received: from localhost (kali.cs.cr [10.10.10.50])
        by eliott1.cs.cr (Postfix) with ESMTP id E99906AAB7
        for <alice@cs.cr>; Thu, 4 Dec 2025 13:50:03 +0100 (CET)
Subject: Hello de Kali

Test
<
```

Installation Dovecot et accès IMAP sécurisé (SSL/TLS)

Pour pouvoir consulter ses mails d'un accès distant, il va nous falloir un serveur IMAP, on va utiliser ici Dovecot. Pour l'installer (en root) apt install dovecot-core dovecot-imapd.

Ensuite un petit systemctl status dovecot :

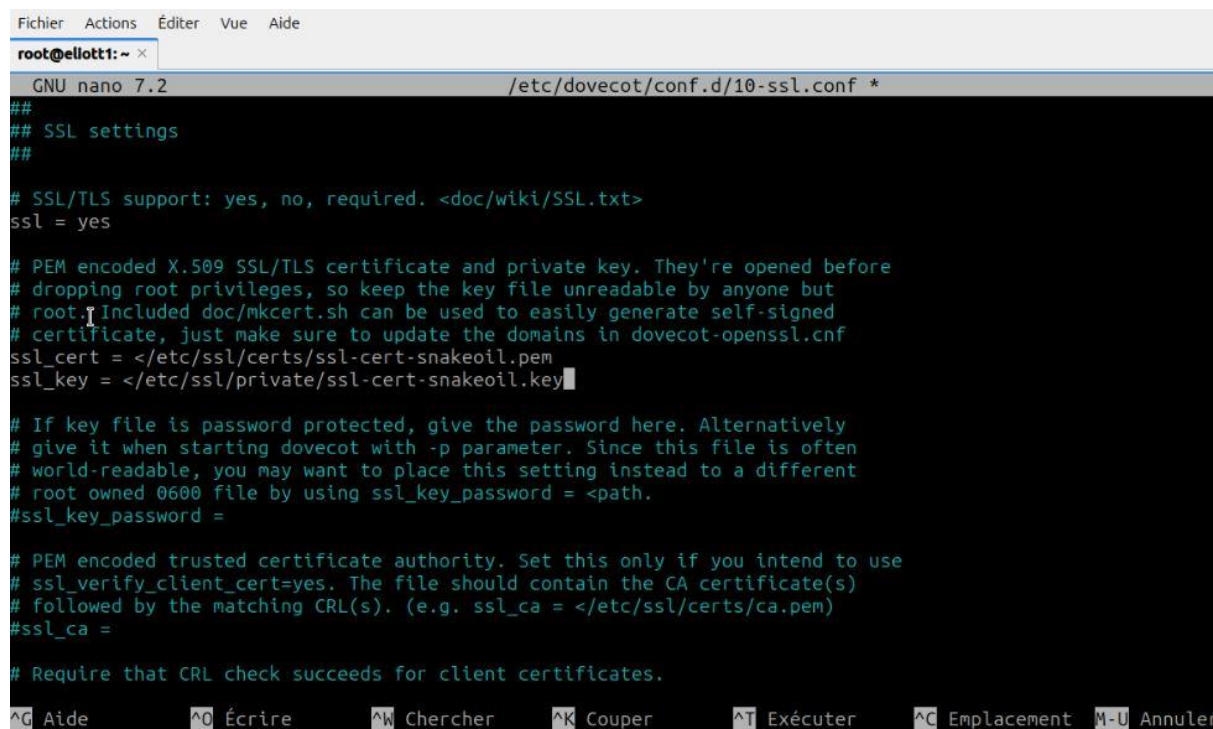
```
root@eliott1:~# systemctl status dovecot
● dovecot.service - Dovecot IMAP/POP3 email server
   Loaded: loaded (/usr/lib/systemd/system/dovecot.service; enabled; preset: enabled)
   Active: active (running) since Thu 2025-12-04 13:58:58 CET; 24s ago
     Docs: man:dovecot(1)
           https://doc.dovecot.org/
  Main PID: 25907 (dovecot)
    Status: "v2.3.21 (47349e2482) running"
     Tasks: 4 (limit: 4590)
    Memory: 3.2M (peak: 3.4M)
       CPU: 24ms
    CGroup: /system.slice/dovecot.service
            └─25907 /usr/sbin/dovecot -F
              └─25908 dovecot/anvil
                └─25909 dovecot/log
                  └─25910 dovecot/config

déc. 04 13:58:58 eliott1 systemd[1]: Starting dovecot.service - Dovecot IMAP/POP3 email server...
déc. 04 13:58:58 eliott1 dovecot[25907]: master: Dovecot v2.3.21 (47349e2482) starting up for imap (core
déc. 04 13:58:58 eliott1 systemd[1]: Started dovecot.service - Dovecot IMAP/POP3 email server.
lines 1-19/19 (END)
```

Voilà c'est bon dovecot écoute par défaut sur le port 143.

Sécurisation par TLS

Pour sécuriser l'IMAP, on utilise STARTTLS sur le port 143 pour le sécuriser (ou IMAPS sur le port 993). Dovecot fournit un certificat auto-signé « snakeoil » à l'installation. On vérifie juste s'il est bien pointé dans `/etc/dovecot/conf.d/10-ssl.conf` :



```
Fichier Actions Éditer Vue Aide
root@elliott1: ~ x
GNU nano 7.2 /etc/dovecot/conf.d/10-ssl.conf *
##
## SSL settings
##
# SSL/TLS support: yes, no, required. <doc/wiki/SSL.txt>
ssl = yes

# PEM encoded X.509 SSL/TLS certificate and private key. They're opened before
# dropping root privileges, so keep the key file unreadable by anyone but
# root. Included doc/mkcert.sh can be used to easily generate self-signed
# certificate, just make sure to update the domains in dovecot-openssl.cnf
ssl_cert = </etc/ssl/certs/ssl-cert-snakeoil.pem
ssl_key = </etc/ssl/private/ssl-cert-snakeoil.key

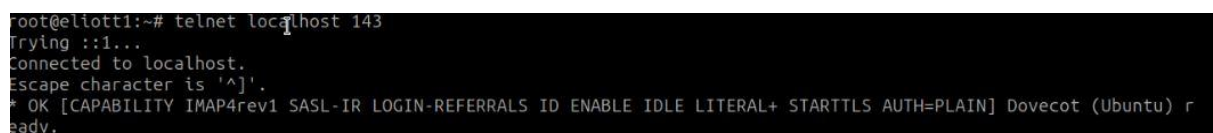
# If key file is password protected, give the password here. Alternatively
# give it when starting dovecot with -p parameter. Since this file is often
# world-readable, you may want to place this setting instead to a different
# root owned 0600 file by using ssl_key_password = <path>.
#ssl_key_password =

# PEM encoded trusted certificate authority. Set this only if you intend to use
# ssl_verify_client_cert=yes. The file should contain the CA certificate(s)
# followed by the matching CRL(s). (e.g. ssl_ca = </etc/ssl/certs/ca.pem)
#ssl_ca =

# Require that CRL check succeeds for client certificates.
```

Ensuite on redémarre Dovecot (`systemctl restart dovecot`), et on va tester dovecot localement.

On réutilise telnet mais cette fois sur le port d'écoute de Dovecot, le 143 :



```
root@elliott1:~# telnet localhost 143
Trying ::1...
Connected to localhost.
Escape character is '^['.
* OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LITERAL+ STARTTLS AUTH=PLAIN] Dovecot (Ubuntu) ready.
```

Pour sortir “a LOGOUT”

Si ce texte ci-dessus apparaît, dovecote est opérationnel en clair, maintenant passons en TLS avec cette commande : `openssl s_client -connect localhost:143 -starttls imap`

```

root@eliott1:~# openssl s_client -connect localhost:143 -starttls imap
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = eliott1
verify return:1
---
Certificate chain
 0 s:CN = eliott1
  i:CN = eliott1
  a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
  v:NotBefore: Nov 18 08:55:04 2025 GMT; NotAfter: Nov 16 08:55:04 2035 GMT

```

Vous verrez le handshake TLS puis vous pourrez taper des commandes IMAP dans ce tunnel chiffré. Par exemple, après le handshake, entrez :

a LOGIN [NOMUSER] [MDPUSER] pour tester une connexion de votre utilisateur.

```

a LOGIN alice root
* CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD=REFS THRE
AD=ORDEREDSUBJECT MULTIAPPEND URL-PARTIAL CATENATE UNSELECT CHILDREN NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 C
ONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS BINARY MOVE SNIPPET=FUZZY PREVIEW=FUZZY
PREVIEW STATUS=SIZE SAVEDATE LITERAL+ NOTIFY SPECIAL-USE
a OK Logged in

```

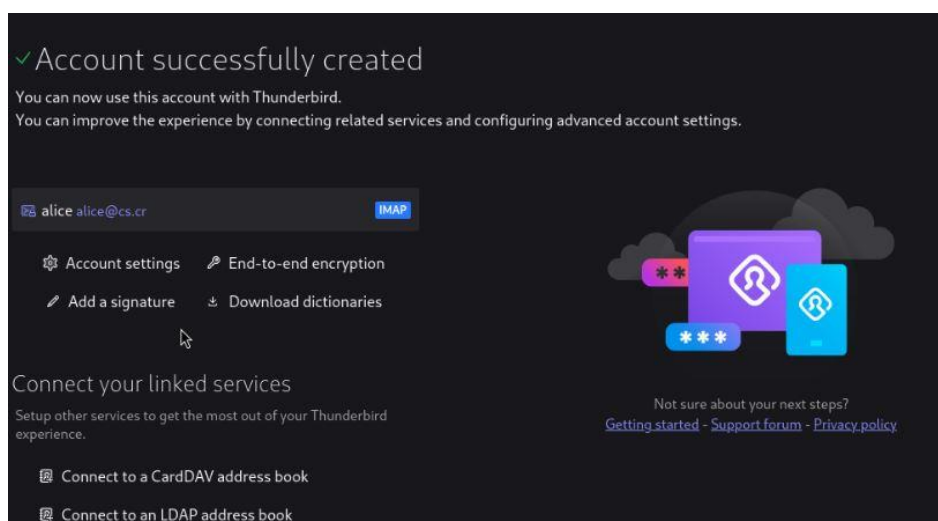
Configuration de Thunderbird (optionnel)

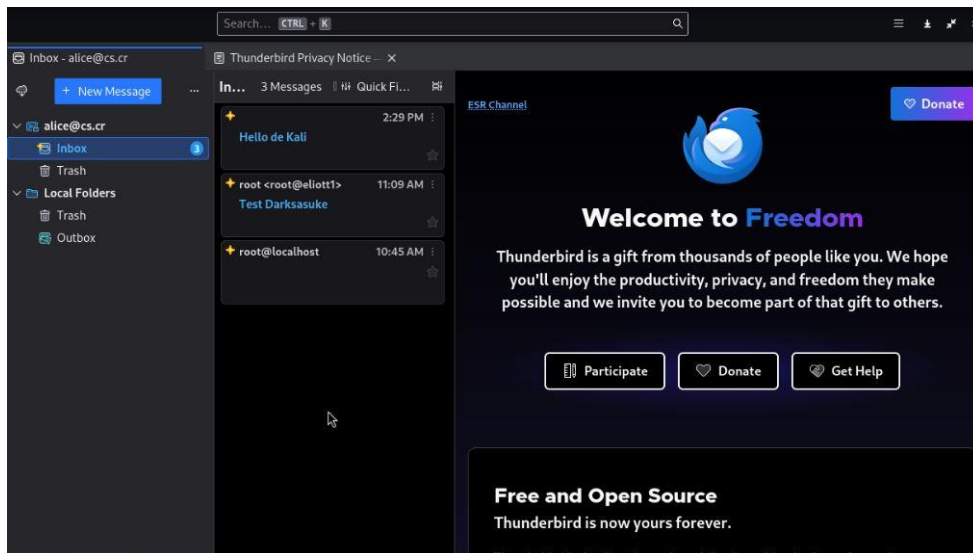
Pour avoir un client mail graphique et consulter nos mails avec plus de confort nous allons installer thunderbird sur notre machine kali.

Pour cela (toujours en root) : apt install thunderbird

(thunderbird & pour ouvrir thunderbird et garder le terminal disponible).

Configurons donc le compte mail d'Alice par exemple :





Accès depuis le WAN et tunnelisation via SSH

Accès depuis le WAN

Maintenant le but c'est que notre messagerie soit accessible depuis n'importe quel réseau et pour ça il va falloir revenir sur notre serveur Pfsense (cf : TP2)

On va exposer notre serveur SMTP (port 25) au WAN, ce n'est pas une bonne solution en règle générale on risque du spam, tout le monde peut nous envoyer des mails. Il faut vraiment que dans le `$mynetwork` du postfix soit renseigné que nos réseaux voulus ainsi que dans `$mydestination` seulement notre domaine.

Donc dans Pfsense : Firewall / NAT / Port Forward on va rajouter la règle du port SMTP :

Firewall / NAT / Port Forward / Edit

Edit Redirect Entry

Disabled ☐ Disable this rule

No RDR (NOT) ☐ Disable redirection for traffic matching this rule
This option is rarely needed. Don't use this without thorough knowledge of the implications.

Interface WAN
Choose which interface this rule applies to. In most cases "WAN" is specified.

Address Family IPv4
Select the Internet Protocol version this rule applies to.

Protocol TCP
Choose which protocol this rule should match. In most cases "TCP" is specified.

Source Display Advanced

Destination ☐ Invert match. WAN address Type Address/mask

Destination port range SMTP From port Custom SMTP To port Custom
Specify the port or port range for the destination of the mapping. The 'to' field may be left empty if only mapping a single port.

Redirect target IP Address or Alias Type 10.10.10.10 Address
Enter the internal IP address of the server on which to map the ports. e.g.: 192.168.1.12 for IPv4
In case of IPv6 addresses, in must be from the same "scope",
i.e. it is not possible to redirect from link-local addresses scope (fe80:*) to local scope (::1)

Redirect target port SMTP Port Custom
Specify the port on the machine with the IP address entered above. In case of a port range, specify the beginning port of the range (the end port will be calculated automatically)

Activer V
Bordereaux

On teste depuis un autre réseau si la connexion passe en SMTP (ici teste sur powershell) :

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\depra> Test-NetConnection 192.168.1.130 -Port 25










ComputerName      : 192.168.1.130
RemoteAddress     : 192.168.1.130
RemotePort        : 25
InterfaceAlias    : Ethernet
SourceAddress     : 192.168.1.33
TcpTestSucceeded  : True







PS C:\Users\depra>
```

Puis plutôt qu'exposer notre port IMAP(143) aussi, on va plutôt tout de suite faire un port forward pour le port 22 (SSH) dans le but de tunneliser nos échanges IMAP + STARTTLS dans une connexion SSH. On évite une trop grosse exposition et des attaques (brut force etc) faciles.

Tunnelisation IMAP+STARTTLS via SSH

Dans un premier temps, comme avec le port SMTP, on crée une règle pour le port 22 (SSH) :

Rules										
<input type="checkbox"/>	Interface	Protocol	Source Address	Source Ports	Dest. Address	Dest. Ports	NAT IP	NAT Ports	Description	Actions
<input type="checkbox"/>	<input checked="" type="checkbox"/> WAN	TCP	*	*	WAN address	22 (SSH)	10.10.10.10	22 (SSH)	SSH/IMAP	  
<input type="checkbox"/>	<input checked="" type="checkbox"/> WAN	TCP	*	*	WAN address	1443	10.10.10.10	443 (HTTPS)	Apache2	  
<input type="checkbox"/>	<input checked="" type="checkbox"/> WAN	TCP	*	*	WAN address	25 (SMTP)	10.10.10.10	25 (SMTP)	Smtpt	  

 Add  Add  Delete  Toggle  Save  Separator

Ah et si votre Pfsense est dans le même réseau que votre machine distante cochez le NAT reflection dans System > Advanced > Firewall & NAT; car Pfsense galère sinon :

Network Address Translation

NAT Reflection mode for port forwards

Pure NAT

- The Pure NAT mode uses a set of NAT rules to direct packets to the target of the port forward. It has better scalability, but it must be possible to accurately determine the interface and gateway IP used for communication with the target at the time the rules are loaded. There are no inherent limits to the number of ports other than the limits of the protocols. All protocols available for port forwards are supported.
- The NAT + Proxy mode uses a helper program to send packets to the target of the port forward. It is useful in setups where the interface and/or gateway IP used for communication with the target cannot be accurately determined at the time the rules are loaded. Reflection rules are not created for ranges larger than 500 ports and will not be used for more than 1000 ports total between all port forwards. This feature does not support IPv6. Only TCP and UDP protocols are supported.

Individual rules may be configured to override this system setting on a per-rule basis.

Reflection Timeout

2000

Enter value for Reflection timeout in seconds.
Note: Only applies to Reflection on port forwards in NAT + proxy mode.

Enable NAT Reflection for 1:1 NAT

☒ Automatic creation of additional NAT redirect rules from within the internal networks.
Note: Reflection on 1:1 mappings is only for the inbound component of the 1:1 mappings. This functions the same as the pure NAT mode for port forwards. For more details, refer to the pure NAT mode description above. Individual rules may be configured to override this system setting on a per-rule basis.

Enable automatic outbound NAT for Reflection

☒ Automatic create outbound NAT rules that direct traffic back out to the same subnet it originated from.
Required for full functionality of the pure NAT mode of NAT Reflection for port forwards or NAT Reflection for 1:1 NAT. Note: This only works for assigned interfaces. Other interfaces require manually creating the outbound NAT rules that direct the reply packets back through the router.

TFTP Proxy

WAN
LAN

Ensuite sur la machine Postfix, on va installer un serveur SSH et l'activer :

apt install open ssh-server

puis 

Désactivez ufw s'il pose trop de problèmes

Ensuite on va ouvrir le tunnel via la machine distante avec la commande suivante

```
Ssh -L XXXX:localhost:143 [NomUser]@[IPSERVERMAIL]
```

-L = permet le transfert de port

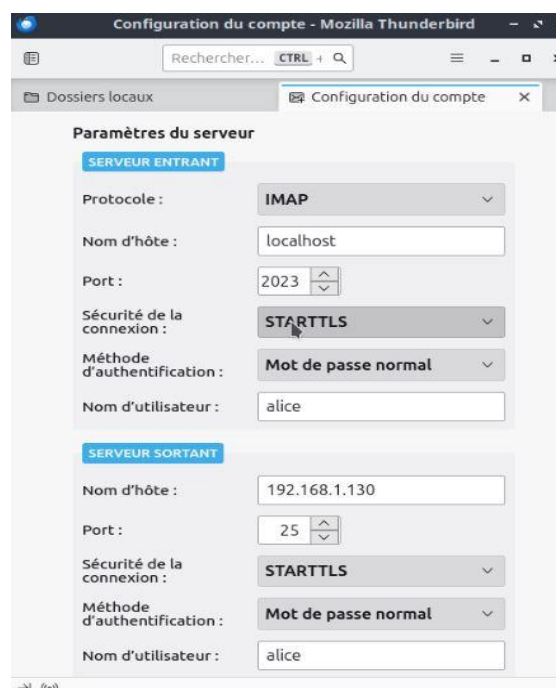
XXXX = port utilisé par la machine locale, ici 2023

Tout ce qui s'y connecte doit être redirigé sur localhost port 143

Désactivez ufw sur le serveur mail s'il pose trop de problèmes comme Connection Timed out.

```
root@bob-standardpc:~# ssh -L 2023:localhost:143 alice@10.10.10.10
ssh: connect to host 10.10.10.10 port 22: Connection timed out
root@bob-standardpc:~# ssh -L 2023:localhost:143 alice@192.168.1.130
ssh: connect to host 192.168.1.130 port 22: Connection timed out
root@bob-standardpc:~# ssh -L 2023:localhost:143 alice@192.168.1.130
The authenticity of host '192.168.1.130 (192.168.1.130)' can't be established.
ED25519 key fingerprint is SHA256:9cA4xtRgfP7mg2NVrnw/LLEY1q3QCuDRPoR6PErvUJo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? █
```

À la suite de cette connexion on va pouvoir paramétrer par exemple thunderbird pour s'envoyer des mails :



Sur wireshark ça donne ça :

The image shows a Wireshark packet capture of an SSH session. The packet list on the left shows multiple packets, with packet 534 selected. The packet details pane on the right shows the structure of packet 534: Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and SSH Protocol. The packet bytes pane shows the raw data of the packet, which is an encrypted SSH packet.

No.	Time	Source	Destination
519	80.546561272	192.168.1.130	192.168.1.138
520	80.564706692	192.168.1.138	192.168.1.130
521	80.566071855	192.168.1.130	192.168.1.138
522	80.584239256	192.168.1.138	192.168.1.130
523	80.584825769	192.168.1.130	192.168.1.138
524	80.603709671	192.168.1.138	192.168.1.130
525	80.604995438	192.168.1.130	192.168.1.138
526	80.623756402	192.168.1.138	192.168.1.130
527	80.624320811	192.168.1.130	192.168.1.138
528	80.642333271	192.168.1.138	192.168.1.130
529	80.642857306	192.168.1.130	192.168.1.138
530	80.661273042	192.168.1.138	192.168.1.130
531	80.661902511	192.168.1.130	192.168.1.138
532	80.680730595	192.168.1.138	192.168.1.130
533	80.682001766	192.168.1.130	192.168.1.138
534	80.700557217	192.168.1.138	192.168.1.130
535	80.701098354	192.168.1.130	192.168.1.138
536	80.720811149	192.168.1.138	192.168.1.130
537	80.721554941	192.168.1.130	192.168.1.138
538	80.739453260	192.168.1.138	192.168.1.130

Frame 534: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface ens18, id 0
Ethernet II, Src: ProxmoxServe_72:ce:9c (bc:24:11:72:ce:9c), Dst: ProxmoxServe_43:56:1c (bc:24:11:43:56:1c)
Internet Protocol Version 4, Src: 192.168.1.138, Dst: 192.168.1.130
Transmission Control Protocol, Src Port: 52052, Dst Port: 22, Seq: 7273, Ack: 7477, Len: 36
SSH Protocol

No.: 534 - Time: 80.700557217 - Source: 192.168.1.138 - Destination: 192.168.1.130 - Protocol: SSH - Length: 102 - Info: Client: Encrypted packet (len=36)

Afficher les octets du paquet

Mon Client distant 192.168.1.138 -- Mon Serveur 192.168.1.130

Pas de trace du port 143, seulement du 22.

Maintenant qu'on a sécurisé les connexions à distance en évitant la surexposition du port 143, si on s'occupait de la cryptographie des mails !

Utilisation de GnuPG pour chiffrer et signer les mails

Création des clefs (1)

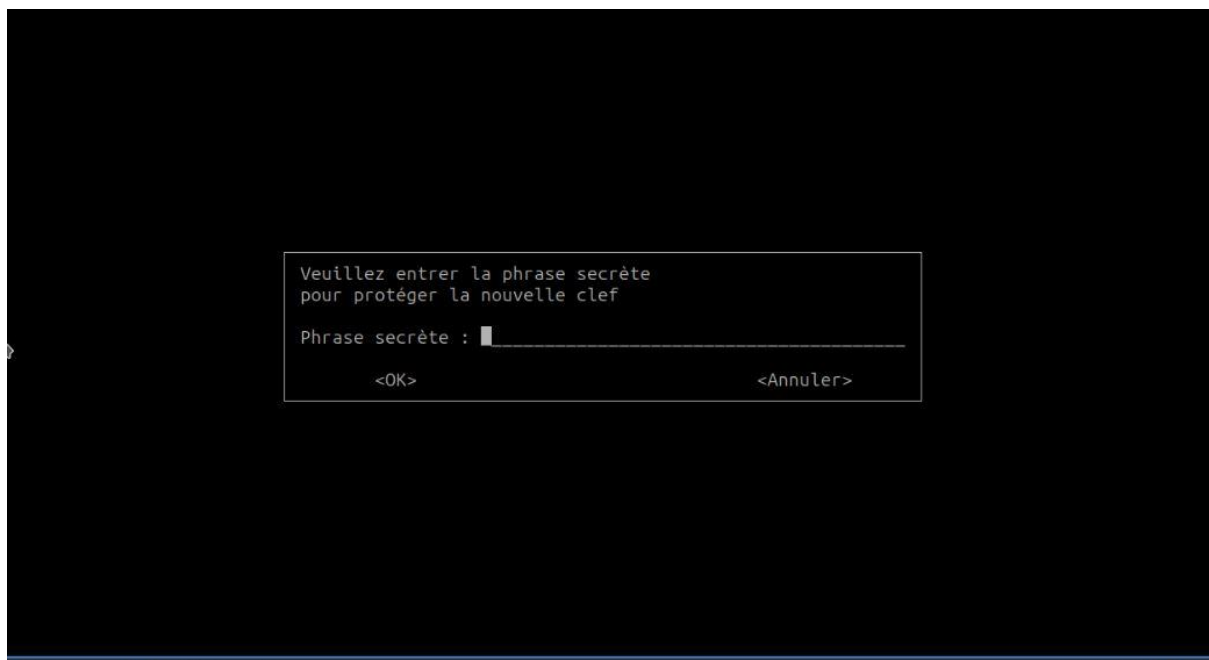
L'objectif est de permettre à deux utilisateurs (Alice et Bob) d'échanger des messages chiffrés et signés, garantissant confidentialité, intégrité et authenticité.

Dans un premier temps on va générer les clefs des deux utilisateurs. Pour les créer assurez-vous que l'utilisateur qui va générer sa clef soit propriétaire du terminal avec `chown [nom_user] $(tty)` et `ls -la $(tty)`.

```
liotte@eliotti:~$ su alice
Mot de passe :
alice@eliotti:/home/liotte$ cd
alice@eliotti:~$ gpg --full-generate-key
gpg (GnuPG) 2.4.4; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: répertoire « /home/alice/.gnupg » créé
gpg: le trousseau local « /home/alice/.gnupg/pubring.kbx » a été créé
Sélectionnez le type de clef désiré :
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (signature seule)
 (14) Existing key from card
Quel est votre choix ? 1
Les clefs RSA peuvent faire une taille comprise entre 1024 et 4096 bits.
Quelle taille de clef désirez-vous ? (3072) 2048
La taille demandée est 2048 bits
Veuillez indiquer le temps pendant lequel cette clef devrait être valable.
  0 = la clef n'expire pas
  <n> = la clef expire dans n jours
  <n>w = la clef expire dans n semaines
  <n>m = la clef expire dans n mois
  <n>y = la clef expire dans n ans
Pendant combien de temps la clef est-elle valable ? (0) 0
```

Saisissez RSA and RSA; la taille de clef que vous désirez puis le délai d'expiration de la clef



Saisissez ensuite la phrase secrète et bien sûr, souvenez-vous en.

On va aussi créer un certificat de révocation en cas de perte, compromission de clef, utilisez la commande

gpg --output nom_certificat_a_definir.asc --gen-revoke [nomUser]@[domaine]

Ceci crée un fichier ASCII permettant d'invalider la clef si nécessaire. Le mieux c'est de le conserver hors ligne.

Export des clefs publics (2)

Maintenant nos deux utilisateurs doivent exporter leur clef publique pour se les échanger.

On utilise la commande suivante :

```
bob@eliott1:~$ gpg --export -a "bob@cs.cr" > bob_public.asc
bob@eliott1:~$ ls
bob_public.asc
bob@eliott1:~$ cat bob_public.asc
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBGkyxd4BCADVMQ1egTVM5Y2n+YWBDm39iVuWs7kyDV+RUgaCXvIHRWEf1Nkp
g+PrHFfrPD/E5Ug9GaRfkoINEYi+FR9nYZW7xu5+2bCj+Ov1ALJDegHO/gyaiHlX
Wj8Ep4EXPGtA1d9cFyMBBq7blml+cCM5iNvnBGi5BqdEFyk5mSJT9K+M8anoMFWX
8J8n0kPPA6yZ3Zfy5X4cm/mUFm6HwLbveXgfNd65n14QpXp2LcdS9ts/7JKENv4o
```

gpg --export -a "[NOMUSER]@[DOMAINE]" > [NOM_CLEF_PUBLIC_A_DEFINIR].asc

Ensuite on se les échange via thunderbird en pièce jointe.

Importation des clefs (3)

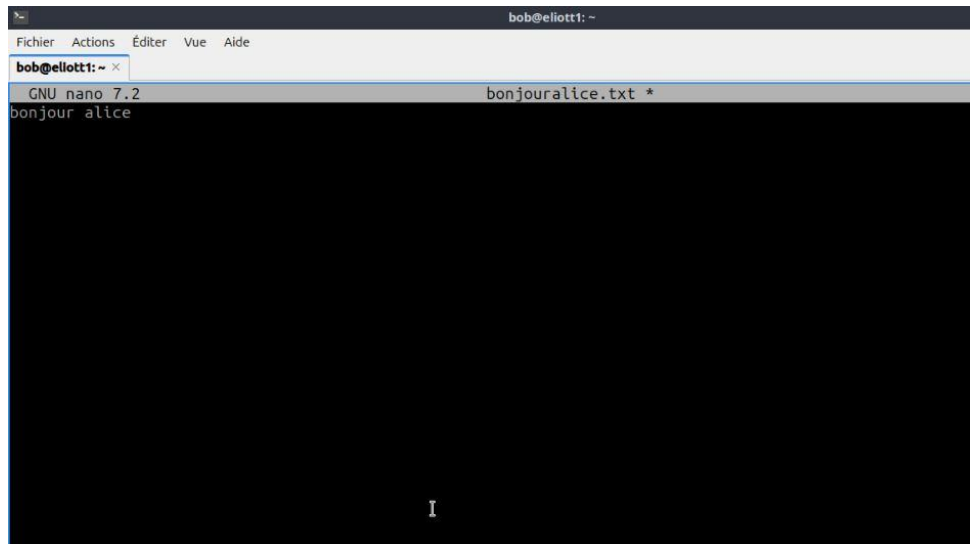
Une fois en possession de la clef publique de l'autre, il faut l'importer avec la commande **gpg --import [Nomdeclef].asc**

```
alice@eliott1:~$ gpg --import bob_public.asc
gpg: clef 5CBD6FC90144F779 : clef publique « bob <bob@cs.cr> » importée
gpg: Quantité totale traitée : 1
gpg:      importées : 1
alice@eliott1:~$
```

Puis chacun peut vérifier leur trousseau avec **gpg --list-key** et **gpg --list-secret-key** (1 pour les clefs publiques et l'autres pour les privées).

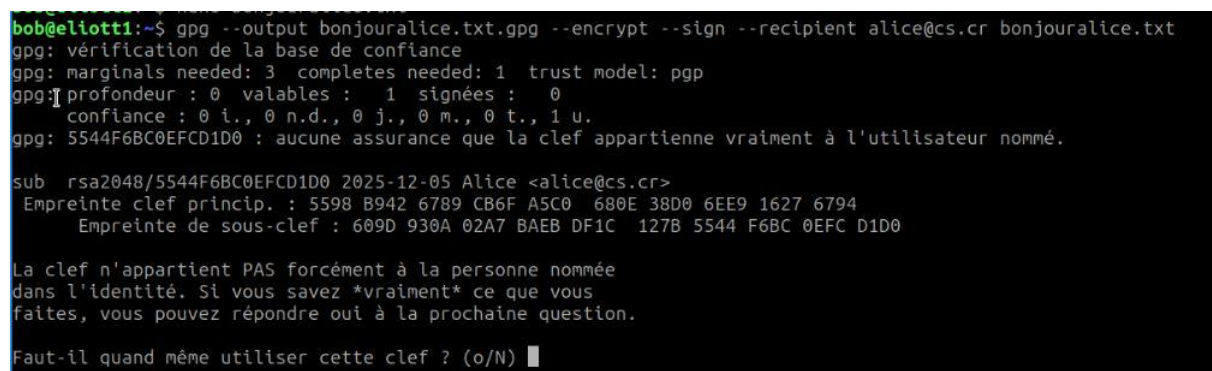
Envoi d'un message chiffré et déchiffrement (4)

Bob va donc vouloir envoyer un message chiffré à Alice :



```
bob@eliott1: ~  
Fichier Actions Éditer Vue Aide  
bob@eliott1: ~ x  
GNU nano 7.2 bonjouralice.txt *  
bonjour alice  
I
```

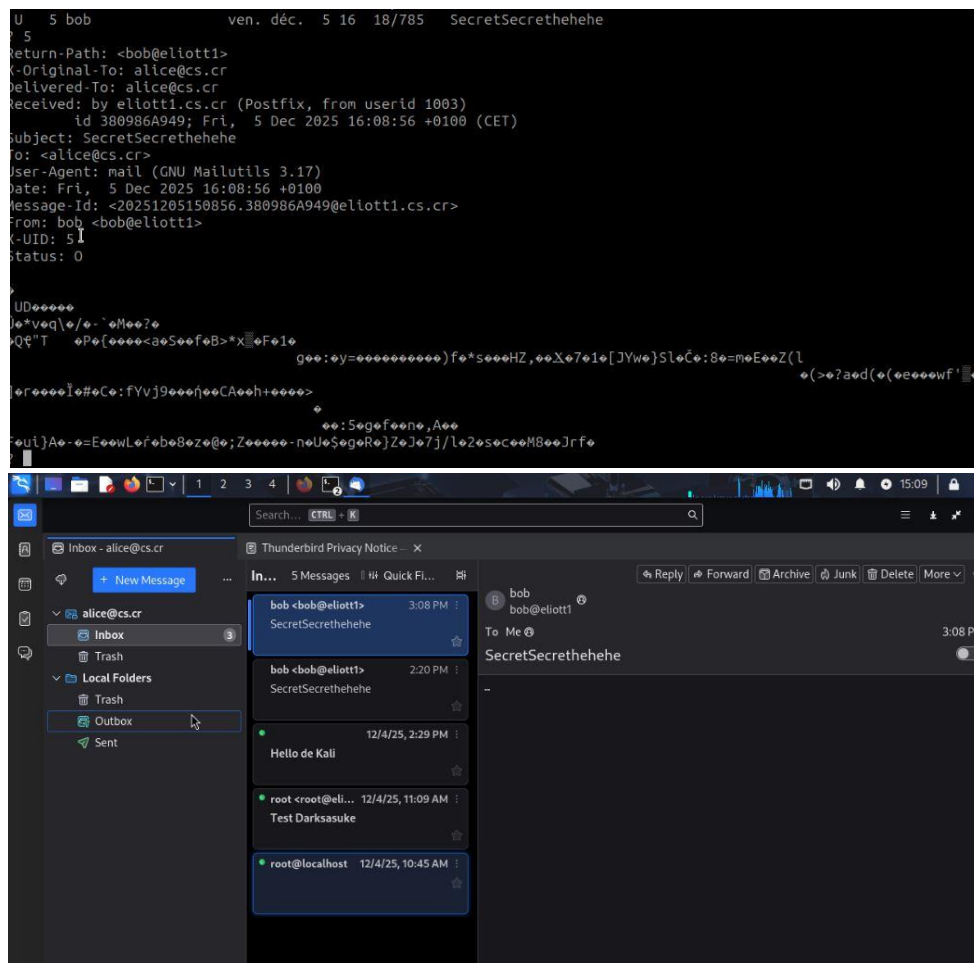
Pour cela il va devoir chiffrer le message avec la clef publique d'Alice :



```
bob@eliott1:~$ gpg --output bonjouralice.txt.gpg --encrypt --sign --recipient alice@cs.cr bonjouralice.txt  
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: 5544F6BC0EFC1D0 : aucune assurance que la clef appartienne vraiment à l'utilisateur nommé.  
  
sub rsa2048/5544F6BC0EFC1D0 2025-12-05 Alice <alice@cs.cr>  
Empreinte clef princip. : 5598 B942 6789 CB6F A5C0 680E 38D0 6EE9 1627 6794  
Empreinte de sous-clef : 609D 930A 02A7 BAEB DF1C 127B 5544 F6BC 0EFC D1D0  
  
La clef n'appartient PAS forcément à la personne nommée  
dans l'identité. Si vous savez *vraiment* ce que vous  
faites, vous pouvez répondre oui à la prochaine question.  
Faut-il quand même utiliser cette clef ? (o/N) █
```

On voit un message de prévention, effectivement la clef n'est pas signée et donc nous ne sommes pas sûr de la provenance de la clef.

Alice quand elle reçoit le mail, reçoit ça :



Il va falloir qu'elle déchiffre le message :

```

alice@eliott1:~$ gpg --output msg.txt --decrypt bonjouralice.txt.gpg
gpg: encrypted with rsa2048 key, ID 5544F6BC0EFCDD1D0, created 2025-12-05
« Alice <alice@cs.cr> »
gpg: Signature faite le ven. 05 déc. 2025 15:07:54 CET
gpg:          avec la clef RSA 1FAFAA6BE3E0CD9251A79D1B5CBD6FC90144F779
gpg: vérification de la base de confiance
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: profondeur : 0  valables : 1  signées : 0
gpg: confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
gpg: Bonne signature de « bob <bob@cs.cr> » [inconnu]
gpg: Attention : cette clef n'est pas certifiée avec une signature de confiance.
gpg:          Rien n'indique que la signature appartient à son propriétaire.
gpg: Empreinte de clef principale : 1FAF AA6B E3E0 CD92 51A7 9D1B 5CBD 6FC9 0144 F779
alice@eliott1:~$

```

Le message devient visible. Cependant pour rassurer bob, elle souhaite signer son prochain message.

```
Fichier Actions Éditer Vue Aide
alice@eliott1:~ x
GNU nano 7.2 msg.txt
bonjour alice

I
```

Signature de clef (5)

Avec la commande `gpg --clearsign [Nomfichier]`, Alice va pouvoir signer son fichier et l'envoyer à Bob :

```
root@eliott1:~# chown alice $(tty) I
root@eliott1:~# ls -la $(tty)
crw----- 1 alice tty 136, 1 déc.  6 12:50 /dev/pts/1
root@eliott1:~# su alice
alice@eliott1:/root$ cd
alice@eliott1:~$ gpg --clearsign signer.txt
alice@eliott1:~$
```

Elle se retrouve donc avec un fichier `signer.txt.asc` :

```
alice@eliott1:~$ ls
alice_public.asc bob_public.asc bonjouralice.txt.gpg mail mbox msg.txt signer.txt signer.txt.asc
alice@eliott1:~$
```

Une fois envoyé à Bob, Bob va vouloir vérifier la signature d'Alice, il va utiliser `--verify` :

```
bob@eliott1:~$ gpg --verify signer.txt.asc
gpg: Signature faite le sam. 06 déc. 2025 12:50:47 CET
gpg:      avec la clef RSA 5598B9426789CB6FA5C0680E38D06EE916276794
gpg: Bonne signature de « Alice <alice@cs.cr> » [inconnu]
gpg: Attention : cette clef n'est pas certifiée avec une signature de confiance.
gpg:      Rien n'indique que la signature appartient à son propriétaire.
Empreinte de clef principale : 5598 B942 6789 CB6F A5C0 680E 38D0 6EE9 1627 6794
bob@eliott1:~$
```

Comme avant on reçoit le message inquiétant : La signature n'appartient peut être pas à son propriétaire.

Mais Bob lui, il est sûr que ça appartient à Alice, il va donc signer la clef publique d'Alice pour lui renvoyer :

```
bob@eliott1:~$ gpg --sign-key alice@cs.cr

pub rsa2048/38D06EE916276794
   créé : 2025-12-05  expire : jamais      utilisation : SC
   confiance : inconnu      validité : inconnu
sub rsa2048/5544F6BC0EFCDD1D0
   créé : 2025-12-05  expire : jamais      utilisation : E
[ inconnue ] (1). Alice <alice@cs.cr>

pub rsa2048/38D06EE916276794
   créé : 2025-12-05  expire : jamais      utilisation : SC
   confiance : inconnu      validité : inconnu
Empreinte clef princip. : 5598 B942 6789 CB6F A5C0  680E 38D0 6EE9 1627 6794

    Alice <alice@cs.cr>

Voulez-vous vraiment signer cette clef avec votre
clef « bob <bob@cs.cr> » (5CBD6FC90144F779)

Voulez-vous vraiment signer ? (o/N) o

Options controlling key import and export:
  --auto-key-locate MÉCANISMES      utiliser les MÉCANISMES pour localiser les clefs
  --auto-key-import                  import missing key from a signature
  --include-key-block                include the public key in signatures
  --disable-dirmngr                  désactiver tous les accès au dirmngr

Options to specify keys:
  -r, --recipient IDENTITÉ          chiffrer pour l'IDENTITÉ
  -u, --local-user IDENTITÉ          utiliser l'IDENTITÉ pour signer ou déchiffrer

(Consultez la page de manuel pour obtenir une liste complète des commandes
et options)

Exemples:
  -se -r Bob [file]                  sign and encrypt for user Bob
  --clear-sign [file]                make a clear text signature
  --detach-sign [file]               make a detached signature
  --list-keys [names]                show keys
  --fingerprint [names]              show fingerprints

Veuillez signaler toutes anomalies sur <https://bugs.gnupg.org> (en anglais)
et tout problème de traduction à <traduc@traduc.org>.
bob@eliott1:~$ gpg --export -a alice@cs.cr > alice_public_signee_bob.asc
bob@eliott1:~$ ls
alice_public.asc          bob_public.asc      bonjouralice.txt.gpg
alice_public_signee_bob.asc  bonjouralice.txt  signer.txt.asc
bob@eliott1:~$
```

Et juste avant il fait à nouveau -- verify sur signer.txt.asc et cette fois pas de message douteux :

```
bob@eliott1:~$ gpg --verify signer.txt.asc
gpg: Signature faite le sam. 06 déc. 2025 12:50:47 CET
gpg:          avec la clef RSA 5598B9426789CB6FA5C0680E38D06EE916276794
gpg: vérification de la base de confiance
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: profondeur : 0  valables : 1  signées : 1
   confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
gpg: profondeur : 1  valables : 1  signées : 0
   confiance : 1 i., 0 n.d., 0 j., 0 m., 0 t., 0 u.
gpg: Bonne signature de « Alice <alice@cs.cr> » [totale]
bob@eliott1:~$
```

Renvoi d'un message avec clef signée (6)

Alice va à nouveau chiffrer un message pour l'envoyer à Bob, mais cette fois avec sa clef signée par bob :

```
root@eliott1:~# chown alice $(tty)
root@eliott1:~# su alice
alice@eliott1:/root$ cd
alice@eliott1:~$ nano truc.txt
alice@eliott1:~$ gpg --encrypt -a -r bob@cs.cr truc.txt
gpg: 92319CF2B265DF6B : aucune assurance que la clef appartienne vraiment à l'utilis
sub rsa2048/92319CF2B265DF6B 2025-12-05 bob <bob@cs.cr>
Empreinte clef princip. : 1FAF AA6B E3E0 CD92 51A7 9D1B 5CBD 6FC9 0144 F779
Empreinte de sous-clef : ADCE 9C6F 74E8 60D8 DDB7 5F9B 9231 9CF2 B265 DF6B

La clef n'appartient PAS forcément à la personne nommée
dans l'identité. Si vous savez *vraiment* ce que vous
faites, vous pouvez répondre oui à la prochaine question.
I
Faut-il quand même utiliser cette clef ? (o/N) o
alice@eliott1:~$ exit
exit
root@eliott1:~# chown bob $(tty)
root@eliott1:~# cp /home/alice/truc.txt.asc /home/bob
root@eliott1:~# su bob
bob@eliott1:/root$ cd
bob@eliott1:~$ gpg --decrypt truc.txt.asc
gpg: encrypted with rsa2048 key, ID 92319CF2B265DF6B, created 2025-12-05
« bob <bob@cs.cr> »
secret pour bob
```

Et cette fois lorsque Bob déchiffre le message, aucun message alarmant.

Et si une tierce personne veut lire les messages d'Alice et Bob ?

Si un troisième utilisateur Charles entrait dans l'échange, il lui faudrait obtenir la clé publique d'Alice et de Bob pour lire les messages chiffrés qu'ils pourraient lui envoyer ou vérifier leurs signatures.

En pratique, Charles devrait faire confiance aux clés d'Alice et Bob soit en les faisant signer par une personne qu'il connaît, soit en comparant leur empreinte en personne, etc. Il existe des serveurs de clés publics (par ex. keys.openpgp.org autrefois pgp.mit.edu, etc.) où les utilisateurs publient leurs clés.

Dans un contexte local (domaine cs.sr isolé), on procède manuellement comme on l'a fait. Dans la réalité, on aurait pu enregistrer les clés de Alice et Bob sur un serveur de clés, et chacun aurait pu les récupérer de là.

Dans le prochain TP on va utiliser OpenVPN sous Pfsense.