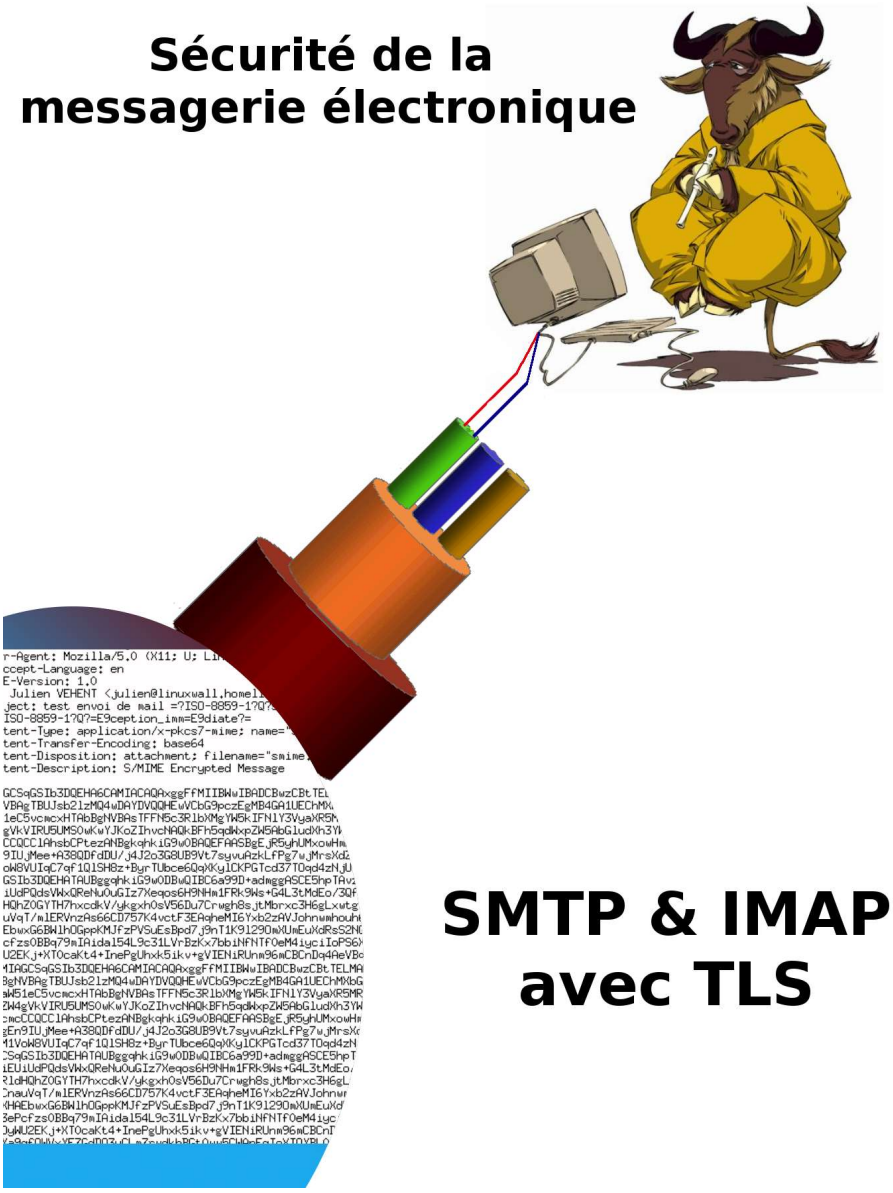
Julien VEHENT



Licence Professionnelle Sécurité et Qualité en Télécommunications
Université de Blois

-

Session 2004-2005

INTRODUCTION

Simple Mail Transfert Protocol (SMTP) est défini dans la RFC 2821. Il a été conçu pour être facilement compréhensible à la fois par les machines et les humains, ce qui l'a amené à devenir le protocole universel pour l'envoi des e-mails sur les réseaux.
Toutefois, la simplicité de SMTP implique une grande facilité de falsifications et d'interception des messages. Ainsi, sans l'ajout de mécanismes de sécurité, il est presque impossible de s'assurer de l'intégrité ou de la confidentialité d'un e-mail.

Transport Layer Security est la version 3.1 du bien connu protocole SSL.
Ce protocole est basé sur les certificats X.509, que nous reverrons plus tard, et les principes de cryptographie symétriques et assymétriques. La principale extension exploitée dans ce projet est définie dans la RFC 3207: il s'agit de "STARTTLS" qui active le chiffrement du canal de données entre un Mail User Agent (MUA) et un Mail Transfer Agent (MTA).

Bien que le sujet initial de ce projet était initialement "SMTP avec SSL", j'ai choisi d'élargir légèrement le champ d'étude pour y inclure IMAP et S/MIME, briévement.

Internet Message Access Protocol (IMAP) est un protocole de réception des messages. Par rapport à POP, IMAP apporte des améliorations importantes pour le stockage et la gestion des boites mails.

Le protocole S/MIME permet de chiffrer le corps des messages électronique avec des certificats TLS. Nous verrons pourquoi S/MIME est presque indispensable, même lorsque "STARTTLS" est déjà implémenté.

Enfin, nous parlerons de la configuration du serveur. Ce dernier est basé sur Debian GNU/Linux avec un noyau 2.6.8. Le MTA est Postfix qui utilise les composants de OpenSSL et Cyrus-Imap.

SOMMAIRE

Présentation de SMTP

> "The objective of the Simple Mail Transfer is to transfer mail reliably and efficiently.
> SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel.
> [...]
> An important feature of SMTP is its capability to transport mail across networks."
>
> RFC 2821

L'extrait de la RFC 2821, qui définie SMTP, nous permet d'isoler trois considérations importantes pour ce protocole:

- transfert de mails sûrement et efficacement
- abstraction de toute dépendance à un protocole de couche inférieure
- capacité à transporter les mails au travers des réseaux

Ainsi, la structure SMTP est la plus basique possible. On identifie deux types d'acteurs:

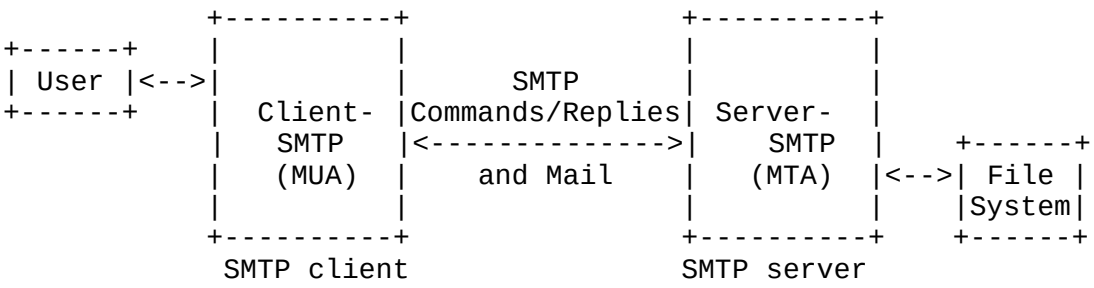
- MTA: Mail Transfer Agent. Ce sont généralement des daemons systèmes dont la tâche est d'acheminer les messages depuis l'émetteur vers le MTA du destinataire ou un MTA de relais intermédiaire.

- MUA: Mail User Agent. Ce sont des programmes locaux qui offrent à l'utilisateur les moyens d'interagir avec le système de messagerie, par le biais d'un jeu de commandes en ligne ou d'un interface graphique.

Processus d'envoi du courrier

SMTP est un protocole de couche 5 (session) du modèle OSI. Il se base sur TCP dans la pluspart des cas (mais rien dans son implémentation ne l'empêche d'utiliser un autre protocole de transport).

La structure basique de SMTP est la suivante:



Quand un MUA a un message à transmettre, il établit un canal de transmission avec son MTA.

Syntaxe SMTP

Il existe deux syntaxes d'envoie de messages: HELO et EHLO (Extended HELLO).

EHLO inclut les extensions de services, dont "STARTTLS".

Nous pouvons recréer le canal de données qu'utilise un MUA via l'outil "telnet".

```
$telnet linuxwall.homelinux.org 25
Trying 82.237.216.204
Connected to linuxwall.homelinux.org.
Escape character is '^]'.
220 linuxwall.homelinux.org ESMTP Postfix
EHLO pctest.exemple.com
250-linuxwall.homelinux.org
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250 8BITMIME
mail from:<jvehent@free.fr>
250 OK
rcpt to:<julien@linuxwall.homelinux.org>
250 OK
DATA
354 End data with <CR><LF>.<CR><LF>
Subject: test envoi de message en telnet
Cette ligne commence le corps du message.
Le point qui vas suivre en début de ligne vas terminer le corps du message.
.
250 OK: Queued as 5FA26B3DFE
```

La transmission se fait en clair, n'importe quel intrus sur le réseau peut sniffer la communication et accéder au contenu du mail.

Internet Message Access Protocol

Présentation de IMAP

> "The Internet Message Access Protocol allows a client to access and manipulate electronic mail messages on a server. IMAP permits manipulation of remote message folders,called "mailboxes", in a way that is functionally equivalent to local mailboxes. IMAP also provides the capability for an offline client to resynchronize with the server.
> IMAP includes operations for creating, deleting, and renaming mailboxes; checking for new messages; permanently removing messages; [...]. Messages in IMAP4rev1 are accessed by the use of numbers. These numbers are either message sequence numbers or unique identifiers.
> IMAP does not specify a means of posting mail; this function is handled by a mail transfer protocol such as [SMTP]."
>
> RFC 2060

Cette extrait de la RFC 2060 (IMAP-v4.rev1) nous spécifie les fonctions que gére ce protocole:

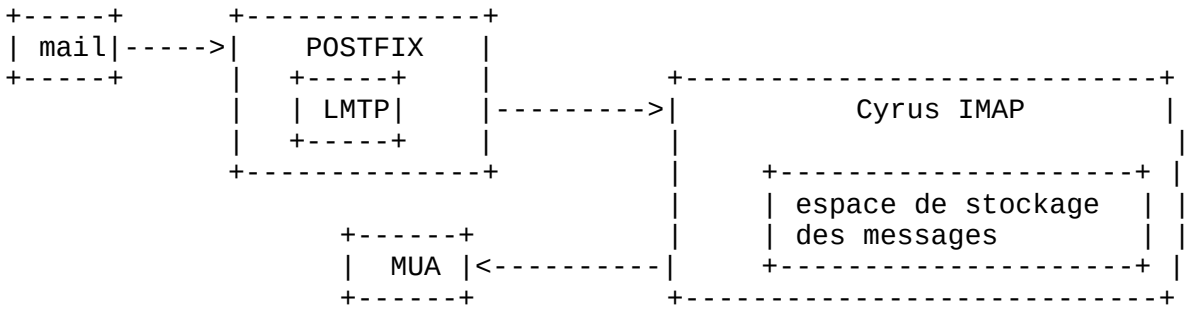
- Permet a un client d'accéder et de manipuler une mailbox sur le serveur
- Permet toute les opérations classiques de gestion des mails reçus
- Stockage des mails sur le serveur IMAP, classés et numérotés par utilisateur
- IMAP ne spécifie pas de méthode d'envoie de mail

Pourquoi Cyrus IMAP ?

Cyrus Imap est une implémentation du protocole Local Mail Transfer Protocol. LMTP fournit un moyen de passer les messages d'un service local de courrier à un autre sans dépendre d'un espace de stockage commun. LMTP est une version simplifiée de SMTP qui va permettre à un MTA de distribuer le courrier entrant sur plusieurs sous-systèmes, soit sur la même machine, soit sur le même LAN.
Cyrus IMAP est développé par l'université de Carnegie Mellon, aux USA. Il a été conçu pour tourner sur des serveurs ne fournissant qu'un accès IMAP, où les utilisateurs n'ont pas besoin d'un compte shell. Il peut attendre les distributions LMTP sur des sockets de domaine UNIX ou des sockets TCP (nous utiliserons les sockets UNIX car nos deux serveurs sont sur la même machine).
C'est la bibliothèque Cyrus SASL qui permet d'effectuer l'authentification.

Réception du courrier

Dans notre configuration, le serveur SMTP et le serveur IMAP sont sur la même machine. Voici comment se passe une réception de mail avec postfix comme MTA:



Le nombre de commandes mises à disposition par un serveur IMAP est beaucoup plus important qu'avec un serveur POP (25 contre 12).

Je ne vais pas détailler l'ensemble des commandes ici mais seulement les principales.

Les commandes présentées ici sont dans l'ordre chronologique d'utilisation.

- Il faut tout d'abord s'authentifier auprès du serveur :
  `Login <user> <mot_de_passe>`
- Il faut ensuite choisir la boite aux lettres que l'on souhaite utiliser :
  `Select inbox`
- La commande Fetch est très puissante. Elle permet d'effectuer de nombreuses et diverses sélections auprès de la liste des messages et des messages eux-mêmes :
  `Fetch <id(s)_msg(s)> <action>`
- Enfin pour quitter la session de dialogue avec le serveur :
  `Logout`

L'utilisation de la bibliothéque SASL permet d'imposer plusieurs méthodes, pour l'authentification, plus sûre que l'envoi classique du mot de passe en PLAIN TEXT. On retiendra la méthode DIGEST-MD5 où l'échange des informations d'authentification commence par un défi lancé par le serveur. Le client utilise ce défi et le mot de passe pour génèrer une réponse unique via l'algorithme MD5. Le serveur effectue aussi le défi et compare son résultat avec celui renvoyé par le client. Cette méthode permet d'éviter que le mot de passe soit envoyé sur le réseau, même chiffré.

Transport Layer Security

Présentation de TLS

La structure complète du protocole TLS est décrite dans la RFC 2246. Voici comment TLS y est présenté:

Le rôle principal de TLS est de fournir la confidentialité et l'intégrité des données entre deux applications en communication. Le protocole est composé de deux niveaux: TLS Record Protocol & TLS Handshake Protocol.

Le plus bas est TLS Record Protocol. Il permet de sécuriser les connections grâce à deux propriétés importantes:

1) **La connection est privée**. Le chiffrement symétrique est utilisé pour chiffrer les données. Les clés sont à usage unique et basée sur un secret partagé négocié par un autre protocol (i.e. Handshake Protocol).

2) **La connection est sûre**. L'intégrité des messages est assurée par un Message Authentication Code (MAC) calculé a partir d'une fonction hash.

TLS Record Protocol est utilisé pour encapsuler des protocoles de plus au niveau, comme par exemple SMTP et IMAP.

Juste au dessus de ce dernier se trouve TLS Handshake Protocol. Il permet au serveur et au client de s'authentifier l'un l'autre puis de négocier un algorithme de chiffrement et une clé cryptographique avant que l'application ne transmette son premier octet. Ce protocole posséde trois propriétés importantes:

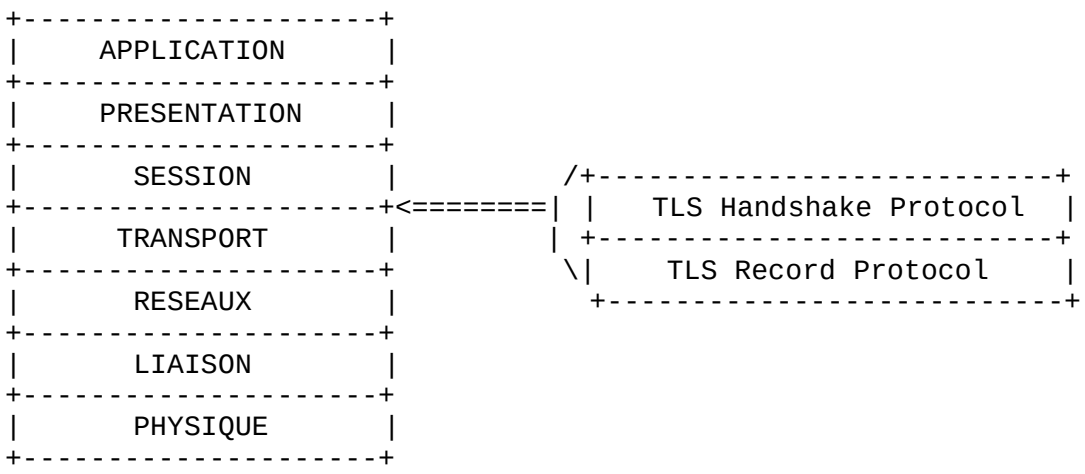
1) **Les participants sont authentifiés** en utilisant un algorithme assymétrique.

2) **La négociation du secret partagé est sécurisée**. Il est complétement à l'abris des intrus, même dans le cas d'un "man in the middle".

3) **La négociation est sûre**. Un intrus ne peut pas modifier les informations de négociation sans être repéré par une des parties.

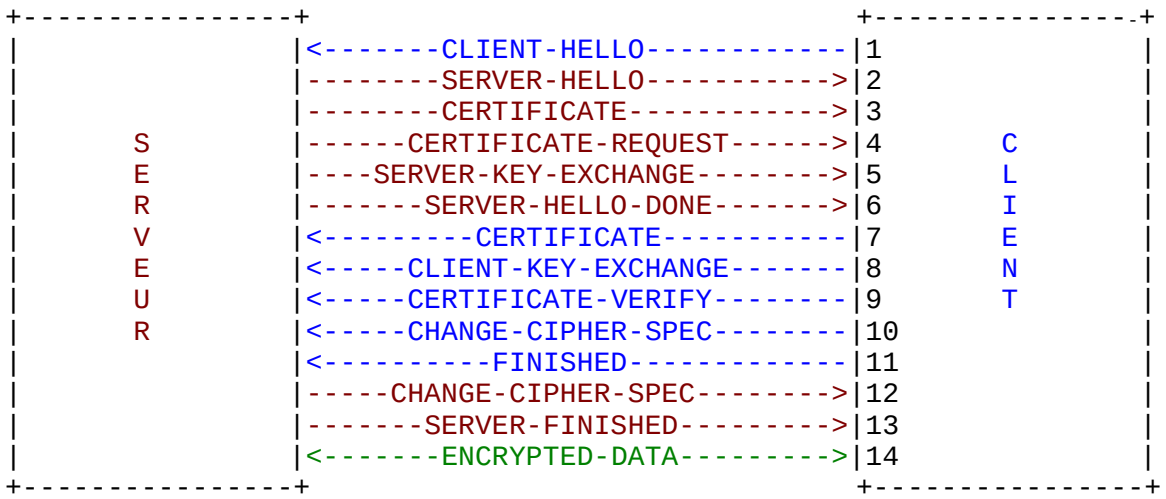
Un des avantages de TLS est qu'il est complétement indépendant de l'application utilisée dans les niveaux supérieurs du modèle OSI, et donc complétement transparent.

Voici comment se situe TLS dans le modèle OSI:



Handshake Protocol

Le protocole Handshake TLS sert à établir des communications privées. Les messages constituant le protocole Handshake TLS doivent être présentés dans un ordre bien précis; dans le cas contraire, il en résulte une erreur fatale.
Voici comment se passe le Handshake:



1. CLIENT HELLO (émetteur ↦ client):

   envoi de la version maximale de SSL supportée (TLS=3.1), la suite des algorithmes supportés (par ordre de préférence décroissant) et une valeur aléatoire d'une taille de 32 octets.

2. SERVER HELLO (serveur):

   choix de la version , de la suite d'algorithmes (Cipher Suite) et d'une valeur aléatoire.

Ci-dessous, un paquet "SERVER HELLO" sniffé via ethereal lors d'un Handshake avec Linuxwall.HomeLinux.org:

```
Handshake Protocol: Server Hello

  Handshake Type: Server Hello (2)
  Length: 70
  Version: TLS 1.0 (0x0301)
  Random.gmt_unix_time: Mar 13, 2005 19:31:22.000000000
  Random.bytes
  Session ID Length: 32
  Session ID (32 bytes)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
                |suite de chiffrement choisie|
```

3. CERTIFICATE (serveur):

   envoi d'une chaîne de certificats par le serveur – *optionnel*

   le premier certificat est celui du serveur, le dernier est celui du CA (voir page 10).

4. CERTIFICATE REQUEST (serveur):

   demande un certificat au client pour l'authentifier – *optionnel*

5. SERVER KEY EXCHANGE (serveur):

   message complémentaire pour l'échange des clés – *optionnel*

   ce message contient la clé publique du serveur utilisée par le client pour chiffrer les informations de clé de session

6. SERVER HELLO DONE (serveur):

   fin des émissions du serveur

7. CERTIFICATE (client):

   certificat éventuel du client si le serveur demande une authentification – *optionnel*

8. CLIENT KEY EXCHANGE (client):

   le client produit un secret pré-maître (*encrypted pre-master key*) et le chiffre avec la clé publique du certificat du serveur. Ces informations sont chiffrées une deuxiéme fois avec la clé publique du serveur reçus dans le message SERVER KEY EXCHANGE (et non la clé publique du certificat).

9. CERTIFICATE VERIFY (client):

   message contenant un hash signé numériquement et effectué à partir des informations de clé et de tous les messages précédents. Ce message permet de confirmer au serveur que le client posséde bien la clé privée correspondant au certificat client (message 7) – *optionnel*

10. CHANGE CIPHER SPEC (client):

    passe en mode chiffrée avec la clé master générée

11. FINISHED (client):

    fin des émissions du client, ce message est chiffrée à l'aide des paramètres de la suite de chiffrement.

12. CHANGE CIPHER SPEC (serveur):

    passe en mode chiffrée avec la clé master générée

13. FINISHED (serveur):

    confirmation au client du passage en mode chiffré, ce message est chiffrée à l'aide des paramètres de la suite de chiffrement.

14. ENCRYPTED DATA:

    tunnel TLS établi, chiffrement du canal de données

15. Authentification du serveur

L'authentification du serveur est assurée par un "Certificate Authority" (CA). Cette autoritée vérifie l'identité rattachée à la clé publique du serveur (par téléphone, ...) lors de l'ajout du certificat dans la base de données du CA.

Une fois que le CA à vérifié cette identité, il calcule une empreinte du certificat public du serveur. Cette empreinte est ajoutée à la fin du certificat puis chiffrée avec la clé privée du CA.

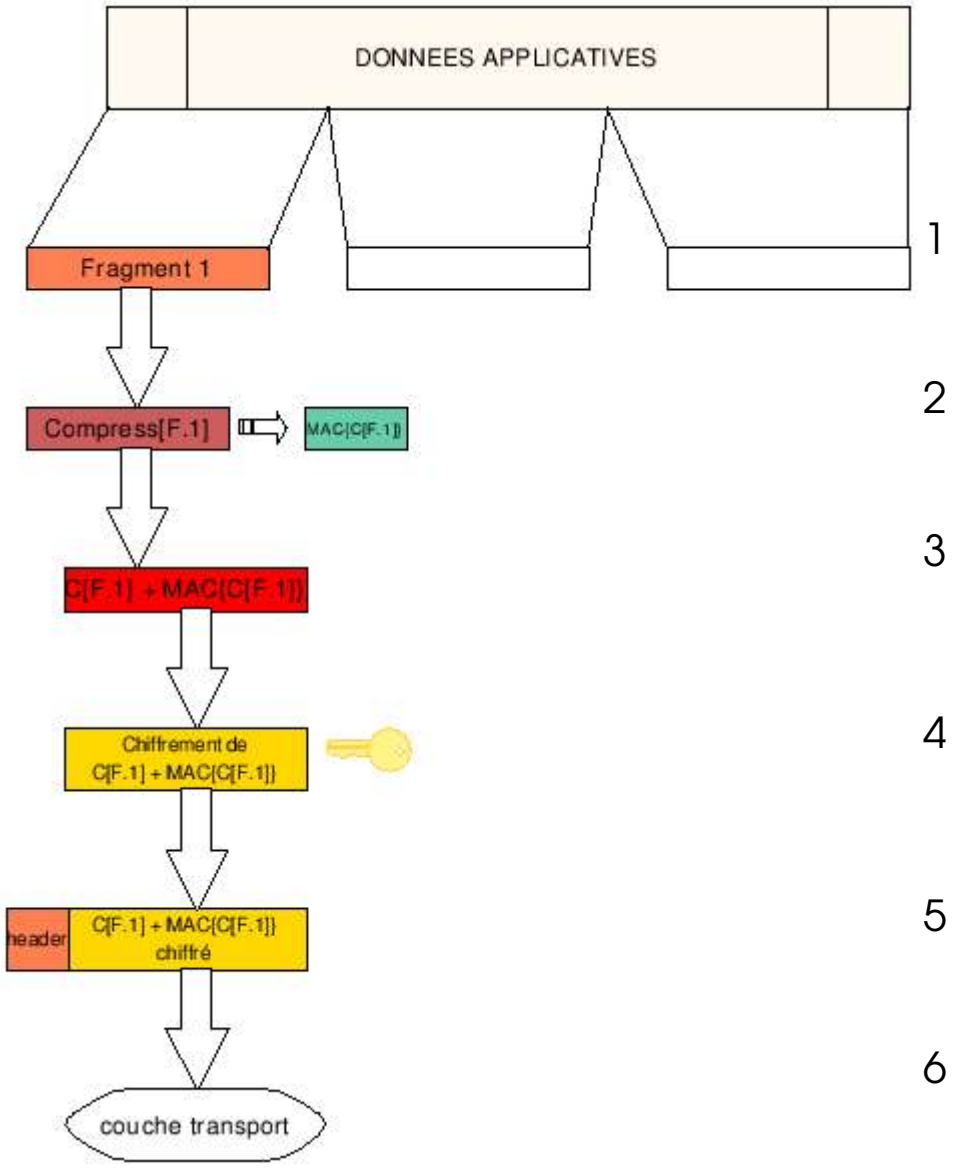
Ainsi, le client qui reçoit ce certificat signé par un CA (qu'il connait forcément puisqu'il doit en posséder la clé publique) a l'assurance que la clé publique du serveur appartient bien à l'identité désignée. Il vérifie tout de même cette signature ainsi que la période de validité du certificat et contrôle que ce dernier n'est pas inscrit dans une

CRL (Certificate Revocation List). Les CRL sont diffusées par les CA.

Enfin, le client contrôle que le certificat identifie sans ambiguïté le serveur avec qui il souhaite établir le canal sécurisé en comparant les noms de domaines (celui spécifié dans le certificat et celui du serveur).

Record Protocol

Le niveau inférieur de TLS, Record Protocol, permet l'encapsulation des messages, assurant ainsi la confidentialité et l'intégrité. Voyons les étapes de ce protocole:



1. Les blocs de données applicatives sont tout d'abord découpés en fragments de 16Koctets (2^{14} octets) maximum.

2. Chaque blocs subit une compression (facultative). Dans le cas de blocs trés courts, cette compression peut augmenter la taille du bloc initial. Toutefois, la longueur du contenu ne peut être augmentée de plus de 1024 octets.

3. Un MAC est calculé sur le bloc compressé (nécessite l'utilisation d'une clé secrète partagée). Le MAC est ajouté a la fin du bloc.

4. Le bloc (et le MAC) sont chiffré par un algorithme symétrique

5. Un en-tête de 5 octets est ajouté. Le champ "Type" de cet en-tête définit le type de protocole de niveau supérieur à la couche d'enregistrement auquel est destiné le message.

   Les types sont les suivants:

Valeur	Protocole
20	ChangeCipherSpec Protocol
21	Alert Protocol
22	Handshake Protocol
23	Application Protocol Data

6. Le message est transmis à la couche 4 (transport). La pluspart du temps, au

   protocole TCP.

X.509

Les certificats X.509 présenté ici utilisent la 3éme version du standard. Cette version est définie dans la RFC 3280.

Les champs des certificats X.509 sont:

```
Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }
```

Le champ `tbsCertificate` contient les noms du sujet et de l'émetteur, une clé publique associée au sujet, une période de validité, un numéro de version et un numéro de série ainsi que des extensions.
Les extensions sont les suivantes:

- `Basic Constraints`: ce booléen est vrai si la clé publique du certificat est celle d'un CA.
- `Subject Key Identifier`: ce champ contient un empreinte SHA-1 de 160 bits de la clé publique du sujet.
- `Authority Key Identifier`: ce champ identifie la clé publique correspondant à la clé privée qui a signée le certificat.

Le champ `signatureAlgorithm` contient le nom de l'algorithme utilisé par le CA pour signer le certificat.

Le champ `signatureValue` contient une signature digitale du champ tbsCertificate. Cette signature est sous la forme d'une suite de bits. C'est grâce à cette signature qu'un CA certifie la validité des informations du champ tbsCertificate.

L'intêret des certificats X.509 est qu'ils fournissent une méthode d'authentification dans les systèmes distribués qui est indépendante d'une implémentation. Le système de codage des certificats X.509 est spécifié dans le système de notification ASN-1.

La création des certificats X.509 via OpenSSL est abordée dans la configuration du serveur.

Extension de sécurité: STARTTLS

La RFC 3207 définit une extension à SMTP, connue sous le nom de STARTTLS. Cette extension permet à un client et un serveur d'utiliser le protocole TLS pour le chiffrement du canal de données.

Le mot clé STARTTLS est utilisé pour indiquer au client que le serveur supporte le protocole TLS. Pour engager les négociations TLS, le client envoie la commande STARTTLS (sans paramètres) au serveur.

Ce dernier renvoie alors l'un de ces 3 codes:

- `220 Ready to start TLS`
- `501 Syntax error (no parameters allowed)`
- `454 TLS not available due to temporary reason`

Si une réponse 454 est retournée, le client doit décider de continuer ou non la sessions.
Un serveur SMTP référencé sur un réseau public ne peut exiger l'utilisation de l'extension de service STARTTLS afin de distribuer un courrier. Cette régle empêche l'extension STARTTLS de remettre en cause l'interopérabilité de l'infrastructure SMTP sur internet.

En revanche, un serveur SMTP privé peut imposer au client une négociation TLS avant d'accepter la moindre commande. Dans ce cas, le serveur répond, à toutes autres commandes que NOOP, QUIT, EHLO et STARTTLS, avec le code:
`530 Must issue a STARTTLS command first`

Aprés le Handshake, les deux parties doivent décider immédiatement si elles continuent ou non avec les paramètres de sécurité négociés. Si le client SMTP estime que le niveau d'authentification ou de confidentialité est insuffisant, il peut transmettre une commande SMTP QUIT, juste aprés les échanges TLS. Dans les mêmes conditions, le serveur retournera un code de réponse:
`554 Command refused due to lack of security`

Une fois le Handshake terminé, le protocole SMTP est réinitialisé à son état initial,

c'est-a-dire l'état SMTP aprés que le serveur ait délivré une annonce:

```
220 service ready
```

Le client doit alors envoyer, en guise de 1ére commande consécutive à une négociation TLS réussie, un EHLO. La liste des extensions de service retournée aprés ce EHLO peut être différente de celles affichées avant les négociations TLS.

Voici un exemple de début de session TLS avec SMTP:

```
Serveur:      <waits for connection on TCP port 25>
Client:       <opens connection>
S:            220 mail.imc.org SMTP service ready
C:            EHLO mail.example.com
S:            250-mail.imc.org offers a warm hug of welcome
S:            250-8BITMIME
S:            250-STARTTLS
S:            250 DSN
C:            STARTTLS
S:            220 Go ahead
C:            <starts TLS negotiation>
C & S:        <negotiate a TLS session>
C & S:        <check result of negotiation>
C:            EHLO mail.example.com
S:            250-mail.imc.org touches your hand gently for a moment
S:            250-8BITMIME
S:            250 DSN
```

Les étapes de négociations ont déjà étaient décrites précedemment.

MIME

Le type MIME (Multipurpose Internet Mail Extensions) est un standard qui a été proposé par les laboratoires *Bell Communications* en 1991 afin d'étendre les possibilités du courrier électronique, c'est-à-dire d'ajouter une structure au corps du message et de définir dess règles de codage pour les messages non ASCII. Le format MIME permet également d'insérer des documents (images, sons, texte, ...) dans un courrier.

Le format MIME définit cinq nouveaux en-têtes de message:

- `MIME-Version`: identifie la version du format MIME utilisé
- `Content-Description`: chaîne ASCII décrivant la teneur du message
- `Content-ID`: identifiant unique
- `Content-Transfer-Encoding`: indique comment le message est encapsulé pour la transmission
- `Content-Type`: type et format du contenu

MIME est décrit dans les RFC 1521 et 1522.

Présentation de S/MIME

S/MIME est un protocole orienté objet permettant de sécuriser les envois de messages électroniques. Le protocole permet de signer ou de chiffrer les messages envoyés en utilisant le protocole CMS.
Le protocole MIME permet de définir le contenu d'un message électronique à l'aide de la balise « Content-Type ». Le protocole S/MIME définit le contenu de cette balise à la valeur « application/pkcs7-mime » pour indiquer que le contenu du message est au format CMS, lui même contenant un contenu MIME.

S/MIME présente notamment les avantages suivants :

- mécanisme de sécurité de bout en bout :le protocole S/MIME présente une assurance de sécurité de bout en bout. En effet, les messages sont signés et/ou chiffrés par l'émetteur du message et déchiffrés par le destinataire. Ces messages

transitent à travers le système de messagerie de manière sécurisée.

- non répudiation : les mécanismes de signature du protocole S/MIME permettent de véhiculer un message signé depuis un émetteur sans transformation de la signature.
- utilisation de relais de messagerie : le protocole S/MIME, du fait de la composition même du protocole permet l'utilisation de relais de messagerie. Le message est relayé sans que le corps ne soit inspecté.

S/MIME est définit dans les RFC 2630 à 2633.

Le protocole CMS

CMS est un protocole fournissant des mécanismes d'encryption et de signature de données. Chaque message CMS possède un type décrivant le mécanisme cryptographique utilisé pour chiffrer les données. Pour S/MIME, nous utiliserons le protocole CMS pour les types de messages suivants :

- les messages signés :

  un message signé contient des données accompagnées d'une ou de plusieurs signatures du contenu. Le protocole effectue un hachage du contenu signé par la clé privée de l'émetteur. Ce type de message peut inclure les certificats permettant la vérification des signatures utilisées ainsi que l'adresse de la CRL permettant de vérifier la validité des dits certificats.

- les messages chiffrés :

  un message chiffré contient des données chiffrées. Les données sont chiffrées via le certificat public signés du destinataire du message.

Certificats signés et certificats PKCS#12

Pour utiliser le protocole S/MIME avec un client de messagerie, nous avons besoin de deux types de certificats par utilisateur de la messagerie.

Le certificats privés de l'utilisateur de la messagerie va contenir, entre autres, la clé privée de cet utilisateur et le certificat signé, le tout empaqueté au format PKCS#12 qui définit un standard pour le stockage sécurisé des clés privées et des

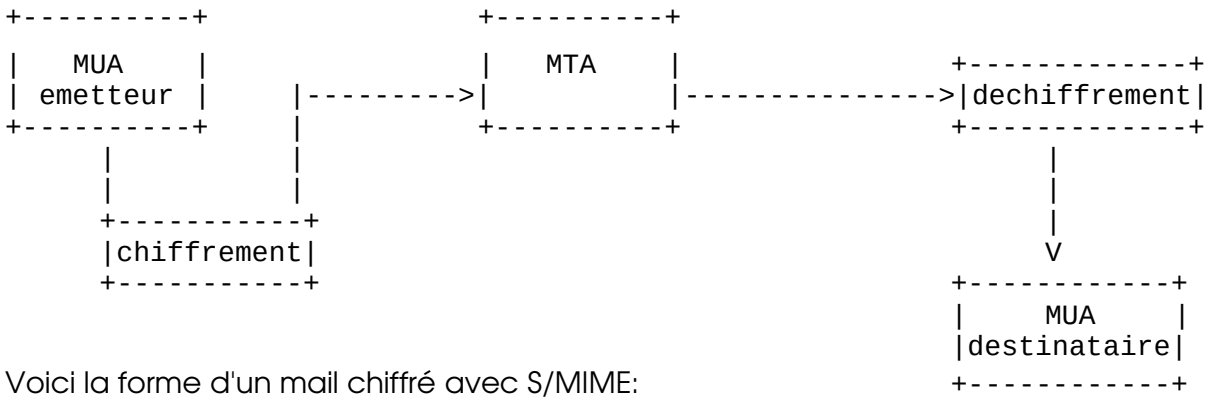
certificats. Ce fichier prendra l'extension ".p12" et sera protégé par un mot de passe.

Le certificat publics signés est un certificat contenant la clé publique d'un utilisateur et est signés par un CA de confiance. Ce certificat est au format X.509 comme vu précedemment.

Un émetteur qui envoie un message doit posséder le certificat public signés du destinataire pour chiffrer le message avec la clé publique de ce dernier.

Un destinataire qui reçoit un message chiffré doit posséder son certificat p12 pour déchiffrer le message avec sa clé privée.

Dans notre configuration, c'est la cryptographie assymétrique via l'algorithme RSA qui est utilisée pour chiffrer les messages.



Voici la forme d'un mail chiffré avec S/MIME:

```
linuxwall:/home/julien# cat /var/spool/cyrus/mail/j/user/julien/50.
Return-Path: <cyrus@linuxwall.linuxwall.homelinux.org>
Received: from linuxwall.linuxwall.homelinux.org ([unix socket])
       by linuxwall.linuxwall.homelinux.org (Cyrus v2.1.17-IPv6-Debian-2.1.17-3) with
LMTP; Sun, 06 Mar 2005 20:15:49 +0100
X-Sieve: CMU Sieve 2.2
Return-Path: <julien@linuxwall.homelinux.org>
Received: from [192.168.1.2] (unknown [192.168.1.2])
        by linuxwall (Postfix) with ESMTP id A9872460FF
        for <julien@linuxwall.homelinux.org>; Sun,  6 Mar 2005 20:15:48 +0100 (CET)
Message-ID: <422B553E.3080601@linuxwall.homelinux.org>
Date: Sun, 06 Mar 2005 20:08:46 +0100
From: Julien VEHENT <julien@linuxwall.homelinux.org>
User-Agent: Mozilla/5.0 (X11; U; Linux i686; fr-FR; rv:1.7.5) Gecko/20050105
Debian/1.7.5-1
X-Accept-Language: en
MIME-Version: 1.0
To: Julien VEHENT <julien@linuxwall.homelinux.org>
Subject: envoi de mail avec corps crypte par certificat tls
Content-Type: application/x-pkcs7-mime; name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"
Content-Description: S/MIME Encrypted Message

MIAGCSqGSIb3DQEHA6CAMIACAQAxggFiMIIBXgIBADCBxjCBuDELMAkGA1UEBhMCRlIxFjAU
BgNVBAgTDUxvaXJlLWV0LUNoZXIxDjAMBgNVBAcTBUJsb2lzMSAwHgYDVQQKExdMaW51eHdh
bGwuSG9tZUxpbnV4Lm9yZzEYMBYGA1UECxMPRS1tYWlsIFNlY3VyaXR5MRYwFAYDVQQDEw1K (...)
```

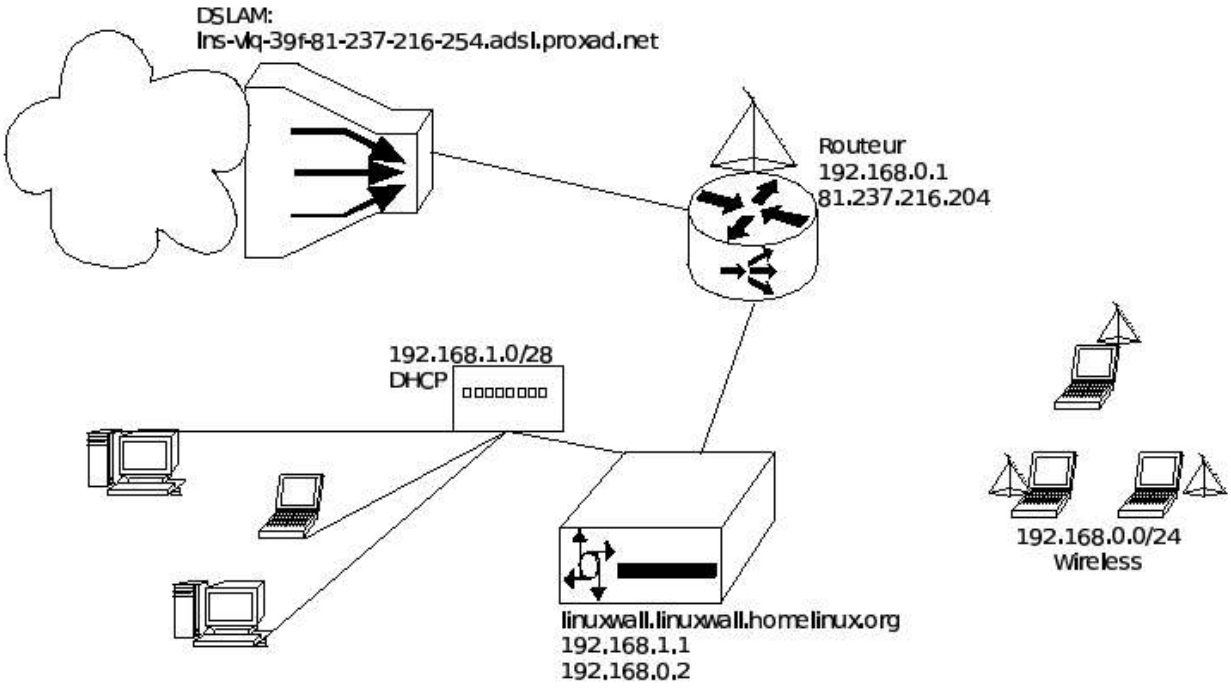
Système Debian

La distribution choisie pour déployer le serveur de messagerie est une Debian dans sa version NetInstall. Cette version permet d'installer une base de système trés légère (inférieur à 250Mo) et de continuer le déployement via l'outil de gestion de package "apt".

Le noyau compris dans la Net Install est la version 2.6.8. N'ayant pas besoin de recompiler celui-ci (aucun problème de drivers à l'installation), le système continuera de tourner sur ce noyau.

Du point de vue matériel, le serveur est basé sur un processeur Intel Pentium II cadencé à 500Mhz, 128Mo de mémoire SDRAM et un disque dur de 8Go. Les besoins en puissance de calcul sont largement satisfait par cette architecture.

La configuration du LAN est la suivante:



Le domaine est linuxwall.homelinux.org, redirigé via dyndns sur 82.237.216.204.
Le routeur forward différents ports dont 25 et 993 (SMTP et IMAPS) sur le serveur.

Voici les principaux packages installés sur le serveur dont la présence est requise pour la fonction de MTA:

```
linuxwall:/home/julien# dpkg -l "*bind*" "*dhcp*" "*postfix*" "*cyrus*" "*ssl*"

||/ Nom                          Version         Description-
================================================================================
ii  bind9                        9.2.4-1         Internet Domain Name Server
ii  dhcp                          2.0pl5-19.1     DHCP server
ii  dhcp-client                   2.0pl5-19.1     DHCP Client
ii  dhcp-dns                      0.50-3.2        Dynamic DNS updates for DHCP
ii  postfix                       2.1.5-6         A high-performance MTA
ii  postfix-tls                   2.1.5-6         TLS and SASL support for Postfix
ii  cyrus21-admin                 2.1.17-3        Cyrus mail system (admin tool)
ii  cyrus21-common                2.1.17-3        Cyrus mail system (common files)
ii  cyrus21-imapd                 2.1.17-3        Cyrus mail system (IMAP support)
ii  libssl0.9.7                   0.9.7e-2        SSL shared libraries
ii  openssl                       0.9.7e-2        Secure Socket Layer (SSL) binary
```

Les quatres premiers packages ne seront pas détaillés, Bind est le serveur DNS (indispensable a tout MTA, à cause de l'enregistrement MX). Le serveur DHCP permet de forcer l'appartenance au domaine et les serveurs DNS à utiliser.

Configuration de Postfix et OpenSSL

Postfix a été écrit par Wietse Venema, bien connnu pour les outils qu'il a créé et pour ses articles sur la sécurité. Le logiciel a été mis à disposition sous licence open source en décembre 1998. IBM Research a sponsorisé sa version initiale et continue de participer à son développement. Dés le départ, la conception et le développement de Postfix ont été pilotés par les critères suivants:

➔ Fiabilité: Postfix détecte les conditions difficiles (manque de mémoire, d'espace disque) et donne un chance au système de ce rétablir.

➔ Sécurité: Postfix suppose qu'il s'exécute dans un environnement hostile. Le concept sécuritaire de privilège minimum est employé dans tout le système Postfix, afin que chaque processus, qui peut être lancé dans un compartiment étanche (chroot), s'exécute avec les privilèges minimaux nécessaires à son fonctionnement. Les processus tournant avec des privilèges plus élevés ne font jamais confiance aux processus non privilégiés. De même, les modules inutiles peuvent être désactivés.

➔ Performances, souplesse (fonctionnement par sous-programmes), facilité d'emploi

et compatibilité avec Sendmail.

Pour fonctionner avec TLS, Postfix à besoin d'un patch connu sous le nom de postfix-tls. Ce patch inclut également le support SASL pour notre MTA.

Le package de ce patch est présent dans "apt":

```
linuxwall:/home/julien# apt-cache show postfix-tls
Package: postfix-tls
[...]
Filename: pool/main/p/postfix/postfix-tls_2.1.5-6_i386.deb
Size: 154564
MD5sum: 54586984e8b159c73136e7d9357ab601
Description: TLS and SASL support for Postfix
 Postfix is Wietse Venema's mail transport agent that started life as an
 alternative to the widely-used Sendmail program.  Postfix attempts to
 be fast, easy to administer, and secure, while at the same time being
 sendmail compatible enough to not upset existing users. Thus, the
outside
 has a sendmail-ish flavor, but the inside is completely different.
 .
 This package adds support for TLS (see RFC 3207) and SASL (see RFC 2554)
to Postfix.
```

La configuration de Postfix s'effectue dans le répertoire "/etc/postfix". Le fichier "main.cf" contient la configuration du MTA, le fichier "master.cf" contient les paramètres des sous-programmes de Postfix.

Fichier main.cf:

```
mydomain = linuxwall.homelinux.org
myhostname = linuxwall
myorigin = $mydomain
mydestination = $mydomain
mynetworks = 127.0.0.0/8 192.168.1.0/28
#-------RESTRICTION--RELAYAGE----------------------
smtpd_recipient_restrictions = permit_mynetworks,
reject_unauth_destination
#--------------------------------------------------
relay_domains = $mydomain
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
#------support-TLS-sur-SMTP-------------------------
smtpd_use_tls = yes
smtpd_tls_key_file = /etc/postfix/mailkey.pem
smtpd_tls_cert_file = /etc/postfix/mail_signed_cert.pem
smtpd_tls_CAfile = /etc/postfix/cacert.pem
#-----------support-Cyrus-IMAP---------------------
mailbox_transport = cyrus
local_recipient_maps =
```

Cette configuration permet d'utiliser TLS, à condition que l'on ait déjà les certificats.

OpenSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS qui offre :

1. une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS.
2. une commande en ligne (`openssl <arguments>`) permettant
    - la création de clés RSA, DSA (signature)
    - la création de certificats X509
    - le calcul d'empreintes (MD5, SHA, RIPEMD160, ...)
    - le chiffrement et déchiffrement (DES, IDEA, RC2, RC4, Blowfish, ...)
    - la réalisation de tests de clients et serveurs SSL/TLS
    - la signature et le chiffrement de courriers (S/MIME)

Dans notre configuration, nous avons besoin de créer un certificat signé pour le serveur Postfix. C'est ce certificat qu'il enverra aux MUA désireux d'utiliser STARTTLS.

La première chose à faire est de définir un CA pour nos certificats. Ici, c'est directement le serveur Postfix qui assurera également la fonction de CA. Dans un environnement professionnel, il est préférable de faire appel à des sociétés comme Verisign, RSA Security ou encore Baltimore. Ces sociétés sont spécialisés dans ce type de sécurité et assurent un excellent niveau de confiance.

La commande "`#misc/CA.pl -newca`" lancée dans le répertoire "/usr/lib/ssl/" nous permet de créer les fichiers nécessaires afin que notre serveur soit un CA. Plusieurs questions sont posées qui correspondent au champ X.509 du certificat du CA.

Ensuite, nous allons créer le certificat du serveur Postfix. Il est important que les clés contenus dans ce certificats ne soient pas chiffrés, sinon les différents daemons de Postfix ne pourront pas lire ces clés et donc ne pourront pas les utiliser.

La commande de génération du certificat du serveur est la suivante:

```
$openssl req -new -nodes -keyout mailkey.pem -out mailreq.pem -days 365
```

- "openssl req -new" crée à la fois une clé privée et une Certificate Signing Request (CSR) qui sera envoyé au CA pour signer le certificat.
- L'option "-nodes" demande de ne pas chiffrer la clé.

- "-keyout" et "-out" indique les noms des fichiers où seront stockées la clé privée et la CSR.
- "-days" précise que le certificat sera valide un an.

Il faut maintenant signer le certificat. Dans notre configuration, ce fichier sera signé par le serveur lui-même via la commande:

```
#openssl ca -out mail_signed_cert.pem -infiles mailreq.pem
```

Cette commande produit le certificat signé "mail_signed_cert.pem" ci-dessous:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            86:03:3b:67:ba:52:c0:33
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=FR, ST=Loire-et-Cher, L=Blois, O=Linuxwall.HomeLinux.org, OU=E-mail
Security, CN=Julien VEHENT/emailAddress=julien@linuxwall.homelinux.org
        Validity
            Not Before: Feb 26 21:49:42 2005 GMT
            Not After : Feb 26 21:49:42 2006 GMT
        Subject: C=FR, ST=Loire-et-Cher, O=Linuxwall.HomeLinux.org, OU=E-mail
Security, CN=Julien VEHENT/emailAddress=julien@linuxwall.homelinux.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c8:68:9b:ca:94:a0:e8:74:32:7c:7a:80:5a:19:
                    c2:5c:bc:c3:08:17:af:66:a4:f3:3d:a6:6c:a3:53:
                    b4:38:ce:ec:a6:b4:4d:9f:cd:5e:48:31:ab:fa:cd:
                    71:aa:ab:53:7e:d6:51:fa:e1:b7:92:f4:45:7e:ea:
                    74:c6:73:4b:5c:fb:59:62:bd:9a:41:8f:0f:85:2e:
                    d9:39:14:9c:82:49:ae:20:04:08:1e:fe:63:cb:42:
                    c1:4d:c7:b1:8e:09:eb:b4:9c:f7:4f:ab:62:62:94:
                    f9:0c:fc:a1:5c:7a:b6:54:bd:3d:e3:7b:bf:24:92:
                    4a:1f:84:1a:08:7b:87:a4:13
                Exponent: 65537 (0x10001)


       X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate

             X509v3 Subject Key Identifier:
                22:63:9F:73:01:BC:4A:CA:90:CB:BC:9E:0D:2A:1D:3F:1C:E8:75:BC
            X509v3 Authority Key Identifier:
            keyid:BF:A7:5A:63:B1:C9:C8:F9:E8:1D:58:6B:43:69:BA:2B:54:4F:7C:0A
             DirName:/C=FR/ST=Loire-et-Cher/L=Blois/O=Linuxwall.HomeLinux.org/OU=E-
mail Security/CN=Julien VEHENT/emailAddress=julien@linuxwall.homelinux.org
                serial:86:03:3B:67:BA:52:C0:32
    Signature Algorithm: md5WithRSAEncryption
        0d:5a:fb:3c:6e:4e:a7:79:c2:22:0b:ae:30:ec:70:57:2b:8e:
        21:27:e4:ae:17:0c:90:96:d0:1d:04:3d:43:a9:7f:67:e0:28:
        7a:12:d6:d6:d9:5f:ba:20:f5:4f:11:fa:ab:13:6b:c5:dd:68:
        2d:9c:11:85:f3:82:57:df:b5:cd:e5:9f:82:b6:c4:38:b3:d2:
        ac:ef:22:24:19:8f:34:b0:10:34:b5:6f:98:60:7d:33:b3:18:
        5c:fc:2a:89:10:eb:65:e8:16:a4:58:15:a7:35:c7:6d:8b:08:
        b8:d8:9b:cf:8b:de:73:81:8f:58:1f:35:a8:70:81:49:75:40:
        9b:cd
```

```
-----BEGIN CERTIFICATE-----
MIIELjCCA5egAwIBAgIJAIYDO2e6UsAzMA0GCSqGSIb3DQEBBAUAMIG4MQswCQYD
VQQGEwJGUjEWMBQGA1UECBMNTG9pcmUtZXQtQ2hlcjEOMAwGA1UEBxMFQmxvaXMx
IDAeBgNVBAoTF0xpbnV4d2FsbC5Ib21lTGludXgub3JnMRgwFgYDVQQLEw9FLW1h
[...]
tc3ln4K2xDiz0qzvIiQZjzSwEDS1b5hgfTOzGFz8KokQ62XoFqRYFac1x22LCLjY
m8+L3nOBj1gfNahwgUl1QJvN
-----END CERTIFICATE-----
```

On retrouve les champs X.509 présentés précedemment avec, en vert, la clé publique du serveur Postfix; en bleu, la signature du CA et, en rouge, le certificat en base 64.

Configuration de Cyrus-IMAP

Cyrus (580-528) était un roi intelligent, qui ne gouverna pas son gigantesque empire par la cruauté comme le firent les Assyriens. Bien au contraire, tout son règne se caractérisa par une politique de tolérance et d'intégration envers les divers peuples qui composèrent son nouvel empire.

Ce, qui retranscrit en termes informatiques, peut donner ceci:

Cyrus-IMAP se caractérise par sa capacité d'intégration et de tolérance envers la multitude de composants qui peuvent former l'ensemble du serveur de mail.

C'est Cyrus-IMAP qui va recevoir les mails locaux, gérer les boites aux lettres des utilisateurs et servir à ces derniers leurs messages via le protocole IMAP (ou POP3 mais il ne nous intéresse pas ici).
Cyrus-IMAP, pour authentifier les clients, s'appuie sur SASL évitant ainsi la création de comptes UNIX sur le serveur.

- Cyrus-IMAP supporte donc tous les mécanismes d'authentification proposés par le protocole d'authentification SASL, à savoir: sasldb, getpwent, kerberos4, kerberos5, pam, rimap, shadow, ldap.
- il utilise le format *Maildir*, plus souple et plus sûr que Mailbox.
- il permet de gérer une arborescence de boites aux lettres multi-niveaux.
- il permet, depuis la version 2.x, de gérer des noms d'utilisateurs et de répertoires contenant des points "." en remplaçant le séparateur de hiérarchie d'origine (le point, justement) par le séparateur standard d'UNIX (le "slash" /).
- il permet la création de filtres qui rejetteront les messages jugés indésirables.

- il permet l'accès aux boite aux lettres de façon très souple, lorsqu'il y a plusieurs serveurs dans le même domaine.
- il est aussi capable de proposer une structure permettant d'exposer des groupes usenet (nntp).
- Cyrus-IMAP sait gérer les dossiers partagés.

L'installation, via "apt", des packages relatifs à Cyrus-IMAP et à SASL créé l'utilisateur "cyrus". Ce dernier est l'administrateur du serveur Cyrus-IMAP. On créé un mot de passe pour cet administrateur via la commande ~# `saslpasswd2 cyrus`.

Les utilisateurs de Cyrus-IMAP sont déclarés dans le fichier "/etc/sasldb2".

```
linuxwall:/# sasldblistusers2
cyrus@linuxwall.homelinux.org: userPassword
julien@linuxwall.homelinux.org: userPassword
tof@linuxwall.linuxwall.homelinux.org: userPassword
franckylourson@linuxwall.homelinux.org: userPassword
njm@linuxwall.homelinux.org: userPassword
[...]
```

La création des utilisateurs passe par les commandes suivantes (que j'ai scriptée pour simplifier la tâche):

```
#! /bin/sh
echo "------------------------------------------------" &&
echo "TAPEZ LE MOT DE PASSE DU COMPTE UTILISATEUR $1" &&
saslpasswd2 -c $1 &&
echo "------------------------------------------------" &&
echo "TAPER LE MOT DE PASSE DE L'ADMINISTRATEUR CYRUS" &&
echo "--PUIS, UNE FOIS LOGUE-------------------------" &&
echo "TAPER createmailbox user.$1 PUIS exit POUR QUITTER" &&
cyradm --user cyrus 127.0.0.1 &&
echo "MAILBOX DE $1 OPERATIONELLE" &&
/etc/init.d/saslauthd restart
```

La syntaxe d'execution du script est:

```
julien@linuxwall:~$ ./creation_mailbox nom_utilisateur
```

C'est le daemon saslauthd qui gére l'authentification des utilisateurs dans la base de données salsdb2.

Maintenant, il faut aller configurer Cyrus-IMAP pour qu'il utilise TLS. Cette configuration se fait dans les fichier "/etc/cyrus.conf" et "/etc/imapd.conf".

Dans le fichier "cyrus.conf", on définit les services que vas utiliser le serveur cyrus. Un détail important: on ne peut pas utiliser IMAPS pour l'administration du serveur cyrus,

il faut donc laisser un accés IMAP non sécurisé pour localhost.

```
#extrait du fichier cyrus.conf
SERVICES {
   # --- Normal cyrus spool, or Murder backends ---
   # add or remove based on preferences
   #imap cmd="imapd -U 30" listen="imap" prefork=0 maxchild=100
   imaps cmd="imapd -s -U 30" listen="imaps" prefork=0 maxchild=100
   #pop3 cmd="pop3d -U 30" listen="pop3" prefork=0 maxchild=50
   #pop3s cmd="pop3d -s -U 30" listen="pop3s" prefork=0 maxchild=50
   imaplocal cmd="imapd -C /etc/imapd-local.conf" listen="127.0.0.1:imap"
prefork=0

   # At least one form of LMTP is required for delivery
   # (you must keep the Unix socket name in sync with imap.conf)
   lmtpunix cmd="lmtpd" listen="/var/run/cyrus/socket/lmtp" prefork=0
maxchild=20
```

La ligne "imaps" active imap sécurisé TLS sur le port 993. La ligne "imaplocal" permet de continuer à administrer la base sasldb2 en local. Enfin, "lmtpunix" désigne quel est l'agent LMTP qui délivre les messages au daemon LMTPD.

Le fichier "imapd.conf" contient les paramètres du serveur IMAP. Voici quelques extraits de ce fichier contenant les paramètres importants:

```
#extrait du fichier cyrus.conf
#Nom du serveur
servername: linuxwall.linuxwall.homelinux.org
# Which partition to use for default mailboxes
defaultpartition: default
partition-default: /var/spool/cyrus/mail
admins: cyrus
# No anonymous logins
allowanonymouslogin: no
sasl_mech_list: digest-md5
#
# SSL/TLS Options
#
#Certificat signe du serveur imap
tls_imap_cert_file: /etc/postfix/mail_signed_cert.pem
#clé privée du serveur imap
tls_imap_key_file: /etc/postfix/mailkey.pem
#certificat du CA
tls_ca_file: /etc/postfix/cacert.pem
```

La configuration de Cyrus-IMAP est terminée.

Il nous reste à éditer le fichier "master.cf" de Postfix pour qu'il prenne en compte Cyrus-IMAP. On ajoute les lignes suivantes dans le fichier:

```
# ==========================================================================
# service type  private unpriv  chroot  wakeup  maxproc command + args
#               (yes)   (yes)   (yes)   (never) (100)
# ==========================================================================
# only used by postfix-tls
tlsmgr    fifo  -       -       n       300     1       tlsmgr
#------------Cyrus support----------------------
cyrus   unix    -       n       n       -       -       pipe
```

La configuration du serveur Linuxwall est terminée. Aprés avoir configuré un client mail (Thunderbird: *http://juliensqt.free.fr/perso/postfix/*), nous constatons que le client demande l'acceptation d'un certificat au lancement. Le sniff des communications nous montre que le MUA utilise bien la commande STARTTLS:

```
Frame 79 (64 bytes on wire, 64 bytes captured)
Ethernet II, Src: 02:00:02:00:00:00, Dst: 34:8a:20:00:02:00
Internet Protocol, Src Addr: 62.147.55.4 (62.147.55.4), Dst Addr:
82.237.216.204 (82.237.216.204)
Transmission Control Protocol, Src Port: 1985 (1985), Dst Port: smtp
(25), Seq: 21, Ack: 141, Len: 10
    Source port: 1985 (1985)
    Destination port: smtp (25)
    Sequence number: 21    (relative sequence number)
    Next sequence number: 31    (relative sequence number)
    Acknowledgement number: 141    (relative ack number)
    Header length: 20 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 8620
    Checksum: 0xdcf5 (correct)
    SEQ/ACK analysis
Simple Mail Transfer Protocol
    Message: STARTTLS\r\n
```

Le serveur répond par l'engagement de la procédure TLS:

```
Frame 80 (78 bytes on wire, 78 bytes captured)
Ethernet II, Src: 34:8a:20:00:02:00, Dst: 02:00:02:00:00:00
Internet Protocol, Src Addr: 82.237.216.204 (82.237.216.204), Dst Addr:
62.147.55.4 (62.147.55.4)
Transmission Control Protocol, Src Port: smtp (25), Dst Port: 1985
(1985), Seq: 141, Ack: 31, Len: 24
    Source port: smtp (25)
    Destination port: 1985 (1985)
    Sequence number: 141    (relative sequence number)
    Next sequence number: 165    (relative sequence number)
    Acknowledgement number: 31    (relative ack number)
    Header length: 20 bytes
    Flags: 0x0018 (PSH, ACK)
    Window size: 5840
    Checksum: 0xa56b (correct)
    SEQ/ACK analysis
Simple Mail Transfer Protocol
    Response: 220 Ready to start TLS\r\n
```

Considérations de sécurité

Maintenant que notre serveur est configuré, il est important de percevoir les faiblesses d'une telle architecture.

Tout d'abord, il est important de noter que SMTP n'est pas un protocole de type bout en bout. Ainsi, si un binôme client/serveur décide d'activer le chiffrement par TLS, ils ne peuvent sécuriser le transport du message du MUA de départ jusqu'au MUA de destination. Si le MUA et/ou le MTA de destination ne supportent pas TLS, le message sera transmis en clair sur ces parties du réseaux.

D'autre part, une attaque de type man-in-the-middle peut être lancée en supprimant la réponse `220 Ready to start TLS` envoyée par le serveur. De cette façon, le client n'essaiera pas d'ouvrir une session TLS.

De plus, lorsqu'un mail est stockée sur un MTA de destination, il apparait en clair dans le système de fichiers. Ainsi, un intrus (ou même un administrateur) peut facilement lire le mail et accèder aux données qu'il contient. C'est ici que S/MIME trouve toute son utilité. En effet, en utilisant ce protocole, les messages sont chiffrés sur le serveur et un intrus, comme un administrateur, ne peut pas les lire.

Enfin, une boite à lettre de type Cyrus-IMAP est sensible aux attaques de type brute force. Un intrus peut essayer de forcer l'accès par mot de passe en utilisant soit un dictionnaire soit un générateur aléatoire de mot de passe. Ce type d'attaque est bien entendu dépendant de la taille des mots de passes, il apparait donc indispensable d'imposer aux utilisateurs des mots de passes forts (7 caractères: 4 lettres, 2 chiffres et 1 caractère spécial).

Conclusion

Comme nous avons pu l'établir au cours de ce projet, la sécurisation des communications par TLS est efficace mais connait certaines faiblesses trés importantes. Ainsi, il apparait trés difficile de garantir une sécurisation complète du transport des messages de part la disparitée des configurations de serveurs de messagerie sur Internet.

Pour une sécurisation de type bout en bout, il est préférable de se tourner vers des solutions telles que S/MIME ou encore PGP. Néanmoins, par rapport à PGP, S/MIME à l'avantage de décharger l'utilisateur de la gestion des clés. Il est en effet trés simple d'ajouter de nouveaux certificats publics signés à un MUA, ce qui n'est pas forcément le cas pour des clés PGP.

Toutefois, l'énorme avantage du protocole TLS reste sa complète transparence vis à vis des couches supérieures du modèle OSI. La théorie relative à TLS présentée au cours de ce rapport reste donc vrai pour toutes les applications qui choisirons d'implémenter ce protocole.

Enfin, la configuration déployée ici est prévue pour fonctionner sur des réseaux importants et est donc capable de tenir une augmentation de la charge SMTP et IMAP. La prochaine étape de ce projet sera d'ailleurs sa réalisation en milieu professionnel au cours de mon stage de fin de Licence.