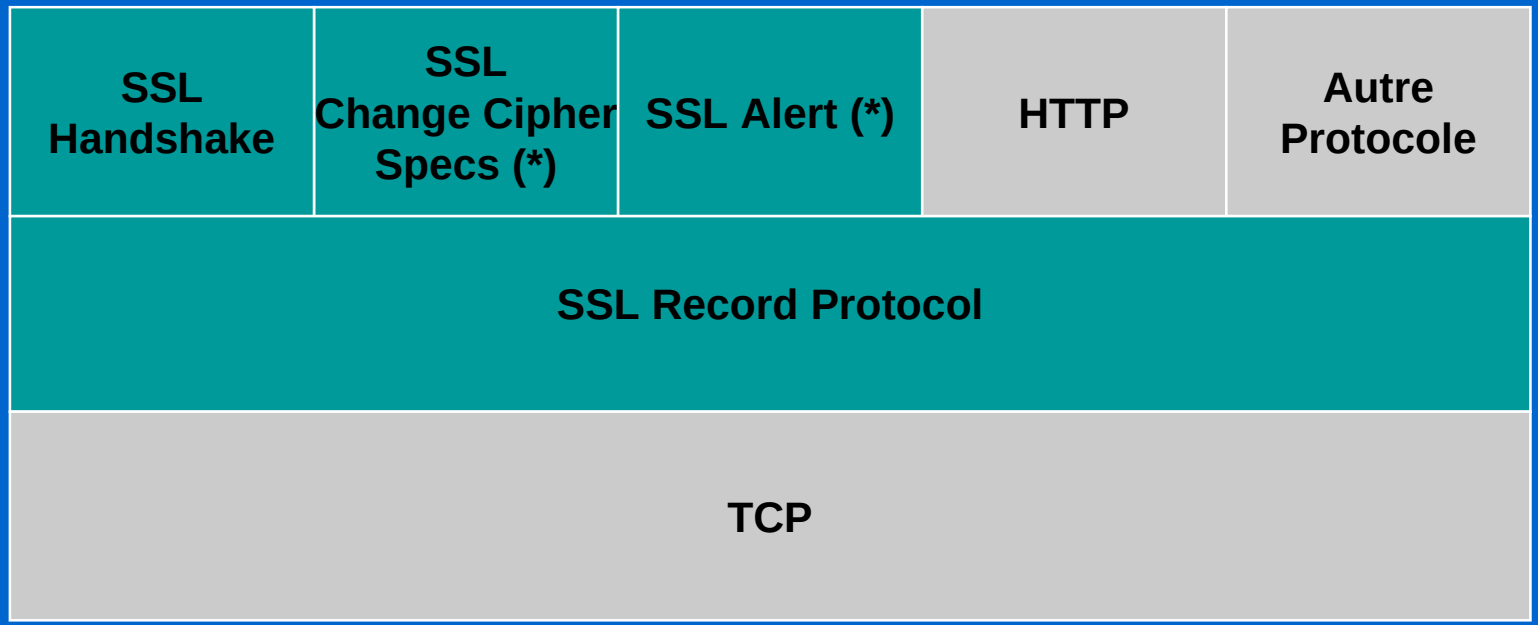
L'accès sécurisé aux données

Les applications du chiffrement

Le protocole SSL - objectifs

- Sécuriser des protocoles existants (HTTP, SMTP, IMAP, …) de façon transparente
- Caractéristiques
  - **confidentialité** via le chiffrement, **intégrité** via empreintes (*Message Authentication Code*), **authentification** via certificats X.509 (serveur et/ou client)
  - **rapidité** : usage de clés de sessions
  - **extensibilité**, **interopérabilité** : négociation des algorithmes de chiffrement
  - **résistance** aux attaques usuelles

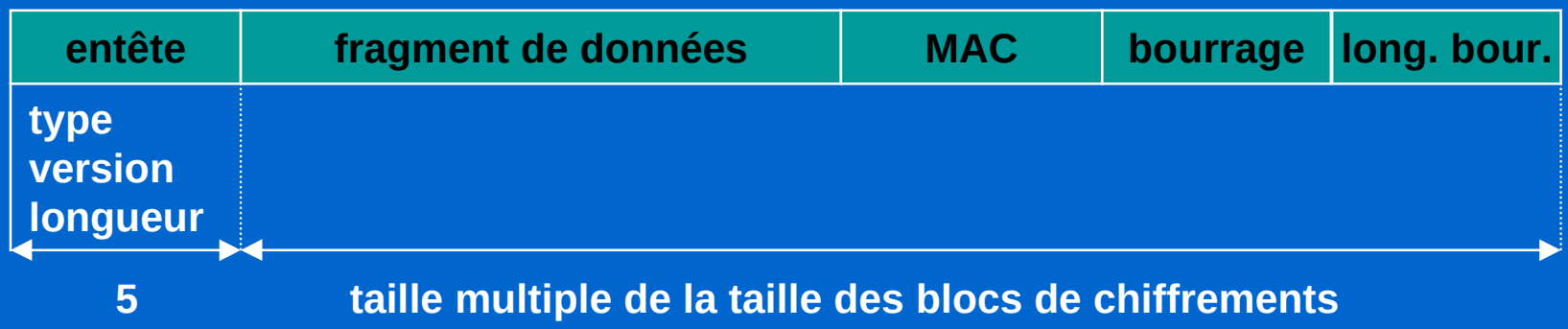
Le protocole SSL v3 - architecture



(*) Inexistant en SSL v2

SSL v3 - Record Protocol

- **fragmente** le flot de données en unités de 16K - 5 octets d'entête et les **compresse** (non implémenté)
- calcule une **empreinte** (MAC) et la rajoute au fragment
- rajoute éventuellement des caractères de bourrage
- **chiffre** le résultat avec une clé de session



SSL v3 - Record Protocol

- les algorithmes utilisés pour la compression, le chiffrement et pour le calcul du MAC ainsi que la clé de chiffrement sont déterminés dans la phase de connexion (*SSL Handshake Protocol*).

- le calcul du MAC inclut, entre autres, le numéro du fragment et une clé secrète dérivée de la clé maître de session. L'empreinte est chiffrée avec les données ⇒ pour l'homme du milieu :
  - impossible d'insérer un faux fragment
  - impossible de supprimer un fragment

SSL v3 - Handshake Protocol

- implémente la phase de connexion
- s'appuie sur le *SSL Record Protocol* avec au départ compression, chiffrement, empreinte = identité.
- rôle :
  - détermine les spécifications de chiffrement (*Cipher Specs*), i.e. les algorithmes pour le calcul des MAC, pour le chiffrement des données. Dans le futur : algorithme de compression
  - échange et vérifie les certificats du serveur et/ou du client
  - génère une clé de session maître et en dérive une clé de chiffrement pour le serveur, une clé de chiffrement pour le client, des clés pour sécuriser les MAC

SSL v3 - Handshake Protocol

- Tous les messages commencent par un octet indiquant le **type**, puis trois octets donnant la longueur du message, puis des **données spécifiques** du message. Le tout est acheminé par le *SSL Record Protocol*.

- Certaines étapes sont optionnelles :
  - authentification du client, si le serveur ne l'oblige pas
  - échange de clé, si le client et le serveur ont les informations nécessaires dans leur cache.

SSL v3 - Handshake Protocol

Change Cipher	*Handshake Protocol*	c ↔ s	*Informations échangées*
Specs	**ClientHello**	→	version**, nbre aléatoire (256 bits),** ident session**, liste de spécifications de chiffrement**
	ServerHello	←	version, nombre aléatoire (256 bits), ident session, spécification de chiffrement
	Certificate	←	certificat du serveur, chaîne de CA
	Certificate Request	←	types de certificats acceptables, liste de CA autorisés.
	ServerHeloDone	←	
	Certificate	→	certificat du client, chaîne de CA.
	ClientKeyExchange	→	version + 46 octets aléatoires chiffrés avec la clé publique du serveur
	CertificateVerify	→	empreinte clé maître + traces des échanges précédents.
Change →	**Finished**	→	empreinte clé maître + traces des échanges précédents.
Change ←	**Finished**	←	empreinte clé maître + traces des échanges précédents

SSL v3 - La clé maître

- Dans la phase ClientKeyExchange, le client a fabriqué une **pré-clé maître** : 2 octets de la version + 46 octets aléatoires. Cette clé est transmise de façon sécurisée au serveur.

- Le client et le serveur calculent la **clé maître**. C'est une empreinte mêlant MD5 et SHA, appliquée trois fois, avec en entrée :
  - la pré-clé maître
  - les nombres aléatoires des ClientHello et ServerHello
  - les chaînes "A", "BB", "CCC"

SSL v3 - Les clés de session

- Une connection SSL v3.0 utilise quatre clés de 128 bits, toutes dérivées de la clé maître :
  - deux clés secrètes de chiffrement : une pour le serveur, une pour le client
  - deux clés secrètes pour rendre confidentiel le calcul des MAC : une pour le serveur, une pour le client

- Pour respecter la réglementation US sur l'exportation de logiciels de chiffrement, les versions "export" n'ont que 40 bits sur les 128 qui sont réellement secrets.

SSL v3 - Optimisations

- Le déroulement complet du *Handshake Protocole* est très **pénalisant**. Dans le cas de HTTPS : une page WWW, 10 images => 11 connexions SSL
- **Optimisation** : le client et le serveur maintiennent un cache indexé par un identificateur de session (ID)
  - **ClientHello** : le client envoie l'ID au serveur.
  - **Serverhello** :
    - l'ID est dans le cache sur serveur => confirme l'ID, pas d'échange de certificats, pas d'échange de clés
    - l'ID n'est pas dans le cache => protocole complet
- **Apache/mod_ssl** : cache dans `/var/run/ssl_cache.db` (à effacer si on change le certificat du serveur)

HTTPS, Apache et mod_ssl

- **HTTPS** est **HTTP** sur une connexion **SSL**. Utilise un port (443 par défaut) et des URL particuliers :
  - HTTP : le client envoie une requête "**`GET url HTTP/1.1`**"
  - HTTPS : le client SSL envoie un message SSL **`ClientHello`**

  => utilisation d'URL de la forme "**`https://www.domaine.fr`**"

- Apache n'inclut pas SSL d'origine => ajouter le module mod_ssl (**`http://www.modssl.org`**)
  - installer openssl : bibliothèque SSL, génération de certificats
  - installer mod_ssl : extension de l'API Apache + module dynamique
  - configure mod_ssl

Installation de openssl

- télécharger à partir de **http://www.openssl.org**.
- extraire dans **/sources/openssl-0.9.3a**
- aller dans **/sources/openssl-0.9.3a** et taper :

```
# ./Configure linux-elf
# make
# make install
```

- installe une hiérarchie dans **/usr/local/ssl**

Installation de mod_ssl sans Apache

- télécharger à partir de **http://www.modssl.org**.
- extraire dans **/sources/mod_ssl-2.3.5-1.3.6**
- aller dans **/sources/mod_ssl-2.3.5-1.3.6**. Taper :

```
# ./configure linux-elf \
    --with-apache=/sources/apache_1.3.6 \
    --with-ssl=/sources/openssl-0.9.3a \
    --prefix=/etc/httpd --enabled-shared=ssl \
    --enabled-shared=max \
    --with-layout=config.layout:Apache
```

Recompilation d'Apache

- aller dans **/sources/apache-1.3.6**. Taper :

```
# make
# make install
```

- installe un exemple de fichier de configuration dans **/etc/httpd/conf/httpd.conf.default**.

- les lignes suivantes sont extraites de ce fichier. Elles sont à rajouter au fichier **httpd.conf** courant.

Configuration de mod_ssl : `httpd.conf`

```
LoadModule ssl_module /usr/lib/apache/libssl.so
AddModule mod_ssl.c

Listen 80
Listen 443

SSLSessionCache dbm:/var/run/ssl_scache
SSLSessionCacheTimeout  300
SSLMutex        file:/var/run/ssl_mutex
SSLRandomSeed   startup builtin
SSLRandomSeed   connect builtin
SSLLog          /var/www/logs/ssl_engine_log
SSLLogLevel     info
```

Configuration de mod_ssl : `httpd.conf`

```
<VirtualHost _default_:443>
    DocumentRoot            /var/www/ssl_htdocs
    ErrorLog                /var/www/logs/ssl_error_log
    TransferLog             /var/www/logs/ssl_access_log
    SSLEngine               on
    SSLCertificateFile      conf/ssl.key/server.crt
    SSLCertificateKeyFile conf/ssl.key/server.key
    SSLCACertificatePath  conf/ssl.crt
    SSLCACertificateFile  conf/ssl.crt/ca-bundle.crt
    SSLVerifyClient         require
    SSLVerifyDepth          5
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive \
                                   ssl-unclean-shutdown
</VirtualHost>
```

Génération du certificat du serveur

- En principe, on génère une **requête de certificat** (*Certificate Signing Request*) pour le serveur qu'on fait signer par une autorité de certification. On peut aussi fabriquer un **certificat autosigné** pour le serveur :

```
# cd /etc/httpd/conf ; mkdir ssl.key
# cd ssl.key
# openssl req -new -x509 -days 365 -nodes \
          -keyout server.key -out server.crt
```

Entrez les champs pays, ville, organisation, division, nom, mél

```
# chmod 600 server.key
```

Installation des autorités de certification

- Installation des CA courants :

```
# cd /sources/mod_ssl-2.3.5-3.6/pkg.sslcfg
# cp ca-bundle.crt /etc/httpd/conf/ssl.crt
# cp Makefile.crt \
            /etc/httpd/conf/ssl.crt/Makefile
```

- Ajouts de CA :

```
# cp nouveau-ca.crt /etc/httpd/conf/ssl.crt
# cd etc/httpd/conf/ssl.crt
# make
```

Première connexion HTTPS !

- Redémarrer le serveur Apache

```
# apachectl configtest
# apachectl restart
```

- Vérifier la configuration SSL
  - depuis un navigateur, **https://www.domaine.fr**
- Rajouter des accès
  - protégés par mots de passe
  - contrôlés par les certificats des clients

La protection par mot de passe

- Rappel : authentification basique dans HTTP consiste à fabriquer une requête HTTP comme suit :

codage de "utilisateur:motdepasse**" en base 64**

```
GET /pages/confidentielles HTTP/1.0
Authentication: Basic xxxxxxxxxxxx
```

- Idem dans HTTPS, mais la requête est chiffrée par la couche SSL => le mécanisme d'authentification basique devient sûr.

La protection par mot de passe

- Créer un fichier **`httpd.passwd`**, contenant :

```
utilisateur1:XYabcdefghijk
utilisateur2:AB12345678901
```

- Rajouter dans **`httpd.conf`**, dans la section **`<virtualhost _default_:443>`** les lignes :

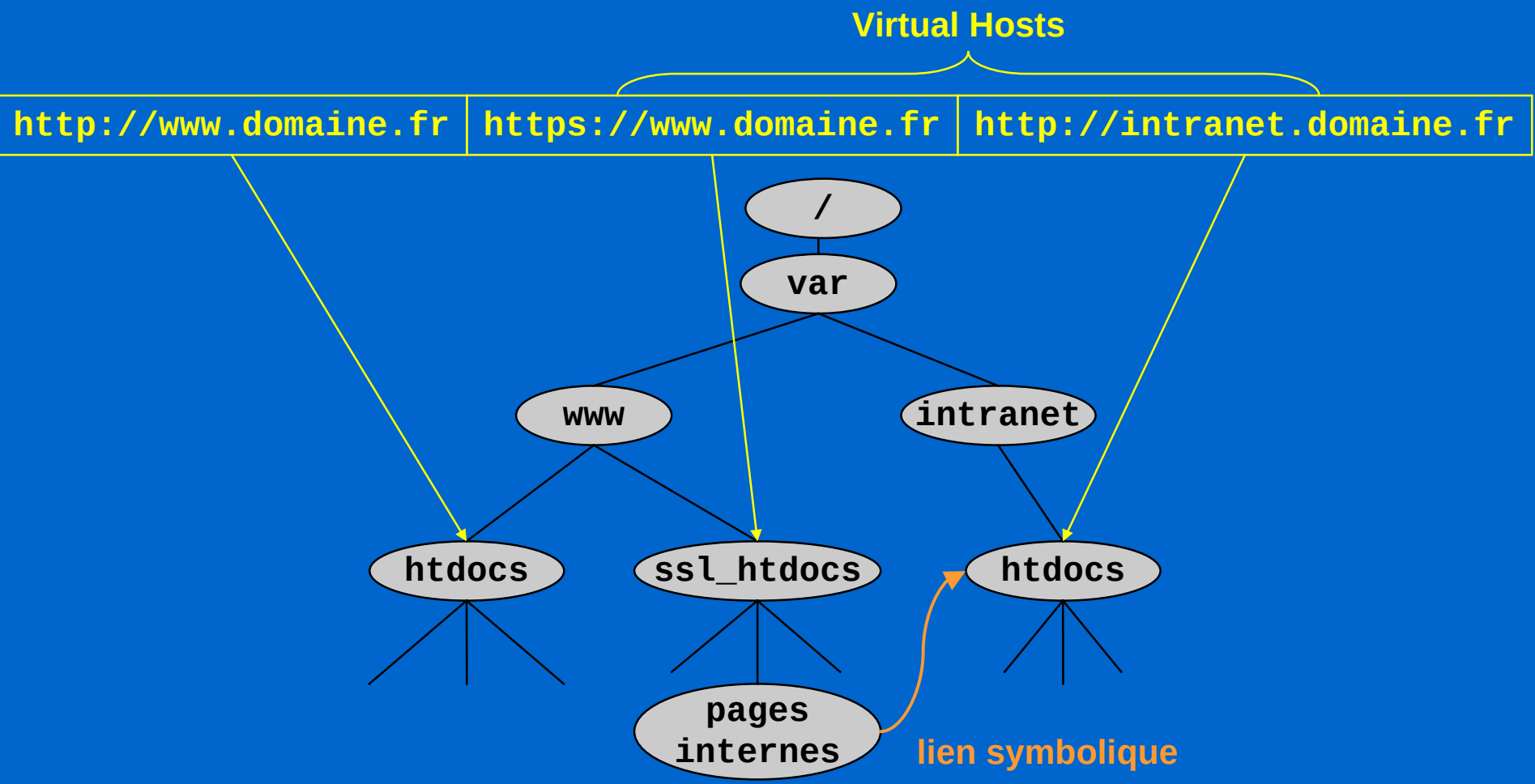
```
<Location /pages/confidentielles>
    AuthType        Basic
    AuthName        Pages-Confidentielles
    AuthUserFile    /etc/httpd/conf/httpd.passwd
    require         valid-user
</Location>
```

La protection par certificat

- Rajouter dans **`httpd.conf`**, dans la section **`<virtualhost _default_:443>`** les lignes :

```
<Location /pages/confidentielles>
    SSLVerifyClient require
    SSLVerifyDepth  5
    SSLRequireSSL
    SSLRequire  %{SSL_CLIENT_S_DN_C} eq "FR"and \
                %{SSL_CLIENT_S_DN_O} eq "Ma Boite"
</Location>
```

Exemple d'utilisation



Les scripts CGI

- Rajouter dans **`httpd.conf`**, dans la section **`<virtualhost _default_:443>`** les lignes :

```
<Location /scripts/securises>
    SSLVerifyClient require
    SSLVerifyDepth  5
    SSLRequireSSL
    Options         +ExecCGI
    SetHandler      cgi-script
</Location>
```

si on souhaite authentifier le client par un certificat

Les scripts CGI et SSL

- La requête est chiffrée, le résultat aussi
  - on peut envoyer et recevoir des données confidentielles via des formulaires HTML
- Apache complète l'environnement du script avant de le lancer :
  - variables standards : **REMOTE_ADDR**, **HTTP_USER_AGENT**, ...
  - variables SSL : **SSL_CLIENT_S_DN_*xx***, **SSL_CLIENT_I_DN_*xx***, **SSL_CIPHER_USEKEYSIZE**, …
- Attention *: SSL sécurise le transfert, pas l'application*

Une application : le Webmail

- **Objectif** : accéder à sa messagerie de façon relativement sûre, depuis des environnements non sûrs, sans installer de logiciels particuliers

- **Solution** : **webmail**
  - **sur le client** : navigateur WWW avec SSL : MSIE 3.0, Netscape 1.0, Lynx 2.8 ou supérieurs
  - **sur le serveur :** serveur HTTPS, script CGI
    - client IMAP du serveur de messagerie
    - génère des pages WWW avec le contenu de la BAL et des dossiers
    - exemples : IMP / Apache, IMHO / Roxen

Autres utilisations de SSL

- **S/ IMAP** : IMAP sur SSL, port 993
  - **exemples de clients** : Netscape Messenger, Eudora Pro 4
  - **exemples de serveurs** : tout serveur IMAP avec WrapSSL
  - **caractéristiques** : meilleure ergonomie que le Webmail, bien adapté aux personnes "excentrées" : antennes d'un service, portables raccordés par fournisseurs d'accès Internet public
- **S/ SMTP** : SMTP sur SSL, port 465
- **S/ NNTP** : NNTP sur SSL, port 563
- **...**

S/ MIME

S/MIME utilise le format et des types **MIME** pour

- **Signature** électronique de messages
  Type MIME du message : **`application/signed`**
  Deux parties :
  - le message en clair : type MIME classique
  - la signature : type MIME **`application/pkcs7-signature`**

- **Chiffrement** des messages
  Type MIME du message : **`application/pkcs7-mime`**
  - encodé en base64

- Les deux : signature, puis chiffrement ou inversement

S/ MIME- exemple de signature

```
MIME-Version: 1.0
Subject: message avec signature
Content-Type: multipart/signed;
     protocol="application/x-pkcs7-signature"; micalg=sha1;
     boundary="------------@@@"

--------------@@@
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: quoted-printable

Ceci est un message sign=E9.
--------------@@@
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIIF2wYJKoZIhvcNAQcCoIIFzDCCBcgCAQExCzAJBgUrDgMCGJLKJH
…
cT/jBEirPG0M3z+4O9idrhSHftctWDCgaxyo1tNw4pQ/rG==
--------------@@@--
```

S/ MIME- exemple de message chiffré

```
MIME-Version: 1.0
Subject: chiffrement du message
Content-Type: application/pkcs7-mime; name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"
Content-Description: S/MIME Encrypted Message

MIAGCSqGSIb3DQEHA6CAMIACAQAxgf0wgfoCAQAwgaMwgZ0xCzA
VQQHEwdUYWxlbmNlMTgwNgYDVQQKEy9DTlJTIC0gRGFpbmUgZXQ
…
9RZB2CTfXWCGHOCw0ZdVxfmK9zJoUSkSjwQoXCHb2YkMk9PR+17
wqA8znuswA55CzxsIgQIJ0L/K5GtRn4E1zwAAAAAAAAAAAAA==
```

S/ MIME- fonctionnement

- **Signature** PKCS#7 (RFC 2315):
  - calcul de l'**empreinte** du message, **chiffrement** avec la clé privée de l'émetteur
  - **certificat** de l'émetteur
- **Vérification** :
  - **vérification** du **certificat** de l'émetteur, **comparaison** du champ **Email=** avec le champ **From**
  - calcul de l'**empreinte** du message, **comparaison** avec **l'empreinte déchiffrée** avec la clé publique de l'émetteur
- **Effet de bord** :
  - le certificat de l'émetteur est conservé dans l'agent de messagerie du destinataire => le chiffrement devient possible

S/ MIME- fonctionnement

- **Chiffrement** PKCS#7 (RFC 2315):
  - chiffrement du message avec la clé publique contenue dans le certificat du destinataire

- **Comment obtenir le certificat du destinataire ?**
  - le destinataire envoie un message signé
  - on interroge un annuaire LDAP

- **Et pour les listes de diffusion ?**
  - émetteur chiffre avec la clé publique de la liste de diffusion
  - le robot de la liste déchiffre et chiffre avec la clé publique de chaque destinataire

Installation du certificat de l'utilisateur

- Importation :
  - à partir du serveur de l'autorité de certification
  - à partir d'un fichier PKCS#12 :
    - dans **Communicator** : *Sécurité -> Vos certificats -> importe*r.
    - dans **Explorer** : *Affichage -> options Internet -> Contenu -> Personnel*.
- Exportation :
  - pour réinstaller le certificat sur une autre machine

Installation du certificat de l'utilisateur

- Communicator :
  - le certificat est dans **`cert7.db`**
  - la clé privée dans **`key7.db`** : ce fichier est chiffré par une clé (*mot de passe Communicator*)
  - ces fichiers peuvent être effacés : ils sont automatiquement reconstruits

- Explorer :
  - utilise la base de registre
  - la clé privée est chiffrée. Par quelle clé ?

S/ MIME

- **S/ MIME + Certificats + Annuaire LDAP = solution complète pour une messagerie sécurisée ?**
  - les logiciels existent, fonctionnent, sont gratuits
  - mais, ...