TABLE DES MATIÈRES

Mots clés	Bioserveur, Medxfer, protocole, sécurisé, certificat	
Révision	**Date**	**Auteur : Action**
1	03/12/07	CVT : Rédaction de la spécification externe du protocole sécurisé Medxfer

Documents de référence		
Type	**Référence**	**Commentaire**

1 Références

- http://tools.ietf.org/html/rfc2068 : « Hypertext Transfer Protocol -- HTTP/1.1 »
- http://tools.ietf.org/html/rfc2045 : « Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies »

2 Conventions

3 Objet

L'objectif de ce projet est de définir un protocole de transfert de fichiers sécurisé de manière à pouvoir pallier les problèmes liés à HPRIM Net.

Ce protocole permettra une communication directe entre un client et un serveur.

La finalité de ce projet est de devenir Open Source pour permettre aux éditeurs d'utiliser ce protocole sans aucune contrainte liée au code.

Obligatoire:

- utilise des protocoles standards
- possibilité d'une authentification forte basée sur x-509 (voir PKI GIP-CPS)
- le client envoie les fichiers au serveur après s'être authentifié.
- signature du transfert de manière à garantir l'intégrité et l'origine des fichiers (le fichier ne doit pas être exploitable avant que le transfert total et les vérifications de sécurité aient été effectués).
- facilement déployable au travers des firewall
- développer un projet opensource avec des APIs et un client et un serveur compatibles sur toutes les plates formes (unix, windows, linux, macosX, AS400)
- utilisable coté logiciel médecin
- gestion des distributions de versions (applet java par ex)

Optionnellement :

- possibilité de reprendre un transfert interrompu dans le cas de gros fichiers
- possibilité de upload ou de download
- compresser les fichiers avant le transfert

Remarques :

HSC déconseille ftp sur SSL en raison de la difficulté pour paramétrer les firewall:

http://www.hsc.fr/ressources/breves/ftp-ssl.html.fr:

"l'utilisation de SSH (avec sftp) ou HTTPS (avec par exemple des outils en ligne de commande et serveurs d'upload spécialisés) ont sans doute un meilleur avenir"

4 Fonctionnalités

4.1 Description du service

Le service fourni est le transfert de fichiers avec authentification forte du client et du serveur et contrôle d'intégrité du transfert.

Pour cela, on s'appuie sur des standards; plus précisément:

Protocole	*Caractéristiques*
HTTPS	protocole standard largement utilisé (par tous les sites Web sécurisés) passe facilement à travers les firewall chiffrage des données pendant le transfert authentification forte du serveur par certificat X509 authentification forte du client par certificat X509 gestion de listes de révocation de certificats X509 transfert bidirectionnel de fichiers entre le client et le serveur (HTTP v1.1) compression possible des données lors de la transmission ***Ne garantit pas l'intégrité des données échangées***
MIME	Structuration des données échangées Ce format est utilisé pour associer une signature SHA1 à chaque fichier échangé afin de valider son transfert et garantir l'intégrité des données.

Les fonctions attendues sont:

- envoi d'un fichier sur le serveur avec contrôle d'intégrité (upload),
- récupération d'un groupe de fichiers à partir du serveur avec contrôle d'intégrité (download).
- récupération de la liste de groupes de fichiers disponibles sur le serveur.

Ces différentes fonctions ne sont réalisables qu'en enchaînant plusieurs requêtes HTTPS. Nous décrivons ci- dessous:

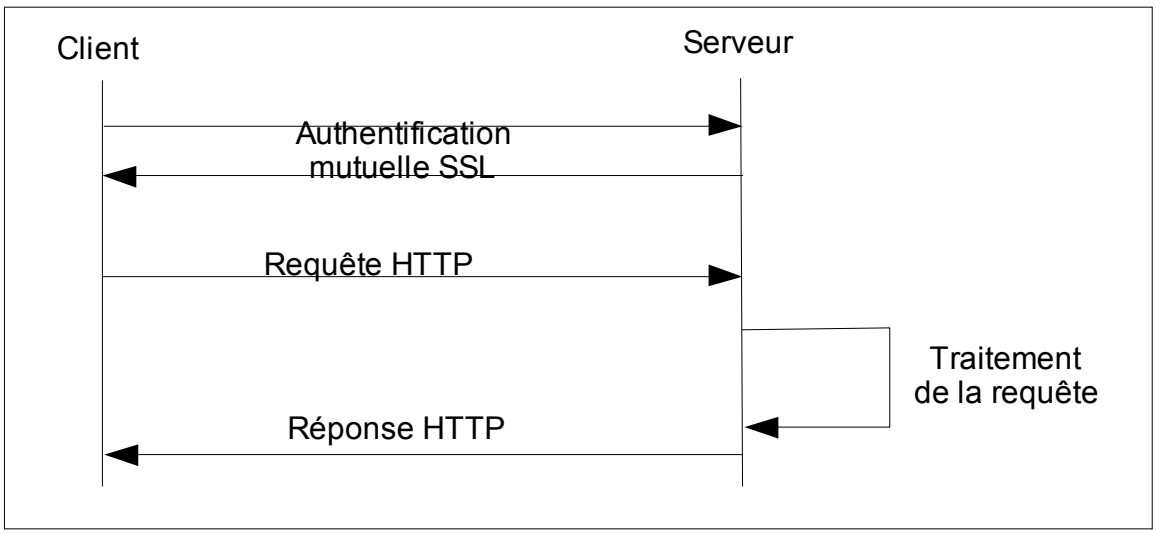
- le déroulement d'une requête HTTPS,
- les séquences de requêtes HTTPS nécessaires pour réaliser les différentes fonctions.

4.1.1 Description générale d'une requête HTTPS

Une requête HTTPS se déroule en deux phases:

- une phase d'authentification basée sur le protocole SSL
- une phase d'échange de données utilisant le protocole HTTP.

Le diagramme ci-dessous décrit le déroulement d'un requête.



4.1.1.1 Authentification SSL

Le serveur Web DOIT être configuré pour n'accepter la connexion qu'après réception d'un certificat client valide.

Le serveur DOIT gérer une liste de révocation de certificats.

Cette phase a pour buts de :

- valider par le client de l'identité du serveur (définie par son certificat X509), le client dispose pour cela du certificat de l'autorité de certification (CA) ayant signé le certificat du serveur;
- valider par le serveur de l'identité du client (définie par son certificat X509), le serveur dispose des certificats des clients autorisés ainsi que d'une liste de révocation de certificats (« CRL ou Certificate Revocation List »);
- échanger des clés pour le chiffrement des échanges HTTP.

La version 1.0 du protocole HTTP prévoit qu'une nouvelle connexion entre le client et le serveur est établie à chaque requête. Ceci implique pour HTTPS une phase d'authentification, et donc d'échange de certificats à chaque requête.

La version 1.1 du protocole HTTP, introduit une optimisation (« pipelining ») qui permet d'exécuter plusieurs requêtes; on a alors une seule phase d'authentification pour ce groupe de requêtes. Il faut néanmoins noter que, pour ne pas saturer les ressources sur les serveurs, ceux-ci imposent des limites sur des paramètres tels que le nombre maximum de requêtes par connexion, la durée maximum d'une connexion ou le délai entre deux requêtes successives.

Le client PEUT donc utiliser le « pipelining » d'HTTP 1.1 pour limiter le nombre d'authentifications mais il DOIT être capable de gérer plusieurs séquences d'authentification lorsque plusieurs requêtes sont nécessaires pour réaliser une des fonctions prévues dans cette spécification.

4.1.1.2 Requête HTTP

Lors de la phase d'authentification SSL, les informations d'authentification du client sont enregistrées par le serveur. Ces informations sont accessibles lors du traitement de la requête. Le mode d'accès à ces informations dépend du serveur Web.

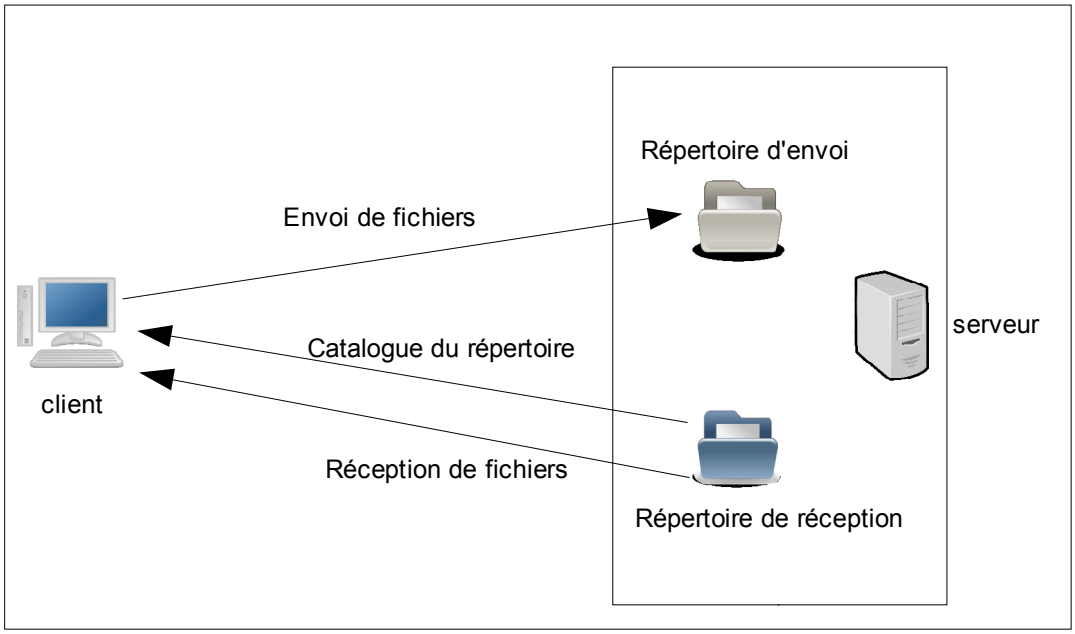
Dans le cas du serveur Apache, ces informations sont accessibles sous forme de variables d'environnement (cf § « Variables d'environnement Apache/mod_ssl »). La variable SSL_CLIENT_S_DN_x509 définit le DN (« distinguished name ») du client.

4.2 Protocole de communication

Ce chapitre décrit les séquences d'échanges entre le client et le serveur pour réaliser les fonctions suivantes:

- envoi d'un fichier au serveur,
- réception d'un groupe de fichiers à partir du serveur,
- catalogue du répertoire sur le serveur.

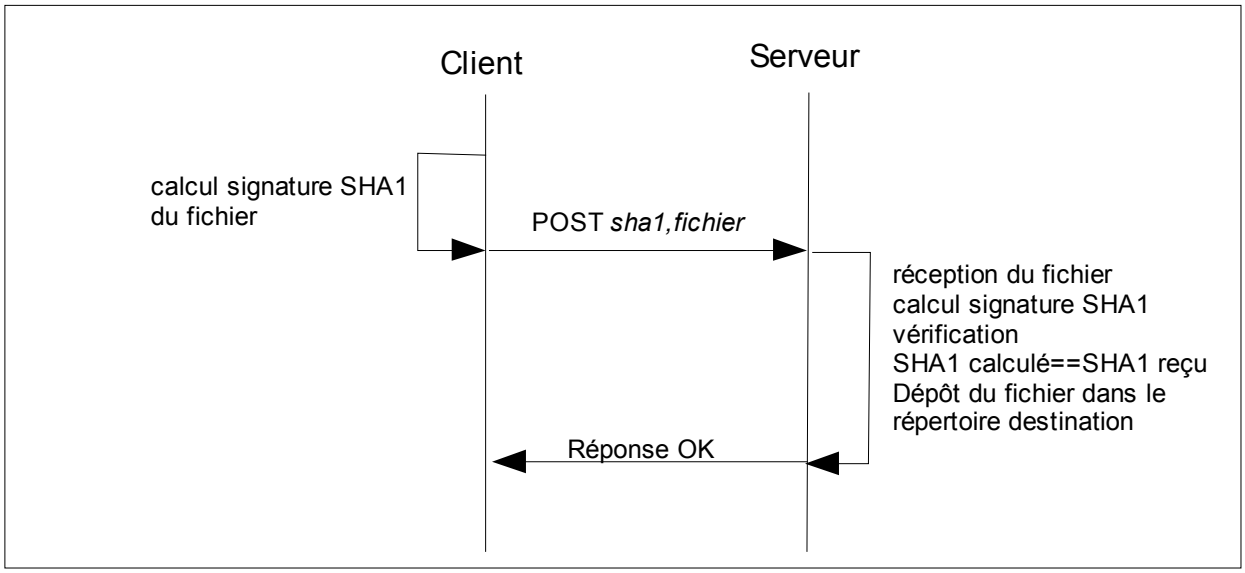
Cette spécification ne prévoit pas de fonctions pour naviguer dans une arborescence de répertoires. Elle suppose que chaque client dispose d'un répertoire personnel sur le serveur pour l'envoi de fichiers et un autre pour la réception. Le serveur utilise les informations obtenues lors de la phase d'authentification (cf §« Authentification SSL ») pour identifier le client et lui associer ces deux répertoires personnels.



Cette spécification modélise le transfert de données entre le client et le serveur sous la forme d'envoi et de récupération de fichiers sur des répertoires du serveur. Elle n'implique pas que ces répertoires ont une existence réelle. Ce protocole peut en effet être utilisé par une application sur le serveur qui traite à la volée les données reçues et/ou génère dynamiquement les données transmises à partir d'informations qui ne sont pas stockées dans des fichiers (base de données,...).

4.2.1 Envoi d'un fichier avec contrôle d'intégrité (upload)

Le schéma ci-dessous décrit la séquence correspondant à un envoi correct.



Le fichier et sa signature SHA1 sont envoyés dans une même requête POST en utilisant la structure de données MIME suivante:

```
POST /upload.cgi HTTP/1.1
Content-type: multipart/related; boundary=boundary;

--boundary
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="filename"
Content-Transfer-Encoding: base64


         ... contenu de filename ...
--boundary
Content-type: application/sha1-signature
Content-Transfer-Encoding: 7bit

signature_SHA1_de_filename
--boundary--
```

L'URL */upload.cgi* dans la directive POST est donnée à titre d'exemple.

Le contenu du fichier est codé en BASE64 afin de s'affranchir des problèmes de transmission de caractères spéciaux qui pourraient perturber le transfert dans le cas de données binaires.

Si aucune erreur n'est détectée, la réponse du serveur est:

```
HTTP/1.1 200 OK
```

A l'issue du transfert, le fait que le serveur réponde OK (code HTTP 200) indique que le transfert est complet et que la signature SHA1 a été vérifiée.

Le fichier reçu NE DOIT PAS apparaître dans le répertoire de destination tant que son transfert n'est pas complet ET que sa signature SHA1 n'est pas validée. Le serveur ne doit donc pas enregistrer de données dans le répertoire de destination tant que le fichier reçu n'est pas validé; la méthode utilisée dépendra de l'implémentation du service (traitement en mémoire, utilisation d'un répertoire temporaire,...).

Dans le cas où un fichier ayant le même nom existait sur le serveur avant le transfert, le contenu de l'ancien fichier est remplacé par les données reçues si le nouveau fichier reçu est valide. Si le nouveau fichier n'est pas valide, l'ancien fichier est conservé.

Le client peut compresser le contenu du fichier avant envoi. Dans ce cas, les entêtes sont modifiées de la façon suivante:

```
POST /upload.cgi HTTP/1.1
Content-type: multipart/related; boundary=boundary;

--boundary
Content-type: application/sha1-signature
Content-Transfer-Encoding: 7bit

signature_SHA1_de_filename

--boundary
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="filename"
Content-Transfer-Encoding: base64
Content-Transfer-Encoding: gzip


        ... contenu de filename compressé...
--boundary--
```

La compression a pour seul but de réduire le temps de tranfert. Le serveur doit donc décompresser les données reçues avant de les stocker dans le fichier *filename*.

La compression se fait avant la conversion en BASE64.

De plus, la signature SHA1 doit être calculée par le client avant compression et conversion en BASE64 des données et vérifiée par le serveur après décadage BASE64 et décompression. Elle est donc indépendante du mode de transfert (compressé ou non compressé).

4.2.1.1 Gestion des erreurs

Des erreurs peuvent être générées à différentes étapes pendant le transfert:

- à l'ouverture de la connexion:
  - serveur injoignable : provient généralement d'un problème réseau entre le client et le serveur;
  - connexion refusée : le serveur a été contacté mais le service n'est pas disponible, le serveur Web est probablement momentanément indisponible;
  - « timeout » à la connexion : aucune réponse n'a été reçue du serveur lors de l'ouverture de la connexion;
- à l'authentification:
  - certificat serveur invalide : le certificat envoyé par le serveur ne correspond pas à celui qu'attend le client;
  - certificat client invalide : le serveur refuse le certificat envoyé par le client, il n'a peut-être pas encore été ajouté dans la liste des certificats clients autorisés ou a été révoqué;
- en cours de transfert :
  - transfert interrompu prématurément : le serveur a coupé la communication avant que le transfert de données soit terminé;
  - « timeout » à l'envoi :

  la durée maximum définie pour un envoi de données a été atteinte avant que le client ait pu terminer le transfert;
  - « timeout » à la réception de la réponse :

la durée maximum d'attente de réponse de la part du serveur a été atteinte avant que le serveur ne réponde;

- dans la réponse du serveur :
  - conformément au standard HTTP, tout code d'erreur différent de 2xx doit être considéré comme une erreur, le serveur Web peut générer des erreurs autres que celles spécifiques à l'application de transfert de fichier (ex: 404 « Not Found » si le client n'a pas spécifié la bonne URL dans la requête POST);
  - l'application de transfert de fichier utilise le code HTTP 409 (« Conflict ») pour indiquer qu'elle a détecté une erreur, la cause de l'erreur est détaillée dans la réponse:

```
HTTP/1.1 409 Conflict
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<transfer_error>
  <error_code>code_erreur</error_code>
  <error_message>message_erreur</error_message>
</transfer_error>
```

La liste des codes d'erreurs et les messages associés sont les suivants (les messages sont spécifiés en anglais) :

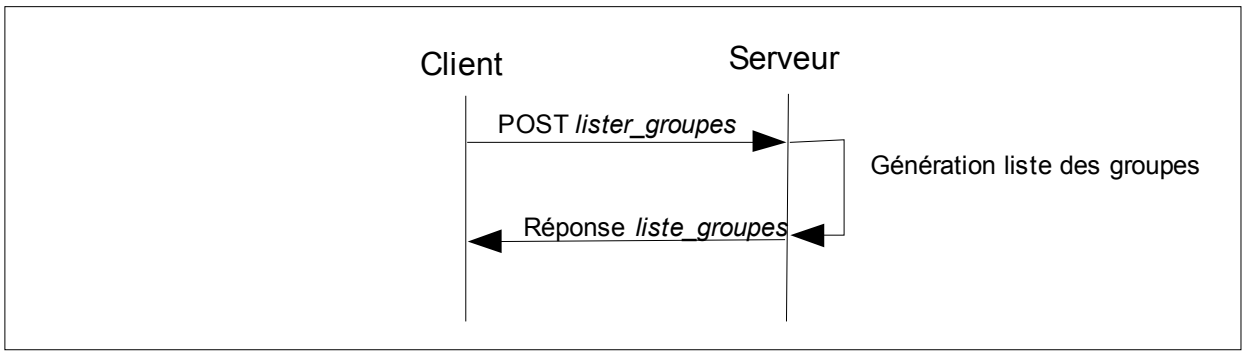
code_erreur	***message_erreur***	***commentaire***
101	Invalid content-type	L'entête « Content-type » n'est pas de la forme « multipart/related; boundary=*boundary*; »
102	Invalid structure	Les données n'ont pas une structure de type MIME (pas de délimiteur « --*boudary* »)
103	Incomplete data	Les données transmises sont incomplètes, il manque le dernier délimiteur « --*boundary*-- ».
104	No SHA1	Les données ne contiennent pas de bloc de type « Content-type: application/sha1-signature »
105	No file	Les données ne contiennent pas de bloc de type « Content-type: application/octet-stream »
106	Bad checksum	La signature calculée par le serveur ne correspond pas à celle définie dans le bloc « sha1-signature »
107	Cannot save file	Le serveur ne peut pas enregistrer le fichier. Peut être dû à un problème d'espace disque

Dans tous les cas d'erreur,

- le serveur ne conserve pas le fichier reçu;
- dans le cas où un fichier ayant le même nom existait sur le serveur avant le transfert, l'ancien fichier est conservé;
- le client ne réessaye pas de transférer du fichier : il retourne un code d'erreur détaillant la cause de l'erreur.

4.2.2 Récupération de la liste de fichiers disponibles sur le serveur

Le schéma ci-dessous décrit la récupération par le client de la liste des groupes de fichiers disponibles sur le serveur dans le dossier de réception du client.



Le serveur renvoie la liste des groupes de fichiers disponibles. Les identifiants de groupes de fichiers fournis par cette requête sont destinés à être utilisés pour la récupération d'un groupe (cf §« Récupération d'un groupe de fichiers à partir du serveur avec contrôle d'intégrité (download) »).

La requête HTTP a la forme suivante:

```
GET  /list groups.cgi HTTP/1.1
Host: serveur_hostname
```

L'URL */list_groups.cgi* dans la directive GET est donnée à titre d'exemple.

La réponse du serveur a la forme suivante :

```
HTTP/1.1 200
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<file_groups>
  <file_group>identifiant_groupe1</file_group>
  ....
  <file_group>identifiant_groupeN</file_group>
</file_groups>
```

Si aucun groupe de fichiers n'est disponible pour le téléchargement, la réponse est:

```
HTTP/1.1 200
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<file_groups>
</file_groups>
```

4.2.2.1 Traitement des erreurs

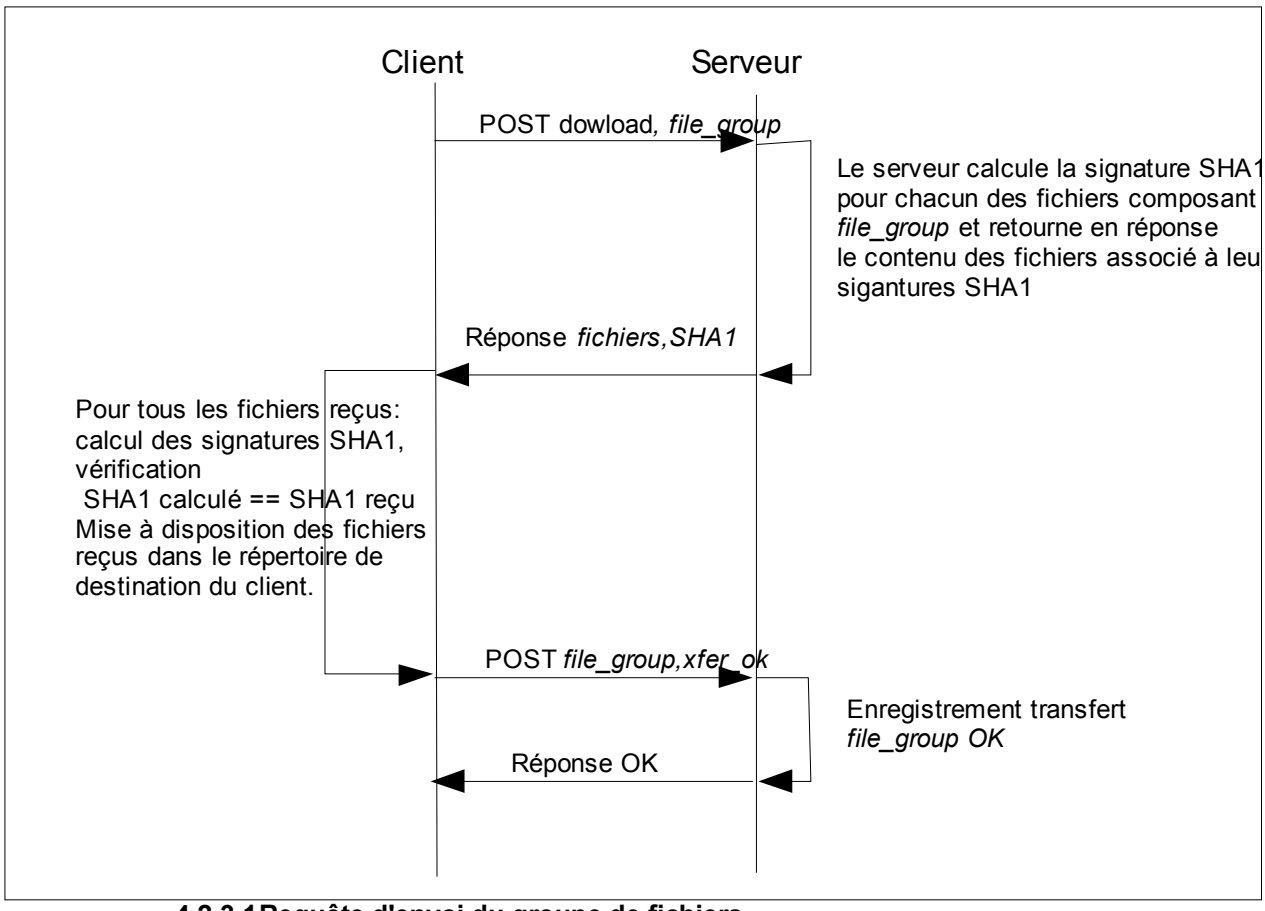
Le traitement des erreurs est identique à celui décrit dans le § « Envoi d'un fichier avec contrôle d'intégrité (upload) ». Il n'y a pas d'erreur applicative retournée avec le code HTTP 409 (Conflict).

4.2.3 Récupération d'un groupe de fichiers avec contrôle d'intégrité (download)

Cette fonction est utilisée pour récupérer plusieurs fichiers formant un ensemble cohérent dont le transfert n'est valide que si tous les fichiers ont été correctement transférés.

Le principe du transfert est le suivant:

- le client demande au serveur de lui envoyer un groupe de fichiers identifié par *file_group*,
- le serveur répond en envoyant tous les fichiers du groupe avec leurs signatures SHA1,
- le client vérifie la validité des données transmises en calculant les signatures SHA1 des fichiers reçus et en les comparant avec celles reçues,
- si tous les fichiers sont validés, ils sont mis à disposition dans le répertoire de destination du client,
- le client informe le serveur de la validité du transfert.



4.2.3.1 Requête d'envoi du groupe de fichiers

```
POST /download.cgi HTTP/1.1
Host: server_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<download>
 <file_group>identifiant_groupe</file_group>
</download>
```

L'URL */download.cgi* dans la directive POST est donnée à titre d'exemple.

Une structure de données XML est associée à la requête POST. Elle définit le groupe de fichiers à télécharger; qui doit correspondre à un des identifiants de groupe de fichiers retournés par la requête de demande de liste de fichiers.

La réponse à une structure MIME, chaque fichier est suivi de sa signature SHA1 :

```
HTTP/1.1 200 OK
Content-type: multipart/related; boundary=boundary;

--boundary
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="filename1"
Content-Transfer-Encoding: base64


         ... contenu de filename1 ...
--boundary
Content-type: application/sha1-signature
Content-Transfer-Encoding: 7bit

signature_SHA1_de_filename1
--boundary
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="filename2"
Content-Transfer-Encoding: base64


         ... contenu de filename2 ...
--boundary
Content-type: application/sha1-signature
Content-Transfer-Encoding: 7bit

signature_SHA1_de_filename2
--boundary--
```

Le contenu du fichier est codé en BASE64 afin de s'affranchir des problèmes de transmission de caractères spéciaux qui pourraient perturber le transfert dans le cas de données binaires.

Le client peut demander au serveur l'envoi du groupe de fichiers en mode compressé:

```
POST /download.cgi HTTP/1.1
Host: server hostname
Content-type: text/plain
Accept-encoding: gzip

file_group=identifiant_groupe
```

La réponse du serveur est :

```
HTTP/1.1 200 OK
Content-type: multipart/related; boundary=boundary;

--boundary
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="filename1"
Content-Transfer-Encoding: base64
Content-Transfer-Encoding: gzip


        ... contenu de filename1 compressé...
--boundary
Content-type: application/sha1-signature
Content-Transfer-Encoding: 7bit

signature_SHA1_de_filename1
--boundary
Content-Type: application/octet-stream
Content-Disposition: attachment; filename="filename2"
Content-Transfer-Encoding: base64
Content-Transfer-Encoding: gzip


        ... contenu de filename2 compressé...
--boundary
Content-type: application/sha1-signature
Content-Transfer-Encoding: 7bit

signature_SHA1_de_filename2
--boundary--
```

La compression a pour seul but de réduire le temps de transfert. Le client doit donc décompresser les données reçues avant de les stocker dans le fichier.

La compression se fait avant la conversion en BASE64.

De plus, la signature SHA1 doit être calculée par le serveur avant compression et conversion en BASE64 des données et vérifiée par le client après décodage BASE64 et décompression. Elle est donc indépendante du mode de transfert (compressé ou non compressé).

4.2.3.1.1 Traitement des erreurs

Le traitement des erreurs est semblable à celui décrit dans le § « Envoi d'un fichier avec contrôle d'intégrité (upload) ». La réponse est donc de la forme:

```
HTTP/1.1 409 Conflict
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<transfer_error>
  <error_code>code_erreur</error_code>
  <error_message>message_erreur</error_message>
</transfer_error>
```

Les seules erreurs applicatives associées au code d'erreur HTTP 409 sont:

code_erreur	*message_erreur*	*commentaire*
101	Invalid content-type	L'entête « Content-type » n'est pas de la forme « text/plain »
108	Unknown file_group	La valeur de la variable *file_group* ne correspond à un groupe de fichiers existant sur le serveur.
109	No file_group	La requête POST ne contient pas de variable *file_group*

Dans tous les cas d'erreur, le client ne conserve aucun des fichiers reçus.

4.2.3.2 Requête de compte-rendu

Cette requête n'est exécutée par le client que si le serveur n'a pas généré d'erreur à la requête précédente.

Envoi du compte-rendu de validation du fichier:

```
POST /download_result.cgi HTTP/1.1
Host: serveur_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<download_result>
  <file_group>identifiant groupe</file_group>
  <error_code>code_erreur</error_code>
  <error_message>message_erreur</error_message>
  <error_detail>detail_erreur</error_detail>
</download_result>
```

L'URL */download_result.cgi* dans la directive POST est donnée à titre d'exemple.

Pour des raisons d'homogénéité avec les codes d'erreurs prévus dans les autres échanges, on se base sur ces codes d'erreurs avec les spécificités suivantes:

- l'identification du groupe de fichiers est identifié dans l'attribut « file_group »,
- le code d'erreur « 0 » (OK) indique qu'il n'y a pas eu d'erreur,
- un troisième attribut « error_detail » peut être utilisé pour préciser la cause de l'erreur,
- dans le cas où plusieurs causes d'erreur sont détectées, seule la première est reportée.

code_erreur	*message_erreur*	*detail_erreur*	*commentaire*
0	OK	*Aucun*	
101	Invalid content-type	*Aucun*	L'entête « Content-type » n'est pas de la forme « text/xml; charset="utf-8" »
102	Invalid structure	*Aucun*	Les données n'ont pas une structure de type MIME (pas de délimiteur « *--boudary* »)
103	Incomplete data	*Aucun*	Les données transmises sont incomplètes, il manque le dernier délimiteur « *--boundary--* ».
104	No SHA1	Nom du fichier qui n'est pas de signature SHA1 associée	Il manque un bloc de type « Content-type: application/sha1-signature »
105	No file	*Aucun*	Il manque un bloc de type « Content-Type: application/octet-stream »
106	Bad checksum	Nom du fichier qui n'est pas de signature SHA1 associée	La signature calculée par le serveur ne correspond pas à celle définie dans le bloc « sha1-signature »
107	Cannot save file	Nom du fichier	Le client ne peut pas enregistrer le fichier. Peut être dû à un problème d'espace disque

La requête de compte-rendu en l'absence d'erreur est :

```
POST /download_result.cgi HTTP/1.1
Host: serveur_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<download_result>
  <file_group>group1</file_group>
  <error_code>0</error_code>
  <error_message>OK</error_message>
</download_result>
```

En cas d'erreur, la requête de compte-rendu a la forme suivante:

```
POST /download result.cgi HTTP/1.1
Host: serveur_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<download_result>
  <file_group>group1</file_group>
  <error_code>6</error_code>
  <error_message>Bad checksum</error_message>
  <error_detail>filename3</error_detail>
</download_result>
```

Cet exemple correspond à la détection d'une erreur de signature SHA1 sur le fichier « filename3 ».

Si aucune erreur n'est détectée, la réponse du serveur est:

```
HTTP/1.1 200 OK
```

4.2.3.2.1 Traitement des erreurs

Le traitement des erreurs est identique à celui décrit dans le § « Envoi d'un fichier avec contrôle d'intégrité (upload) ». La réponse est donc de la forme:

```
HTTP/1.1 409 Conflict
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<transfer_error>
  <error code>code erreur</error code>
  <error_message>message_erreur</error_message>
</transfer_error>
```

Les seules erreurs applicatives associées au code d'erreur HTTP 409 sont:

code_erreur	*message_erreur*	*commentaire*
101	Invalid content-type	L'entête « Content-type » n'est pas de la forme « text/xml; charset="utf-8" »
106	No file_group	La requête POST ne contient pas de variable *file_group*
108	Unknown file_group	La valeur de la variable *file_group* ne correspond à un groupe de fichiers existant sur le serveur.

Si une erreur se produit à cette étape dans la séquence d'échanges, le serveur a envoyé le groupe de fichiers au client, celui-ci les a reçus et validés. Néanmoins, le serveur n'a pas été correctement informé de la validité du transfert. Le client considère donc que le transfert n'est pas correct et ne conserve pas les fichiers reçus.

4.2.4 Récupération de tous les groupes de fichiers avec contrôle d'intégrité (downloadall)

Cette fonction a pour but de récupérer en une seule opération la totalité des groupes de fichiers disponibles en téléchargement.

Elle s'appuie sur les fonctions « list » et « download ».

- « list » est utilisé pour récupérer la liste des groupes disponibles.
- « download » est utilisé pour récupérer tous les groupes en une seule requête. Pour cela, cette requête est modifiée pour pouvoir demander le téléchargement de tous les groupes de fichiers retournés par la commande « list » :

```
POST /download.cgi HTTP/1.1
Host: server_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<download>
 <file_group>identifiant_groupe1</file_group>
 <file_group>identifiant_groupe2</file_group>
 ...
 <file_group>identifiant_groupeN</file_group>
</download>
```

Le compte-rendu est modifié pour renvoyer tous les identifiants de groupes de fichiers téléchargés:

```
POST /download result.cgi HTTP/1.1
Host: serveur_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<download_result>
  <file_group>identifiant_groupe</file_group>
  <file group>identifiant groupe2</file group>
  ...
  <file_group>identifiant_groupeN</file_group>
  <error_code>code_erreur</error_code>
  <error_message>message_erreur</error_message>
  <error_detail>detail_erreur</error_detail>
</download_result>
```

5 Spécification du client

5.1 Fonctionnalités

Le client est un programme en ligne de commande qui réalise les trois fonctions de base du protocole d'échange :

- envoi d'un fichier,
- réception d'un groupe de fichiers,
- liste des groupes de fichiers disponibles.

Afin de faciliter son intégration dans des applicatifs, ce programme n'est pas interactif : il effectue une des trois fonctions ci-dessus en fonction de ses paramètres d'appel spécifiés sur la ligne de commande et retourne un compte-rendu.

L'exécution de programme est synchrone : le lancement du programme est bloquant jusqu'à ce que l'opération demandée soit terminée.

5.1.1 Utilisation

Ce chapitre décrit succinctement la syntaxe d'appel du programme.

5.1.1.1 Envoi d'un fichier

L'argument « upload » indique que l'opération est un envoi de fichier. Le programme prend un argument supplémentaire qui est le chemin complet du fichier à envoyer.

```
medxfer upload chemin_fichier
```

Le résultat de cette commande est une ligne de compte-rendu (cf « Récapitulatif des retours d'erreur »).

(NB: Le nom *medxfer* est utilisé à titre d'exemple)

5.1.1.1.1 Exemple

Envoi du fichier « c:\repertoire\fichier1 »:

```
medxfer upload c:\repertoire\fichier1
```

Le transfert s'est effectué sans erreur:

```
0,OK
```

5.1.1.2 Envoi d'un groupe de fichiers

L'option « upload » peut être utilisée pour envoyer plusieurs fichiers.

```
medxfer upload chemin_fichier1  chemin_fichier2 .. chemin_fichierN
```

Si tous les fichiers sont transférés sans erreur, le compte-rendu est:

```
0,OK
```

Sinon, le transfert s'arrête dès qu'un des transferts échoue et le compte-rendu d'erreur est celui retourné par ce transfert.

NB : Le transfert d'une liste de fichiers est géré comme une suite d'envois de fichiers indépendants. Lorsqu'un des transferts échoue, les fichiers ayant été transférés sans erreur sont conservés sur le serveur.

5.1.1.3 Liste des groupes de fichiers

L'argument « list » indique que l'opération à effectuer est de lister les groupes de fichiers disponibles.

```
medxfer list
```

Le résultat de cette commande est:

- une ligne de compte-rendu (cf « Récapitulatif des retours d'erreur »),
- le nombre de groupes de fichiers disponibles,
- suivie de la liste des groupes de fichiers disponibles, à raison d'un identifiant de groupe par ligne.

Si aucun groupe de fichiers n'est disponible, la réponse ne contient que la ligne de compte-rendu:

```
0,OK
0
```

5.1.1.3.1 Exemple

Demande de la liste des groupes disponibles:

```
medxfer list
```

La commande s'est exécutée sans erreur; deux groupes de fichiers identifiés « groupe1 » et « groupe2 » sont disponibles pour le téléchargement:

```
0,OK
2
groupe1
groupe2
```

5.1.1.4 Réception d'un groupe de fichiers

```
medxfer download groupe répertoire_destination
```

Le résultat de cette commande est

- une ligne de compte-rendu (cf « Récapitulatif des retours d'erreur »),

- le nombre de fichiers,
- les noms de fichiers reçus, à raison d'un nom par ligne.

Les fichiers sont stockés dans le répertoire de réception de l'application.

5.1.1.4.1 Exemple

```
medxfer download groupe1 c:\repertoire_reception
```

La commande s'est exécutée sans erreur; trois fichiers ont été créés dans le répertoire de destination:

```
0,OK
3
fichier1
fichier2
fichier3
```

5.1.1.5 Réception de tous les groupes de fichiers

```
medxfer downloadall répertoire_destination
```

Le résultat de cette commande est identique à celui de la commande « download ».

5.1.1.5.1 Exemple

```
medxfer downloadall c:\repertoire_reception
```

La commande s'est exécutée sans erreur; trois groupes de fichiers ont été récupérés dans le répertoire de destination:

La commande s'est exécutée sans erreur; trois fichiers ont été créés dans le répertoire de destination:

```
0,OK
3
fichier1
fichier2
fichier3
```

5.1.1.6 Consultation d'un paramètre

```
medxfer getparam fichier paramètre
```

L'application étant paramétrable (cf § « Configuration » p.19), cette option affiche la valeur du *paramètre* définie dans *fichier*.

5.1.1.6.1 Exemple

```
medxfer getparam etc/medxfer.conf upload_url
```

fournit la valeur du paramètre « upload_url ».

```
https://medxfer.bioserveur.com/cgi/upload.pl
```

5.1.1.7 Modification d'un paramètre

```
medxfer getparam fichier paramètre valeur
```

L'application étant paramétrable (cf § « Configuration » p.19), cette option modifie la *valeur* du *paramètre* définie dans *fichier*.

5.1.1.7.1 Exemple

```
medxfer getparam etc/medxfer.conf upload_url https://medxfer.bioserveur.com/cgi/upload.pl
```

change la valeur du paramètre « upload_url » dans le fichier etc/medxfer.conf.

5.1.1.8 Récapitulatif des retours d'erreur

Retour d'erreur	*Commentaire*
0,OK	pas d'erreur
1,Cannot join server	Le client n'a pas pu ouvrir de connexion sur le serveur
2,Connection refused	Le connexion a été refusée par le serveur
3,Connection timeout	Le délai d'attente à l'ouverture de connexion a été dépassé
4,Invalid server certificate	Le certificat X.509 envoyé par le serveur ne correspond pas à celui accepté par l'application

Retour d'erreur	***Commentaire***
5,Invalid client certificate	Le serveur a refusé le certificat envoyé par le client
6,Connection closed	Le serveur a coupé la communication avant que les échanges soient terminés
7,outgoing transfer timeout	La durée maximum d'envoi de données au serveur a été dépassée
8,incoming transfer timeout	La durée maximum de réception de données depuis le serveur a été dépassée
9,server response timeout	Le client a complètement envoyé une requête au serveur mais n'a reçu aucune réponse dans le délai prévu
10,invalid option *option*	L'option *option* passée en argument n'est pas « upload », « list » ou « download »
11,cannot open *file_path*	Le fichier à envoyer *file_path* n'a pas pu être ouvert par l'application
12,cannot create *file_path*	Le fichier local *file_path* ne peut pas être créé lors de la réception
13,cannot write *file_path*	Impossible d'écrire dans le fichier *file_path*. Peut être dû à un disque local plein.
101,Invalid content-type	L'entête « Content-type » n'est pas de la forme « multipart/related; boundary=*boundary*; »
102,Invalid structure	Les données n'ont pas une structure de type MIME (pas de délimiteur « --*boudary* »)
103,Incomplete data	Les données transmises sont incomplètes, il manque le dernier délimiteur « --*boundary*-- ».
104,No SHA1	Les données ne contiennent pas de bloc de type « Content-type: application/sha1-signature »
105,No file	Les données ne contiennent pas de bloc de type « Content-type: application/octet-stream »
106,Bad checksum	La signature calculée par le serveur ne correspond pas à celle définie dans le bloc « sha1-signature »
107,Cannot create file	Le serveur ne peut pas enregistrer le fichier. Peut être dû à un disque plein.
108,Unknown file_group	La valeur de la variable *file_group* ne correspond à un groupe de fichiers existant sur le serveur.
109,No file_group	La requête POST ne contient pas de variable *file_group*
4xx,*HTTP status*	Toute erreur HTTP autre que 409

5.1.2 Gestion du certificat serveur

Le certificat du CA ayant signé le certificat du serveur change périodiquement. Il est donc intéressant que le client prenne en charge sa mise à jour automatique (voir § « Diffusion des mises à jour »). Le nouveau certificat sera mis à disposition pour téléchargement quelques jours avant sa mise en place effective sur le serveur. Le client devra donc être capable de gérer la transition de l'ancien au nouveau certificat.

5.1.3 Diffusion des mises à jour

Afin de simplifier la diffusion des mises à jour, le client est séparé en deux parties:

- un programme principal « chargeur » qui se charge de vérifier périodiquement si une nouvelle version
  - du programme de gestion des transferts,
  - du certificat du CA ayant signé le certificat du serveur

  sont disponibles et les télécharge;
- le programme de gestion des transferts.

Cette fonctionnalité ajoute une option au programme:

```
medxfer update
```

qui force la vérification immédiate de la disponibilité d'une nouvelle version du client.

Lors de cette vérification, le client envoie les informations suivantes au serveur:

- nom de l'application à vérifier,
- type de système d'exploitation:

```
POST /check_update.cgi HTTP/1.1
Host: serveur_hostname
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<check_update>
  <application>medxfer</application>
  <operating_system>Windows 2000</operating_system>
</check_update>
```

Le serveur répond en indiquant

- le nom de l'application (pour contrôle),
- le système d'exploitation (pour contrôle),
- le numéro de version le plus récent de l'application,
- l'URL du fichier à télécharger,
- la signature SHA1 du fichier à télécharger:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<check_update_result>
  <application>medxfer</application>
  <operating_system>Windows 2000</operating_system>
  <version>1.2.3</version>
  <url>https://nom_du_serveur/fichier_a_telecharger</url>
  <sha1>code_erreur</sha1>
</check_update_result>
```

Le client vérifie que si application installée est à jour (numéro de version égal à celui retourné par le serveur); sinon, le fichier désigné dans l'URL est téléchargé, vérifié en calculant sa signature SHA1 et en la comparant à celle fournie par le serveur et, si celle-ci est correcte, il remplace la version installée par celle reçue.

5.1.4 Configuration

Un certain nombre de paramètres sont nécessaires pour que l'application fonctionne correctement; tels que:

- URLs associées aux fonctions « upload », « list », « download »,
- URL de vérification des mises à jour,
- certificat(s) serveur accepté(s) par l'application,
- certificat client utilisé par l'application,
- fréquence de vérification des mises à jour,
- etc.

Ces paramètres sont stockés dans un fichier de configuration.

5.1.5 Traçabilité

Toutes les opérations seront enregistrées dans un fichier de log avec les informations:

- date et heure de la requête,
- identifiant du processus (PID du programme),
- identifiant du processus qui demandé le transfert (PPID, processus père du programme),
- opération (upload, download, setparam, getparam,...),
- durée d'exécution,
- code de retour.

Les messages d'erreur et de déboguage seront enregistré dans un second fichier. Chaque ligne contiendra les informations:

- date et heure de la requête,
- identifiant du processus.

5.2 Livrables

Les éléments à fournir à l'utilisateur sont :

- Programme « chargeur » (spécifique par plateforme),
- Certificat du CA ayant signé le certificat du serveur (commun à toutes les livraisons),
- Certificat client (spécifique à chaque utilisateur).

5.3 Intégration à la plate-forme client

Le client est développé en Perl et utilise des librairies binaires spécifiques à chaque plate-forme (ex: OpenSSL) éventuellement modifiées (voir §« Intégration des cartes GIP-CPS »).

Deux mode de distribution sont possibles:

- application Perl « classique » : seule l'application est distribuée. Un environnement d'exécution Perl devra avoir été installé par le client. Ces environnements existent pour les plate-formes Windows, Mac OS X, Linux, Solaris, AIX et HP-UX. Ils sont soit fournis avec le système d'exploitation (ex: Linux, HP-UX,..), soit disponibles gratuitement (ex: ActivePerl pour Windows d'ActiveState),
- programme « auto-suffisant » : l'application est convertie en programme binaire spécifique à la plate-forme avec l'utilitaire Perl2exe du « Perl Dev Kit » de la société ActiveState qui permet de générer des exécutables pour Windows, Mac OS X, Linux, Solaris, AIX et HP-UX.

Remarque 1 : Le « Perl Dev Kit » est un produit payant de la société ActiveState. Il n'y a aucune restriction à la distribution des binaires générés par Perl2exe.

Remarque 2 : L'utilisation de Perl2exe permet de s'affranchir des problèmes de compatibilité éventuels entre la version de Perl installée sur la plate-forme et celle nécessaires pour l'application.

5.3.1

6 Intégration des cartes GIP-CPS

Le GIP « CPS » (« Carte de Professionnel de Santé ») a pour but de définir les moyens de sécurisation des échanges électroniques dans le domaine de la santé. Ces moyens sont basés sur l'utilisation de cartes à puces qui, dans leurs dernières versions, se basent sur des certificats X.509.

L'utilisation de ces cartes dans le cadre de ce protocole sécurisé de transfert de fichiers a des implications au niveau du client et du serveur.

6.1 Intégration des cartes GIP-CPS au niveau serveur

Bien que basé sur le standard X.509, le certificat client provenant d'une carte à puces GIP-CPS n'est pas reconnu en standard par le serveur Web Apache.

Le GIP-CPS met à disposition des patchs du module SSL (modssl) d'Apache sur son site (https://editeurs.gip-cps.fr/index.php?page=logiciels : « Patch MODSSL GIP-CPS pour APACHE 2.0.x »).

Remarque 3 : L'intégration de ce patch peut poser des problèmes pour la maintenance du serveur Apache installé sur le serveur. En effet, à chaque publication d'une mise à jour d'Apache par l'éditeur (RedHat), il sera nécessaire de ré-appliquer le patch et valider son fonctionnement avant l'installation en production.

6.2 Intégration des cartes GIP-CPS au niveau client

Le GIP-CPS publie un « Kit API CPS 5.03 » pour différentes plate-formes (Windows, MacOS, Linux, AIX, Solaris, HP-UX,...).

Pour que le client puisse utiliser ces cartes, il faut que ce kit soit intégré dans les routines de gestion des certificats. Dans la mesure où le client est développé à partir de composants logiciels standards tels que OpenSSL, cette intégration implique la modification de ces composants.

Remarque 4 : L'intégration de kit GIP-CPS à OpenSSL implique de modifier des librairies C et donc de disposer de chacune des plate-formes sur laquelle le client doit être supporté pour y compiler et valider ces librairies.

7 Annexes

7.1 Variables d'environnement Apache/mod_ssl

cf: http://httpd.apache.org/docs/2.2/mod/mod_ssl.html

Variable Name	*Value Type*	*Description*
HTTPS	flag	HTTPS is being used.
SSL_PROTOCOL	string	The SSL protocol version (SSLv2, SSLv3,TLSv1)
SSL_SESSION_ID	string	The hex-encoded SSL session id
SSL_CIPHER	string	The cipher specification name
SSL_CIPHER_EXPORT	string	true if cipher is an export cipher
SSL_CIPHER_USEKEYSIZE	number	Number of cipher bits (actually used)
SSL_CIPHER_ALGKEYSIZE	number	Number of cipher bits (possible)
SSL_COMPRESS_METHOD	string	SSL compression method negotiated
SSL_VERSION_INTERFACE	string	The mod_ssl program version
SSL_VERSION_LIBRARY	string	The OpenSSL program version
SSL_CLIENT_M_VERSION	string	The version of the client certificate
SSL_CLIENT_M_SERIAL	string	The serial of the client certificate
SSL_CLIENT_S_DN	string	Subject DN in client's certificate
SSL_CLIENT_S_DN_x509	string	Component of client's Subject DN
SSL_CLIENT_I_DN	string	Issuer DN of client's certificate
SSL_CLIENT_I_DN_x509	string	Component of client's Issuer DN
SSL_CLIENT_V_START	string	Validity of client's certificate (start time)
SSL_CLIENT_V_END	string	Validity of client's certificate (end time)
SSL_CLIENT_V_REMAIN	string	Number of days until client's certificate expires
SSL_CLIENT_A_SIG	string	Algorithm used for the signature of client's certificate
SSL_CLIENT_A_KEY	string	Algorithm used for the public key of client's certificate
SSL_CLIENT_CERT	string	PEM-encoded client certificate
SSL_CLIENT_CERT_CHAIN_n	string	PEM-encoded certificates in client certificate chain
SSL_CLIENT_VERIFY	string	NONE, SUCCESS, GENEROUS or FAILED:reason
SSL_SERVER_M_VERSION	string	The version of the server certificate
SSL_SERVER_M_SERIAL	string	The serial of the server certificate
SSL_SERVER_S_DN	string	Subject DN in server's certificate
SSL_SERVER_S_DN_x509	string	Component of server's Subject DN
SSL_SERVER_I_DN	string	Issuer DN of server's certificate
SSL_SERVER_I_DN_x509	string	Component of server's Issuer DN
SSL_SERVER_V_START	string	Validity of server's certificate (start time)
SSL_SERVER_V_END	string	Validity of server's certificate (end time)
SSL_SERVER_A_SIG	string	Algorithm used for the signature of server's certificate
SSL_SERVER_A_KEY	string	Algorithm used for the public key of server's certificate
SSL_SERVER_CERT	string	PEM-encoded server certificate