rfc3161, rfc5816

PKIX Time-Stamp Protocol

Un service d'horodatage supporte la preuve qu'une donnée existait à une date donnée. Un TSA peut être opéré par un Tiers de confiance (TTP - Trusted Third Party), bien que d'autres modèles opérationels peuvent être appropriés, par ex, une organisation peut exiger un TSA pour de l'horodatage interne.

Les services de non-répudiation exigent la capacité d'établir l'existence d'une donnée avant une date spécifiée. Ce protocole peut être utilisé comme bloque de construction de tels services.

Pour associer une donnée avec un point dans le temps, une autorité d'horodatage (TSA) peut être utilisé. Ce tiers de confiance fournis une preuve d'existence pour cette donnée particulière à une point dans le temps.

Le rôle du TSA est de dater une donnée pour établir l'évidence indiquant qu'une donnée existait avant une date particulière. Cela peut être utilisé, par exemple, pour vérifier qu'une signature numérique a été appliquée à un message avant que le certificat correspondant n'ait été révoqué, permettant à une clé publique révoquée d'être utilisé pour vérifier la signature créé avant la date de révocation. C'est une opération PKI importante. Le TSA peut également être utilisé pour indiquer la date d'émission quand une deadline est critique, ou pour indiquer la date de transaction pour les entrées dans un log.

Le TSA

Le TSA est un TTP qui créé des jetons d'horodatage pour indiquer qu'une donnée a existé à un point dans le temps particulier. Pour le reste du document, une "requête valide" signifie une requête que peut être décodée correctement.

Pre-requis du TSA

Le TSA doit :

- Utiliser une source de temps fiable pour chaque jeton d'horodatage
- Inclure une date fiable pour chaque jeton d'horodatage
- Inclure un entier unique pour chaque jeton d'horodatage généré
- produire un jeton d'horodatage une fois reçu une requête valide, si possible
- Inclure dans chaque jeton d'horodatage un identifiant pour indiquer de manière unique la stratégie de sécurité sous laquelle le jeton a été créé.
- horodater seulement une représentation hashée des données
- examiner l'OID d'une fonction de hash résistante aux collisions et vérifier que la longueur du hash est consistante avec l'algorithme
- Ne pas example l'empreinte à horodater de toute autre manière
- Ne pas inclure d'identification de l'identité demandeur dans les jetons d'horodatage
- Signer chaque jeton d'horodatage en utilisant une clé générée exclusivement dans ce but et avoir cette propriété de clé indiquée dans le certificat correspondant.
- Inclure les informations additionnelles dans le jeton d'horodatage, si demandé en utilisant le champs extensions, seulement pour les extensions qui sont supportés par le TSA. Si ce n'est pas possible, le TSA doit répondre avec un message d'erreur.

Transactions TSA

Comme premier message de ce mécanisme, l'entité demande un jeton d'horodatage via une requête (qui est ou inclus un TimeStampReq) au TSA. Comme second message, le TSA répond en envoyant une réponse (qui est ou inclus un TimeStampResp) à l'entité.

Une fois reçu la réponse (qui est ou inclus un TimeStampResp qui contient normalement un TimeStampToken (TST)), l'entité doit vérifier le status retourné dans la réponse et en l'absence d'erreur il doit vérifier les divers champs contenus dans le TimeStampToken et la validité de la signature numérique du TimeStampToken. En particulier, il doit vérifier que l'horodatage correspond à ce qui était demandé. Le demandeur doit vérifier que le TimeStampToken contient l'identifiant de certificat correcte du TSA, l'empreinte des données correcte, et l'OID de l'algorithme de hashage correcte. Il doit ensuite vérifier le délai de la réponse en vérifiant soit la date incluse dans la réponse avec une source de temp locale fiable de référence, si disponible, ou la valeur du nonce (un grand nombre aléatoire avec une grande probabilité qu'il a été généré par le client seulement une seule fois) inclus dans la réponse avec la valeur inclus dans la demande. Si une de ces vérification échoue, le TimeStampToken doit être rejeté.

Ensuite, le certificat du TSA peut avoir été révoqué, le status du certificat doivrait être vérifié pour s'assurer que le certificat est valide.

Ensuite, l'application cliente devrait vérifie le champ policy pour déterminer si la stratégie sous laquelle le jeton a été émis est acceptable pour l'application.

Identification du TSA

Le TSA doit signer chaque message avec une clé réservée spécifiquement dans ce but. Un TSA peut avoir des clé privées distincts, par ex, pour gérer différentes stratégie, différents algorithmes, différentes tailles de clé ou pour augmenter les performances. Le certificat correspondant doit contenir seulement une instance du champ d'utilisation de clé avec KeyPurposeIP ayant la valeur : **id-kp-timeStamping**. Cette extension doit être critique.

L'identifiant d'objet suivant identifie le KeyPurposeIP ayant la valeur id-kp-timeStamping

```
id-kp-timeStamping OBJECT IDENTIFIER ::= {iso(1)
  identified-organization(3) dod(6)
  internet(1) security(5) mechanisms(5) pkix(7)
  kp (3) timestamping (8)}
```

Format des requêtes

Une requête time-stamping est comme suit :

```
TimeStampReq ::= SEQUENCE {
 version INTEGER { v1(1) },
 messageImprint MessageImprint,
  -Un OID d'algorithme de hash et la valeur du hash des donnée à horodater
 reqPolicy TSAPolicyId OPTIONAL,
 nonce INTEGER OPTIONAL,
 certReq BOOLEAN DEFAULT FALSE,
 extensions [0] IMPLICIT Extensions OPTIONAL }
```

Le champ version (actuellement v1) décris la version de la demande d'horodatage.

Le champ messageImprint devrait contenir le hash des données à horodater. Le hash est représenté comme chaîne d'octets. Sa longueur doit correspondre à la longueur de la valeur du hash pour cet algorithme. (20 octets pour SHA1, 16 pour MD5).

```
MessageImprint ::= SEQUENCE {
 hashAlgorithm AlgorithmIdentifier,
```

```
 hashedMessage OCTET STRING }
```

L'algorithme de hashage indiqué dans le champ hashAlgoritm devrait être un algorithme de hashage connu. Cela signifie qu'il doit être one-way et résistant aux collisions. Le TSA devrait vérifier si l'algorithme de hashage donné est connu pour être suffisant. Si le TSA ne reconnait pas l'algorithme de hashe ou sait qu'il est faible, le TSA devrait refuser de fournir le jeton d'horodatage en retournant un pkiStatusInfo de 'bad_alg'.

Le champ reqPolicy, si inclus, indique la stratégie TSA sous laquelle le TimeStampTocke devrait être fournis. TSAPolicyId est définis comme suit : **TSAPolicyId : := OBJECT IDENTIFIER**

Le nonce, si inclus, permet au client de vérifier la date de la réponse quand aucune horloge locale n'est disponible. Le nonce est un grand nombre aléatoire avec une forte probabilité que le client l'a généré seulement une seule fois. Dans ce cas la même valeur nonce doit être incluse dans la réponse, sinon la réponse doit être rejetée.

Si le champ certReq est présent et à true, le certificat du TSA qui est référencé par l'identifiant ESSCertID dans un attribut SigningCertificate ou ESSCertIDv2 dans la réponse doit être fournie par le TSA dans le champ certificates de la structure SignedData dans cette réponse. Ce champ peut également contenir d'autres certificats.

Si le champ certReq est manquant ou s'est est présent est à false, le champ certificates ne doit pas être présent dans la réponse.

Le champ extensions est un manière générique d'ajouter des informations additionnelles aux requêtes dans le future. Les extensions sont définies dans la rfc2459. Si une extension, qu'elle soit marquée critique ou non, est utilisée par un demandeur mais non reconnue par le TSA, le serveur ne doit pas émettre de jeton et doit retourner une erreur (unacceptedExtension)

La requête d'horodatage ne doit pas identifier le demandeur, vu que cette information n'est pas validée par le TSA. Dans les situations où le TSA exige l'identité du demandeur, une identification/authentification alternative doit être mise en place.

Format de réponse

Une réponse d'horodatage est comme suit :

```
TimeStampResp ::= SEQUENCE {
 status PKIStatusInfo,
 timeStampToken TimeStampToken OPTIONAL }
```

Le status est basé sur la définition des status dans la rfc2510 comme suit :

```
PKIStatusInfo ::= SEQUENCE {
 status PKIStatus,
 statusString PKIFreeText OPTIONAL,
 failInfo PKIFailureInfo OPTIONAL }
```

Quand le status contient la valeur 0 ou 1, un TimeStampToken doit être présent. Quand le status contient une autre valeur, un TimeStampToken ne doit pas être présent. Une des valeurs suivantes doit être contenus dans le status :

```
PKIStatus ::= INTEGER {
 granted (0),
 grantedWithMods (1),
 rejection (2),
 waiting (3),
 revocationWarning (4), // indique une révocation imminente
 revocationNotification (5) // indique une révocation
```

Les serveurs conformes ne doivent pas produire d'autres valeur. Les clients conforme doivent générer une erreur si des valeurs qu'il ne

comprend pas sont présents.

Quand le TimeStampToken n'est pas présent, failInfo indique la raison du rejet de le requête et peut être une des valeurs suivantes :

```
PKIFailureInfo ::= BIT STRING {
 badAlg (0), - identifiant d'algorithme non supporté ou inconnu
 badRequest (2), - transaction non permise ou non supportée
 badDataFormat (5), - Mauvais format de données transmises
 timeNotAvailable (14), - Source de temps du TSA non disponible
 unacceptedPolicy (15), - La stratégie TSA demandée n'est pas supportée par le TSA
 unacceptedExtension (16), - extension non supportée par le TSA
 addInfoNotAvailable (17), - Les informations additionnelles demandées ne sont pas comprises ou non
disponible
 systemFailure (25) - La requête ne peut pas être gérée à cause d'une erreur système
}
```

Ce sont les seules valeurs de PKIFailureInfo qui doivent être supportées.

Les serveurs conformes ne doivent pas produire d'autres valeurs. Les clients conformes doivent générer une erreur si des valeurs qu'il ne comprend pas sont présents.

Le champ statusString de PKIStatusInfo peut être utilisé pour inclure un texte de raison.

Un TimeStampToken est comme suit. Il est définis comme ContentInfo et doit encapsuler un type de contenu de données signée.

```
TimeStampToken ::= ContentInfo
 - contentType is id-signedData ([CMS])
 - content is SignedData ([CMS])
```

Les champs de type EncapsulatedContentInfo de la construction SignedData ont la signification suivante :

eContentType est un identifiant d'objet qui spécifie uniquement le type de contenu. Pour un jeton d'horodatage il est définis comme suit :

```
id-ct-TSTInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) ct(1) 4}
```

eContent est le contenu, un chaîne d'octet, un TSTInfo encodé DER.

Le jeton d'horodatage ne doit pas contenir de signature autre que la signature du TSA. L'identifiant de certificat (ESSCertID ou ESSCertIDv2) du certificat TSA doit être inclus comme attribut signerInfo dans l'attribut SigningCertificate.

```
TSTInfo ::= SEQUENCE {
 version INTEGER { v1(1) },
 policy TSAPolicyId,
 messageImprint MessageImprint,
  - doit avoir la même valeur que le champ similaire dans TimeStampReq
 serialNumber INTEGER,
 genTime GeneralizedTime,
 accuracy Accuracy OPTIONAL,
 ordering BOOLEAN DEFAULT FALSE,
 nonce INTEGER OPTIONAL,
  - Doit être présent si le champ similaire est présent dans TimeStampReq, et doit avoir la même valeur
 tsa [0] GeneralName OPTIONAL,
 extensions [1] IMPLICIT Extensions OPTIONAL }
```

Le champ version (v1) décris la version du jeton d'horodatage. Les serveurs conformes doivent être capable de fournir des jetons version 1.

Parmis les champs optionnels, seul nonce doit être supporté. Les demandeurs conforme doivent être capable de reconnaitre les jeton d'horodatage version 1 avec tous les champs optionnels présents, mais ne sont pas obligés de comprendre les sémantiques des extensions. si présent.

Le champ policy doit indiquer la stratégie TSA sous laquelle la réponse a été produite. Si un champ similaire est présent dans TimeStampReq, il doit avoir la même valeur, sinon une erreur unacceptedPolicy doit être retournée. Cett stratégie peut inclure les types d'information suivantes (bien que cette liste n'est certainement pas exhaustive) :

- Les condition sous lesquels le jeton peut être uilisé
- La disponibilité de log de jeton d'horodatage, pour permettre une vérification ultérieur de l'authenticité du jeton.

Le messageImprint doit avoir la même valeur que le champ similaire dans le TimeStampReq, indiquant que la taille de la valeur du hash matche la taille attendue de l'algorithme de hashage identifié dans hashAlgorithm.

Le champ serialNumber est un entier assigné par le TSA pour chaque TimeStampTocken. Il doit être unique pour chaque TimeStampToken émis par un TSA donné.

genTime est la date à laquelle le jeton a été créé par le TSA. Il est exprimé en temp UTC pour réduire la confusion avec le timezone local. UTC est une échelle de temp, basée sur le secon (SI), comme définis et recommandé par le CCIR, et maintenu par le Bureau International des Poids et Mesures (BIPM). Un synonime est le temp "Zulu" qui est utilisé par l'aviation civile et représentée par la lettre Z.

accuracy représent la déviation de temp autour du temps UTC contenus dans GeneralizedTime.

```
Accuracy ::= SEQUENCE {
 seconds INTEGER OPTIONAL,
 millis [0] INTEGER (1..999) OPTIONAL,
 micros [1] INTEGER (1..999) OPTIONAL }
```

Si un des éléments est manquant, la valeur est considéré comme 0. En ajoutant la valeur accuracy dans le GeneralizedTime, une limite supérieur du temp auquel le jeton d'horodatage a été créé par la TSA peut être obteni. De la même manière, en soustrayant l'accuracy au GeneralizedTime, une limite inférieur du temp auquel le jeton a été créé par le TSA peut être obtenu.

Si le champ ordering est manquant, ou s'il est présent et mis à false, le champ genTime indique seulement le temp auquel le jeton a été créé par le TSA. Dans un tel cas, l'ordre des jetons d'horodatage émis par le même TSA ou différents TSA est seulement possible quand la différence entre le genTime du premier jeton et le genTime du second jeton est supérieur à la somme des accuracy du genTime pour chaque jeton.

Si le champ ordering est présent et à true, tout jeton du même TSA peut toujours être ordonné basé sur le champ genTime, sans regarder l'accuracy GenTime.

Le champ nonce doit être présent s'il est présent dans le TimeStampReq. Dans ce cas, il doit être recopié dans la structure TimeStampReq.

Le but du champ tsa est de donner une indication pour identifier le nom du TSA. Si présent, il doit correspondre à un des nom de sujet inclus dans le certificat qui est utilisé pour vérifier le jeton. Cependant, l'identification de l'entité qui a signé la réponse se produit toujours via l'utilisation de l'identifiant de certificat (ESSCertID) dans l'attribut SigningCertificate ou ESSCertIDv2 dans l'attribut SigningCertificateV2 qui fait partie du signerInfo.

Les extensions sont une manière générique d'ajouter des informations additionnelles dans le future.

Transports

Il n'y a pas de mécanisme de transport obligatoire pour les messages TSA dans ce document. Les mécanismes décris ci-dessous sont optionnels, d'autres mécanisme pouvant être définis dans le future.

E-mail

Cette section spécifie un moyen de transporter des messages encodés ASN.1 pour l'échange de protocole via les mails. 2 objets MIME sont spécifiés comme suit :

```
Content-Type: application/timestamp-query
Content-Transfer-Encoding: base64
«the ASN.1 DER-encoded Time-Stamp message, base64-encoded»
Content-Type: application/timestamp-reply
Content-Transfer-Encoding: base64
«the ASN.1 DER-encoded Time-Stamp message, base64-encoded»
```

Ces objets MIME peuvent être respectivement envoyés et reçus en utilisant les moteurs de traitement MIME qui fournissent un simple transport par mail des messages d'horodatage.

Pour les types MIME application/timestamp-query et application/timestamp-reply, les implémentations devraient inclure les paramètres optionnels "name" et "filename". Inclure un nom de fichier aide à préserver le type d'information quand les requêtes et réponses d'horodatage sont sauvées en tant que fichier. Quand ces paramètres sont inclus, un nom de fihier avec l'extension approprié devrait être sélectionné :

```
______MIME Type_____________________File Extension
application/timestamp-query_____________.TSQ
application/timestamp-reply_____________.TSR
```

Fichier

Un fichier contenant un message timestamp doit contenir seulement l'encodé DER d'un message TSA. De tels fichiers peuvent être utilisé pour transporter les messages timestamp en utiisant ftp par exemple.

Une requête timestamp devrait être contenue dans un fichier avec l'extension .tsq, et une réponse devrait être contenue dans un fichier avec l'extension .tsr

Socket

Le protocole suivant basé sur TCP est utilisé pour transporter les messages TSA. Ce protocole est utilisable pour les cas où une entité initie une transaction et peut demander un résultat. Le protocole assume qu'un processus d'écoute dans un TSA peut accepter les messages TSA sur un port définis (318).

Généralement un initiateur lie ce port et envoie le message TSA initiale. Le répondeur répond avec un message TSA et/ou avec un numéro de référence à utiliser ultérieurement pour demander la réponse.

L'initiateur d'une transaction envoie un "direct TCP-based TSA message" au destinataire. Le destinataire répond avec un message similaire. Un "direct TCP-based TSA message" consiste de : **length (32-bits), flag (8-bits), value (defined below)**

Le champ longueur contient le nombre d'octets du reste du message. Toutse les valeurs 32bits dans ce protocole sont spécifiés pour être dans l'ordre d'octet réseaux

```
Message name flag value
tsaMsg '00'H DER-encoded TSA message - message TSA
pollRep '01'H polling reference (32 bits),
     time-to-check-back (32 bits)
     - Réponse quand aucun message TSA n'est prête. Utilise la valeur polling reference (et la valeur de
date estimée) pour la demande ultérieur.
```

```
pollReq ’02’H polling reference (32 bits)
      - demande d’une réponse TSA pour le message initial
negPollRep ’03’H ’00’H
       pas d’autre réponses ultérieur (ex: transaction complète)
partialMsgRep ’04’H next polling reference (32 bits),
      time-to-check-back (32 bits),
      DER-encoded TSA message
      - réponse partielle du message initial plus une nouvelle référence d’attente à utilise pour obtenir la
prochaine partie de la réponse
finalMsgRep ’05’H DER-encoded TSA message
      - réponse finale du message initial
errorMsgRep ’06’H human readable error message
      - produit quand une erreur est détectée
```

La séquence des messages qui se produit est :

a) L’entité envoie tsaMsg et reçoit un pollRep, negPollRep, partialMsgRep, ou finalMsgRep dans la réponse.

b) L’entité envoie un message pollReq et reçois negPollRep, partialMsgRep, finalMsgRep, ou errorMsgRep dans la réponse

HTTP

2 objets MIME sont spécifiés comme suit :

```
Content-Type: application/timestamp-query

 «the ASN.1 DER-encoded Time-Stamp Request message»

Content-Type: application/timestamp-reply

 «the ASN.1 DER-encoded Time-Stamp Response message»
```

Ces 2 objets MIME peuvent être envoyés et reçus en utilisant un moteur de traitement HTTP sur des liens WWW et fournis un transport simple via navigateur internet. Une fois un requête valide reçue, le serveur doit répondre avec soit une réponse valide avec le type application/timestamp-response, ou avec une erreur HTTP.

Considérations de sécurité

Tout le document est concerné par les considérations de sécurité. En concevant un service TSA, les considérations suivantes ont été identifiées comme ayant un impacte sur la validité ou la confiance dans le jeton d’horodatage.

1. Quand un TSA ne doit plus être utilisé, mais la clé privée TSA n’a pas été compromise, le certificat de l’autorité doit être révoqué. Quand l’extension reasonCode relatif au certificat révoqué du TSA est présent dans les extensions d’entrée de CRL, il doit être mis soit à unspecified, affiliationChanged, superseded ou cessationOfOperation. Dans ce cas, les jetons signés avec la clé correspondante seront considérés comme invalides, mais les jetons générés avant la date de révocation resteront valides. Quand l’extension reasonCode n’est pas présent dans la CRL, tous les jetons qui ont été signés avec la clé correspondante doivent être considérés comme invalides. Pour cette raison, il est recommandé d’utiliser l’extension reasonCode.

2. Quand la clé privée TSA a été compromise, le certificat correspondant doit être révoqué. Dans ce cas, l’extension reasonCode peut être présent et dans ce cas doit être à keyCompromise. Tout jeton signé par le TSA utilisant cette clé privée ne peuvent plus être validés. Pour cette raison, il est impératif que la clé privée du TSA soit conservé en lieu sûr et contrôlé pour minimiser la possibilité d’être compromise. Dans le cas où la clé privée est compromise, un audit de tous les jetons générés par le TSA peut fournir un moyen de discriminer les jetons. 2 jetons d’horodatage de 2 TSA différents est une autre manière d’adresser ce problème.

3. La clé de signature TSA doit être suffisamment longue pour une durée de vie confortable. Même si c'est le cas, la clé a une durée de vie définie. Donc, tout jeton signé par le TSA devrait être horodatées de nouveau (si des copies authentique des anciennes CRL sont disponibles) ou notarisé (s'il elle ne le sont pas) à une date ultérieur pour renouveler la confiance qui existe dans la signature du TSA. Les jetons d'horodatage peuvent également être conservés avec un "Evidence Recording Authority" pour maintenir ce trust.

4. Une application client utilisant seulement un nonce et sans horloge locale devrait être concerné par la quantité de temp qu'il est prêt à attendre la réponse. Une attaque MITM peut introduire un délai. Donc, un TimeStampResp qui prend trop de temp devrait être considéré comme suspect. Vu que chaque méthode de transport spécifié dans ce document a différentes caractéristiques de délai, la durée considérée acceptable dépend du transport utilisé, ainsi que les facteurs environnementaux.

5. Si différentes entités obtiennent des jetons d'horodatage sur le même objet en utilisant le même algorithme de hashage, ou une simple entité obtient plusieurs jetons sur le même objet, les jetons générés incluent le même message ; en résultat, un observateur avec accès à ces jetons peut en déduire que l'horodatage réfère aux même données.

6. Le replays délibérés ou par inadvertance pour les demandes incorporant le même algorithme de hashage et de valeur peut se produire. Les replays par inadvertance se produisent quand plus d'une copie de la même requête est envoyée au TSA à cause de problèmes réseau. Les replays délibérés se produisent via MITM. Pour détecter ces situations, de nombreuses techniques peuvent être utilisé. Utiliser un nonce permet toujours de détecter les replay, et est recommandé. Il est également possible d'utiliser une horloge local, et une fenêtre de temps durant laquelle le demandeur mémorise tous les hash envoyés . En reçevant une réponse, le demandeur s'assure que l'heure de la réponse est dans la fenêtre et qu'il y a une seule occurence de la valeur de hash dans cette fenêtre de temps.

APPENDIX A - attributs Time-stamp utilisant CMS

Une des utilisations majeur de l'horodatage est d'horodater une signature pour prouver qu'une signature numérique a été créée avant une date donnée. Le certificat correspondant est révoqué, cela permet à un vérificateur de savoir si la signature a été créé avant ou après la date de révocation.

Un endroit sensible où stocker un timestamp est dans une structure CMS comme un attribut non-signé. Cette appendice définis un attribut Signature Time-stamp qui peut être utilisé pour horodater une signature numérique.

Les identifiant d'objets suivants identifient cet attribut :

```
id-aa-timeStampToken OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) aa(2) 14 }
```

La valeur de cet attribut a le type ASN.1 SignatureTimeStampToken :

```
SignatureTimeStampToken ::= TimeStampToken
```

La valeur du champ messageImprint dans TimeStampToken doit être un hash de la aleur du champ signature dans SignerInfo pour le signedData à horodater.

APPENDIX 1 - Placer une signature à un point dans le temps

On présent un exemple d'une utilisation possible de ce service d'horodatage généraliste. Il place une signature à un point dans le temps, depuis lequel les informations de status du certificat appropri(ex, CRL) doit être vérifié. Cette application est prévue pour être utilisée en conjonction avec des preuves générées en utilisant un mécanisme de signature numérique.

Les signature peuvent seulement être vérifiées en accord avec une stratégie de non-répudiation. Cette stratégie peut être implicite ou explicite (ex, indiquée dans la preuve fournie par le signataire). La stratégie de non-répudiation peut spécifier, entre-autre, la période permise par un signataire pour déclarer la compromission d'un clé de signature utilisée pour la génération des signatures numérique. Donc une signature ne peut pas garantir d'être valide jusqu'à la fin de cette période.

Pour vérifier une signature numérique, la technique basique suivante peut être utilisée :

A) Les informations d'horodatage doivent être obtenus juste après la production de la signature

1) La signature est présentée au TSA. Le TSA retourne un TimeStampToken (TST).

2) Le demandeur du service doit vérifier que le TimeStampToken est correct.

B) La validité de la signature numérique peut ainsi être vérifiée de la manière suivante :

1) Le jeton d'horodatage lui-même doit être vérifié et qu'il s'applique à la signature du signataire

2) La date indiquée par le TSA dans le TimeStampToken doit être retrouvée.

3) Le certificat utilisé par le signataire doit être identifié et récupéré

4) La date indiquée par le TSA doit être dans la période de validité du certificat du signataire

5) Les informations de révocation sur ce certificat, à la date de l'opération d'horodatage, doit être récupéré

6) Si le certificat est révoqué, la date de révocation doit être ultérieur à la date indiquée par le TSA.

Si toutes ces conditions sont réussies, la signature numérique doit être déclarée comme valide.

APPENDIX C - Module ASN.1 utilisant la syntaxe 1988

```
PKIXTSP {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-tsp(13)}

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

- EXPORTS ALL -

IMPORTS
Extensions, AlgorithmIdentifier FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit-88(1)}
GeneralName FROM PKIX1Implicit88 {iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit-88(2)}
ContentInfo FROM CryptographicMessageSyntax {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) modules(0) cms(1)}
PKIFreeText FROM PKIXCMP {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5)
pkix(7) id-mod(0) id-mod-cmp(9)} ;

- Locally defined OIDs -

- eContentType for a time-stamp token

id-ct-TSTInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) ct(1) 4}

- 2.4.1

TimeStampReq ::= SEQUENCE {
 version INTEGER { v1(1) },
 messageImprint MessageImprint,
  -a hash algorithm OID and the hash value of the data to be time-stamped
 reqPolicy TSAPolicyId OPTIONAL,
 nonce INTEGER OPTIONAL,
 certReq BOOLEAN DEFAULT FALSE,
 extensions [0] IMPLICIT Extensions OPTIONAL }
```

```
MessageImprint ::= SEQUENCE {
 hashAlgorithm AlgorithmIdentifier,
 hashedMessage OCTET STRING }

TSAPolicyId ::= OBJECT IDENTIFIER

- 2.4.2

TimeStampResp ::= SEQUENCE {
 status PKIStatusInfo,
 timeStampToken TimeStampToken OPTIONAL }

PKIStatusInfo ::= SEQUENCE {
 status PKIStatus,
 statusString PKIFreeText OPTIONAL,
 failInfo PKIFailureInfo OPTIONAL }

PKIStatus ::= INTEGER {
 granted (0),
  - when the PKIStatus contains the value zero a TimeStampToken, as requested, is present.
 grantedWithMods (1),
  - when the PKIStatus contains the value one a TimeStampToken, with modifications, is present.
 rejection (2),
 waiting (3),
 revocationWarning (4),
  - this message contains a warning that a revocation is imminent
  revocationNotification (5)
- notification that a revocation has occurred }

PKIFailureInfo ::= BIT STRING {
 badAlg (0),
 badRequest (2),
 badDataFormat (5),
 timeNotAvailable (14),
 unacceptedPolicy (15),
 unacceptedExtension (16),
 addInfoNotAvailable (17)
 systemFailure (25)

TimeStampToken ::= ContentInfo

TSTInfo ::= SEQUENCE {
 version INTEGER { v1(1) },
 policy TSAPolicyId,
 messageImprint MessageImprint,
  - MUST have the same value as the similar field in TimeStampReq
 serialNumber INTEGER,
  - Time-Stamping users MUST be ready to accommodate integers up to 160 bits.
 genTime GeneralizedTime,
 accuracy Accuracy OPTIONAL,
 ordering BOOLEAN DEFAULT FALSE,
 nonce INTEGER OPTIONAL,
  - MUST be present if the similar field was present in TimeStampReq. In that case it MUST have the same
value.
 tsa [0] GeneralName OPTIONAL,
 extensions [1] IMPLICIT Extensions OPTIONAL }
Accuracy ::= SEQUENCE {
 seconds INTEGER OPTIONAL,
```

```
 millis [0] INTEGER (1..999) OPTIONAL,
 micros [1] INTEGER (1..999) OPTIONAL }
END
```