MASTER EN SCIENCES INFORMATIQUES

Analyse et implémentation de services de sécurité dans une messagerie X400/84

Decloux, Bernard

Award date:
1994

Awarding institution:
Universite de Namur

Analyse et implémentation de services de sécurité dans une messagerie X400/84

Mémoire présenté pour l'obtention
du grade de Licencié et Maître en
Informatique par

Bernard DECLOUX
1993-1994

Facultés Universitaires Notre-Dame de la Paix
INSTITUT D'INFORMATIQUE
Rue de Bruxelles 61 - 5000 NAMUR
Tél.081/72.41.11 - Telex 59222 facnam-b - Telefax 081/23.03.91

Analyse et implémentation de services de sécurité dans une messagerie X400/84

DECLOUX Bernard

Résumé

Bien qu'offrant de nombreux services à ses utilisateurs, une messagerie X400/84 ne leur donne aucune garantie quant à la confidentialité de leurs communications : quiconque au sein du réseau pourrait être à l'écoute de leurs échanges de messages. Ce problème a été résolu en adoptant l'architecture de travail Generalized Secure Message Handling System (GSMHS) qui permet d'envisager facilement l'implémentation de services de sécurité au sein de MHS. Travaillant avec le logiciel EAN 3.04, nous avons implémenté un UA sécurisé qui, de manière analogue à ce que propose PEM pour la représentation de messages sécurisés, convertit tout message sous un format particulier. Cette représentation permet alors un transfert de messages via le réseau qui respecte les services de confidentialité et de contrôle d'intégrité du contenu du message.

Mémoire de licence et maîtrise en Informatique
Septembre 1994
Promoteur : J. Ramaekers

Au terme de ce travail de fin d'études, je tiens à remercier tout particulièrement les personnes suivantes :

- Joël Hubin qui s'est donné la peine de s'intéresser à mon mémoire;

- Monsieur Ramaekers, promoteur de ce mémoire, qui tout au long de l'année a toujours suivi de près l'avancement de mon travail et qui m'a donné bon nombre de conseils utiles en ce qui concerne notamment la rédaction et l'organisation du travail;

- Suchun Wu sans qui ce mémoire n'aurait jamais eu une seule chance de voir le jour. Un tout grand merci à toi, pour ta bonne humeur, ta serviabilité et ta qualité de travail !!

- ainsi que toutes les personnes qui ont lu ou liront un jour ce travail.

Introduction

Partant de la constatation qu'une messagerie X400 sans service de sécurité ne peut pas décemment satisfaire tous ses utilisateurs (comment en effet, un utilisateur de X400 peut-il par exemple se satisfaire du fait que chacun des messages qu'il envoie peut être lu par n'importe qui?), ce mémoire s'est attaché à étudier et à résoudre ce problème ainsi qu'à en proposer une solution implémentée.

Dans un premier chapitre sont présentées les notions de base du Message Handling System qui sert de modèle fonctionnel à la messagerie X400. Quels sont les services proposés par cette messagerie, quelles sont les différentes étapes comprises entre la rédaction et la réception d'un message, sous quelle forme un message est-il représenté, comment l'adressage y est-il traité: tels seront les principaux points exposés.

Le deuxième chapitre donne un premier aperçu des liens qui existent ou ... qui devraient exister entre une messagerie X400 et le domaine de la sécurité. Il aborde ainsi les risques auxquels tout message est soumis lorsqu'il est envoyé via la messagerie X400 et démontre de ce fait la nécessité cruciale d'inclure des services de sécurité au sein de X400.

Le troisième chapitre présente une palette d'outils de sécurité sur lesquels le mémoire va pouvoir par la suite se reposer : chiffrements à clé secrète et à clé publique, signature digitale, ...

Le chapitre 4 se focalise sur la mise en œuvre, grâce aux outils de sécurité, du service de confidentialité du contenu du message. C'est ce service qui va être l'objet principal de notre travail d'analyse avec comme perspective finale son implémentation et son intégration dans une messagerie X400.

Avant de passer à une étape d'analyse plus fine du problème, nous avons jugé utile d'étudier ce qui existait déjà par ailleurs dans le domaine de la sécurité au sein de messageries d'autres types. Ceci nous a amené à étudier dans le chapitre 5 la solution proposée par Internet dans son standard PEM

(Privacy Enhancement for Internet Electronic Mail) de représentation de messages mis sous forme sécurisée. C'est de cette solution que le mémoire s'inspirera pour mettre en œuvre sa solution au problème de manque de sécurité au sein de X400.

Le sixième chapitre présente le cadre dans lequel le travail d'analyse s'est développé. Ce cadre de travail est désigné sous le terme de Generalized Secure Message Handling System (GSMHS) et il consiste en une présentation hiérarchisée en interfaces des fonctions qui permettent de faire le lien entre la messagerie X400 et le domaine de la sécurité. En choisissant ensuite de se focaliser sur l'interface du GSMHS : Security Application Program Interface, le mémoire offre une découpe du travail en différents modules.

Le chapitre 7 passe en revue chacune des fonctions présentées lors du chapitre précédent et en offre une analyse détaillée.

Le huitième chapitre aborde quelques problèmes d'ordre plus pratique et technique liés à l'insertion de nouvelles fonctions à l'intérieur d'un logiciel de messagerie X400 préexistant : EAN version 3.04. Ce sera aussi l'occasion de présenter dans les très grandes lignes quelques caractéristiques du fonctionnement de ce logiciel au niveau User Agent.

Le mémoire se conclut finalement en proposant quelques conclusions et perspectives pour des développements ultérieurs.

Chapitre 1 : Recommandation X400/84

1. Description générale du modèle fonctionnel MHS

En 1984, le CCITT publie une série de recommandations X400 proposant un modèle fonctionnel régissant les messageries électroniques (cfr. : [MYE,84] et [CUN,83]).

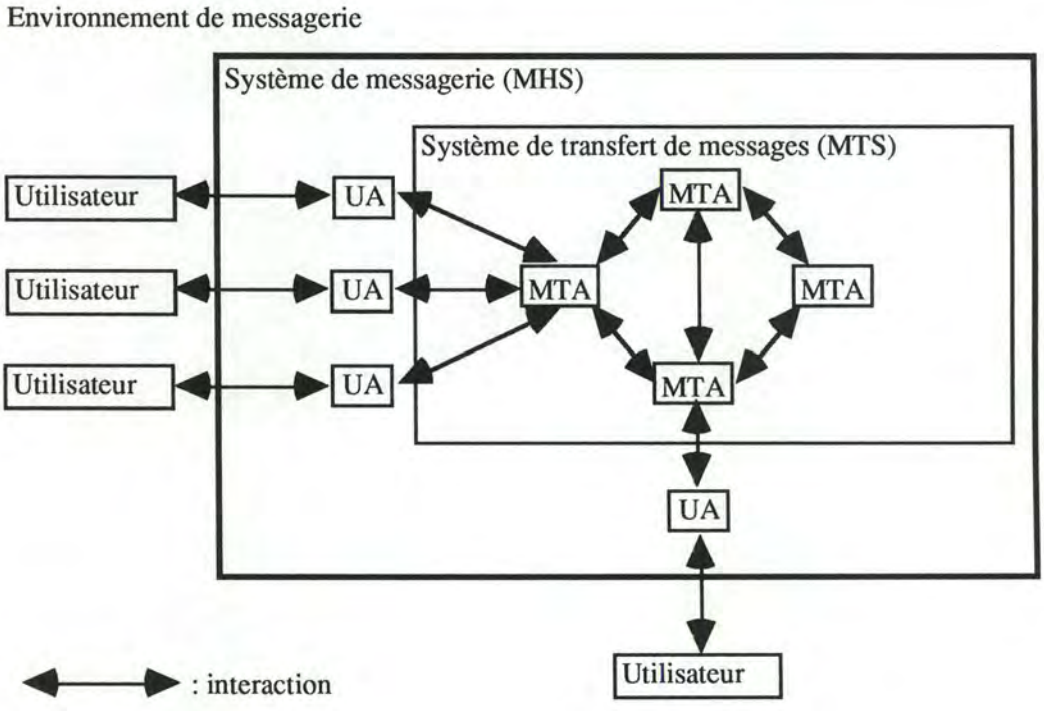


Figure 1 : Modèle fonctionnel d'un système de messagerie X400/84.

Dans une messagerie de type X400, à chaque utilisateur (programme d'application ou être humain) du système de messagerie électronique (Message Handling System (MHS)) est associé ce qu'on appelle un User Agent (UA); ces UA ont pour fonctions d'assister l'utilisateur à la préparation, la mise en forme et la soumission de messages.

Un autre type d'agent au sein du MHS est le MTA (Message Transfer Agent) qui sert au transfert proprement dit des messages.

Le système de transfert de message (Message Transfer System (MTS)) est défini comme étant composé de l'ensemble des MTA interconnectés. Il s'agit donc du réseau par lequel transite les messages.

Le MHS est donc constitué du MTS et de tous les UA qui y sont reliés.

Par la suite, nous nous placerons exclusivement dans le cadre d'un système de courrier électronique plutôt que dans celui d'un système de messagerie électronique : le terme de courrier électronique (E-Mail) étant restreint aux seules communications entre utilisateurs humains. L'utilisation du terme messagerie électronique se référera donc dans ce travail uniquement au contexte d'un courrier électronique.

2. Cheminement d'un message(1)

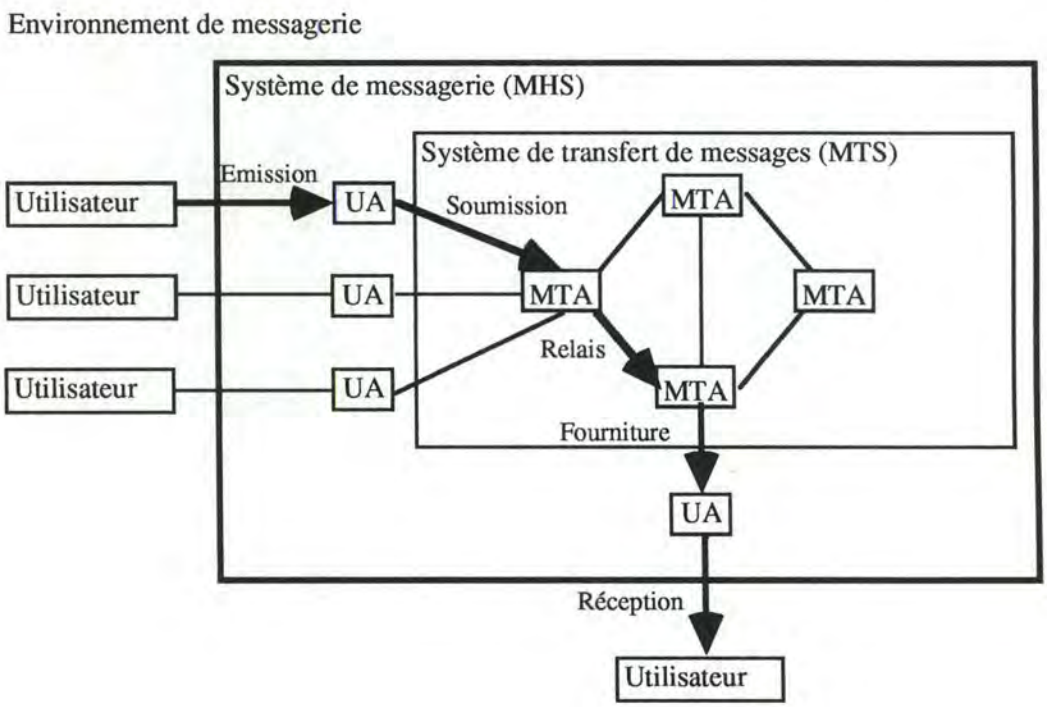


Figure 2 : Cheminement d'un message au sein d'une messagerie X400.

L'émetteur fait tout d'abord savoir à son UA via une opération d'émission qu'il désire émettre un message. L'UA soumet alors le message préparé au MTA auquel il est relié et c'est grâce à une série de relais entre

(1) Par cheminement d'un message, nous voulons signifier trajet du message depuis son émission jusqu'à sa réception.

différents MTA le long d'un trajet que le message est fourni au UA du destinataire. Et, à cette étape, le receveur n'a plus qu'à réceptionner le message auprès de son UA. C'est ce mécanisme où chaque agent enregistre temporairement le message pour ensuite le transmettre vers un autre agent qu'on nomme "store-and-forward".

Dans le cas où un même message est envoyé à plusieurs destinataires, l'UA ne soumet qu'un seul message au MTS mais, en cours de route lorsque le besoin s'en fait ressentir, un MTA peut dupliquer le message pour l'expédier ensuite vers plusieurs UA ou MTA. Chacune des copies poursuivra alors son chemin de manière indépendante.

3. Représentation en couches du modèle MHS

Le modèle MHS vient s'inscrire dans la couche application (couche 7) du modèle OSI.

Cette couche peut elle-même être divisée en deux sous-couches : la sous-couche agent utilisateur (UAL) s'occupant des fonctions relatives au contenu des messages et la sous-couche transfert de message (MTL).

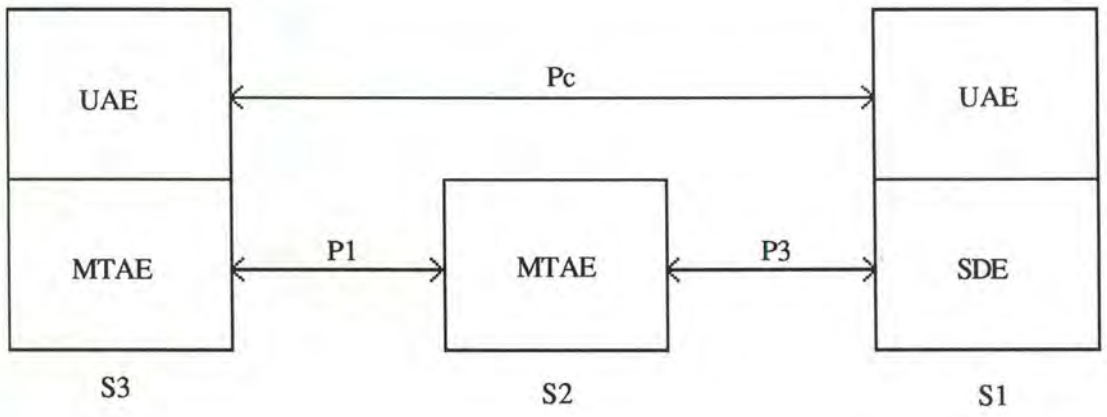


Figure 3 : Représentation en couches du modèle MHS.

Les MTAE et UAE sont les entités fonctionnelles correspondant respectivement à MTA et à UA.

Le SDE (Submission and Delivery Entity) permet à la couche UAL de S1 de disposer des services de la couche MTL.

S1, S2 et S3 représentent trois configurations élémentaires possibles selon lesquelles l'ensemble des UA et des MTA peut être structuré :

- S1 est un système ne fournissant que les fonctions relatives à l'UA.
- S2 est un système contenant uniquement les fonctions de MTA.
- S3 est un système fournissant à la fois des fonctions d'UA et de MTA.

Une architecture de messagerie reprenant chacun de ces trois systèmes pourrait être la suivante :

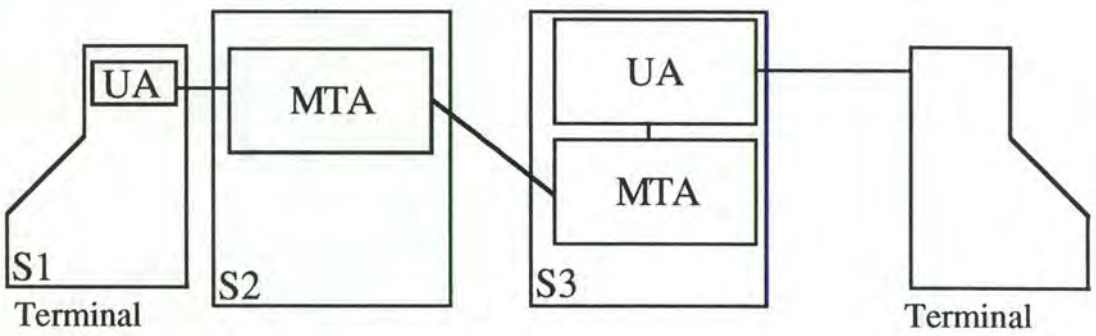


Figure 4 : Architecture possible pour une messagerie X400.

Dans cette représentation en couches, trois protocoles sont définis :

- le protocole P1 (Message Transfer Protocol) définit les interactions entre les MTA;
- le protocole P3 (Submission and Delivery Protocol) permet à un SDE de S1 de fournir à l'UA l'accès aux services de la couche MTL;
-la classe Pc de protocoles définit la syntaxe et la sémantique des messages à transférer. Parmi ces protocoles, on retrouve ainsi le protocole P2 concernant le courrier électronique.

4. Eléments de service

Une messagerie X400 ne se limite pas à réaliser un simple transfert de messages, elle agrémente ce transfert de nombreux services :

4.1. Services de base de transfert de messages

Ces services qui concernent l'échange de messages entre Agents Utilisateurs permettent d'offrir de nombreuses possibilités :

- gestion de l'accès;
- indication du type du contenu du message;
- indication de conversion;

- indication des date et heure de remise (plutôt que de parler de réception, on utilise aussi le terme de remise);
- identification du message par un numéro;
- avis de non remise;
- indication des types de codage d'origine;
- indication des date et heure de dépôt (plutôt que de parler d'émission, on utilise aussi le terme de dépôt);
- enregistrement des caractéristiques utilisateur/UA.

En plus de ces services de base à toute messagerie X400/84, d'autres services peuvent aussi être offerts comme par exemple : le choix de l'urgence avec laquelle un message doit être remis au destinataire ou la désignation d'un destinataire suppléant.

4.2. Services de base de messagerie de personne à personne

Ces services sont désignés sous le terme de IPM (Interpersonal Messaging services) et reprennent, en plus des 9 services de base de transfert de messages, les services relatifs à l'identification de message IP et à l'indication de type de corps.

De nombreux autres services optionnels peuvent être présents :

- indication de destinataires principaux et de copie;
- indication de destinataires de copie muette;
- indication d'annulation (ce message en annule un autre);
- indication de demande de réponse;
- indication de retransmission automatique;
- indication de l'importance du message;

...

5. Structure d'un message

De même qu'une analogie entre courrier électronique et service postal peut être trouvée, on peut rapprocher la structure d'un message de celle d'une lettre postale.

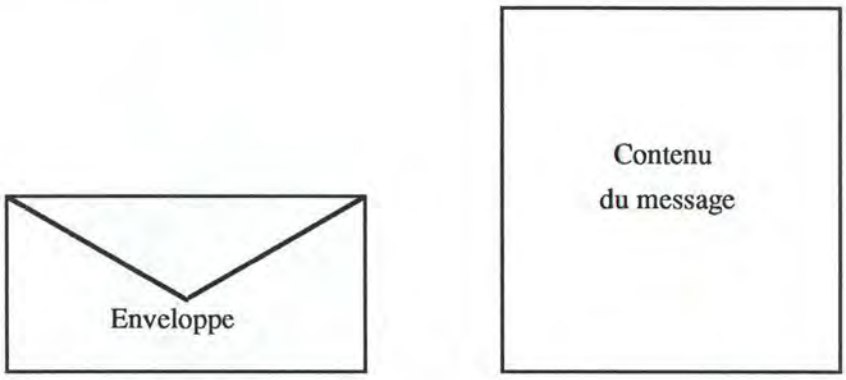


Figure 5 : Structure générale d'un message X400.

Le message est constitué d'une part d'une enveloppe reprenant les informations nécessaires (adresses de l'émetteur et du destinataire, priorité associée au message, ...) pour le transfert au sein du MTS et d'autre part du contenu proprement dit du message qu'un utilisateur désire envoyer à un ou plusieurs correspondants.

Dans un premier temps, l'UA aide l'émetteur à construire le contenu de son message; ensuite, ce contenu ainsi que des informations destinées à la construction de l'enveloppe sont soumises à un MTA : c'est ce MTA qui se charge alors de la fabrication de l'enveloppe.

Au fil des différentes opérations de stockage et de retransmission du message dans les MTA intermédiaires, l'enveloppe est susceptible de changer.

Trois types d'enveloppe peuvent être distingués :

- l'enveloppe de soumission;
- l'enveloppe de relais;
- l'enveloppe de fourniture.

L'enveloppe de soumission est l'enveloppe construite au départ des informations fournies par l'UA de l'émetteur.

L'enveloppe de relais est modifiée par chacun des MTA qui y ajoute des informations destinées au bon cheminement de l'information. Ainsi, les MTA

y indiquent par exemple leur identité pour signaler que le message est déjà passé par eux : ce qui évite le bouclage.

L'enveloppe de fourniture contient quant à elle des informations utilisées par l'UA du récepteur.

Dans le cas où le message est envoyé à plusieurs destinataires :

- l'enveloppe de soumission contient l'ensemble des références des différents UA destinataires;
- lorsque le message doit être dupliqué pour pouvoir emprunter des chemins différents, les différents messages créés contiennent alors dans leur enveloppe de relais les références des UA qu'ils doivent joindre sur cette nouvelle route.

En se référant à la représentation en couches du modèle MHS, les messages que se communiquent les UA et les MTA peuvent être classés en deux grandes classes : les UAPDU (User Agent Protocol Data Unit) et les MPDU (Message Protocol Data Unit).

Les UAPDU sont les unités de données que les UAE s'échangent selon la syntaxe et la sémantique définie par un des protocoles de la classe Pc. Dans notre cadre d'étude restreint au courrier électronique, les unités de données que s'échangent les UAE en respectant le protocole P2 sont appelées IM-UAPDU (Interpersonal Messaging-UAPDU). C'est à l'intérieur de ces unités de données que le contenu du message est placé par l'UA de l'émetteur.

Les MPDU sont les unités de données que s'échangent les MTAE en respectant le protocole P1. C'est à l'intérieur de ces unités de données que les UAPDU sont enveloppées par le MTA auquel a été soumis le message.

Parmi les MPDU, deux types de message sont distingués : les User-MPDU qui contiennent les messages qui doivent être transférés et les Service-MPDU qui contiennent des informations à propos de ces messages. Il y aura ainsi par exemple typiquement un Service-MPDU pour signaler à son émetteur la non livraison d'un message.

Nous sommes maintenant en mesure d'affiner davantage la structure d'un message; plus précisément examinons la structure d'un User-MPDU contenant un IM-UAPDU :

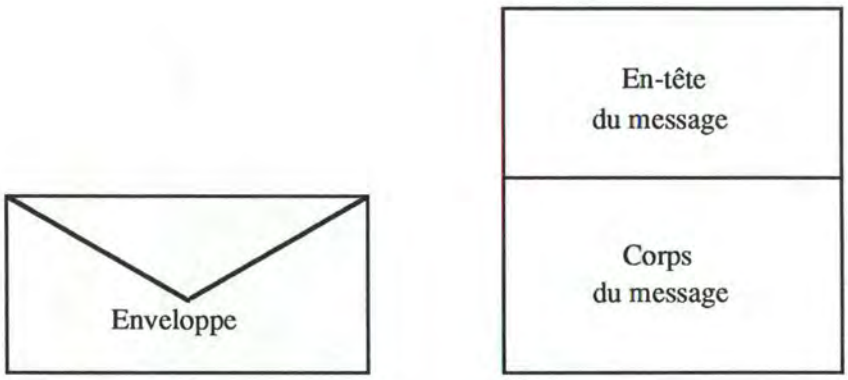


Figure 6 : Structure générale d'un IM-UAPDU à l'intérieur d'un User-MPDU.

L'en-tête du message ("heading") contient une série de champs créés par l'UA de l'émetteur et il reprend des informations telles que l'identifiant de l'IP-message, le sujet du message ou l'importance du message.

Le corps du message constitue l'information même que l'émetteur désire transmettre à son(ses) destinataire(s). Le corps du message peut être lui-même subdivisé en plusieurs parties (body part) correspondant à des types de messages différents : fichier binaire, son codé, document fax, ...

Dorénavant, lorsque le terme de message sera utilisé, il faudra comprendre IM-UAPDU si on se place au niveau de la sous-couche UAL ou User-MPDU contenant un IM-UAPDU si on se place au niveau de la sous-couche MTL.

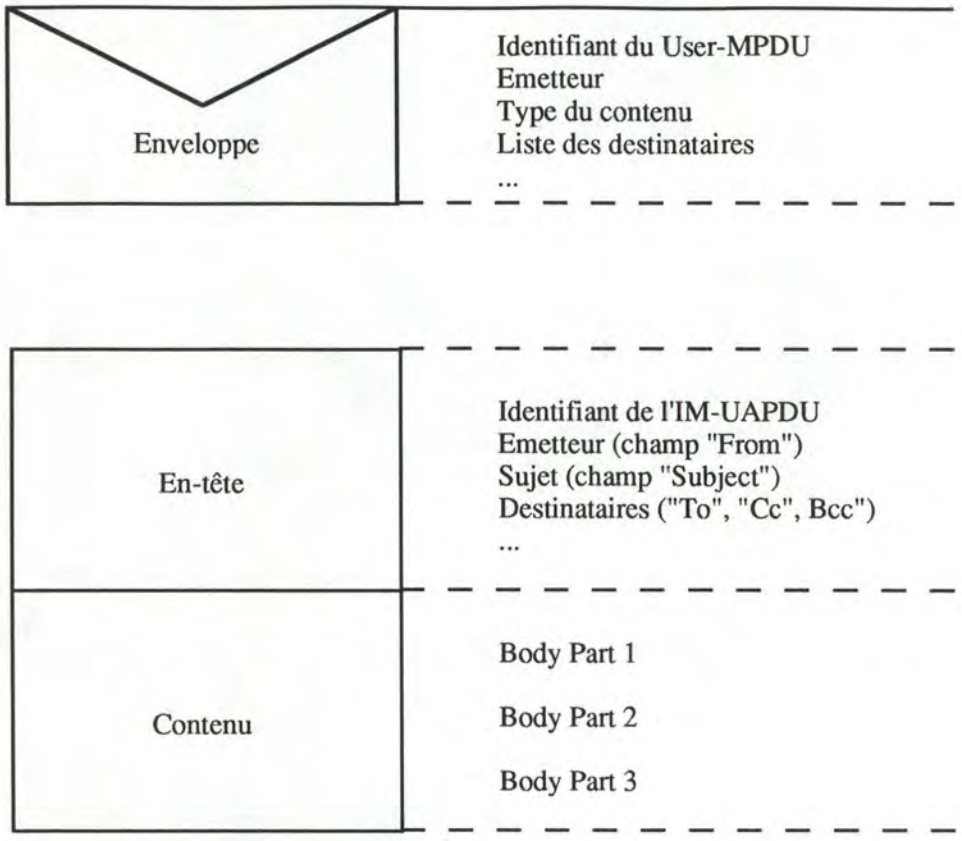


Figure 7 : Schéma détaillé d'un User-MPDU contenant un IM-UAPDU.

6. Adressage

Au sein d'un MHS, chaque utilisateur est identifié par un O/R name (Originator/Recipient name) qui, dans la recommandation X400/84, joue aussi le rôle d'adresse O/R (Originator/Recipient address). Ces adresses doivent donc contenir les informations qui permettent d'identifier de manière univoque un utilisateur au sein de la messagerie.

Une adresse O/R est construite comme étant une séquence de différents éléments, ces éléments étant organisés de manière hiérarchique.

Prenons-en un exemple :

C = be, ADMD = rtt, PRMD = fundp, O = info, OU = ts, S = bdc.

Ceci a comme signification :

C (Country) : nom du pays,

ADMD (ADministration Management Domain) : nom du domaine de gestion d'administration,

PRMD (PRivate Management Domain) : nom du domaine de gestion privé,
O (Organization) : nom de l'organisation,
OU (Organizational Unit) : nom de l'unité organisationnelle,
S (Subject) : nom de l'utilisateur.

La recommandation X400 définit la notion de domaine de gestion (MD : Management Domain) comme étant un ensemble d'au moins un MTA et d'un nombre (positif ou nul) quelconque de UA. Lorsque le domaine de gestion est géré par une administration, on parle de ADMD tandis que lorsque le gérant est privé , on parle de PRMD.

Chapitre 2 : Sécurité dans X400

1. Besoin de sécurité dans une messagerie

Nous sommes actuellement de plus en plus souvent amenés à communiquer via ce qu'on appelle des messageries électroniques. Ces messageries nous permettent de transmettre des informations à des correspondants pouvant se trouver à l'autre bout du monde grâce à l'échange de messages, tout ceci se réalisant via un réseau. Ces messageries sont donc en quelque sorte le pendant électronique du traditionnel service postal; service pour le moins amélioré si on se réfère au grand nombre de fonctionnalités offertes ainsi qu'à la rapidité et l'efficacité des échanges.

Les informations à échanger peuvent appartenir à des domaines d'application très différents : il peut s'agir d'une simple communication privée, d'un secret d'Etat ou encore d'une transaction entre un utilisateur et sa banque. Mais, dans chacun des cas, un minimum de sécurité devrait toujours être assuré.

Etant donnés la nature partagée par définition d'un réseau et le nombre ainsi que la diversité des lignes et des commutateurs qui le composent, le réseau se retrouve en première ligne face aux attaques délibérées de personnes malhonnêtes. Il suffit simplement de penser aux personnes qui "piratent" un réseau pour détourner des informations qui ne leur sont pas du tout destinées.

Dès lors, comment une messagerie (que ce soit d'ailleurs une messagerie de type X400 ou une autre) peut-elle assurer à ses utilisateurs que le contenu d'un message délivré à un destinataire est bien identique à celui qui a été rédigé par l'émetteur?

De plus, les correspondants sont-ils assurés que leur message sera gardé secret vis-à-vis d'une tierce personne qui se montrerait trop curieuse? Le destinataire dispose-t-il de moyens pour vérifier que l'émetteur est bien celui qui prétend l'avoir envoyé? La personne qui envoie le message a-t-elle des garanties d'être protégée contre un destinataire qui modifierait à son avantage le contenu du message?

2. Introduction de la sécurité dans X400 : norme X400/88

Une messagerie décrite comme dans le premier chapitre ne présenterait que très peu d'intérêt auprès de ses utilisateurs si elle ne pouvait leur fournir un minimum de confidentialité, de secret dans leurs échanges de messages.

C'est pourquoi, par rapport à sa version 1984 où la question de la sécurité était omise, la version 1988 des recommandations X400 a, entre autres améliorations(2), introduit la notion de sécurité dans une messagerie (cfr. : [X400,88]).

La sécurité y est présentée comme un élément d'extension venant s'ajouter au protocole P1 de transfert de message : la sécurité apparaît donc comme une information figurant sur l'enveloppe.

Plus précisément, chaque élément d'extension à la recommandation X400 de 1984 est composé d'une séquence de trois parties :

- un entier indiquant le type de l'élément d'extension.
- un drapeau (flag) indiquant le caractère critique de l'extension.
- une valeur permettant d'identifier l'élément d'extension.

Un message comprenant des extensions dites critiques est un message dont chaque MTA par lequel il transite doit comprendre la sémantique de ces extensions, sinon le message ne pourra être délivré et l'expéditeur recevra un avertissement de la non livraison. Dans le cas où il ne s'agit pas d'une extension critique, les MTA n'ont pas à se préoccuper de ces éléments lors du transfert du message.

---

(2) Puisque le logiciel utilisé dans le cadre de ce travail est une implémentation d'une messagerie X400/84, nous n'entrerons pas plus en détail dans les améliorations apportées par X400/88 par rapport à X400/84.

De manière plus formelle, un élément d'extension est défini par la séquence :

```
Extension Field ::= SEQUENCE {
        Type          [0]INTEGER;
        Criticality   [1]BITSTRING;
        Value         [2]ANY
    }
```

3. Services de sécurité dans X400/88

En plus de définir la sécurité comme un élément d'extension, les recommandations X400/88 décrivent un ensemble de services de sécurité qu'une messagerie devrait être à même d'offrir à ses utilisateurs(3).

3.1. Authenticité de l'origine du message

Lors d'une communication via une messagerie électronique, un des utilisateurs pourrait envoyer un message sous une fausse identité et tromper ainsi le destinataire sur la véritable identité de son émetteur. C'est cette situation où une personne se fait passer pour une autre qui est appelée mascarade (en anglais : masquerade) et qui peut être mise ne évidence grâce au service d'authenticité de l'origine du message.

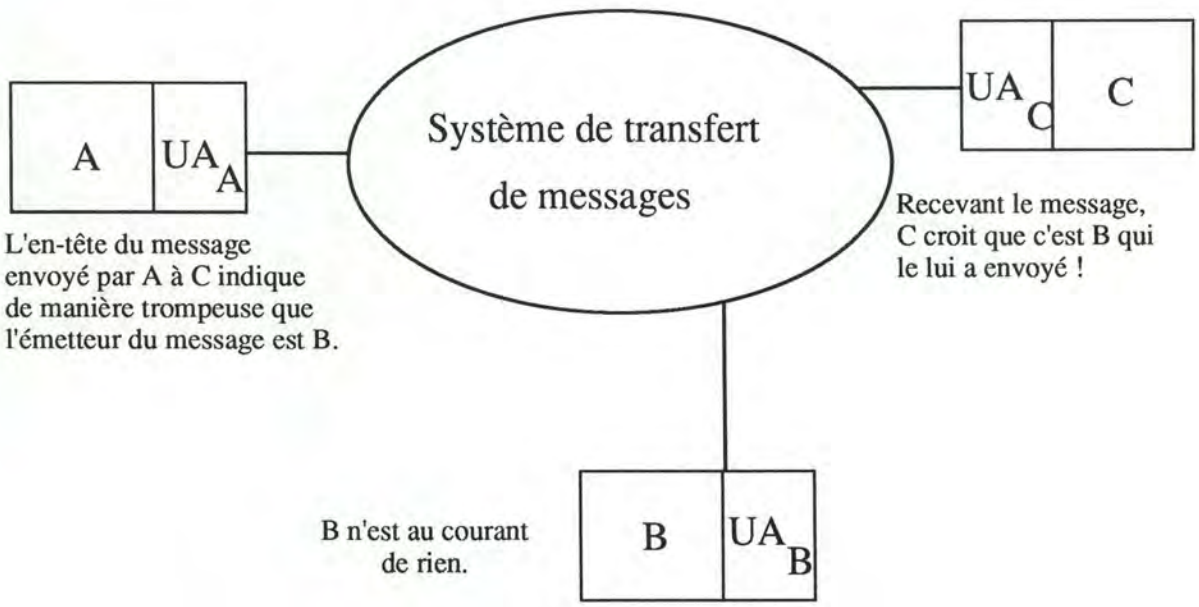


Figure 8 : Mascarade.

Plus généralement, le destinataire mais aussi l'ensemble des différents MTA par lesquels transite le message doivent être capables de vérifier

---

(3) Remarquons que les différentes recommandations ne disent rien quant à la manière de mettre ses services de sécurité en pratique.

l'authenticité de l'identité de celui qui leur transmet le message : c'est ce que permet le service d'authenticité pour chacun des MTA.

Notons dès à présent qu'un tel service pourra être mis en pratique grâce au mécanisme de signature digitale (ce mécanisme sera revu lors du chapitre relatif au chiffrement) : la signature digitale est au message transmis par messagerie électronique ce que la signature manuscrite est à la lettre envoyée par la poste. Elle permet notamment d'identifier de façon univoque et sans ambiguïté le signataire du message.

3.2. Contrôle d'intégrité du contenu du message

Un autre problème pouvant surgir est caractérisé par la situation où le message arrivé chez le destinataire diffère de celui réellement envoyé.

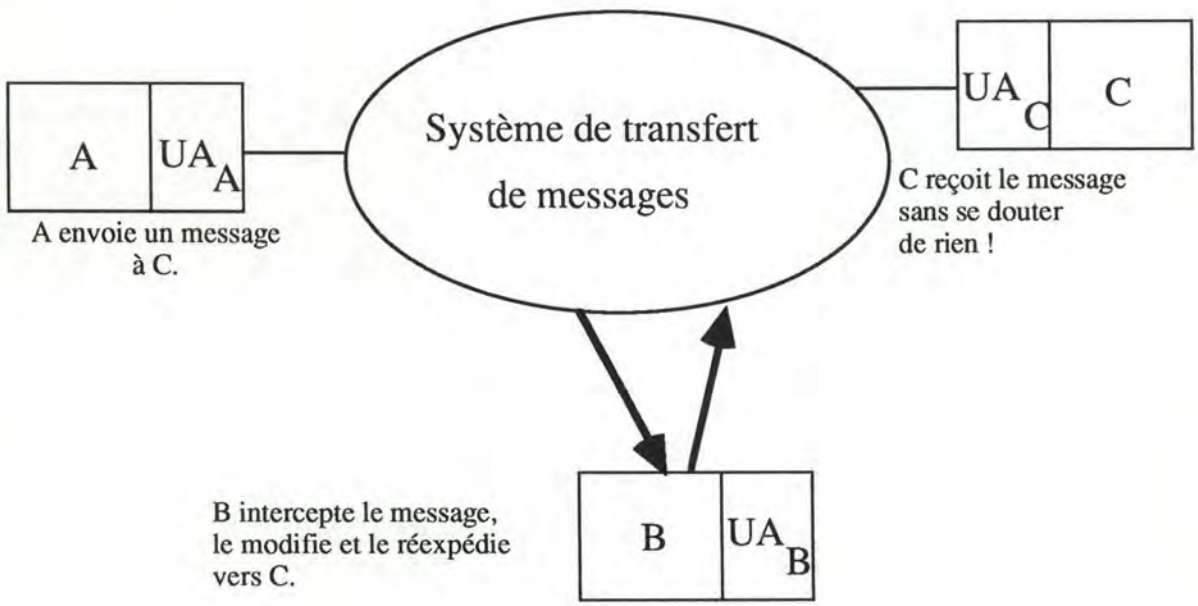


Figure 9 : Modification du message lors de son transfert.

Le service de contrôle d'intégrité du contenu du message permet au destinataire de vérifier que le contenu du message n'a pas été modifié en cours de trajet au sein de la messagerie.

En pratique, le mécanisme de signature digitale sera utilisé pour mettre en œuvre ce service.

3.3. Confidentialité du contenu du message

Un troisième type de menace qui plane sur le bon fonctionnement d'une messagerie est la situation où deux partenaires s'échangent un message à caractère privé sans se douter qu'une troisième personne prend note de manière toute à fait indiscrète de leurs communications.

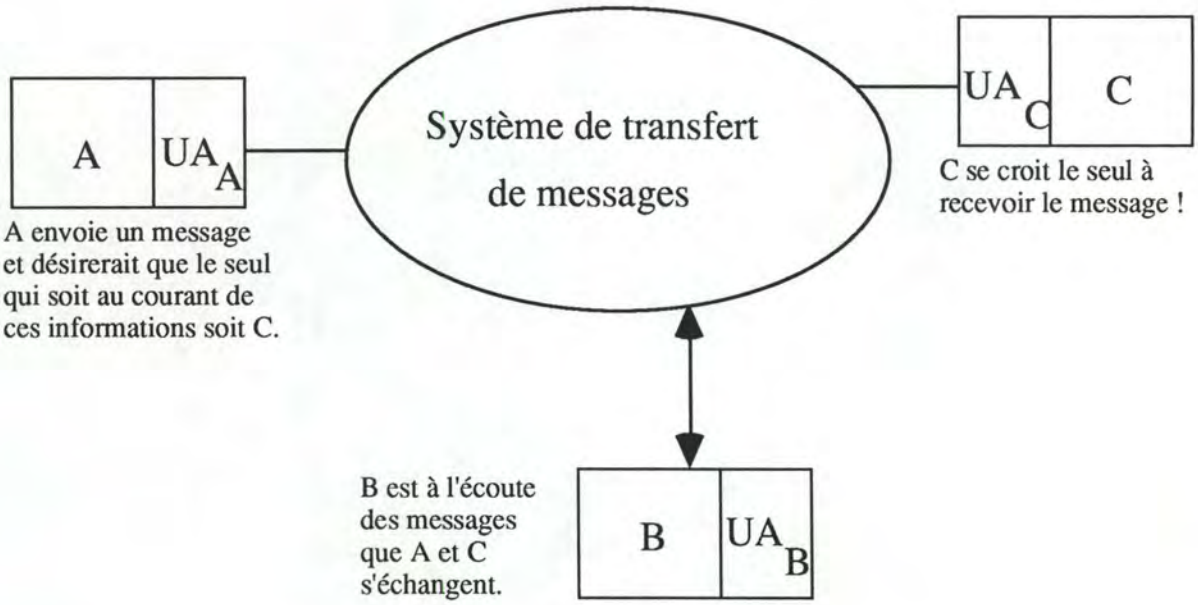


Figure 10 : Non confidentialité du contenu du message.

Le service de confidentialité du contenu du message empêche que le contenu du message ne soit divulgué à un autre utilisateur que le(s) destinataire(s) prévu(s).

En pratique, ce service est réalisé grâce à des techniques de chiffrement qui transforment sur base d'une clé le message de départ en un message illisible pour toute personne ne disposant pas de la clé.

3.4. Services de non répudiation

3.4.1. Non répudiation de l'origine du message

Une autre menace possible est caractérisée par la situation où, après avoir envoyé un message, un utilisateur nie en être l'expéditeur.

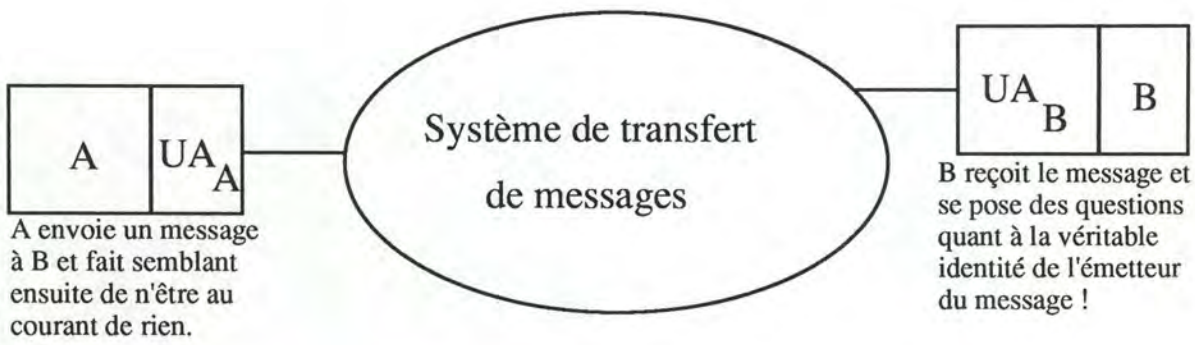


Figure 11 : Répudiation de l'origine du message.

Le service de non répudiation de l'origine du message permet de mettre en évidence et d'éviter cette situation.

3.4.2. Non répudiation de la livraison du message

De manière symétrique à la situation précédente, un destinataire pourrait nier avoir reçu un message.

Figure 12 : Répudiation de la livraison du message.

Le service de non répudiation de la livraison(4) du message permet donc de mettre en évidence et d'éviter cette situation.

Remarquons aussi que les services de non répudiation pourront être mis en pratique grâce à la signature digitale.

4. Emplacement de la sécurité dans une messagerie X400/84

Chacun des services de sécurité décrits pourra être mis en œuvre de manière purement bout à bout (end-to-end) : il n'y aura nul besoin de s'inquiéter de la plus ou moins grande sécurité du système de transfert de messages proprement dit et toute l'attention devra se focaliser au niveau des UAs. C'est cette optique d'intégration des services de sécurité à l'intérieur même de la couche UAL que nous adopterons par la suite.

Notons qu'au lieu d'intégrer les services de sécurité à l'intérieur de la sous-couche UAL, d'autres choix auraient été aussi possibles comme par exemple l'intégration de la sécurité au sein de la sous-couche MTL ou entre les sous-couches UAL et MTL.

---

(4) Par livraison d'un message, il faut comprendre réception d'un message par un utilisateur.

Chapitre 3 : Chiffrement et mécanismes s'y rapportant

La mise en œuvre de services de sécurité dans une messagerie électronique repose sur différents mécanismes propres au domaine de la sécurité et parmi ceux-ci, le chiffrement en est un des plus importants (cfr. : [DAVIES,89], [MAR,87] et [SECURE]).

Le principe de base du chiffrement est le suivant :

Soient - M : le texte écrit en clair par l'émetteur.
- M' : le texte chiffré.
- E : l'algorithme de chiffrement.
- D : l'algorithme de déchiffrement.
- k : une clé.

Le mécanisme de chiffrement du message s'exprime par : $M' = E_{k1}(M)$ où k_1 est une clé de chiffrement. Le message chiffré M' résultant du chiffrement est donc généré à partir de deux paramètres : le texte d'origine et la clé.

Inversement, l'opération de déchiffrement peut être symbolisée par :
$M = D_{k2}(M')$ où k_2 est une clé de déchiffrement.

Deux classes de mécanismes de chiffrement/déchiffrement peuvent être distinguées : le chiffrement à clé secrète appelé aussi chiffrement symétrique et le chiffrement à clé publique appelé aussi chiffrement asymétrique.

Par la suite, un réseau sera dit non sécurisé lorsqu'un intrus aura la possibilité d'y lire, d'y modifier ou d'y détruire des messages ainsi que d'y introduire de nouveaux messages.

1. Chiffrement à clé secrète

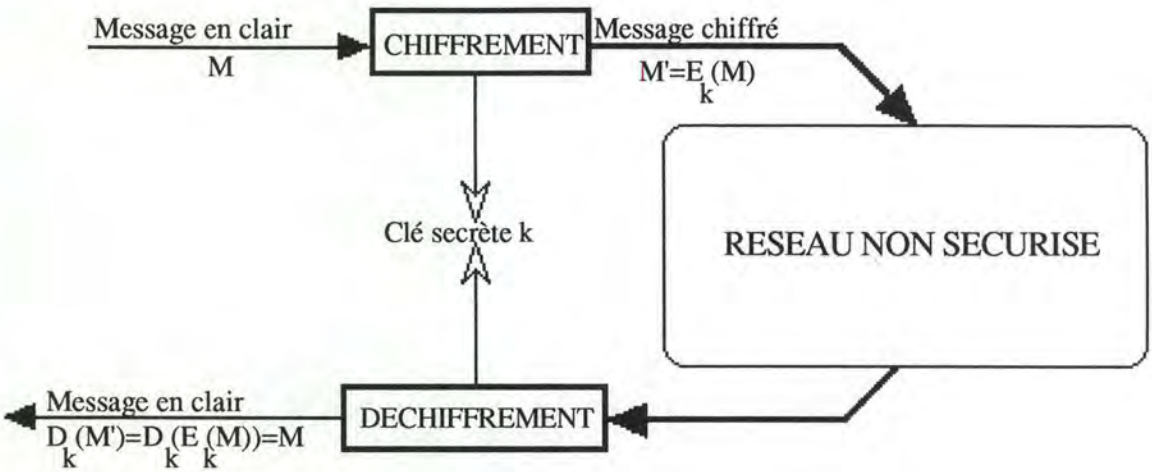


Figure 13 : Chiffrement à clé secrète.

Les opérations de chiffrement et de déchiffrement peuvent se résumer par : $M' = E_k(M)$ et $M = D_k(M')$. L'émetteur et le receveur du message utilisent ici la même clé pour, respectivement, chiffrer et déchiffrer le message. Ce chiffrement est appelé chiffrement symétrique car la connaissance de la clé par chacun des communicants permet la communication dans les deux sens.

Actuellement, l'algorithme de chiffrement à clé secrète le plus souvent utilisé est le standard DES (Data Encryption Standard).

Plusieurs inconvénients peuvent être relevés à l'encontre de la méthode de chiffrement à clé secrète :

- devant posséder la même clé, les deux partenaires ont dû se l'échanger auparavant et cet échange a dû lui-même demander des conditions parfaites de sécurité.
- une "fuite" chez l'un des deux communicants peut se révéler très vite catastrophique pour l'ensemble des deux partenaires; cette grande dépendance mutuelle pourrait parfois se révéler dangereuse. C'est pourquoi, il est préférable que ces clés aient une durée de vie aussi courte que possible, limitée à par exemple une seule communication.
- puisqu'une même clé est partagée par au moins deux utilisateurs, elle ne peut pas être utilisée (comme nous verrons que c'est le cas avec le chiffrement à clé publique) pour éviter la situation de la mascarade.

2. Chiffrement à clé publique

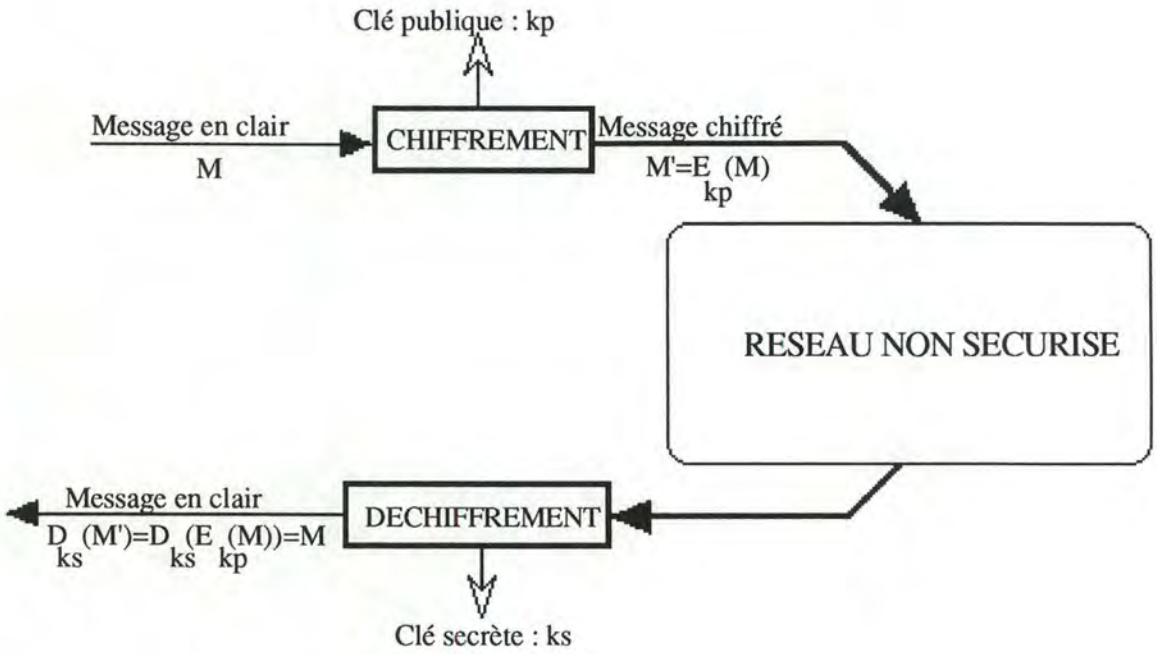


Figure 14 : Chiffrement à clé publique.

Les opérations de chiffrement et de déchiffrement peuvent être résumées par : $M' = E_{kp}(M)$ et $M = D_{ks}(M')$ où k_p est une clé publique et où k_s est une clé secrète.

L'idée originale de ce type de chiffrement est donc d'utiliser une clé k_p publique c'est-à-dire une clé dont tout le monde peut avoir connaissance. Cette clé k_p est telle que tout message chiffré grâce à elle ne peut être déchiffré qu'avec la clé secrète k_s du destinataire. Le point délicat est donc de générer deux clés k_p et k_s liées entre elles de telle manière que l'utilisateur de la clé k_p ne puisse en déduire k_s mais dont k_p a pu être construite à partir de k_s! L'algorithme par excellence qui entre dans cette catégorie est le RSA (Rivest, Shamir et Adelman).

Ce type de chiffrement présente cependant l'inconvénient d'être basé sur le calcul de fonctions comme par exemple la mise en puissance de nombres de plus de 50 chiffres, ce qui demande un équipement hardware assez performant pour éviter des temps CPU trop coûteux.

Par contre, un des avantages par rapport au chiffrement symétrique est évident : il n'est plus nécessaire que les partenaires se soient auparavant échangé de manière risquée une clé secrète.

De plus, le chiffrement asymétrique permet aussi de mettre en œuvre les principes de signature digitale, de certificat et de jeton.

2.1. Principe de la signature digitale

Le mécanisme de signature digitale est très proche de celui de chiffrement à clé publique à la différence près que l'ordre des clés est inversé : le message est cette fois chiffré (on dira que le message est signé) avec la clé secrète : $E_{ks}(M)=M'$ et le déchiffrement (on parlera de vérification de la signature) est réalisé grâce à la clé publique : $D_{kp}(M')=M$.

A défaut d'assurer le caractère confidentiel du message (puisque tout le monde peut avoir accès à la clé publique k_p), la signature peut permettre à une personne quelconque de vérifier l'identité de la provenance ainsi que l'intégrité des informations transmises : la signature digitale est donc plus que le simple équivalent électronique de la signature manuscrite apposée au bas d'une lettre écrite.

Essentiellement, la signature est un mécanisme générant une grandeur qui dépend de l'ensemble des bits du message et qui ne peut avoir été créée que par l'émetteur du message tandis que le(s) destinataire(s) pourra(ont) en vérifier la validité.

Dans le cas où après déchiffrement à la réception, le message obtenu est *incohérent* (5), le destinataire pourra en déduire que le message transmis est différent du message envoyé.

---

(5) Cette notion d'incohérence est fort subjective : dans le cas par exemple où le message attendu par le destinataire est un nombre (par exemple un montant bancaire), comment le destinataire pourra-t-il, et en fonction de quoi, décider que le message reçu est bien cohérent ?

Afin de faciliter la détection de la cohérence du message reçu, la procédure suivante est souvent utilisée :

- une valeur appelée Manipulation Detection Code (MDC) (on parle aussi parfois de Message Integrity Check (MIC)) est calculée à partir de la valeur des informations à transmettre; cette valeur constitue une représentation compacte du message et elle est utilisée pour détecter d'éventuelles modifications dans le message;
- cette valeur est ensuite chiffrée grâce à la clé secrète : $E_{ks}(MDC)$;
- grâce à la clé publique k_p associée à la clé secrète k_s, un utilisateur peut retrouver la valeur du code détecteur : $D_{kp}(E_{ks}(MDC))$ = MDC et comparer ensuite cette valeur avec la valeur directement calculée à partir du message reçu. Dans le cas où ces deux valeurs sont identiques, l'intégrité du contenu du message est effectivement vérifiée.

Notons dès à présent que la signature digitale et le chiffrement sont très fréquemment combinés pour assurer aux messages à la fois le service de confidentialité du contenu du message et celui de contrôle de l'intégrité du contenu du message.

2.2. Certificat

Bien qu'il ne faille pas protéger les clés publiques contre leur divulgation aux yeux de tous, il est important de veiller à ce qu'elles ne soient pas falsifiées. Il faut à tout prix avoir la certitude qu'une clé publique appartient bien à la personne qui prétend en être l'initiatrice.

Imaginons la situation où A désire envoyer un message chiffré à B : pour ce faire, A doit connaître la clé publique associée à B : k_{pB} (en chiffrant le message avec k_{pB}, B sera le seul à pouvoir le déchiffrer grâce à sa clé secrète k_{sB}). Supposons de plus que C a généré de manière frauduleuse une clé publique : k_{pC} en se faisant passer pour B et que k_{pB} a été remplacé par k_{pC}. Donc, ignorant tout de cette fraude, A va chiffrer son message avec k_{pC}.

L'intrus C est maintenant à même de déchiffrer un message qui ne lui est pas destiné et, comble de l'imposture, il peut émettre un tout nouveau message vers B en utilisant cette fois k_{pB}!

Le seul véritable moyen d'empêcher ces agissements est d'éviter que quiconque ait l'occasion de falsifier une des clés publiques mises à la disposition de l'ensemble des utilisateurs : c'est le rôle de l'autorité de certification (Certification Authority(CA)).

Le CA est un site particulier au sein du réseau qui offre des certificats d'identité à propos des clés publiques c'est-à-dire que le CA permet à un utilisateur de vérifier l'authenticité de la clé publique qui lui est fournie.

Au départ, il "suffit" que chacun des utilisateurs possède une copie sûre de la clé publique de CA : k_{pCA} et ainsi, lorsqu'à une demande de clé publique, CA renvoie la clé signée avec sa propre clé secrète (k_{sCA}), la vérification de l'authenticité de la clé publique pourra se réaliser grâce à k_{pCA}.

Le certificat de la clé publique utilisé par l'utilisateur A peut se noter :

$CA(A) = E_{ksCA}$ (sgnAlg , CA , A , k_{pA} , T^A)

où : - sgnAlg désigne l'algorithme de génération de la signature grâce à la clé secrète de CA.

- T^A représente la période de validité du certificat(6).

2.3. Jeton (token)

Tout comme le certificat, le token est une structure signée de données mais, au lieu d'être générée par une autorité de certification, elle l'est par l'émetteur du message lui-même : A { sgnAlg , t^A , B , sgnData , encAlg , k_{pB}[encData] }

où : - sgnAlg désigne l'algorithme utilisé pour générer la signature.

- encAlg désigne l'algorithme utilisé pour le chiffrement.
- B est le nom du destinataire.
- k_{pB} est la clé publique utilisée par B.
- sgnData reprend des paramètres de sécurité signés par k_{sA}.

---

(6) En restreignant la durée de vie du certificat d'une clé publique, le paramètre T^A renforce encore davantage le caractère sécurisé des certificats.

- encData reprend des paramètres de sécurité chiffrés par k_{pB}.
- les paramètres apparaissant dans sgnData et encData dépendent des services de sécurité que l'utilisateur désire appliquer à son message.
- t^A est un indicateur de temps(7).

Puisque les données sont chiffrées grâce à la clé publique du destinataire B, seul ce dernier pourra les déchiffrer (opération réalisée grâce à sa clé secrète : k_{sB}); de plus, en utilisant la clé publique k_{pA} correspondant à la clé secrète de l'émetteur pour vérifier la signature apparaissant sur le token, B pourra de ce fait vérifier l'intégrité du contenu du message.

La structure de jeton permet donc de combiner le service de confidentialité avec celui de contrôle d'intégrité.

---

(7) t^A permet d'éviter la situation où un ennemi ayant trouvé la valeur de kt_A se fait passer pour A en employant d'anciennes valeurs de la clé de session.

Chapitre 4 : Analyse du service de confidentialité

Ne pouvant pas implémenter tous les services de sécurité, nous nous sommes particulièrement intéressés à l'analyse et à la mise en œuvre du service de confidentialité du contenu du message.

Comme son nom l'indique, le service de confidentialité du contenu du message permet de garder secret vis-à-vis d'un tiers trop curieux le contenu du message.

En partant d'une situation de base très simplifiée, la mise en œuvre de ce service va petit à petit être affinée pour aboutir finalement à des situations relativement complexes qui, en plus du service de confidentialité, vont inclure les services de contrôle d'intégrité et d'authentification (cfr. [SECURE] et [DAVIES,89]) :.

Reprenons les notations :

- M : le message d'origine;
- M' : le message chiffré;
- E : l'algorithme de chiffrement;
- D : l'algorithme de déchiffrement;
- k : une clé.

Et plaçons-nous dans la situation où un utilisateur A désire envoyer un message à un utilisateur B via un système de courrier électronique.

1. Situation 1

La manière la plus simple de dissimuler des informations qu'on souhaite rendre confidentielles est l'utilisation des mécanismes de chiffrement :

Chiffrement symétrique	Chiffrement asymétrique
- A génère M.	- A génère M.
- A envoie à B : $E_k(M)$.	- A envoie à B : $E_{kpB}(M)$.
- B déchiffre M : $D_k(E_k(M))=M$.	- B déchiffre M : $D_{ksB}(E_{kpB}(M))=M$.

2. Situation 2

La situation précédente ne précise rien quant au problème de la gestion des clés : de quelle manière les utilisateurs voulant entrer en communication au sein d'un réseau peuvent-ils par exemple prendre possession de ces clés?

2.1. Cas du chiffrement à clé privée

Pour que A et B puissent utiliser la même clé k dans leurs opérations de chiffrement et de déchiffrement du message, ils ont dû précédemment se l'échanger ... de manière confidentielle!

Ce problème qui, à première vue, semble cyclique et sans solution peut être résolu si par exemple les deux partenaires s'échangent la clé directement de main à main. Il s'agit donc de trouver un canal d'échange sécurisé qui contourne le problème du manque de confidentialité d'un réseau non sécurisé.

2.2. Cas du chiffrement à clé publique

Puisque le chiffrement se réalise via une clé publique, il n'y a pas lieu de tenir compte de conditions de sécurité particulières pour la diffusion de ce genre de clé. Néanmoins (cfr. chapitre relatif au chiffrement et aux mécanismes s'y rapportant), la gestion des clés publiques doit être traitée par une autorité de certification qui permet d'éviter la falsification de ces clés.

3. Situation 3

La situation présentée au point 2.1. présente le désavantage d'associer la même clé de chiffrement à toute communication entre deux partenaires donnés.

Donc, si à un instant donné, un fraudeur découvre d'une manière ou d'une autre cette clé, il pourra par la suite déchiffrer l'ensemble des messages transitant de A vers B.

Une solution serait alors de limiter la durée de vie de la clé de chiffrement de manière à limiter les dégâts engendrés dans le cas de sa

divulgation à un tiers. L'option souvent retenue est de considérer que la clé de chiffrement n'est valide que pendant un seul envoi de message.

Mais, de quelle manière procéder pour que les deux partenaires se mettent d'accord, préalablement à toute communication, sur la valeur des différentes clés de chiffrement?

- une solution peu commode serait de s'échanger de main à main une liste de clés de chiffrement qui, au fur et à mesure des communications, serait parcourue; ainsi, lors de la 14ème communication de A vers B, la 14ème clé de la liste serait utilisée! Mais que se passerait-il lors du vol ou de la perte de cette liste ... et n'est-il pas irréaliste et coûteux de devoir posséder une liste différente pour chacun des utilisateurs avec qui on communique?

- une solution plus pratique est considérer une hiérarchie au sein des clés :

  - soit k_n la clé de chiffrement du message dont la durée de vie est limitée à une seule communication (une seule session) : kn est appelée **clé de session** (session key).
  - soit k_{tX} la clé de chiffrement associée à l'utilisateur X : cette clé est appelée **clé de terminal** (terminal key).
  - soit k_m la **clé maître** (master key) qui se situe au sommet de la hiérarchie.

Les clés de terminal, dont la durée de vie n'est pas aussi limitée que celle des clés de session, sont principalement utilisées pour le transfert des k_n : ce sont donc des clés de chiffrement qui servent à chiffrer d'autres clés de chiffrement. Ces clés sont associées à chacun des utilisateurs du système de telle manière qu'une clé donnée est uniquement connue par son propriétaire et par ce qui est appelé un centre de distribution de clés (cdc).

Le centre de distribution de clés constitue un site privilégié au sein du réseau dont les tâches essentielles sont la création des clés de session ainsi que le stockage sécurisé des clés de terminal. Ce stockage sécurisé consiste à garder les clés de terminal sous la forme chiffrée : $E_{km}(k_t)$ où la clé k_m est uniquement connue par le cdc.

Les clés de session, de terminal et la clé maître permettent donc de chiffrer à trois niveaux différents :

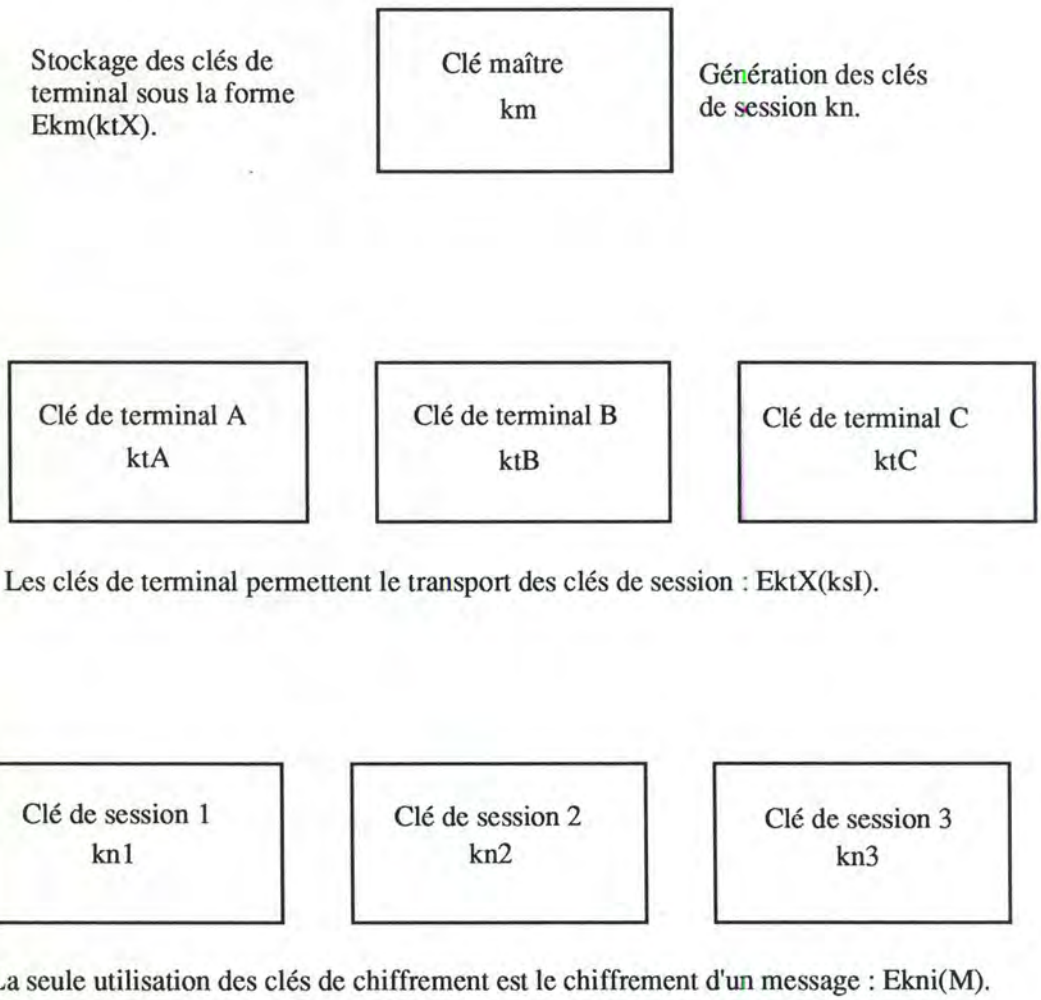


Figure 15 : Hiérarchie des clés de chiffrement.

La mise en pratique du service de confidentialité dans le cadre du chiffrement symétrique peut maintenant être décrit par l'enchaînement des actions :

- A demande au cdc la génération d'une clé de session kn et il lui indique l'identité de son correspondant : B;

- cdc répond à A en lui envoyant $E_{ktA}(k_n)$ de manière à ce que A soit le seul capable d'utiliser k_n : $D_{ktA}(E_{ktA}(k_n)) = k_n$;

- cdc envoie aussi à B : $E_{ktB}(k_n)$, ce qui permet à B de disposer de la clé de déchiffrement du message : $D_{ktB}(E_{ktB}(k_n)) = k_n$;

- A chiffre alors le message : $E_{kn}(M)$;

- lors de la réception, B déchiffre le message : $D_{kn}(E_{kn}(M)) = M$.

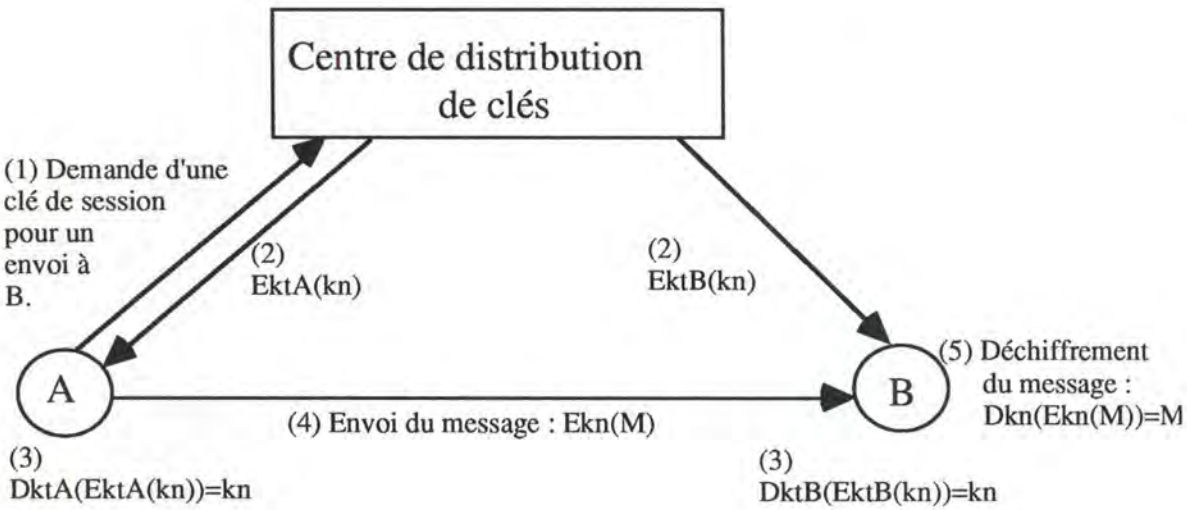


Figure 16 : Service de confidentialité (1).

Cette situation où B prend possession d'une clé de déchiffrement avant même de recevoir le message chiffré pose de gros problèmes dans le cas où plusieurs utilisateurs désirent communiquer avec B. En effet, le destinataire devra choisir dans un ensemble de clés k_n celle qui convient au message reçu. Et, même si le cdc accompagne l'envoi de k_n de l'identité de l'émetteur, B devra gérer un stock de clés et décider des mesures à prendre dans la cas où par exemple A n'enverrait finalement pas de message suite à une panne.

Pour ces raisons, on préférera le scénario :

- A demande au cdc la génération d'une clé de session k_n et il lui indique l'identité de son correspondant : B;

- cdc répond à A en lui envoyant $E_{ktA}(k_n, E_{ktB}(k_n))$;

- A déchiffre la réponse de cdc et obtient : $D_{ktA}(E_{ktA}(k_n, E_{ktB}(k_n)) = k_n, E_{ktB}(k_n)$;

- A chiffre le message et envoie à B : $(E_{kn}(M), E_{ktB}(k_n))$;

- B opère un premier déchiffrement : $D_{ktB}(E_{ktB}(k_n)) = k_n$ qui lui permet alors d'obtenir le message en clair : $D_{kn}(E_{kn}(M)) = M$.

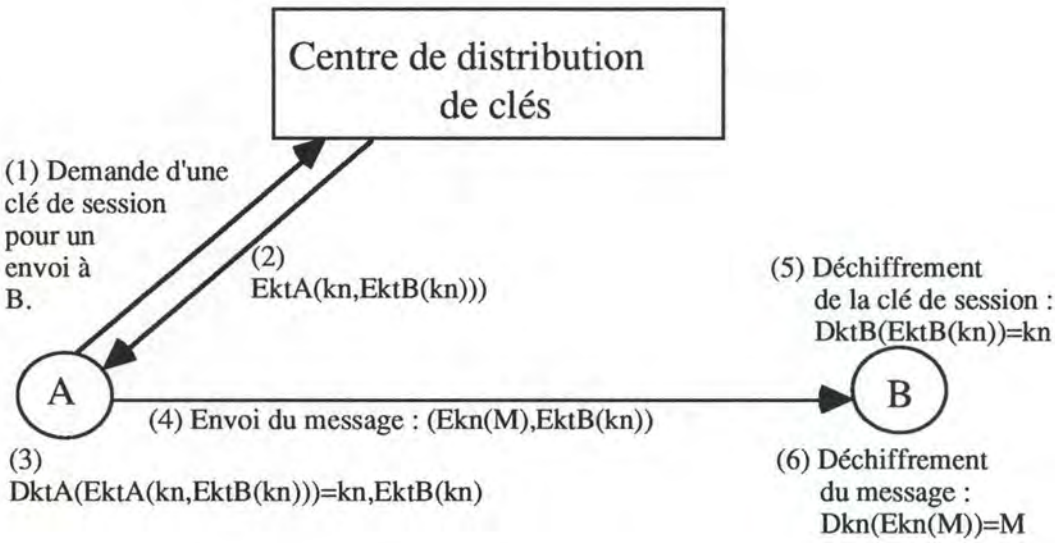


Figure 17 : Service de confidentialité (2).

4. Situation 4

La situation précédente, bien que protégeant la clé de session de toute découverte par un intrus, ne résout pas tous les problèmes :

- comment éviter par exemple qu'un utilisateur X demande en se faisant passer pour A une clé de session au cdc?

- que se passe-t-il aussi si un "ennemi" prend la place du cdc et se met à distribuer des clés qui ont déjà été envoyées?

C'est pourquoi, la situation 4 a pour but de traiter les problèmes liés à l'authenticité de l'identité de A, B et cdc.

Authentification de l'identité de cdc par A

Lors de la phase d'acquisition de la clé de session, A doit s'assurer qu'il est bien en communication avec le cdc et qu'il ne se trouve pas en présence d'un intrus qui tenterait de lui envoyer une ancienne valeur de k_n :

- A envoie sa demande de génération de clé accompagnée d'un nombre aléatoire r_A;

- le cdc répond à A en envoyant : $E_{ktA}(r_A, k_n, E_{ktB}(k_n))$;

- A déchiffre la réponse : $D_{ktA}(E_{ktA}(r_A, k_n, E_{ktB}(k_n))) = r_A, k_n, E_{ktB}(k_n)$.

En comparant la valeur initiale r_A avec celle qui lui a été renvoyée chiffrée, A peut s'assurer qu'il a bien communiqué avec cdc puisqu'à part A, le seul qui puisse chiffrer un message avec la clé de terminal k_{tA} est le cdc.

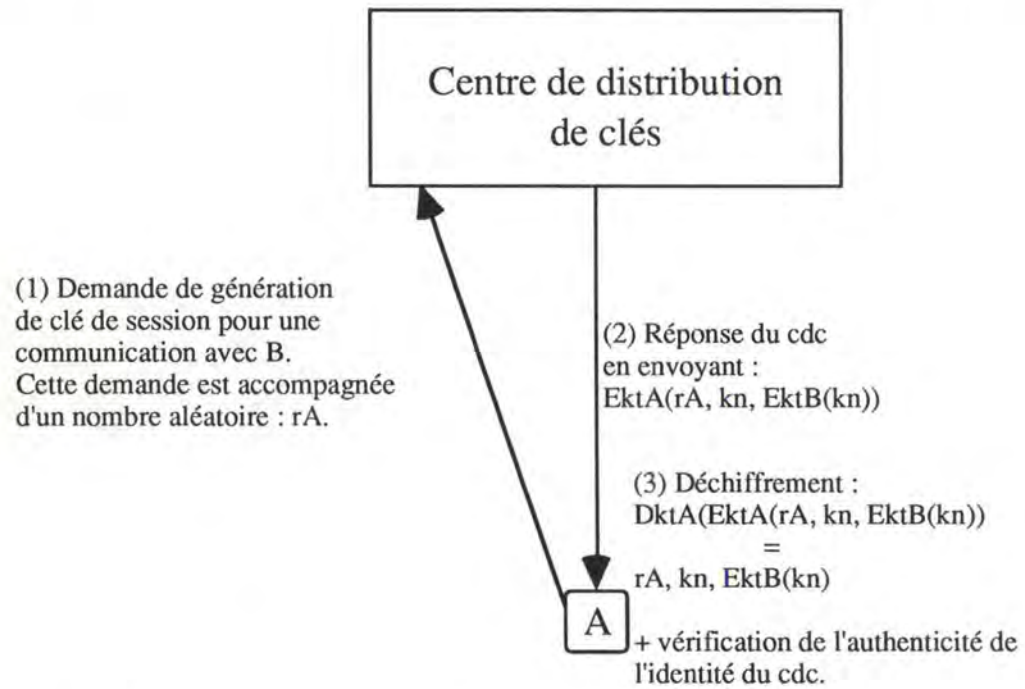


Figure 18 : Phase d'acquisition de la clé de session avec authentification de cdc par A.

Authentification de l'identité du cdc par B

De manière tout à fait analogue à ce que fait A pour s'assurer qu'il est bien en relation avec le cdc, B peut envoyer un nombre aléatoire r_B au cdc.

Authentification de l'identité de A (et de B) par le cdc

Lors de la phase d'acquisition de la clé de session, il n'est pas utile pour le cdc de s'assurer que la demande provient effectivement de A puisque sa réponse est chiffrée avec ktA, clé connue seulement par A.

Authentification mutuelle de A et de B

Lors de la phase d'envoi d'un message de A vers B, les deux utilisateurs doivent être sûrs de l'authenticité de l'identité de leur correspondant. C'est pourquoi, préalablement à l'envoi proprement dit du message et suite aux étapes d'acquisition de la clé de session par A et de transfert de celle-ci jusque B, on peut imaginer les étapes suivantes pour l'authentification mutuelle :

- A génère un nombre aléatoire n_A et envoie à B : $E_{kn}(n_A)$;

- B déchiffre l'envoi de A;

- B prépare sa réponse dans laquelle le nombre n_A est concaténé avec un nouveau nombre aléatoire n_B;

- B envoie à A : $E_{kn}(n_A+n_B)$;

- A déchiffre la réponse de B : $D_{kn}(E_{kn}(n_A+n_B)) = n_A+n_B$;

- A ce stade, A est sûr de communiquer avec B;

- pour que B puisse authentifier l'identité de A, il suffit que A renvoie à B : $E_{kn}(n_B)$ et B n'a plus qu'à effectuer le déchiffrement.

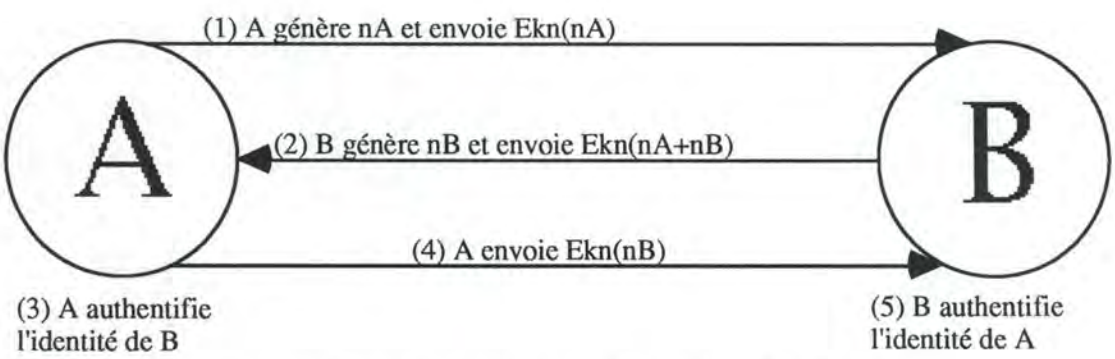


Figure 19 : Authentification mutuelle de A et B.

5. Situation 5

Pour éviter le cas où un même message authentique serait envoyé à plusieurs reprises, il est souvent ajouté aux messages échangés un identifiant unique de transaction (on utilisera le terme de nonce). Cet identifiant peut prendre la forme d'une quantité fonction de la date et de l'heure de l'envoi. En vérifiant "l'état de fraîcheur" des informations échangées, les utilisateurs peuvent éviter cette situation dite situation de rejeu (cfr. : [DENN,81]).

6. Situation 6

Malgré l'ensemble des améliorations apportées, des brèches peuvent encore être trouvées. Imaginons ainsi que le receveur du message soit une personne malhonnête et décide de falsifier lui-même à son avantage le message qui lui a été envoyé : il ajoute par exemple un ou deux zéros au montant d'un chèque qui lui est destiné.

Dans cette situation particulière et dans le cas plus général de dispute entre les deux communiquants, une tierce personne (un notaire) doit disposer de moyens lui permettant d'arbitrer la dispute : c'est ce que permet la signature digitale. Dorénavant, nous nous placerons donc dans le cadre du chiffrement à clé publique qui permet de simplifier grandement les problèmes rencontrés pour la distribution de la clé de session dans le chiffrement à clé secrète.

C'est pour éviter ces problèmes que dans de très fréquents cas, le service de confidentialité du contenu va de pair avec celui de contrôle de l'intégrité du contenu du message :

- les données dont on désire s'assurer de l'intégrité sont signées avec la clé secrète de l'émetteur : EksA(M);

- le tout est ensuite chiffré avec la clé publique associée au destinataire : EkpB(EksA(M));

- B effectue le déchiffrement : DksB(EkpB(EksA(M))) = EksA(M);

- B vérifie ensuite la signature : DkpA(EksA(M)) = M.

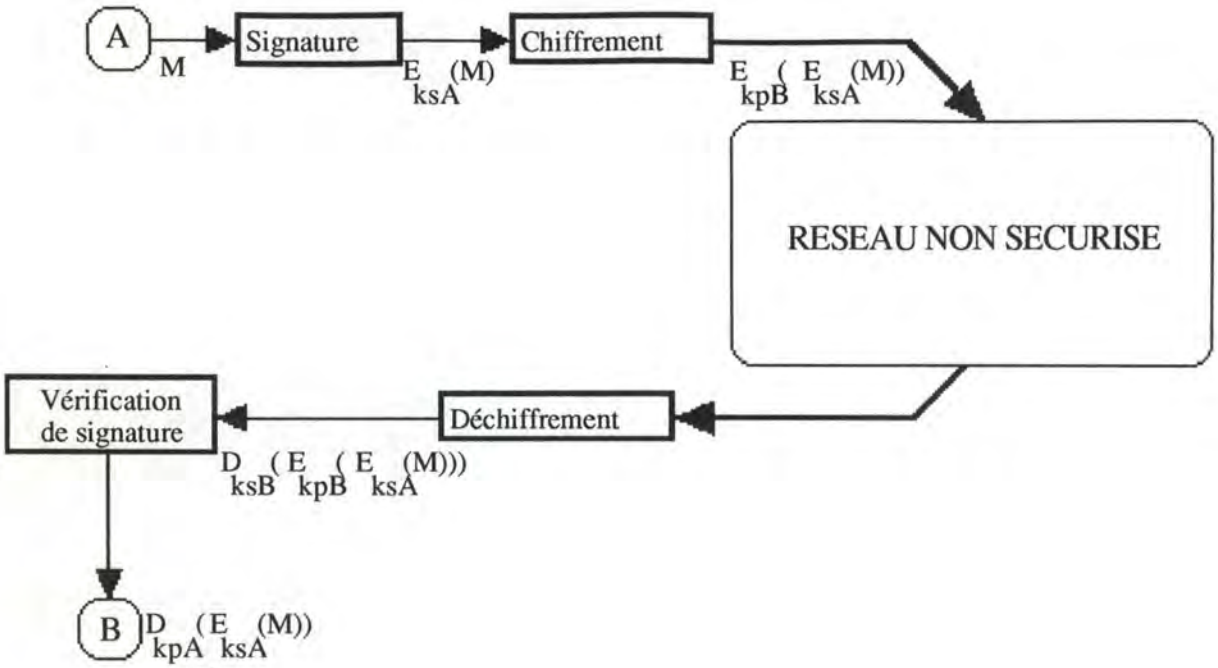


Figure 20 : Combinaison des services de confidentialité et de contrôle d'intégrité.

L'utilisation de la signature digitale (quantité qui ne peut être créée que par l'émetteur du message mais vérifiée par quiconque le désire) permet donc d'éviter les différents cas possibles de disputes :

- le récepteur crée un message de toutes pièces et essaie de faire croire qu'il lui a été envoyé (situation de **forgery**);
- l'émetteur a envoyé un message mais nie ensuite l'avoir fait (situation de **répudiation de l'origine**);
- l'émetteur essaie de faire croire qu'il a envoyé un message alors qu'il n'en a rien fait (situation de **falsification de l'origine**);
- le récepteur nie avoir reçu un message (situation de **répudiation de la livraison**).

7. Situation 7

Une situation finale combinant le service de confidentialité avec ceux d'authentification et de contrôle d'intégrité dans le cas du chiffrement à clé publique peut être la suivante :

- A envoie au CA un identifiant de son identité et un identifiant de B : a,b;

- CA recherche la clé publique associée au destinataire : k_{pB};

- CA renvoie à A : $E_{ks(CA)}(k_{pB},b)$ où ks(CA) est la clé secrète de CA;

- A déchiffre la réponse de CA : $D_{kp(CA)}(E_{ks(CA)}(k_{pB},b)) = k_{pB},b$;
  Cette vérification de signature sur la valeur b permet à A d'authentifier CA; de plus, la clé publique k_{pB} est donc sûre car signée par CA.

- A envoie à B : $E_{kpB}(I_A,a)$ où I_A est un "nonce" qui évite le rejeu;

- B déchiffre : $D_{ksB}(E_{kpB}(I_A,a)) = I_A,a$;

- B envoie à CA : a,b;

- CA recherche la clé publique associée à l'émetteur : k_{pA};

- CA renvoie à B : $E_{ks(CA)}(k_{pA},a)$;

- B déchiffre la réponse de CA : $D_{kp(CA)}(E_{ks(CA)}(k_{pA},a)) = k_{pA},a$;
  Cette vérification de signature permet à B d'authentifier CA et d'être sûr de la valeur de la clé publique k_{pA}.

- B envoie à A : $E_{kpA}(I_B,I_A)$ où I_B est un "nonce";

- A effectue : $D_{ksA}(E_{kpA}(I_B,I_A)) = I_A,I_B$;
  Ceci permet à A d'authentifier B.

- A renvoie alors à B : $E_{kpB}(I_B)$;

- B effectue enfin : $D_{ksB}(E_{kpB}(I_B)) = I_B$, ce qui lui permet d'identifier A;

- l'envoi du message peut alors se faire comme décrit dans la situation 6.

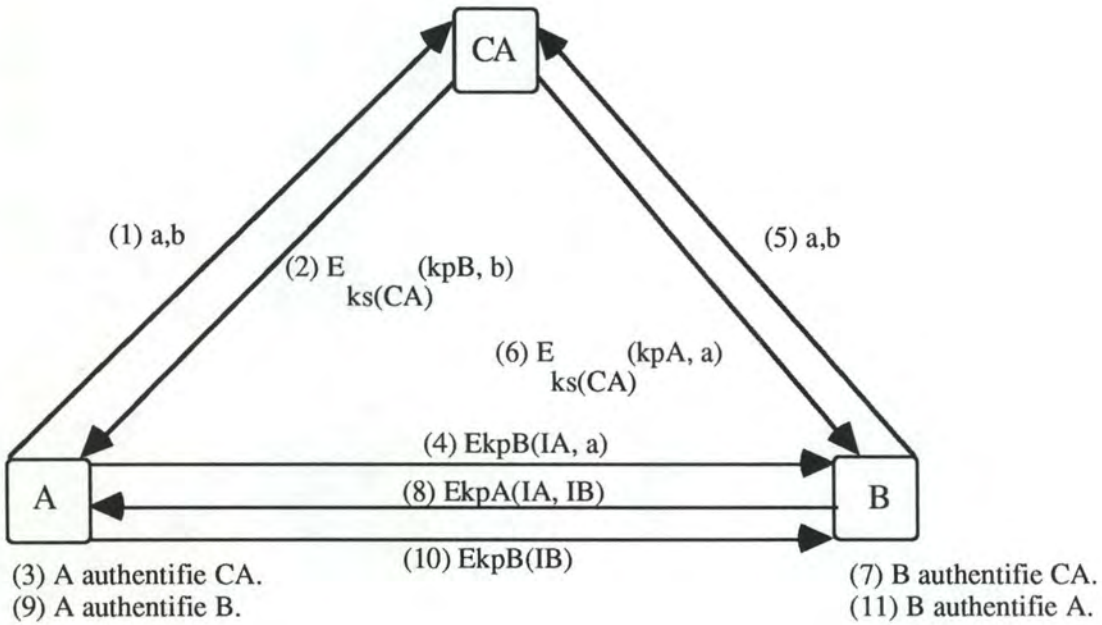


Figure 21 : Mise en œuvre du service de confidentialité (situation 7).

Chapitre 5 : PEM (Privacy Enhanced Mail)

1. Introduction

En vue d'insérer des services de sécurité au sein d'une messagerie X400, nous nous sommes intéressés à ce qui existait déjà par ailleurs au sein du monde Internet.

En février 1993, quatre recommandations ([BAL,93], [KAL,93], [KEN',93] et [LIN,93]) ont donné naissance à la notion de PEM : Privacy Enhancement for Internet Electronic Mail. PEM offre un standard qui permet d'améliorer en terme de sécurité les systèmes de courrier électronique au sein de Internet.

L'idée à la base de PEM est sa compatibilité avec les environnements de courrier électronique déjà existants : ce qui permet de ne pas devoir reconstruire de toutes pièces de nouveaux systèmes de courrier électronique. C'est exactement ce dont nous avons besoin pour notre travail au sein de X400.

PEM offre à ses utilisateurs les services de sécurité concernant la confidentialité, l'intégrité du contenu du message ainsi que l'authentification et la non répudiation de l'émetteur.

Notons en outre que des recherches sont actuellement menées en vue d'étendre PEM pour pouvoir l'utiliser dans le cadre de MIME (Multipurpose Internet Mail Extensions) ce qui permettrait d'allier messagerie sécurisée et possibilités du multimédia.

2. Environnement de messagerie PEM

Voyons à partir d'un exemple comment PEM peut venir se positionner par rapport à une architecture de messagerie préexistante (cfr. : [KEN,93]).

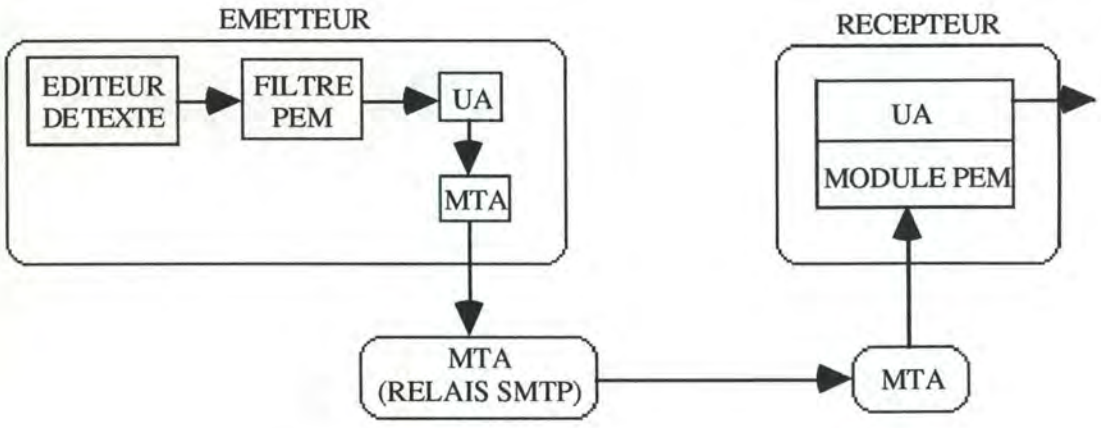


Figure 22 : Environnement de messagerie PEM.

PEM est construit de manière à être transparent en ce qui concerne les différents MTAs assurant ainsi une compatibilité maximale avec les systèmes de transfert de messages déjà existants : ce qui permet donc d'utiliser simplement SMTP pour transférer les messages PEM.

Deux approches pour l'implémentation de PEM se présentent :

- PEM peut être implanté comme un filtre qui traite les données du message édité par l'utilisateur et qui les transmet ensuite à l'UA (c'est ce choix qui est représenté du côté émetteur); cette approche qui dissocie PEM et UA peut s'avérer assez lourde à gérer.
- c'est pourquoi une seconde approche propose une solution plus intégrée qui insère le module PEM au sein même de l'UA (cfr. sur la figure : côté récepteur du message).

3. Hiérarchisation des clés

On appelle clés d'échange (Interchange Keys (IKs)) les clés qui sont utilisées pour chiffrer les clés qui servent au chiffrement proprement dit des données (ces dernières étant appelées Data Encrypting Keys (DEKs)).

Notons qu'en ce qui concerne la dénomination des clés, il y a équivalence entre d'une part les notions de IK et de clé de terminal et d'autre part entre DEK et clé de session.

4. Les messages PEM

4.1. Représentation des messages

PEM utilise une approche où le message chiffré doit pouvoir être représenté de manière uniforme pour l'ensemble des agents utilisateurs du système sans tenir compte de systèmes de représentation particuliers. Le message doit donc être transformé en une forme uniforme particulière pour pouvoir être transmis depuis l'émetteur où il est chiffré jusqu'au destinataire où il est déchiffré sans avoir besoin de transformation en représentation intermédiaire.

La transformation utilisée est une transformation en quatre étapes :

- la forme locale : le message est écrit dans la "langue maternelle" de l'agent utilisateur (c'est-à-dire en respectant par exemple une syntaxe particulière ou un emploi de caractères spéciaux spécifiques au poste de travail utilisé); c'est sous cette forme que le message est prêt à être soumis.

- la forme canonique : le message est converti en une forme canonique universelle. C'est à partir de cette forme que pourra s'effectuer l'étape de chiffrement ou celle de calcul d'une quantité liée à la vérification de l'intégrité du message : MIC (Message Integrity Check). Cette représentation évite donc que dans un environnement interconnectant des ordinateurs disparates chacun d'entre eux doive effectuer une traduction du message en une forme qui lui permette de communiquer avec une autre machine.

Cette forme canonique est celle qui est spécifiée dans la recommandation RFC 822 et qui est utilisée par SMTP dans le cadre d'une

messagerie non sécurisée. Mais PEM ne se limite pas à l'utilisation exclusive de SMTP : nous pouvons tout aussi bien imaginer l'utilisation de PEM dans le contexte d'une messagerie électronique X400. La seule restriction sera de limiter à une seule forme canonique le nombre de formats correspondant à un message soumis. Donc, dans le cas où les destinataires d'un même message appartiendrait à des systèmes de messagerie différents, le message devra être soumis plus d'une fois pour pouvoir être mis successivement sous les différentes formes canoniques requises.

- étape d'authentification et de chiffrement : ces opérations n'agissent donc pas sur la forme locale du texte du message mais bien sur sa forme canonique. Ainsi, lorsqu'un utilisateur reçoit un message, il effectue d'abord le déchiffrement de celui-ci avant de passer à la forme du message adaptée à la station utilisée.

- étape de codage vers une forme imprimable : la suite de bits issue de la troisième étape est transformée en une forme universellement représentable entre tous les sites et composée d'un nombre assez restreint de caractères. Cette étape a essentiellement comme but de transformer le message en une forme qui sera protégée contre les perturbations et les modifications durant le trajet dans les différents "sous-réseaux" traversés. Cette représentation est compatible avec la mise sous forme canonique et avec la plupart des passerelles entre systèmes de messagerie. Insistons sur le fait que cette étape est essentielle car la moindre erreur, ne fût-ce qu'un seul bit erroné, dans la forme canonique du message provoquerait une catastrophe lors de la vérification de l'intégrité du message.

Les différentes transformations peuvent donc se résumer par :

Au niveau de l'émetteur :

Forme-transmise = Codage (Chiffrement (Forme-Canonique (Forme-locale)))

Au niveau du destinataire :

Forme-locale = Forme-Canonique^{-1} (Déchiffrement (Décodage (Forme-transmise)))

4.2. En-tête d'un message PEM

Après être passé par chacune des étapes de transformation, le corps du message est prêt à être envoyé via le réseau. A ce corps de message est associée une série de champs relatifs à la sécurité qui vont apparaître dans l'en-tête.

Figure 23 : Structure schématique d'un message PEM.

Voyons plus précisément quel en est le format :

----- BEGIN PRIVACY-ENHANCED MESSAGE -----
(limite supérieure)

En-tête (partie reprenant les informations relatives à la sécurité comme par exemple la clé de chiffrement du message)

Message chiffré (corps du message)

----- END PRIVACY-ENHANCED MESSAGE -----
(limite inférieure)

Figure 24 : Structure d'un schéma PEM.

Prenons un exemple de message PEM :

```
----- BEGIN PRIVACY-ENHANCED MESSAGE -----
Proc-Type: 4,ENCRYPTED
Content-Domain: RFC822
DEK-Info: DES-CBC,F8143EDE5960C597
Originator-ID-Symmetric: linn@zendia.enet.dec.com,ptf-kmc,3
Recipient-ID-Symmetric: pem-dev@tis.com,ptf-kmc,4
Key-Info: DES-ECB, RSA-MD2, 161A3F75DC82EF29,
        E2EF532CCBCFF79658CBCD245454564987

LLrHBOeJ/yfuyef4684gfuhgf496ef4s4efg6KLLHD4534efhgfeuUUGF5454hgfre
hgfjf6564jghnvfn/vfegf664FYDF41fegfGUGHUI4534ghrugh554ffuegfhfhgf5
44hfugGIHGTDFH45645jhregfhregjegfk545646rehgurzGYUHef54regfg44gefg
6fhhfrhgzGGUHI546g4r==
----- END PRIVACY-ENHANCED MESSAGE -----
```

Voyons brièvement, champ après champ, les différentes informations de sécurité pouvant apparaître dans l'en-tête d'un message PEM.

Le champ **"Proc-Type"** permet d'identifier le processus appliqué au message. Ce champ est composé de deux parties séparées par une virgule :

- un nombre mis à 4 dans le cas où la recommandation RFC 1421 (norme définissant les procédures de chiffrement et d'authentification concernant PEM) est respectée; place est laissée à d'éventuelles extensions ultérieures.

- un (ou un ensemble de) string(s) pouvant prendre les valeurs :
  - "ENCRYPTED" : le message à envoyer a été traité par les services de confidentialité et d'intégrité du contenu ainsi que par ceux d'authentification et de non-répudiation de l'origine du message.
  - "MIC-ONLY" : le message est soumis aux services d'intégrité du contenu du message, d'authentification et de non-répudiation de l'origine; dans ce cas-ci, l'accent est mis sur la protection du message contre d'éventuelles modifications plutôt que sur le caractère confidentiel du contenu du message.

- "MIC-CLEAR" : le message dispose des mêmes services que pour le cas du champ "MIC-ONLY" mais le message n'est pas transformé en une forme imprimable (cfr. : 4ème étape de la représentation des messages). Le message s'expose donc à d'éventuelles modifications lors de son transfert. Par conséquent, un message de type "MIC-CLEAR" ne pourra être correctement validé que s'il n'a subi aucune modification en cours de transport entre l'émetteur et le destinataire (et donc le MIC sera inchangé); la validation sera vouée à l'échec dans le cas contraire. En revanche, l'avantage d'un tel type de message est de permettre à des utilisateurs n'ayant aucune connaissance de PEM de pouvoir tout de même lire le message (seuls les véritables utilisateurs PEM seront néanmoins à même d'en vérifier l'intégrité).

- "CLR" (Certificate Revocation List) : ce string désigne un type spécial de message relatif à l'annulation de listes de certificats; nous y reviendrons plus en détail lorsque nous aborderons le point relatif aux certificats au sein de PEM.

Le champ "**Content-Domain**" définit le type de contenu; nous nous contenterons de constater qu'il est initialisé à "RFC822", signifiant par là que le message respecte cette norme.

Le champ "**DEK-Info**" contient les éléments relatifs au chiffrement utilisé pour le message: son identifiant ainsi que les paramètres utilisés. Dans notre exemple, le message a été chiffré en utilisant DES (Data Encryption Standard) en mode chaînage de blocs chiffrés (CBC). Le paramètre situé après la virgule est utilisé par l'algorithme de chiffrement; c'est ce qu'on appelle le vecteur initialisation.

Le champ "**Originator-ID**" identifie l'émetteur du message ainsi que sa clé d'échange. Dans notre exemple, un chiffrement de type symétrique a été choisi, c'est pourquoi l'intitulé de champ "Originator-ID-Symmetric" apparaît.

Ce champ est composé de trois parties :

- l'identifiant d'une entité liée avec une clé d'échange particulière;
- l'identifiant de l'autorité qui a produit cette clé;
- l'identifiant qui permet de distinguer parmi les différentes clés issues d'une même autorité.

Originator-ID-Symmetric: linn@zendia.enet.dec.com,ptf-kmc,3 signifie donc que le nom qui permet d'identifier l'émetteur du message est :

linn@zendia.enet.dec.com et que la clé d'échange utilisée est la clé n°3 produite par l'autorité de certification pft-kmc.

Le champ "**Originator-Certificate**", seulement présent dans le cas où le chiffrement asymétrique est utilisé par au moins un des destinataires, désigne le certificat de la clé publique de l'émetteur.

Le champ "**MIC-Info**", seulement présent dans le cas où le chiffrement asymétrique est utilisé par un des destinataires, comprend plusieurs arguments : l'identifiant de l'algorithme utilisé pour calculer le MIC, l'identifiant de l'algorithme utilisé pour signer le MIC ainsi que le MIC signé.

Le champ "**Issuer-Certificate**", seulement présent dans le cas du chiffrement asymétrique, contient la plupart du temps le certificat de la clé publique utilisée pour signer le certificat inclus dans le champ "Originator-Certificate". Dans certains cas, ce champ contient plutôt un chemin (un "certificate path") permettant au destinataire de valider de manière récursive le certificat.

Le champ "**Recipient-ID**" identifie le destinataire du message ainsi que sa clé d'échange et sa structure est identique à celle de Originator-ID.

Pour chacun des destinataires, le champ "**Key-Info**" reprend des informations comme la clé utilisée pour chiffrer les données. Dans notre exemple où la gestion des clés est symétrique quatre arguments y apparaissent :

- DES-ECB signifie que le chiffrement de la clé DEK et du MIC est un chiffrement DES en mode Electronic Code Block;
- RSA-MD2 identifie l'algorithme utilisé pour calculer le MIC;
- 161A3F75DC82EF29 est la clé DEK chiffrée par DES-ECB;
- E2EF532CCBCFF79658CBCD245454564987 est le MIC chiffré par DES-ECB.

5. Certificat

La recommandation RFC1421 concerne la gestion des clés et s'articule autour de la notion essentielle de certificat.

5.1. Notion de certificat dans PEM

Rappelons tout d'abord que les certificats (cfr. : chapitre 3) permettent de se protéger contre la falsification des clés publiques.

Un certificat pour une clé publique est décrit au sein de PEM comme une structure signée de données comprenant le nom de l'utilisateur ("subject"), le nom de l'entité ("issuer") qui se porte garante pour certifier que la clé publique est bien liée à l'utilisateur et la clé publique utilisée. Ce certificat est accompagné d'un indicateur de temps spécifiant jusqu'à quand le certificat est valide; le tout étant signé par la clé secrète de l'"issuer".

5.2. Hiérarchisation des autorités de certification

Le système de gestion et de distribution des certificats est organisé de manière hiérarchique où chaque niveau permet de certifier l'identité du niveau inférieur.

Au sommet de la pyramide, nous trouvons l'IPRA (Internet Policy Registration Authority) dont la clé publique constitue en quelque sorte la racine à partir de laquelle toutes les opérations concernant les validation de certificats pourront avoir lieu.

Au niveau directement inférieur, les PCA (Policy Certification Authorities) gèrent généralement des organisations; elles se différencient par les techniques et les mesures de sécurité utilisées pour générer et protéger les paires "clé publique/clé secrète".

En dessous des PCA, se trouvent les CA (Certification Authorities(8)) qui servent à certifier l'identité des utilisateurs ou éventuellement de petites organisations subordonnées à de plus grandes organisations gérées par les PCA.

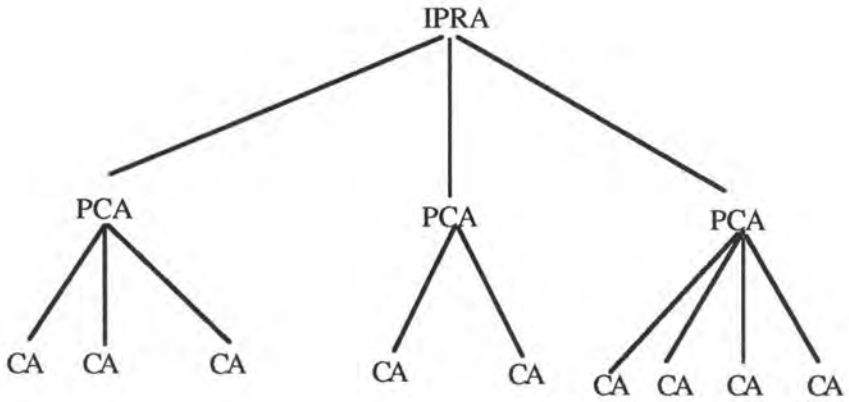


Figure 25 : Hiérarchisation des autorités de certification au sein de PEM.

Remarquons aussi qu'une même autorité de certification pourrait être certifiée par plus d'une PCA.

5.3. Utilisation des certificats

Dans PEM, les certificats sont utilisés pour d'une part, fournir à l'émetteur la clé publique du(des) destinataire(s) et d'autre part, fournir à chaque destinataire la clé publique utilisée par l'émetteur.

Avant de pouvoir effectivement envoyer un message, l'émetteur doit disposer d'un certificat pour chacun des destinataires auxquels il destine son message; ces certificats lui permettront de s'assurer de l'authenticité de l'identité des destinataires. Après les avoir obtenus auprès de l'autorité de certification compétente, ces certificats doivent être soumis à validation càd que la signature digitale apposée sur le certificat doit être vérifiée; cette vérification étant réalisée grâce à la clé publique correspondant à la clé secrète utilisée (par l'autorité de certification) pour la signature. Une fois

---

(8) La définition précise du terme "Certification Authority" est donnée dans la norme X409 : "An authority trusted by one or more users to create and assign certificates."

cette validation réalisée, la clé publique contenue dans le certificat est extraite pour pouvoir ensuite chiffrer la DEK. Cette clé de chiffrement du message est inclue sous sa forme chiffrée dans le champ "Key-Info" de l'en-tête du message.

A la réception du message, le champ "Key-Info" est déchiffré grâce à la clé secrète du destinataire; ce qui lui permet de disposer ainsi de la DEK et de l'utiliser pour déchiffrer le message.

De plus, dans le but de mettre en œuvre les services de sécurité relatifs à l'intégrité du contenu du message et à l'authentification de l'émetteur, un code d'intégrité du message (MIC) chiffré par la clé secrète de l'émetteur est inclus dans l'en-tête sous le champ "MIC-Info". A la réception du message, l'utilisateur n'a plus qu'à déchiffrer ce MIC et à en comparer la valeur avec le résultat d'un calcul qui se fait de manière locale à partir du message reçu.

5.4. Utilisation des chemins de certification

Plaçons-nous dans la situation où les deux partenaires voulant entrer en communication dépendent de PCA différents. Grâce au chemin de certification contenu dans le champ "Issuer-certificate" de l'en-tête du message envoyé (chemin dans le sens : "parcours" dans la hiérarchie des autorités de certification permettant de relier le destinataire à l'émetteur), les opérations suivantes pourront se dérouler :

- le récepteur utilise la clé publique de l'IPRA pour valider le certificat du PCA envoyé (contenu dans le champ d'en-tête "Issuer-Certificate");
- il extrait ensuite de ce certificat la clé publique du PCA, ce qui lui permet de valider à son tour le certificat du CA.
- il répète le même processus pour valider ainsi finalement le certificat de l'émetteur du message (contenu dans le champ "Originator-Certificate" de l'en-tête).

5.5. Liste de révocation de certificats (Certificate Revocation List (CRL))

La validation d'un certificat ne concerne pas uniquement l'analyse de la signature digitale et la vérification du fait que le temps de validité du certificat n'est pas dépassé mais elle comprend aussi la consultation de listes

de révocation de certificats où sont repris les certificats précédemment annulés par les autorités de certification.

Essentiellement, il peut y avoir deux raisons principales à cette révocation :

- un utilisateur change d'adresse et passe ainsi par exemple d'une organisation A à une affiliation dans une organisation B.
- suite à la découverte frauduleuse ou accidentelle d'une clé secrète par une tierce personne, toute la politique de la gestion des clés concernant la personne lésée doit être revue; on rendra par exemple invalide la clé publique correspondant à cette clé secrète.

Chapitre 6 : Cadre de travail

1. Architecture GSMHS

L'architecture qui sert de cadre de travail à ce mémoire est désignée sous le terme de GSMHS, abréviation signifiant Generalized Secure Message Handling System.

Cette architecture développée et imaginée par Suchun Wu fait actuellement l'objet d'un projet de coopération entre l'université de Namur et la Région Wallonne (cfr. : [WU,92] et [WU,94]).

Les objectifs de base poursuivis par cette étude sont :

- permettre le développement de manière modulaire de MHS sécurisés;
- faciliter l'implémentation de services de sécurité au sein de MHS;
- permettre l'interconnexion de différents MHS.

L'architecture GSMHS peut se représenter comme étant construite à partir de trois types d'interfaces :

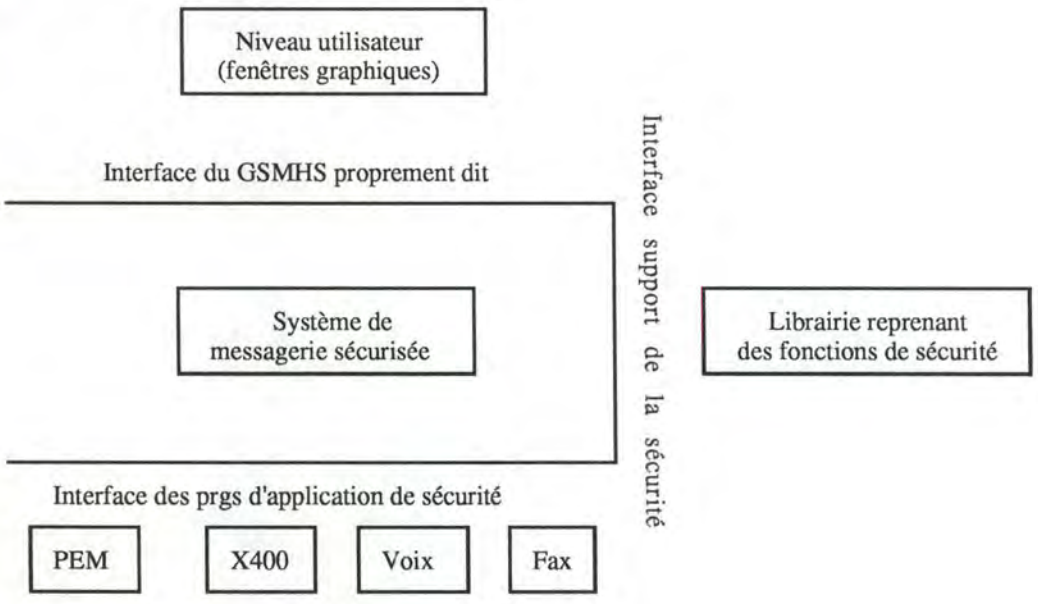


Figure 26 : Structure en interfaces de l'architecture GSMHS.

2. Interfaces du modèle GSMHS

2.1. Interface entre la messagerie sécurisée et l'utilisateur : Interface du GSMHS proprement dit

Cette interface doit permettre à l'utilisateur d'établir et de gérer directement ses communications sécurisées de messages : elle se doit donc d'être la plus conviviale; c'est ainsi qu'on peut l'imaginer sous la forme de fenêtres graphiques.

A ce niveau, il s'agit d'offrir à l'utilisateur le choix entre différents services de sécurité applicables à ses envois de messages. Dans le cas où l'utilisateur désire que ses messages soient rendus confidentiels, il lui suffirait par exemple de valider un choix présenté dans un menu déroulant.

2.2. Interface entre la messagerie sécurisée et la librairie des fonctions de sécurité : Interface support de la sécurité

Cette interface permet d'implémenter les appels aux modules des fonctions de sécurité : fonctions de génération de paires RSA (clé secrète, clé publique), de chiffrement DES de message, de signature de texte grâce à un algorithme particulier, de calcul de codes détecteurs de manipulation, ...

Pour plus de modularité dans notre travail, nous répartirons les fonctions de sécurité en deux grands modules : celui des fonctions de gestion de clés et celui des fonctions cryptographiques.

2.3. Interface entre la messagerie sécurisée et une application MHS : Interface des programmes d'application de sécurité

Par la suite, cette interface sera désignée sous le terme de Security Application Program Interface (SAPI).

Cette interface constitue la passerelle entre une messagerie non sécurisée et une messagerie "améliorée" disposant d'un ensemble de services de sécurité. Elle doit permettre aux programmeurs d'implémenter les services de sécurité correspondant aux demandes des utilisateurs.

L'interface SAPI peut être elle-même décomposée en trois modules; cette décomposition étant réalisée dans le but de rendre l'implémentation la portable et la plus facilement "extensible" :

- un module Security Feature Establishment (SFE);

- un module General Cryptographic Function (GCF);
- un module Application Oriented Security Function (ASF).

Dans le module SFE sont regroupées les routines qui servent à établir le contexte dans lequel la communication sécurisée aura lieu. C'est ainsi qu'on devra par exemple y retrouver une routine permettant à l'utilisateur se situant au niveau SAPI de choisir une application spécifique de messagerie (par exemple : choix de la messagerie électronique de type X400) et d'interagir par la suite avec l'agent utilisateur de cette messagerie.

Le module GCF reprend des routines qui sont totalement indépendantes d'un choix de technique particulière : il regroupe ce qu'on pourrait appeler des fonctions cryptographiques abstraites. Ainsi, peu importe par exemple pour l'utilisateur que son message soit chiffré par un mécanisme de clé privée ou de clé publique et seule apparaîtra dans GCF une fonction générale de chiffrement du corps du message (Encryp_mbody).

Le module ASF regroupe des routines qui sont dépendantes de l'application de messagerie choisie (on parlera de fonctions "orientées application") : il y aura ainsi un module ASF spécifique pour PEM et un module ASF pour X400.

Ce module englobe des fonctions aussi diverses que des fonctions de gestion et de structuration des données, des fonctions de gestion des certificats ou des fonctions de gestion des erreurs.

3. Précisions quant au choix du cadre de travail

Le travail réalisé s'est focalisé essentiellement au niveau de l'interface SAPI c'est-à-dire que le point de vue choisi est celui d'un programmeur qui désire mettre en œuvre un ensemble de services de sécurité.

Les services de sécurité retenus sont ceux de confidentialité et de contrôle d'intégrité du contenu du message dans le cadre d'un envoi de message à un seul utilisateur.

Plus précisément, trois options sont offertes à l'utilisateur :

- service de confidentialité du contenu du message :

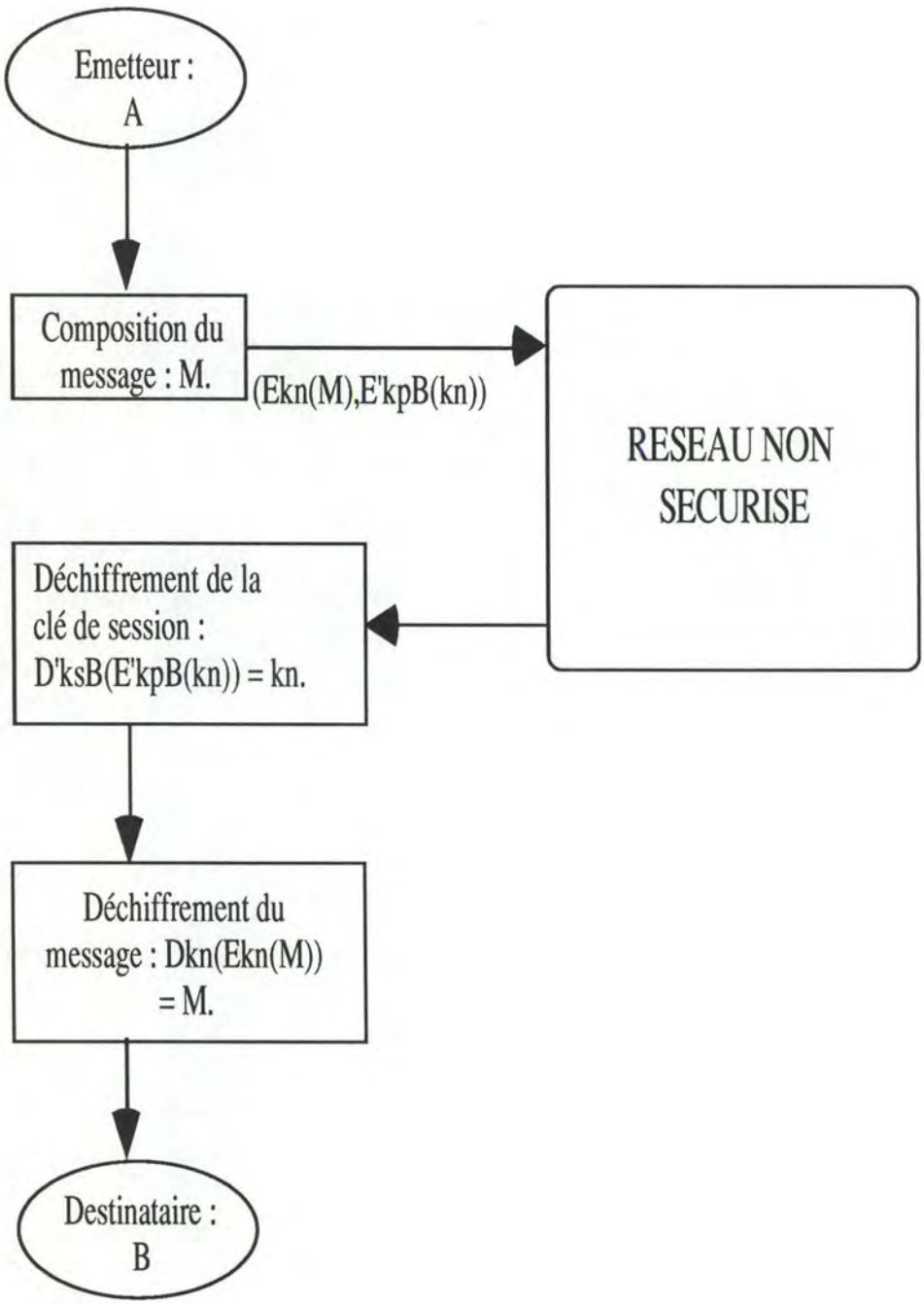


Figure 27 : Représentation schématique du service de confidentialité implémenté.

- service de contrôle d'intégrité du contenu du message :

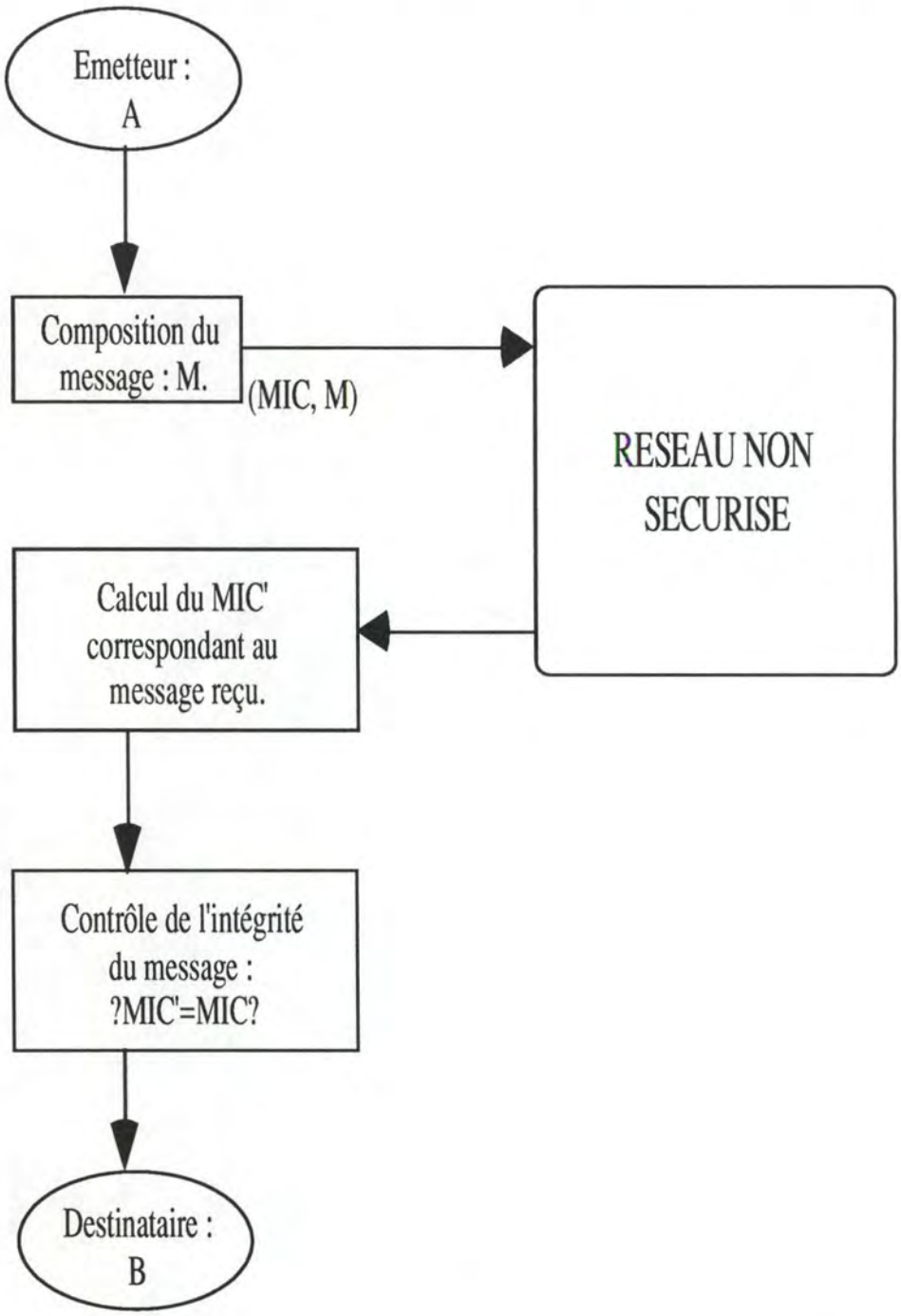


Figure 28 : Représentation schématique du service de contrôle d'intégrité implémenté.

- combinaison des services de confidentialité et de contrôle d'intégrité du contenu du message :

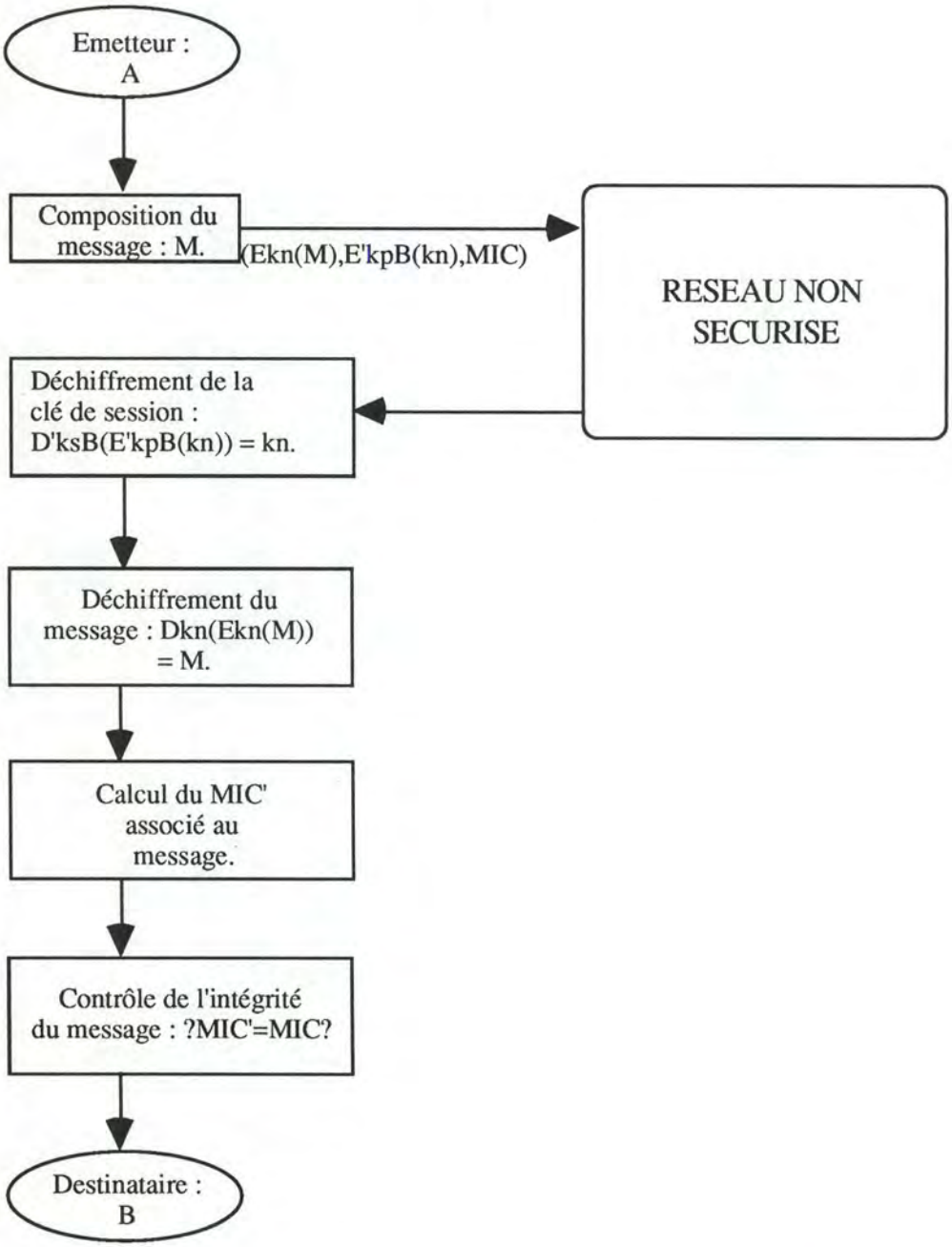


Figure 29 : Combinaison des deux services implémentés.

L'autre partie importante du travail a été de construire une librairie reprenant un ensemble suffisamment important de fonctions de sécurité non seulement pour pouvoir implémenter les services de confidentialité et de contrôle d'intégrité mais aussi pour aider à la réalisation ultérieure d'autres services.

4. Regroupement en modules des procédures implémentées

4.1. Module SAPI

4.1.1. Module SFE

Travaillant uniquement dans le cadre restreint de la version EAN 3.04 de X400/84, le contexte dans lequel la communication sécurisée aura lieu est donc par défaut EAN 3.04 et nous n'avons donc pas eu besoin de procédure à ce niveau.

4.1.2. Module ASF spécifique à EAN

Procédure : EAN_sserv_prep(char *service)

- argument : /.
- résultat : service.
- utilité : procédure de préparation des informations nécessaires au service de sécurité choisi par l'utilisateur.

Procédure : EAN_sserv_cont_send(char *service)

- argument : service.
- résultat : /.
- utilité : procédure coordinatrice permettant d'enchaîner les différentes opérations nécessaires à la mise en œuvre du service de sécurité désigné par le paramètre service lors de l'envoi d'un message.

Procédure : EAN_sserv_cont_rec(char *service)

- argument : service.
- résultat : /.
- utilité : procédure analogue à EAN_sserv_cont_send mais utilisée lors de la réception d'un message.

Procédure : Bldh_sec_msg(char *service, file *headermsg)

- argument : service.
- résultat : fichier headermsg.
- utilité : procédure de construction de l'en-tête (headermsg) comprenant les informations de sécurité associées à un message;

cette construction est uniquement fonction du service de sécurité précédemment choisi.

Procédure : Ubldh_sec_msg(file *headermsg, param_secur)

- argument : fichier headermsg.
- résultat : ensemble de paramètres de sécurité.
- utilité : procédure d'analyse de l'en-tête (headermsg) contenant les informations de sécurité relatives à un message; cette procédure permet d'obtenir des informations aussi diverses que la clé de session chiffrée, l'algorithme de chiffrement utilisé ou le MIC associé à un message.

Procédure : Inclu_sec_msg(file *headermsg, file *bodymsg, file*secmsg)

- arguments : fichiers headermsg et bodymsg.
- résultat : fichier secmsg.
- utilité : procédure de construction du message sécurisé (secmsg) à partir des informations de sécurité contenues dans le header (headermsg) et du texte proprement dit du message (bodymsg).

Procédure : Extra_mparts(file *secmsg, file *bodymsg, file *headermsg)

- argument : fichier secmsg.
- résultats : fichiers bodymsg et headermsg.
- utilité : procédure d'extraction de l'en-tête (headermsg) et du corps (bodymsg) du message à partir de l'entièreté du message sécurisé (secmsg).

4.1.3. Module GCF

Procédure : Encryp_mbody(file *bodymsg, char *id_algo_cip_msg, char *sess_key)

- arguments : fichier bodymsg, id_algo_cip_msg et sess_key.
- résultat : fichier bodymsg modifié.
- utilité : procédure de chiffrement du corps du message (bodymsg) grâce à l'algorithme de chiffrement identifié par id_algo_cip_msg et la clé de chiffrement sess_key.

Procédure : Decryp_mbody(file *bodymsg, char *id_algo_cip_msg, char *sess_key)

- arguments : fichier bodymsg, id_algo_cip_msg et sess_key.
- résultat : fichier bodymsg modifié.
- utilité : procédure de déchiffrement du corps du message (bodymsg) grâce à l'algorithme dont l'identifiant est id_algo_cip_msg et la clé sess_key.

Procédure : Encryp_sess_key(char *session_key, *kp, char *encip_sess_key, char *id_algo)

- arguments : clés session_key et kp ainsi que le paramètre id_algo.
- résultat : clé encip_sess_key.
- utilité : procédure de chiffrement par l'algorithme identifié par id_algo de la clé de session (session_key) grâce à une clé publique (kp) donnant comme résultat encip_sess_key.

Procédure : Decryp_sess_key(char *encip_sess_key, *ks, char *session_key, char *id_algo)

- arguments : clés encip_sess_key et ks ainsi que le paramètre id_algo.
- résultat : clé session_key.
- utilité : procédure de déchiffrement par l'algorithme identifié par id_algo de la clé de session chiffrée (encip_sess_key) grâce à une clé secrète (ks) donnant comme résultat session_key.

Procédure : MIC_compute(file *bodymsg, char *mic,
char *id_algo_mic)

- arguments: fichier bodymsg et id_algo_mic.
- résultat : mic.
- utilité : procédure de calcul du vérificateur d'intégrité du contenu (mic) associé au corps du message (bodymsg) grâce à l'algorithme identifié par id_algo_mic.

Procédure : long Verif_mic_compute(file *bodymsg, char *mic,
char *id_algo_mic)

- arguments : fichier bodymsg, mic et id_algo_mic.
- résultat : long Verif_mic_compute.
- utilité : procédure de vérification du MIC (mic) associé au corps (bodymsg) du message grâce à l'algorithme identifié par id_algo_mic. La procédure renvoie 1 dans le cas où la vérification du MIC s'est effectuée de manière correcte et 0 dans le cas contraire.

Procédure : Signa(file *in, file *out, char *ks, char *id_algo_sign)

- arguments : fichier in et clé ks.
- résultat : fichier out.
- utilité : procédure de signature du texte contenu dans in grâce à l'algorithme identifié par id_algo_sign et à la clé secrète ks, le résultat étant stocké dans le fichier out.

Procédure : long Verif_signa(file *in, char *kp, char *id_algo_sign)

- arguments : fichier in et clé kp.
- résultat : Verif_signa.
- utilité : procédure de vérification de la signature associée au message contenu dans in grâce à l'algorithme identifié par id_algo_sign et à la clé publique kp; Verif_signa vaut 1 dans le cas où la vérification se révèle exacte et 0 sinon.

4.2 Module reprenant un ensemble de fonctions de sécurité

4.2.1. Module de gestion des clés

Procédure : Public_key_generate(char *kp,*ks)

- argument : /.
- résultats : clés kp et ks.
- utilité : procédure de génération d'une paire de clés RSA (clé publique (kp), clé secrète (ks)).

Procédure : Pkeylist_add(file *fpklist, char *kp, char *address)

- arguments : fichier fpklist, kp et address.
- résultat : fichier fpklist modifié.
- utilité : procédure d'ajout de la clé publique (kp) associée à l'utilisateur d'adresse de courrier électronique address(9) à l'ensemble des clés publiques déjà enregistrées dans le fichier fpklist.

Procédure : Store_seckey(file *fseckey, char *ks)

- arguments : fichier fseckey et clé ks.
- résultat : nouveau fichier fseckey.
- utilité : procédure d'enregistrement dans le fichier fseckey de la clé ks.

Procédure : Pkeylist_remove(file *fpklist, char *kp, char *address)

- arguments : fichier fpklist, kp et address.
- résultat : fichier fpklist modifié.
- utilité : procédure de retrait de la clé publique (kp) associée à l'utilisateur d'adresse de courrier électronique address de l'ensemble des clés publiques contenues dans fpklist.

---

(9) Par souci de généralisation, nous avons préféré traiter les adresses sous une représentation RFC822 plutôt que sous la représentation X400 : plutôt que d'avoir dans fpklist : S=eanrec; OU=seclib; O=info; PRMD=fundp; ADMD=rtt; C=be, nous aurons eanrec@seclib.info.fundp.rtt.be.

Procédure : Pkeylist_extrac(file *fpklist, char *address, char *kp)

- arguments : fichier fpklist et id_user.
- résultat : clé kp.
- utilité : procédure de recherche dans le fichier fpklist de la clé publique (kp) associée à l'utilisateur dont l'adresse électronique est address.

Procédure : Pkey_prompt(file *fpklist, char *address)

- arguments : fichier fpklist et id_user.
- résultat : /.
- utilité : procédure d'affichage de la clé publique contenue dans fpklist et associée à l'utilisateur d'adresse électronique address.

Procédure : Skey_prompt(file *fks)

- argument : fichier fks.
- résultat : /.
- utilité : procédure d'affichage de la clé secrète contenue dans fks et associée à l'utilisateur qui en fait la demande.

Procédure : Sess_key_generate(char *sessionkey)

- argument : /.
- résultat : clé sessionkey.
- utilité : procédure de génération d'une clé de session.

Procédure : Seed_generate(long seed)

- argument : /.
- résultat : seed.
- utilité : procédure de génération d'un nombre (seed) qui servira de racine à la génération d'un nombre aléatoire.

Procédure : long Verif_pkey_recip(file *fkplist, char *address)

- arguments : fichier fkplist et address.
- résultat : long Verif_pkey_recip.
- utilité : procédure de vérification de l'enregistrement dans fkplist de l'adresse électronique (address) d'un utilisateur en

tant que possesseur d'une clé publique; la procédure renvoie 1 si le destinataire est déjà enregistré et 0 dans le cas contraire.

4.2.2. Module des fonctions cryptographiques

Procédure : Encryp_RSA_key(char *session_key, *kp,
char *encip_sess_key)

- arguments : clés session_key et kp.
- résultat : clé encip_sess_key.
- utilité : procédure de chiffrement RSA de la clé de session (session_key) grâce à une clé publique (kp) donnant comme résultat encip_sess_key.

Procédure : Decryp_RSA_key(char *encip_sess_key, *ks,
char *session_key)

- arguments : clés encip_sess_key et ks.
- résultat : clé session_key.
- utilité : procédure de déchiffrement RSA de la clé de session chiffrée (encip_sess_key) grâce à une clé secrète (ks) donnant comme résultat la clé session_key.

Procédure : Encryp_DES(char *session_key, file *in, file *out)

- arguments : fichier in et clé session_key.
- résultat : fichier out.
- utilité : procédure de chiffrement DES du contenu du fichier in grâce à la clé session_key donnant comme résultat le fichier out.

Procédure : Decryp_DES(char *session_key, file *in, file *out)

- arguments : fichier in et clé session_key.
- résultat : fichier out.
- utilité : procédure de déchiffrement DES du contenu du fichier in grâce à la clé session_key donnant comme résultat le fichier out.

Procédure : Digest(file *in, char *mic)

- argument : fichier in.
- résultat : mic.
- utilité : procédure de calcul d'une représentation compacte (mic) du message contenu dans le fichier in.

Procédure : Signa_RSA(file *in, file *out, char *ks)

- arguments : fichier in et clé ks.
- résultat : fichier out.
- utilité : procédure de signature grâce à l'algorithme RSA du texte contenu dans in grâce à la clé secrète ks et stockage du résultat dans out.

Procédure : long Verif_signa_RSA(file *in, char *kp)

- arguments : fichier in et clé kp.
- résultat : Verif_signa.
- utilité : procédure de vérification de la signature grâce à l'algorithme RSA et à la clé publique kp, Verif_signa_RSA vaut 1 dans le cas où la vérification se révèle exacte et 0 sinon.

Chapitre 7 : Analyse détaillée des procédures

1. Module SAPI

1.1. Module ASF/EAN

Procédure EAN_sserv_prep(char *service)

Cette procédure est composée de deux parties principales :

a) Verif_new_user;

b) Affich_menu.

a) Verif_new_user : vérification de l'enregistrement préalable de l'utilisateur de la messagerie en tant qu'utilisateur disposant des données nécessaires aux services de sécurité càd vérification du fait qu'une paire de clés RSA est déjà associée à l'émetteur du message.

Dans le cas où il s'agit d'un nouvel utilisateur, un message propose à l'utilisateur de générer une paire de clés RSA (ks, kp). Si l'utilisateur refuse cette opération alors il ne pourra appliquer aucun service de sécurité à ses messages (il se retrouve donc dans la situation du courrier électronique X400/EAN non amélioré); dans le cas contraire, après génération et stockage des clés RSA, l'utilisateur passe à l'étape b).

b) Affich_menu : affichage d'un menu proposant à l'utilisateur de choisir parmi un ensemble de services de sécurité qui pourront être appliqués aux envois de messages:

****

* SECURE EAN *

****

Make your choose between :

1) Content confidentiality

2) Content integrity
3) Content confidentiality and integrity
4) No security service

C'est le caractère identifiant le service de sécurité entré par l'utilisateur qui est renvoyé comme résultat par la procédure EAN_sserv_prep.

Procédure EAN_sserv_cont_send(char *service)

Cette procédure coordonne la suite des opérations à réaliser lors de l'émission du message sécurisé :

Si service = "1" alors Verif_pkey_recip
Si verif OK alors Pkeylist_extrac
Sess_key_generate
Encryp_sess_key
Bldh_sec_msg
Encryp_mbody
Inclu_sec_msg
Sinon un message avertit l'utilisateur que le destinataire du message n'est pas enregistré en tant qu' *utilisateur sécurisé* (10) et qu'il devra se contenter d'un envoi non sécurisé.

Si service ="2" alors MIC_compute
Bldh_sec_msg
Inclu_sec_msg

Si service = "3" alors Mic_compute
Verif_pkey_recip
Si verif OK alors Pkeylist_extrac
Sess_key_generate
Encryp_sess_key

(10) Par utilisateur sécurisé, nous entendons utilisateur dont l'adresse de courrier électronique est associée avec une paire de clés RSA.

Bldh_sec_msg
Encryp_mbody

Sinon un message avertit l'utilisateur que le destinataire du message n'est pas enregistré en tant qu' *utilisateur sécurisé* et qu'il devra se contenter d'un envoi garantissant seulement l'intégrité du contenu du message.

Notons que quel que soit le service de sécurité choisi, le procédé de construction du message est toujours le même :

1) Construction de l'en-tête contenant les informations de sécurité.
2) Construction du corps du message.
3) Construction du message sécurisé à partir de l'en-tête et du corps du message.

Procédure Bldh_sec_msg(char *service,file *headermsg)

Cette procédure remplit les différents champs de sécurité qui doivent être présents dans l'en-tête en fonction du service de sécurité choisi.

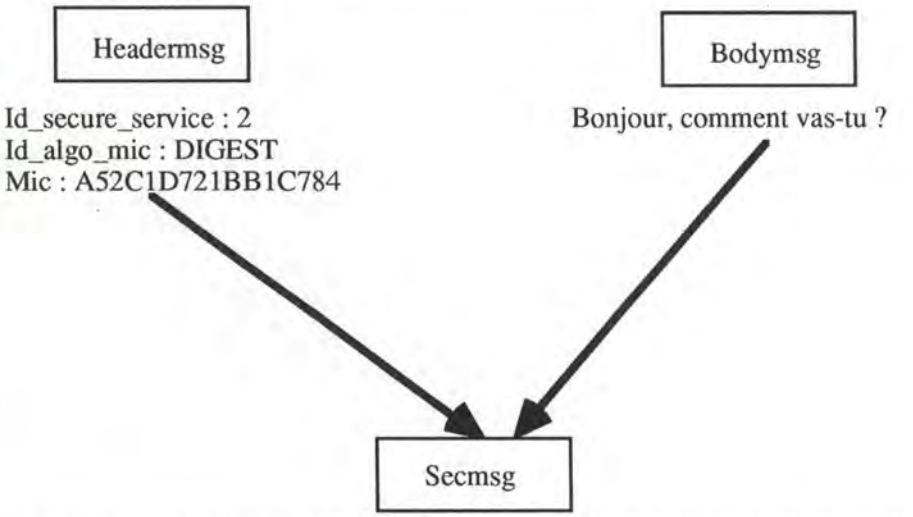
```
Si service = "1" alors Id_secure_service <-"1"
                       Id_algo_cip_key <-"RSA"
                       Id_algo_cip_msg <-"DES"
                       Session_key <- Encryp_sess_key
Si service = "2" alors Id_secure_service <-"2"
                       Id_algo_mic <-"DIGEST"
                       Mic <- MIC_compute
Si service = "3" alors Id_secure_service <-"3"
                       Id_algo_cip_key <-"RSA"
                       Id_algo_cip_msg <-"DES"
                       Id_algo_mic <-"DIGEST"
                       Session_key <- Encryp_sess_key
                       Mic <- MIC_compute
```

Les différents champs disponibles sont :

- Id_secure_service est l'identifiant du service de sécurité appliqué au message.
- Id_algo_cip_key est l'identifiant de l'algorithme de chiffrement de la clé de session.
- Id_algo_cip_msg est l'identifiant de l'algorithme de chiffrement du texte du message.
- Id_algo_mic est l'identifiant de l'algorithme de calcul du MIC associé au message.
- Session_key est la clé de session chiffrée.
- Mic est le MIC associé au message.

Procédure Inclu_sec_msg(file *headermsg, file *bodymsg, file *secmsg)

Cette procédure ne se contente pas de concaténer le corps du message à son en-tête pour construire le message sécurisé mais elle agit de la manière suivante :



**********BEGIN OF SECURE ZONE**********
Id_secure_service : 2
Id_algo_mic : DIGEST
Mic : A52C1D721BB1C784
**********BEGIN OF MESSAGE**********
Bonjour, comment vas-tu ?

**********END OF MESSAGE**********
**********END OF SECURE ZONE**********

Figure 30 : Construction du message sécurisé à partir de son corps et de son en-tête.

Cette procédure imite donc en quelque sorte la syntaxe utilisée dans PEM pour représenter un message mis sous forme sécurisée.

Procédure EAN_sserv_cont_rec(char *service)

Cette procédure coordonne la suite des opérations à réaliser lors de la réception d'un message sécurisé :

Lecture de la première ligne du message afin de déterminer si oui ou non il s'agit d'un message sécurisé:

Si la première ligne est identique à :
**********BEGIN OF SECURE ZONE**********

Alors (cas du message sécurisé) :
Extra_mparts
Ubldh_sec_msg
Si service = "1" alors Decryp_sess_key
Decryp_mbody

Si service = "2" alors Verif_mic_compute

Si service = "3" alors Decryp_sess_key
Decryp_mbody
Verif_mic_compute

Sinon (cas du message non sécurisé) : laisser la main au logiciel EAN non sécurisé.

Procédure Ubldh_sec_msg(file *headermsg, param_secur)

Cette procédure utilisée du côté du destinataire du message sécurisé permet de passer en revue et de retrouver l'ensemble des champs de sécurité stockés dans le fichier headermsg.
Le premier champ rencontré (correspondant à la première ligne du fichier headermsg) est de la forme : Id_secure_service : n où n indique le service de sécurité appliqué au message envoyé.

En fonction de l'identifiant du service de sécurité rencontré, il est très facile d'entrer en possession de l'ensemble des paramètres de sécurité.

Dans la cas où n=1, la lecture ligne par ligne du fichier headermsg donnera successivement les champs :

- Id_algo_cip_key;
- Id_algo_cip_msg;
- Session_key.

Dans le cas où n=2, nous aurons successivement :

- Id_algo_mic;
- Mic.

Dans le cas où n=3, les différents champs seront :

- Id_algo_cip_key;
- Id_algo_cip_msg;
- Id_algo_mic;
- Session_key;
- Mic.

Procédure Extra_mparts(file *secmsg, file *bodymsg, file *headermsg)

Cette procédure agit de manière inverse à la procédure Inclu_sec_msg : à partir du message sous sa forme sécurisée, elle en extrait l'en-tête et le corps.

**********BEGIN OF SECURE ZONE**********
Id_secure_service : 3
Id_algo_cip_key : RSA
Id_algo_cip_msg : DES
Id_algo_mic : DIGEST
Session_key : Aup1\%zs+IL?hg!
Mic : A52C1D721BB1C784
**********BEGIN OF MESSAGE**********
1M%*ölK47£%ki '(IJUnb&i
**********END OF MESSAGE**********
**********END OF SECURE ZONE**********

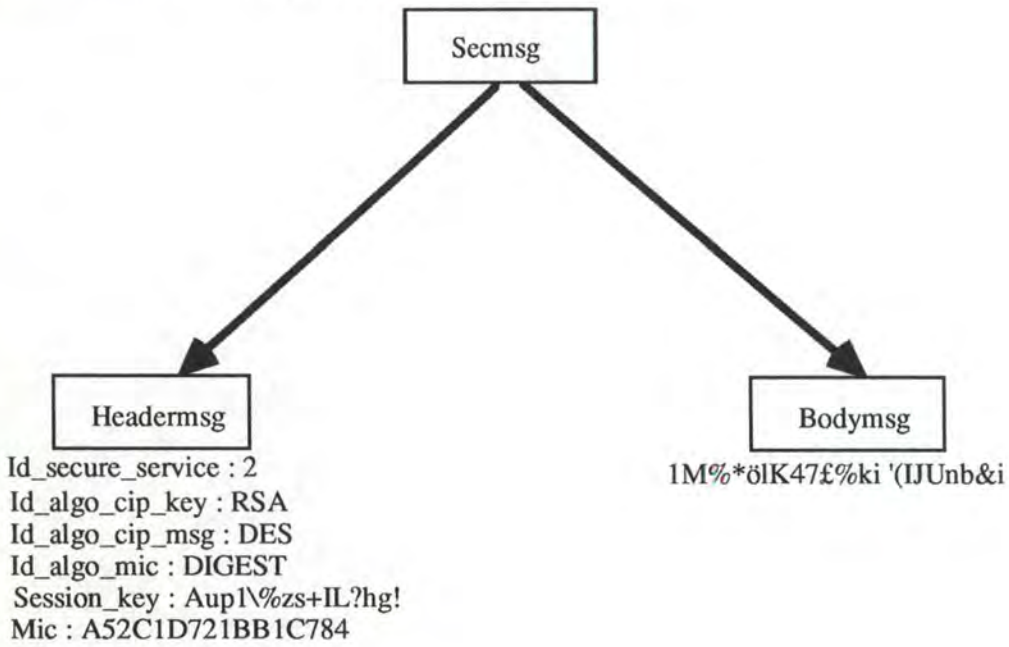


Figure 31 : Extraction du corps et de l'en-tête d'un message sécurisé.

1.2. Module GCF

```
Procédure Encryp_mbody(file *bodymsg, char *Id_algo_
                       cip_msg, char *sess_key)
```

En fonction de l'identifiant de l'algorithme de chiffrement Id_algo_cip_msg utilisé, Encryp_mbody appelle la procédure correspondante dans le module des fonctions cryptographiques. Ainsi, si Id_algo_cip_msg = "DES" alors la procédure Encryp_DES est appelée.

```
Procédure Decryp_mbody(file *bodymsg, char *Id_algo
                       cip_msg, char *sess_key)
```

En fonction de l'identifiant de l'algorithme de chiffrement Id_algo_cip_msg utilisé, Decryp_mbody appelle la procédure correspondante dans le module des fonctions cryptographiques. Ainsi, si Id_algo_cip_msg = "DES" alors la procédure Decryp_DES est appelée.

```
Procédure Encryp_sess_key(char *session_key, *kp,
                          char *encip_sess_key, char* id_algo)
```

En fonction de l'identifiant de l'algorithme de chiffrement id_algo utilisé, Encryp_sess_key appelle la procédure correspondante dans le module des fonctions cryptographiques. Ainsi, si id_algo = "RSA" alors la procédure Encryp_RSA_skey est appelée.

```
Procédure Decryp_sess_key(char *encip_sess_key,*ks,
                          char *session_key, char *id_algo)
```

En fonction de l'identifiant de l'algorithme de chiffrement id_algo utilisé, Decryp_sess_key appelle la procédure correspondante dans le module des fonctions cryptographiques. Ainsi, si id_algo = "RSA" alors la procédure Decryp_RSA_skey est appelée.

```
Procédure Signa(file *in, file *out, char *ks,
                char *id_algo_sign)
```

En fonction de l'identifiant de l'algorithme de signature id_algo_sign utilisé, Signa appelle la procédure correspondante dans le module des fonctions cryptographiques.
Ainsi, si id_algo_sign = "RSA" alors la procédure Signa_RSA est appelée.

```
Procédure long Verif_signa(file *in, char *kp,
                           char *id_algo_sign)
```

En fonction de l'identifiant de l'algorithme de signature id_algo_sign utilisé, Verif_signa appelle la procédure correspondante dans le module des fonctions cryptographiques.
Ainsi, si id_algo_sign = "RSA" alors appel sera fait à la procédure Verif_signa_RSA.

```
Procédure MIC_compute(file *bodymsg, mic,
                      char *Id_algo_mic)
```

En fonction de l'identifiant de l'algorithme de calcul du vérificateur d'intégrité du contenu Id_algo_mic utilisé, MIC_compute appelle la procédure correspondante dans le module des fonctions cryptographiques.
Ainsi, si Id_algo_mic = "DIGEST" alors la procédure Digest est appelée.

```
Procédure long Verif_mic_compute(file *bodymsg,
                                 char *mic, char *Id_algo_mic)
```

En fonction de l'identifiant de l'algorithme de calcul du vérificateur d'intégrité du contenu Id_algo_mic utilisé, MIC_compute appelle la procédure correspondante dans le module des fonctions cryptographiques.
Ainsi, si Id_algo_mic = "DIGEST" alors l'enchaînement des opérations sera le suivant :

Mic_bis <- Digest.

Comparaison de mic et de Mic_bis.
Si (mic=Mic_bis) alors la procédure renvoie 1.
Sinon elle renvoie 0.

2. Module des fonctions de sécurité

2.1. Module de gestion des clés

Procédure Public_key_generate(char *kp,*ks)

Pour générer une paire de clés RSA, la procédure Public_key_generate agit en respectant les étapes suivantes :

Seed_generate
Génération aléatoire d'un nombre premier p d'environ 30 chiffres.
Génération aléatoire d'un second nombre premier q (différent de p) et d'environ 30 chiffres.
Calcul du produit r : r=p*q.
Calcul du produit intermédiaire n : n=(p-1)*(q-1).
Génération de la clé secrète ks[11] (clé d'environ 60 chiffres) de telle manière que le plus grand commun diviseur de ks et de n soit 1.
La clé publique kp est alors obtenue par la relation : kp*ks ≡ 1 mod n.

Procédure Pkeylist_add(file *fpklist, char *kp, char *address)

A chaque clé kp stockée dans le fichier désigné par fpklist correspondent trois lignes du fichier :

- la première ligne reprend le paramètre address càd l'adresse électronique du possesseur de la clé;

---

(11) ks seule ne constitue pas une clé secrète : c'est le couple (ks, n) qui représente véritablement cette clé secrète (cette même remarque concerne aussi la clé publique (kp, n)). Néanmoins, pour plus de facilité dans les notations : ks désignera la clé secrète RSA et kp représentera la clé publique RSA.

- les deux dernières lignes correspondent à la clé publique (kp sur la deuxième ligne et n sur la troisième).

Ce fichier fpklist reprenant les clés publiques est stocké de manière totalement non sécurisée : ce qui permet aux différents utilisateurs de prendre facilement possession de clés. Mais cette solution est loin d'être totalement satisfaisante et devrait être à l'avenir complétée par un système de gestion de certificats de clés publiques.

Procédure Store_seckey(file *fseckey, char *ks)

Contrairement au cas de la clé publique où le fichier de stockage ne dispose d'aucun mécanisme de sécurité, la clé secrète demande plus de protection : la solution choisie a été de stocker la clé ks dans un fichier uniquement accessible par le possesseur de la clé.

Procédure Pkeylist_remove(file *fpklist, char *kp,
char *address)

Cette procédure agit de manière inverse à Pkeylist_add en permettant de retirer du fichier désigné par fpklist la clé kp. Elle ne peut être utilisée que pour effacer sa propre clé (càd la clé correspondant à son adresse électronique); cette procédure doit aussi prévoir l'effacement simultané de la clé secrète correspondante.

Pour être implémentée de manière plus correcte et plus raffinée, cette procédure devrait aussi gérer les situations où le couple (ks, kp) est par exemple effacé après un envoi d'un message utilisant une de ces clés : comment alors le destinataire pourra-t-il entrer en possession du message ?

Procédure Pkeylist_extrac(file *fpklist, char *address,
char *kp)

Cette procédure passe en revue les différentes lignes du fichier désigné par fpklist jusqu'au moment où elle rencontre l'adresse

électronique address : les deux lignes suivantes du fichier constituent alors la clé publique à renvoyer.

Procédure Pkey_prompt(file *fpklist, char *address)

Un utilisateur pourrait trouver intéressant de consulter la valeur d'une clé publique associée à une adresse de courrier électronique address : c'est ce que permet cette procédure.

Procédure Skey_prompt(file *fks)

Un utilisateur pourrait trouver intéressant de consulter la valeur de la clé secrète qui est associée à sa propre adresse électronique : c'est ce que permet cette procédure.

Procédure Sess_key_generate(char *sessionkey)

La génération de la clé de session comprend deux étapes essentielles : Seed_generate.

Génération d'une suite de 16 caractères dont le code ASCII est compris entre 33 ("!") et 126 ("~"); cette suite constituant la clé de session.

Procédure Seed_generate(long seed)

Cette procédure fournit un nombre qui peut utilisé par la suite comme racine d'un générateur de nombre aléatoire. Ce nombre est fonction du temps GMT auquel l'appel à Seed_generate est effectué : ainsi si l'appel est effectué à 10h 24' 56" le nombre seed sera 102456.

La principale utilité de cette procédure est donc d'éviter d'initialiser à chaque fois le générateur aléatoire avec la même racine et donc d'obtenir à chaque fois le même nombre aléatoire. Ceci serait catastrophique pour la génération de clés de session qui fournirait à chaque fois la même clé !

```
Procédure long Verif_pkey_recip(file *fkplist,
                                char *address)
```

Cette procédure de vérification de l'enregistrement de l'utilisateur identifié par l'adresse électronique address en tant qu'utilisateur sécurisé est réalisée grâce à une simple consultation du fichier désigné par fkplist. Dans le cas où une ligne de ce fichier est identique au paramètre address, la procédure renvoie 1 et dans le cas contraire, la procédure renverra 0.

2.2. Module des fonctions cryptographiques

```
Procédure Encryp_RSA_key(char *session_key, *kp,
                         char *encip_sess_key)
```

La relation qui permet de passer de la clé de session (session_key) à la clé de session chiffrée par l'algorithme RSA (encip_sess_key) est : encip_sess_key = zexpmod(session_key, kp) où zexpmod représente l'exponentiation modulaire(12) .

```
Procédure Decryp_RSA_key(char *encip_sess_key, *ks,
                         char *session_key, char *id_algo)
```

De manière analogue à la procédure Encryp_RSA_key, Decryp_sess_key déchiffre la clé de session en effectuant l'opération : session_key = zexpmod(encip_sess_key, ks).

```
Procédure Encryp_DES(char *session_key, file *in,
                     file *out)
```

Cette procédure fait appel à un ensemble de fonctions spécialement implémentées dans le cadre du chiffrement standard DES.

---

(12) La librairie du domaine public verylong permet de traiter assez facilement ce genre d'opération sur des nombres de plusieurs dizaines de chiffres.

Remarquons une nouvelle fois que le chiffrement DES permet d'atteindre un excellent degré de sécurité tout en ne requérant pas d'importants temps de calcul.

Procédure Decryp_DES(char *session_key, file *in, file *out)

De manière totalement symétrique à Encryp_DES, Decryp_DES permet de déchiffrer le message grâce à la clé de session session_key.

Procédure Digest(file *in, char *mic)

La procédure Digest utilise l'algorithme appartenant au domaine public "MD5 message-digest" (cfr. : [RIV,92]).
MD5 traite en entrée d'un message de longueur arbitraire et produit en sortie un condensé (un *digest*) de 128 bits de ce message. Ce condensé est tel que la probabilité que deux messages donnent le même résultat est de 2^{-64}. De plus, la probabilité de trouver un message produisant un condensé spécifié à l'avance est de 2^{-128}. Cet algorithme mérite donc bien sa place parmi les applications concernant le domaine de la sécurité.
Habituellement, MD5 est utilisé pour des raisons de performances : il permet de comprimer un message de taille relativement importante qui doit être chiffré avec l'algorithme RSA.

Procédure Signa_RSA(file *in, file *out, char *ks)

Cette procédure de calcul de la signature associée au message stocké dans le fichier désigné par in agit vérifie la relation : contenu de out = zexpmod(contenu de in, ks).
Vu la relative lenteur de l'algorithme RSA, il serait préférable de calculer la signature non pas à partir du contenu du message mais à partir de la seule adresse de courrier électronique de l'émetteur (choix qui facilitera l'implémentation de Verif_signa_RSA).

Procédure long Verif_signa_RSA(file *in, char *kp)

La vérification de la signature apposée sur le message procède de la manière suivante :

Resul <- zexpmod(contenu de in, kp).

Si Resul = champ de l'en-tête du message correspondant à l'adresse de l'émetteur du message alors Verif_signa_RSA renvoie 1.

Sinon Verif_signa_RSA renvoie 0.

3. Enchaînement des procédures

Pour clarifier davantage les idées, nous reprenons sur les figures suivantes l'enchaînement des procédures qui permettent d'envoyer et de réceptionner un message mis sous forme sécurisée.

3.1. Envoi d'un message dont l'intégrité du contenu est vérifiée

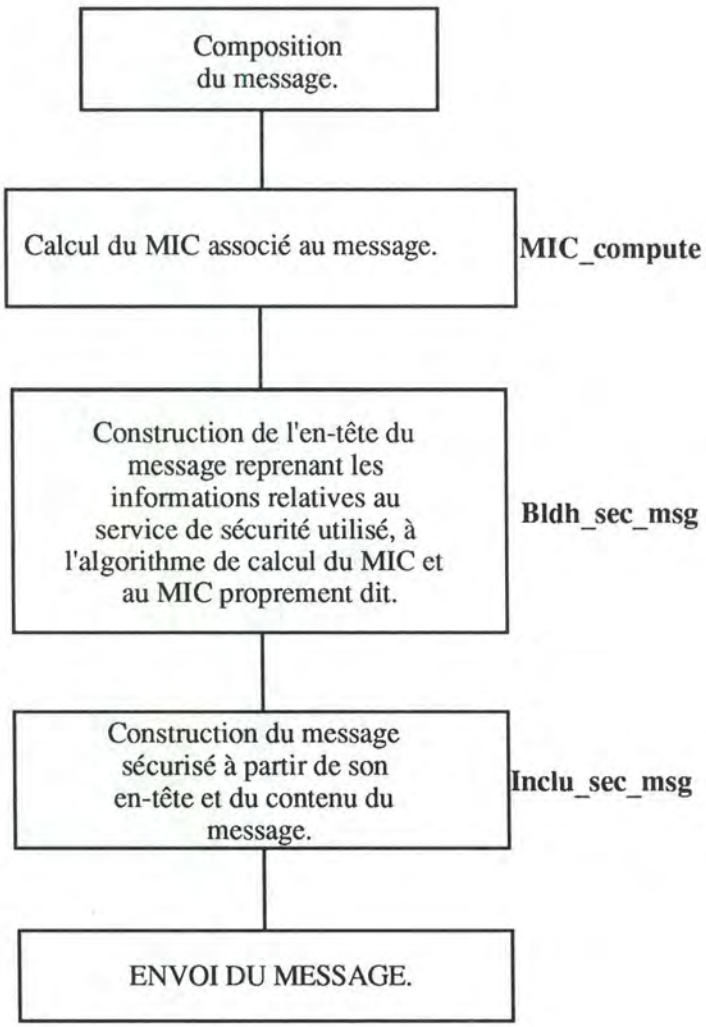


Figure 32 : Enchaînement des procédures lors de l'envoi d'un message soumis au service de contrôle d'intégrité du contenu.

3.2. Envoi d'un message dont le contenu est rendu confidentiel

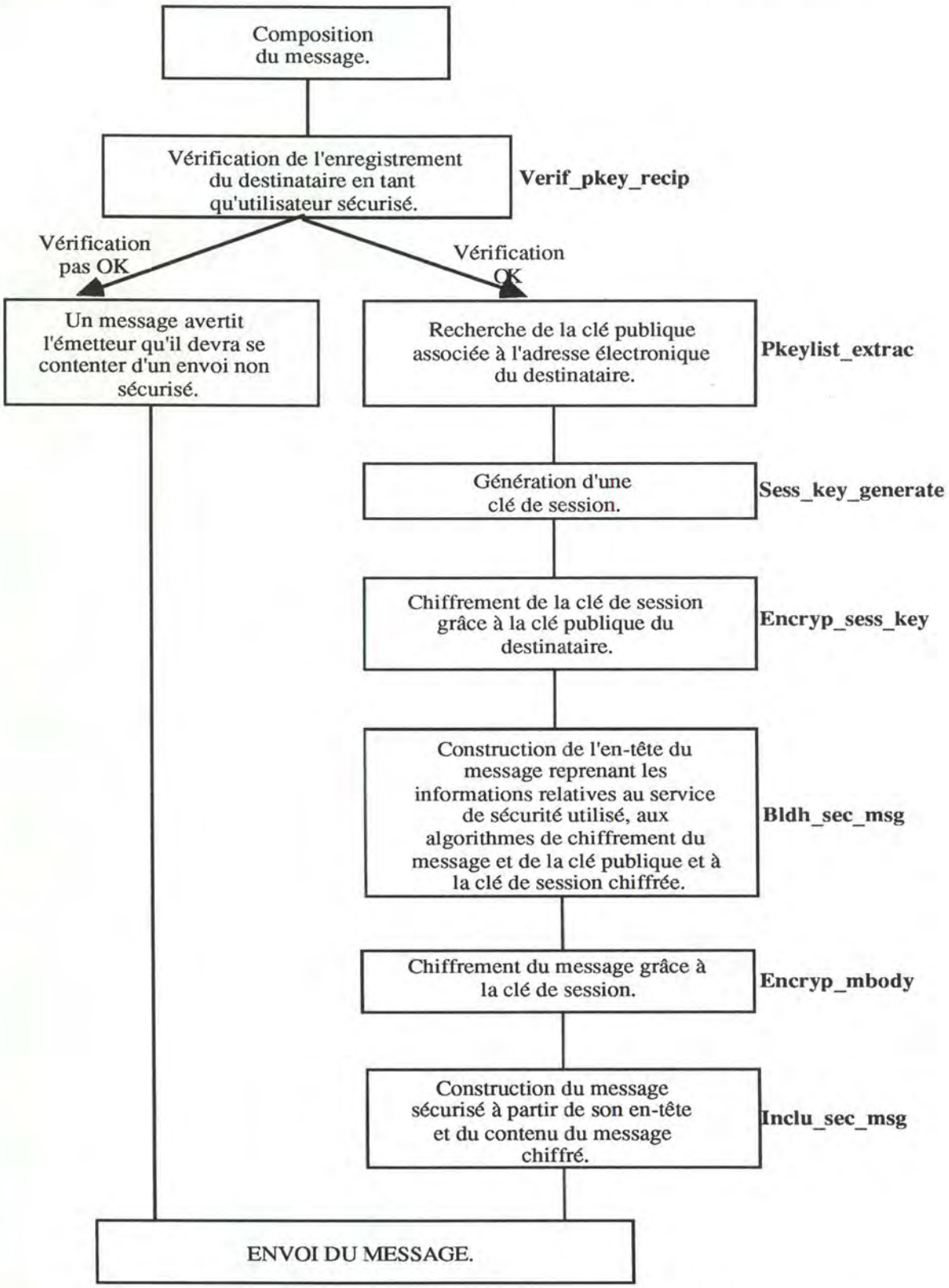


Figure 33 : Enchaînement des procédures lors de l'envoi d'un message confidentiel.

3.3. Envoi d'un message soumis aux services de confidentialité et de contrôle d'intégrité du contenu

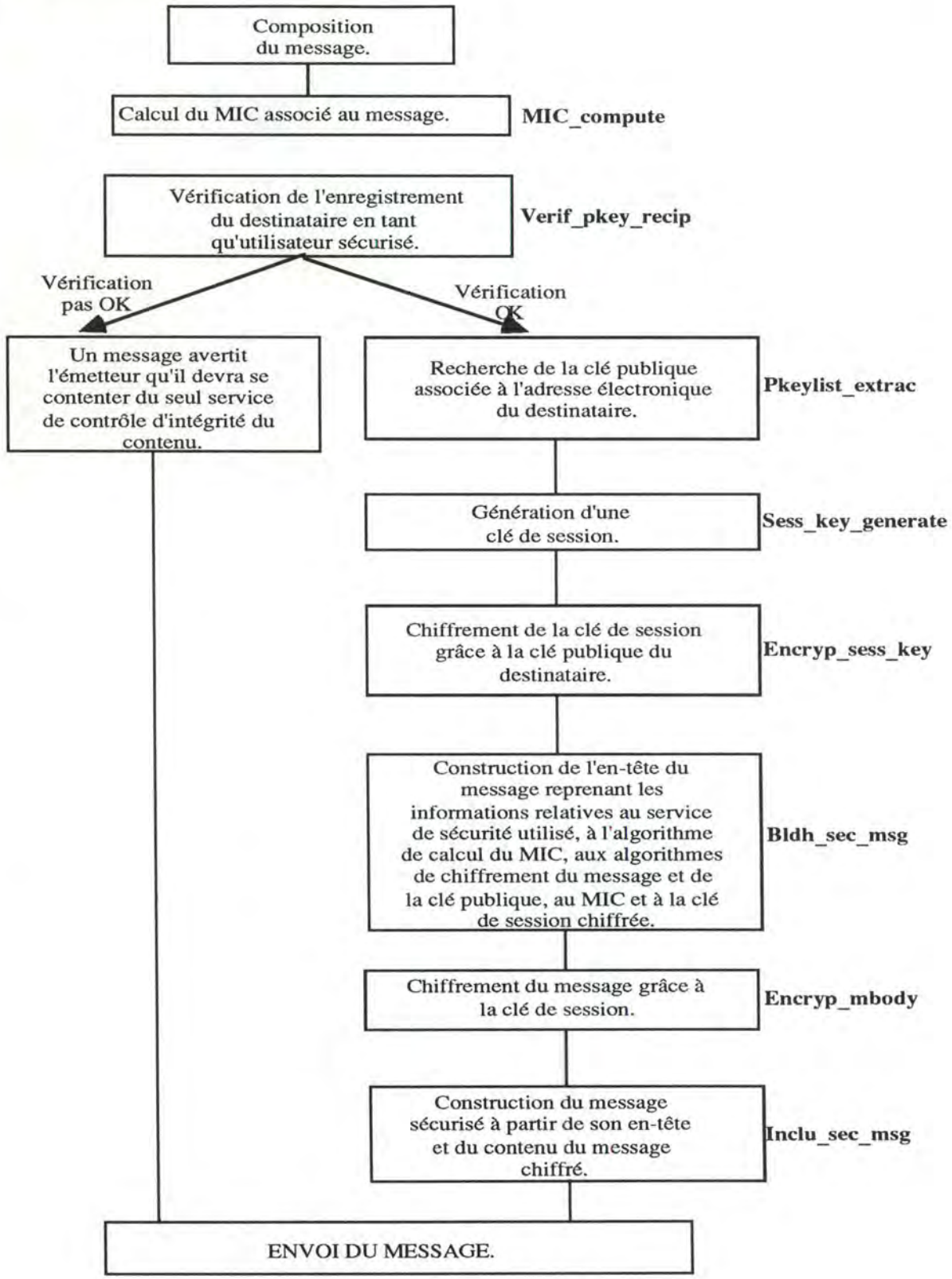


Figure 34 : Enchaînement des procédures lors de l'envoi d'un message soumis aux services de confidentialité et de contrôle d'intégrité du contenu.

3.4. Réception d'un message soumis au service de contrôle d'intégrité du contenu

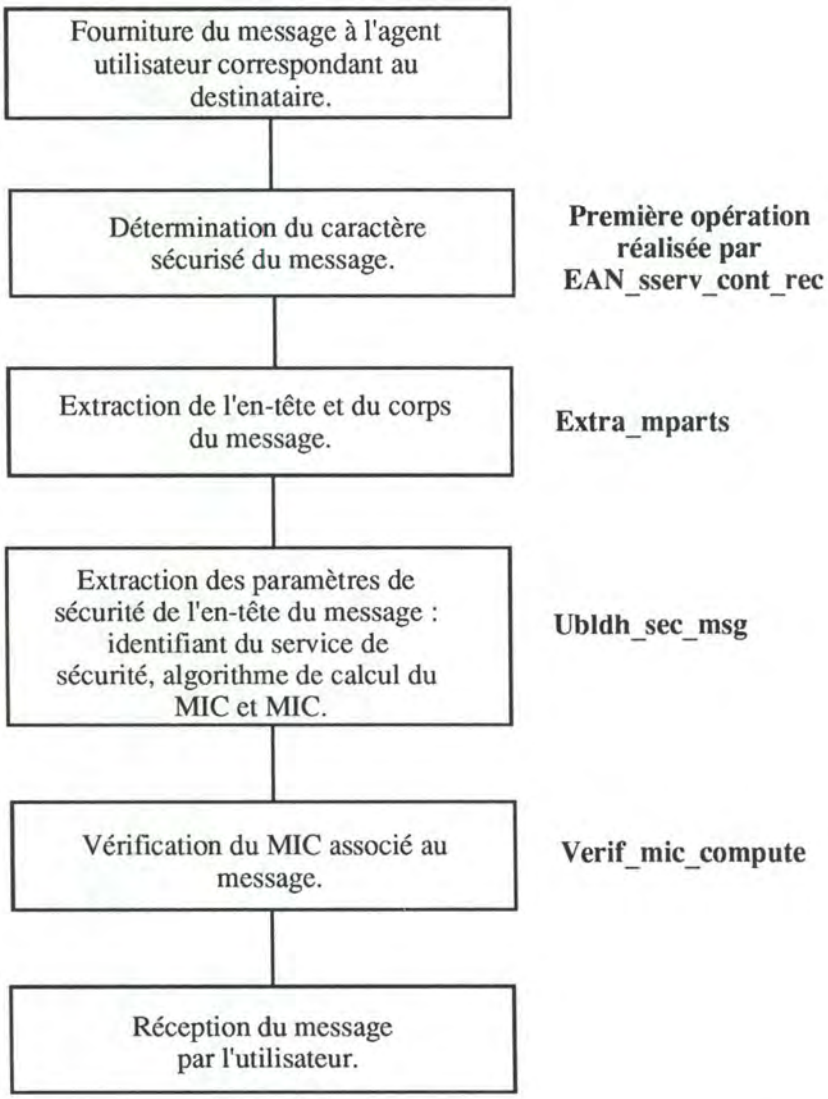


Figure 35 : Enchaînement des procédures lors de la réception d'un message dont l'intégrité du contenu est vérifiée.

3.5. Réception d'un message confidentiel

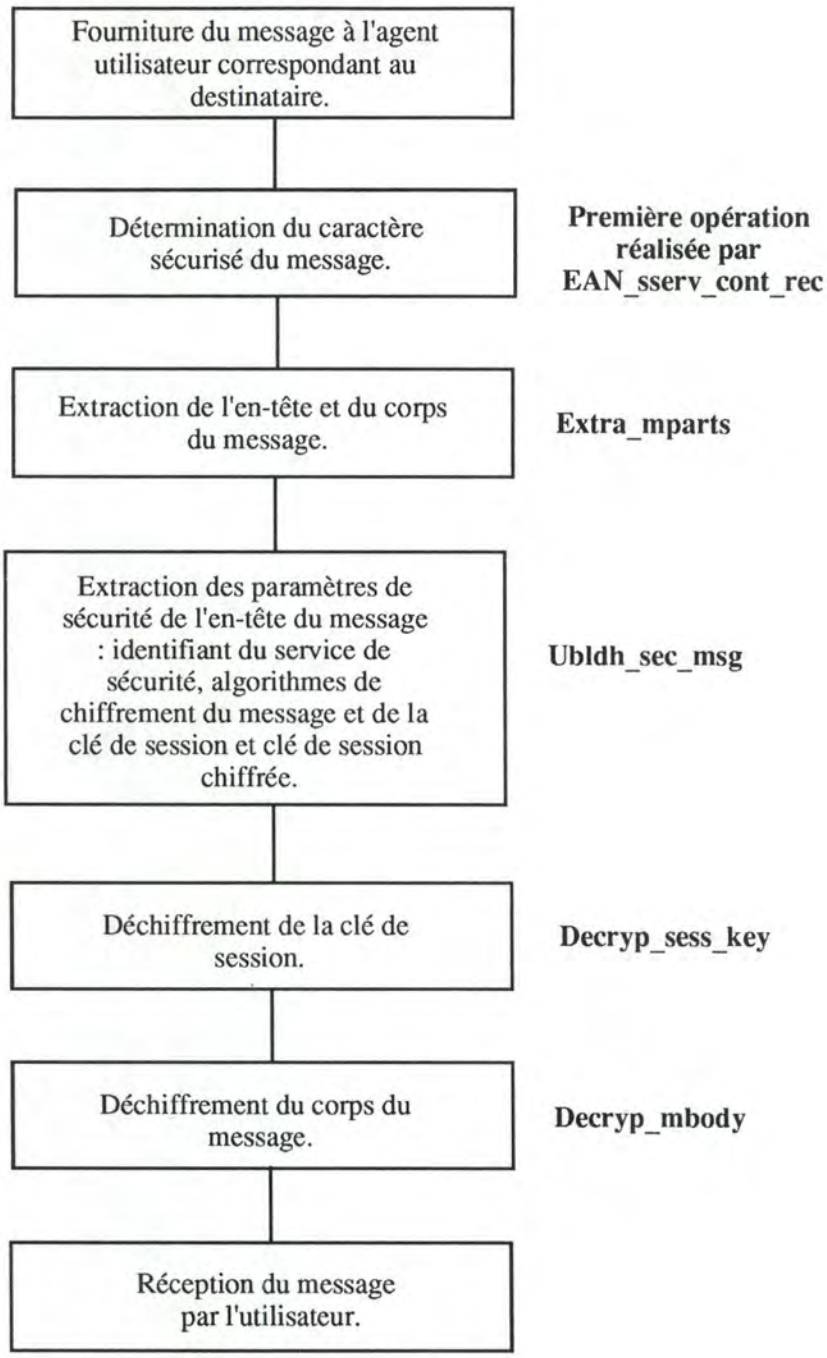


Figure 36 : Enchaînement des procédures lors de la réception d'un message confidentiel.

3.6. Réception d'un message soumis aux services de confidentialité et de contrôle d'intégrité du contenu

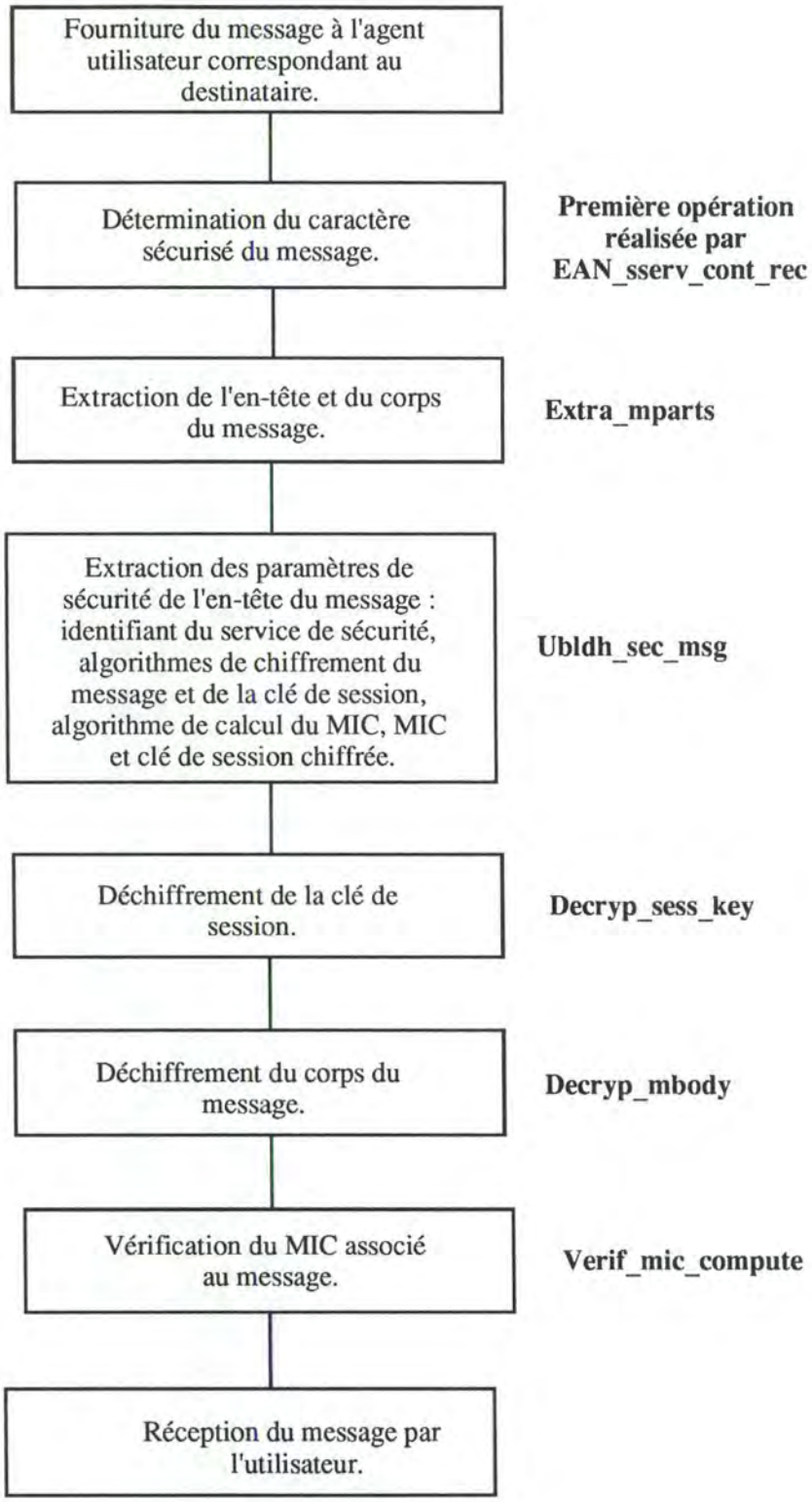


Figure 37 : Enchaînement des procédures lors de la réception d'un message confidentiel et dont l'intégrité est contrôlée.

Chapitre 8 : Logiciel EAN

La difficulté majeure du mémoire a été de tenir compte des particularités du logiciel EAN (cfr. : [EAN,87], [EAN,92] et [RUT,89]) pour pouvoir y insérer de manière optimale les services de sécurité : d'une part, nous avons dû nous familiariser avec de nouvelles structures de données et d'autre part, nous avons dû nous intéresser aux différentes étapes par lesquelles transite un message : depuis son émission jusqu'à sa soumission ainsi que depuis sa fourniture jusqu'à sa réception.

1. Structures de données

1.1. Norme X409

La norme X409 (cfr. : [X409,84]) définit la syntaxe de transfert que doit respecter l'information échangée entre deux entités de la couche application dans un MHS. Selon cette norme, les données sont représentées par trois éléments : le contenu proprement dit de l'information transférée, la longueur de ce contenu et l'identificateur du type de données.

Cet identificateur est codé sur 8 bits :

Les bits 8 et 7 spécifient la classe :

bit 8	bit 7	classe	commentaires
0	0	universal	type indépendant de l'application (ex : un booléen)
0	1	application-wide	type spécifique à une application particulière
1	0	context-specific	type utilisé dans un contexte restreint
1	1	private-use	type réservé pour un typage privé

Le bit 6 spécifie la forme :

bit 6	forme	commentaires
0	primitive	élément dont le contenu est indécomposable
1	constructor	élément dont le contenu est lui-même un élément ou une suite d'éléments (ex : une séquence)

Les 5 premiers bits constituent le code ID : c'est ce code qui permet de distinguer un type de données d'un autre à l'intérieur d'un même classe. Pour un code ID compris entre 0 et 30, ces 5 bits suffisent; sinon les 5 bits sont mis à 1 et le code ID est contenu dans un(des) octet(s) d'extension.

C'est ainsi que l'identificateur d'un type de classe universal, de forme primitive et de code ID64 sera représenté par :

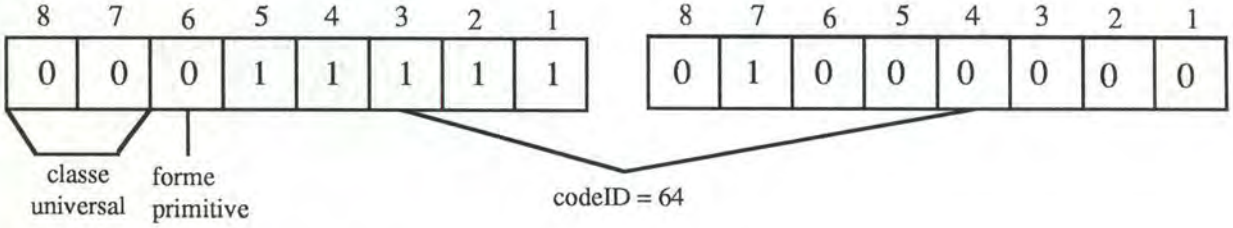


Figure 38 : Exemple de codage de donnée.

1.2. Structure d'ENODE

La structure de données primordiale dans EAN est celle de l'ENODE; à partir d'elle, il est possible de représenter n'importe quelle donnée X409.

Formellement, cette structure peut être définie par :

```
struct ENODE
        { eid               id
          elen              length
          byte              *primitive
          struct ENODE      *constructor
          struct ENODE      *next
        }
```

Dans cette structure d'ENODE :

- le champ id correspond à la partie identificateur de la norme X409:
- le champ length correspond à la partie longueur du contenu en X409;
- lorsque la donnée est de forme primitive, son adresse est placée dans le champ primitive et les champs constructor et next sont mis à NULL;
- lorsque la donnée est de forme constructor, chacun des éléments qui la compose est représenté par un ENODE; ces ENODE sont chaînés entre eux par le champ next, le champ constructor pointant vers le premier des éléments.

1.3. Structure de MESSAGE

Dans le logiciel EAN est définie une structure spéciale MESSAGE :

```
struct MESSAGE
        { int       status;
          ENODE*    content;
          ENODE*    envelope;
          ENODE*    reports;
          ENODE*    infotypes;
          int       has_sig;
        }
```

Dans la structure MESSAGE :

- le champ status indique l'état du message : c'est ce champ qui indiquera par exemple si un message a ou non déjà été envoyé ou s'il s'agit d'un message qui vient juste d'être composé;
- le champ content pointe vers le contenu du message;
- le champ envelope pointe vers l'enveloppe du message;
- le champ reports pointe vers une séquence d'informations comme par exemple la date et l'heure à laquelle le message est réceptionné ou la cause de la non fourniture d'un message;
- le champ infotypes désigne le type de contenu du message : texte, code binaire, voix, graphiques, ...;
- le champ has_sig est mis à 1 (TRUE) dès que la structure MESSAGE associée au texte du message composé par un

utilisateur est effectivement enregistrée dans ce qu'on appelle un folder.

1.4. Procédures

Pour pouvoir manipuler assez facilement ces structures particulières de données, il existe un ensemble de fonctions très pratiques regroupées en librairies; quelques-unes des plus utiles sont :

- ENODE* xe_tag : fonction permettant de construire une donnée de forme constructor;
- ENODE* xe_boolb : fonction permettant de construire un ENODE correspondant à une donnée de type booléen;
- ENODE* xe_intb : fonction permettant de construire un ENODE correspondant à une donnée de type entier;
- ENODE* xe_strb : fonction permettant de construire un ENODE correspondant à une donnée de type string;
- ENODE* xe_qapp : fonction permettant d'ajouter un ENODE à une donnée de forme constructor;
- char* xe_stru : fonction permettant d'extraire une donnée de type string d'un ENODE donné;
- long xe_intu : fonction permettant d'extraire une donnée de type entier d'un ENODE donné;
- int xe_cmpprim : fonction permettant de comparer deux données de forme primitive;
- int xe_cmplist : fonction permettant de comparer deux données de forme constructor;

...

2. Cheminement d'un message sécurisé dans EAN

2.1. Du côté de l'émetteur du message

Du côté de l'émetteur du message, deux problèmes sont à résoudre :

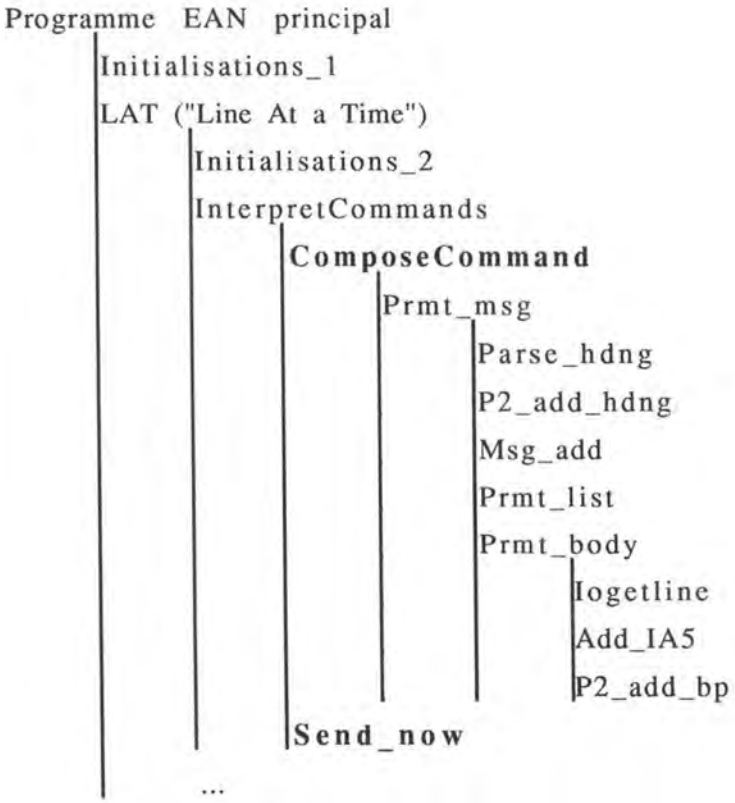
- préalablement à la rédaction d'un message, l'utilisateur doit se voir proposer le menu offert par la procédure EAN_sserv_prep (Problème 1);
- il faut intercepter le message une fois qu'il a été rédigé pour pouvoir le transformer en une forme sécurisée sous laquelle il sera ensuite soumis à un MTA (Problème 2).

2.1.1. Problème 1

Ce problème a été résolu en plaçant la procédure EAN_sserv_prep en tout début du programme EAN parmi les initialisations. Ce choix est assez logique puisque EAN_sserv_prep constitue la procédure d'initialisation des paramètres de sécurité de la version sécurisée de EAN.

2.1.2. Problème 2

Lors de la composition d'un message, l'enchaînement des procédures suivi par un message au sein du logiciel EAN 3.04 peut être schématiquement représenté par :



Après quelques initialisations (Initialisations_1 et Initialisations_2) dont le détail ne nous intéresse pas directement, nous aboutissons à la procédure correspondant à la composition d'un message par un utilisateur : ComposeCommand.

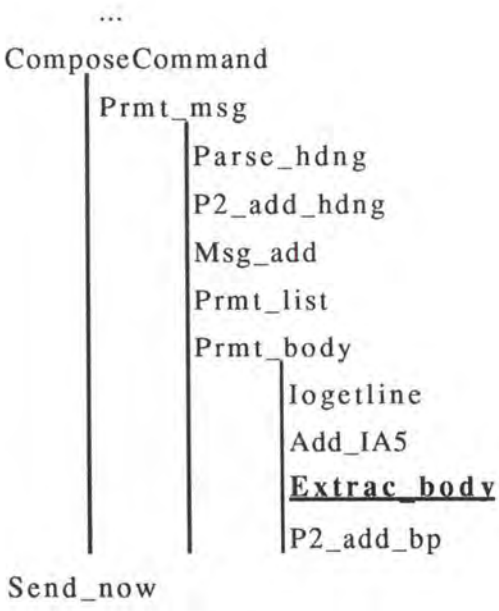
Cette procédure a comme buts principaux la construction d'un message et son enregistrement dans ce qu'on appelle un draft (un "brouillon" de message). Cette construction se fait essentiellement en deux étapes :

- la construction de l'ENODE correspondant à l'en-tête du message comprenant notamment les champs d'en-tête de l'émetteur et du destinataire du message : procédures Parse_hdng jusqu'à Prmt_list;
- l'acquisition et la construction du corps du message : procédure Prmt_body.

La procédure Prmt_body est constituée d'une boucle qui lit le texte du message ligne après ligne (Iogetline) et qui ajoute au fur et à mesure de cette lecture chacune des lignes à la structure d'ENODE correspondant au corps du message (Add_IA5[13]). Une fois que l'ENODE du corps du message est construit, il est ajouté à la structure de MESSAGE par la procédure P2_add_bp.

Ceci étant réalisé, le message est prêt à être envoyé via la procédure Send_now.

L'option retenue pour l'insertion de services de sécurité a été la suivante :



Juste avant l'étape où le corps du message est effectivement ajouté à la structure même du message, nous avons intercalé une procédure appelée Extrac_body. Cette procédure joue le rôle de lien entre le logiciel EAN non

---

(13) Le terme de IA5 (International Alphabet n° 5) fait référence à la notion de IA5 String définie dans [X409,84] : un string IA5 représente une séquence de caractères compris dans l'Alphabet International de référence n° 5.

sécurisé et la procédure EAN_sserv_cont_send du module ASF/EAN : elle permet de passer aux procédures relatives à la sécurité le corps du message.

Une fois que le travail de transformation en une forme sécurisée du corps du message est effectué (cfr. : analyse détaillée de la procédure EAN_sserv_cont_send), la procédure Extrac_body construit un nouvel ENODE correspondant à ce nouveau corps et le transmet en retour au logiciel EAN non sécurisé.

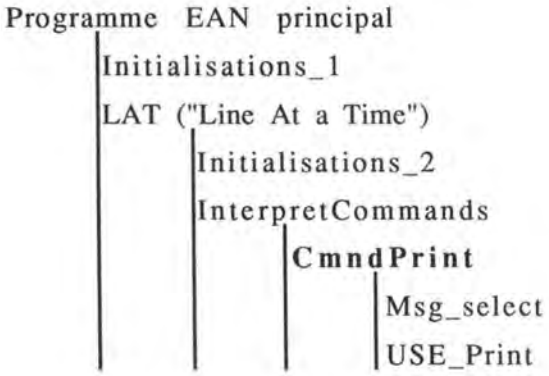
C'est donc ce corps de message sous forme sécurisée qui est ajouté dans la structure du message et qui va être effectivement envoyé.

En fait, la procédure Extrac_body ne se contente pas uniquement de transférer le corps du message : elle transmet aussi à la procédure EAN_sserv_cont_send l'en-tête du message. En effet, dans le cas où le message doit être mis sous forme confidentielle, la clé de session doit être chiffrée par la clé publique correspondant au destinataire du message; pour ce faire, nous avons donc besoin de l'adresse de courrier électronique de ce destinataire. La procédure définie dans EAN P2_find_hdng nous permet d'extraire de l'en-tête du message le champ correspondant au destinataire et donc d'obtenir finalement l'adresse désirée.

2.2. Du côté du récepteur du message

De manière similaire à ce qui se passe du côté de l'émetteur du message, nous avons dû construire une procédure qui permet d'intercepter le corps du message pour le passer à la procédure de transformation EAN_sserv_cont_rec.

L'enchaînement des procédures suivies par un message dans le logiciel EAN 3.0 depuis sa fourniture à un UA jusqu'à sa réception et sa consultation par un utilisateur est le suivant :



...

La réception d'un message par un utilisateur consiste en sa consultation lorsqu'il est affiché à l'écran : la procédure CmndPrint permet de réaliser cet affichage.

Pour un utilisateur donné, les messages envoyés et reçus sont normalement stockés dans un fichier spécial appelé inbox; à l'intérieur de ce fichier, les messages sont désignés par un numéro identificateur. Ainsi, à un moment donné, la liste des messages dans l'inbox d'un utilisateur donné peut être la suivante :

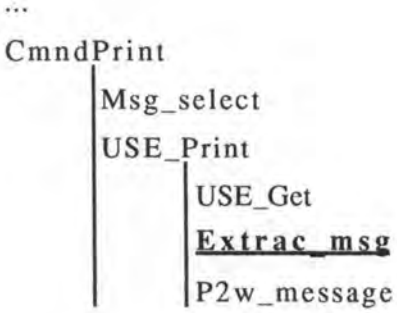
1	Jane Smith	Jun	9	94	Réunion urgente
2	Joe Brown	Jun	10	94	Remise mémoire
3	John Jackson	Jul	1	94	Vacances
4	Billy Johnson	Jul	6	94	Mondial 94

Pour consulter le message relatif aux Mondial 94, il suffira alors simplement à l'utilisateur d'entrer le numéro correspondant : 3.

Lors de l'entrée de ce numéro, la procédure Msg_select permet de sélectionner le message approprié dans le fichier inbox. Puis, la procédure USE_Get permet de se positionner de manière effective sur le pointeur vers la structure de MESSAGE.

Ensuite, la procédure P2w_message permet d'afficher à l'écran l'en-tête ainsi que le corps du message et enfin, la procédure Me_free libère l'espace mémoire alloué à cette structure de MESSAGE.

L'option retenue pour l'insertion de services de sécurité a été logiquement la suivante :



| |Me_free

La procédure Extrac_msg permet de faire le lien entre le logiciel EAN non sécurisé et la procédure EAN_sserv_cont_rec en transférant vers les fonctions de sécurité (cfr. : analyse détaillée de la procédure EAN_sserv_cont_rec) le message sécurisé avant qu'il ne soit affiché à l'écran.

A chaque consultation de message, la transformation de la forme sécurisée du message vers son format non sécurisé imprimable à l'écran est donc effectuée : cela permet donc de stocker les messages reçus sous une forme sécurisée tout en permettant à l'utilisateur de ne se préoccuper en aucune manière des diverses transformations subies par le corps du message et de consulter les messages sécurisés comme s'il s'agissait de simples messages non sécurisés.

REMARQUE IMPORTANTE : puisque les messages envoyés par un utilisateur apparaissent aussi dans son fichier inbox, on peut imaginer la situation où un utilisateur décide de consulter le texte d'un de ses messages après l'avoir envoyé. Dans le cas où le message envoyé a été traité par le service de confidentialité du contenu, on est confronté au problème suivant : puisque le clé de session a été chiffrée grâce à la clé publique correspondant au destinataire du message, la seule personne qui est à même de la déchiffrer est ce destinataire. Donc, l'émetteur du message ne peut plus disposer du contenu originel de ce message et il devra se contenter de l'affichage à l'écran de la forme sécurisée!

3. Utilisation du logiciel EAN sécurisé

3.1. Composition d'un message sécurisé

Après avoir lancé l'exécution du programme de messagerie X400 sécurisée en tapant "eansec" et éventuellement (dans le cas où il s'agit d'un utilisateur qui n'est pas encore enregistré en tant qu'utilisateur sécurisé) accepté de se faire enregistrer en tant que possesseur d'une clé publique, l'utilisateur n'a qu'à choisir le(s) services de sécurité qui l'intéresse(nt) et qui va (vont) être appliqué(s) sur ses différents envois de messages.

Ainsi, s'il désire que le service de confidentialité et de contrôle d'intégrité du contenu soient appliqués, l'utilisateur devra entrer le numéro correspondant : 3.

```
**********************************************************
*                        SECURE EAN                      *
**********************************************************
Make your choose between :
      1) Content confidentiality
      2) Content integrity
      3) Content confidentiality and integrity
      4) No security service
      3
```

Figure 39 : Choix du service de sécurité à appliquer.

Après cette étape d'initialisation des paramètres de sécurité, l'utilisateur se retrouve de manière classique au sein de l'environnement du logiciel EAN non sécurisé et n'a plus qu'à par exemple composer un message grâce à la commande comp.

Après la composition d'un message, un utilisateur possède en outre toujours la possibilité de consulter (par simple curiosité ou par besoin de se rassurer sur l'efficacité des mesures de sécurité) le texte sécurisé de son message qui hélas (ou tant mieux) peut être devenu totalement illisible.

3.2. Réception d'un message sécurisé

Après l'étape habituelle d'initialisation des paramètres relatifs à la sécurité, la réception d'un message sécurisé se déroule de manière totalement transparente pour son utilisateur : un récepteur de message sécurisé n'est même pas conscient des éventuels services de sécurité qui ont été appliqués au message reçu !!

La seule exception à cette transparence est la situation où l'intégrité du contenu du message n'est pas vérifiée; dans ce cas, le destinataire en est averti via le message : "Attention, the content integrity of your message is not OK !!!"

Conclusions et perspectives

L'approche offerte par l'architecture GSMHS a donc finalement permis d'introduire au sein d'une messagerie X400/84 les services de sécurité relatifs à la confidentialité du contenu du message et au contrôle d'intégrité.

Bien que valable dans un premier temps, cette solution au problème du manque de sécurité dans X400 mériterait d'être améliorée et davantage élargie :

- au lieu d'un envoi à un seul destinataire à la fois, il faudrait aussi envisager le cas multi-destinataire et les problèmes de gestion de clés qui en découleraient;

- au lieu de se limiter à l'implémentation d'un service de confidentialité relativement simplifié, il serait aussi intéressant d'implémenter des solutions plus élaborées comme la situation 7 (cfr. : chapitre 4 relatif à l'étude du service de confidentialité);

- pour fournir davantage de sécurité sur le point crucial de l'authenticité des clés publiques, il faudrait étudier la mise en œuvre d'une gestion efficace des certificats;

- l'implémentation de procédures comme Signa ou Verif_signa permettrait d'envisager assez facilement la mise en œuvre d'autres services de sécurité comme l'authenticité de l'identité de l'émetteur ou la non-répudiation de l'émetteur;

- au lieu de se limiter à la présentation simpliste d'une interface "on-line", il serait bien entendu intéressant d'imaginer une solution beaucoup plus conviviale et agréable à utiliser.

Sur base de ce travail, d'autres perspectives d'études peuvent aussi être envisagées:

- l'utilisation de l'EDI qui aurait pour base une messagerie X400 sécurisée;

- l'étude des problèmes d'interconnexion et de compatibilité entre notre messagerie X400 sécurisée et une messagerie PEM;
- l'étude et l'analyse de l'insertion de la sécurité non plus dans une messagerie X400/84 mais dans une messagerie X400/88;
- ...

Table des abréviations

ADMD	ADministration Management Domain.
ASF	Application-oriented Security Function.
CA	Certification Authority.
cdc	Centre de Distribution de Clés
CCITT	Comité Consultatif International pour la Téléphonie et la Télégraphie.
CRL	Certficate Revocation List.
D	Algorithme de déchiffrement.
DEK	Data Encrypting Key.
DES	Data Encryption Standard.
E	Algorithme de chiffrement.
EAN	Electronic Access Network
GCF	General Cryptographic Function.
GSMHS	Generalized Secure Message Handling System.
IA5	International Alphabet n° 5.
IK	Interchange Key.
IM-UAPDU	Interpersonal Messaging-UAPDU.
IPRA	Internet Policy Registration Authority.
k_m	Clé maître.
k_n	Clé de session.
k_{pX}	Clé publique de l'utilisateur X.
k_{sX}	Clé secrète de l'utilisateur X.
k_t	Clé de terminal.
M	Message en texte clair.
M'	Message chiffré.
MD	Management Domain.
MDC	Manipulation Detection Code.
MHS	Message Handling System.
MIC	Message Integrity Check.
MIME	Multipurpose Internet Mail Extensions.
MPDU	Message Protocol Data Unit.
MTA	Message Transfer Agent.
MTAE	Message Transfer Agent Entity.
MTL	Message Transfer Layer.

MTS	Message Transfer System.
O/R	Originator/Recipient.
PCA	Policy Certification Authority.
PEM	Privacy Enhancement for Internet Electronic Mail.
PRMD	Private Management Domain.
RSA	Rivest, Shamir et Adelman (algorithme de chiffrement).
SAPI	Security Application Program Interface.
SDE	Submission and Delivery Entity.
SFE	Security Feature Establishment.
SMTP	Simple Mail Transfer Protocol.
UA	User Agent.
UAE	User Agent Entity.
UAL	User Agent Layer.
UAPDU	User Agent Protocol Data Unit.

Table des figures

Table des procédures

Références

[ACM,94]

National Institute of Standards and Technology
Debating encryption standards
in : Communications of the ACM, Vol. 35, n°7, 1992, pp. 33-54.

[BAL,93]

D. BALENSON
Privacy enhancement for Internet electronic mail :
Part III - Algorithms, modes and identifiers
RFC 1423, February 1993.

[CCITT,88]

Rapporteur Groups 13/I and 14/I of CCITT
F400 : Message Handling Services-Final Draft Text
May 1988.

[CUN,83]

I. CUNNINGHAM
Message Handling systems and protocols
in : Proceedings of the IEEE, Vol. 71, n°2, December 1983, pp. 1425-1430.

[DAVIES,89]

D.W. DAVIES & W.L. PRICE
Security for Computer Networks
"An introduction to Data Security in Teleprocessing and Electronic Funds Transfer (Second Edition)"
John Wiley & Sons, 1989.

[DENN,81]

D.E. DENNING & G.M. SACCO
Timestamps in Key Ditribution Protocols
in : Communication of the ACM, Vol. 24, n°8, 1981, pp. 533-536.

[EAN,87]

G. NEUFELD

The EAN Distributed Message System User's Manual

University of British Columbia, Vancouver, 1987.

[EAN,92]

Centre Universitaire d'Informatique

Guide de l'utilisateur EAN sur UNIX

Université de Genève, Mai 1992.

[JAN,91]

P. JANSON & R. MOLVA

Security in open networks and distributed systems

in : Computer Networks and ISDN Systems, Vol. 22, 1991, pp. 323-346.

[KAL,93]

B. KALISKI

Privacy enhancement for Internet electronic mail :

Part IV - Key certification and related services

RFC 1424, February 1993.

[KEN,93]

S. T. KENT

Internet privacy enhanced mail

in : Communication of the ACM, Vol. 36, August 1993, pp. 48-60.

[KEN',93]

S. KENT

Privacy enhancement for Internet electronic mail :

Part II - Certificate-based key management

RFC 1422, February 1993.

[LIN,93]

J. LINN

Privacy enhancement for Internet electronic mail :

Part I - Message encipherment and authentication procedures

RFC 1421, February 1993.

[MAR,87]

A.D. MARSHALL & H.J. PODELL

Tutorial "Computer and Network Security"
IEEE Computer Society Order Number 756
The Computer Society of the IEEE, 1987.

[MITCH,90]
C. MITCHELL, D. RUSH & M. WALKER
A secure architecture implementing the X400-1988 security features
in : The Computer Journal, Vol. 33, n°4, 1990, pp. 290-295.

[MUF,92]
S. MUFTIC
Implementation of the Comprehensive Integrated Security System for Computer Networks
in : Computer Networks and ISDN Systems, 25, 1992, pp. 469-475.

[MYE,84]
T. H. MYER
Standards for global messaging : a progress report
in : Journal of Telecommunication Networks, pp. 413-433, 1984.

[RED,83]
D. REDELL & J. WHITE
Interconnecting Electronic Mail Systems
in : Computer, September 1983, pp. 55-63.

[RIV,92]
R. RIVEST
The MD5 Message-Digest Algorithm
RFC 1321, MIT Laboratory for Computer Science and RSA Data Security Inc., April 1992.

[RUT,89]
X. RUTTEN & J. VANTHOURNOUT
Etude du logiciel de courrier électronique EAN
Mémoire de licence et maîtrise en informatique, FUNDP Namur, 1989.

[SECUR]
J. RAMAEKERS & J. HUBIN
Sécurité et fiabilité des systèmes informatiques
Syllabus, FUNDP Namur, 1993.

[VAN,93]

Ph. van BASTELAER
Cours de téléinformatique et réseaux : fonctions et concepts
Cours de téléinformatique : matière approfondie
FUNDP Namur, 1993.

[WU,92]

S. WU
MHS Security - a concise survey
in : Computer Networks and ISDN Systems, Vol. 25, 1992, pp. 490-495.

[WU,93]

S. WU & J. RAMAEKERS
Specification of Security Application Program Interface (SAPI)
Work Draft (1.0), FUNDP Namur, March 1993.

[WU,94]

S. WU & J. RAMAEKERS
Integrating Security into Message Handling Systems
International Conference on Communication Technology, Shanghaï,
8-10 June 1994.

[X400,88]

CCITT Recommendation X.400
Data communications networks : Message handling system and service overview, November 1988.

[X409,84]

CCITT Recommendation X.409
Message Handling Systems : Presentation transfer syntax and notation
Malaga-Torremolinos, 1984.

[ZEG,93]

E. ZEGWAART
Privacy Enhanced Mail in more detail
in : Computer Networks and ISDN Systems, 25, Suppl. 2, 1993.

[ZIM,93]

P. ZIMMERMANN
Pretty Good Privacy Public Key Encryption for the Masses
in : PGP User's Guide, Volume I : Essential Topics, March 1993.

MODULE ASF/EAN

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include "util/defs.h"
#include "util/str.h"
#include "ccitt/element.h"
#include "eansec.h"
#include "global.h"
#include <pwd.h>
#include "des.h"
#define size_session_key 16

char* EAN_sserv_prep()
{
      long found;
      char togenerate[1];
      FILE *rsa,*ser,*adr;
      char service[2];
      char *serv,*identity,*address_sender;
      char *ajout="@seclib.info.fundp.rtt.be";
      char usrep[80],address[80],pubkey[80],prodkey[80];
      char rep;
      struct passwd *pwentry;

      umask(000);
      service[0]='0';
      pwentry=getpwuid(getuid());
      if (pwentry->pw_name=="postmaster") pwentry->pw_name="ean";
      address_sender=strcat(pwentry->pw_name,ajout);
      adr=fopen("Adem","wb");
      fprintf(adr,"%s",address_sender);
      fclose(adr);
      rsa=fopen("/home/witloof/users/ean/rsa_key","a+");
      rewind(rsa);
      found=0;
      while ((!feof(rsa))&&(found==0))
            {
                  fgets(address,80,rsa);
                  fgets(pubkey,80,rsa);
                  fgets(prodkey,80,rsa);
                  if (strcmp(address,address_sender)==0)
                        found=1;
            }
      fclose(rsa);
      if (found==0)
            {
                  printf("\n\n*****************************************
                  ***************************\n");
                  printf("You are a new user of the secure e-mail SECEAN.\n");
                  printf("If you want to dispose of security services and\n");
                  printf("be stored as a 'secure user' ask : Y.\n");
                  printf("If you do not, you will be unable to use security
                  services !!\n");
                  printf("*****************************************
                  ************************\n\n");
```

```
                scanf("%s",togenerate);
        }
    if (togenerate[0]=='Y')
        {
                Public_key_generate();
                Store_seckey();
                Pkeylist_add("/home/witloof/users/ean/rsa_key",
                address_sender);
                printf("\nRSA keys have been generated and stored.\n");
                printf("You have now a secret key associated with your
                electronic address.\n");
                printf("\n\n");
                service[0]='1';
        }
    if ((togenerate[0]!='Y')&&(found==0)) service[1]='4';
    if ((found==1)||(togenerate[0]=='Y'))
        {
                while ((rep!='1')&&(rep!='2')&&(rep!='3')&&(rep!='4'))
                        {
                                printf("\n\n\n\n\n**********************
                                ****************************\n");
                                printf("   *              SECURE EAN            *\n");
                                printf("*****************************
                                ***********************\n\n");
                                printf("Make your choose between :\n");
                                printf("1) Content confidentiality.\n");
                                printf("2) Content integrity.\n");
                                printf("3) Content confidentiality and
                                integrity.\n");
                                printf("4) No security service.\n");
                                gets(usrep);
                                if (strlen(usrep)==1)
                                        rep=usrep[0];
                                else
                                        rep='?';
                        }
                service[1]=rep;
        }
    serv=&service[0];
    ser=fopen("Service","wb");
    fprintf(ser,"%s",serv);
    fclose(ser);
    return(serv);
}


void Bldh_sec_msg(char* service, char* infile)
{
    FILE *hdr;
    FILE *hmsg;
    register long i;
    unsigned char mic[32];
    char cip_sess_key[80];
    char servic[1];

    servic[0]=*service;
    hdr=fopen("Header","rb");
    rewind(hdr);
    hmsg=fopen(infile,"wb");
    if (servic[0]=='1')
```

```
        {
                fscanf(hdr,"%s",&cip_sess_key);
                fprintf(hmsg,"Id_secure_service   :   1\n");
                fprintf(hmsg,"Id_algo_cip_key   :   RSA\n");
                fprintf(hmsg,"Id_algo_cip_msg   :   DES\n");
                fprintf(hmsg,"Session_key   :\n");
                fprintf(hmsg,"%s",cip_sess_key);
        }
    if  (servic[0]=='2')
        {
                for (i=0;i<32;i++)
                        fscanf(hdr,"%c",&mic[i]);
                fprintf(hmsg,"Id_secure_service   :   2\n");
                fprintf(hmsg,"Id_algo_mic   :   DIGEST\n");
                fprintf(hmsg,"Mic   :   ");
                for (i=0;i<32;i++)
                        fprintf(hmsg,"%c",mic[i]);
        }
    if  (servic[0]=='3')
        {
                fscanf(hdr,"%s\n",&cip_sess_key);
                for (i=0;i<32;i++)
                        fscanf(hdr,"%c",&mic[i]);
                fprintf(hmsg,"Id_secure_service   :   3\n");
                fprintf(hmsg,"Id_algo_cip_key   :   RSA\n");
                fprintf(hmsg,"Id_algo_cip_msg   :   DES\n");
                fprintf(hmsg,"Id_algo_mic   :   DIGEST\n");
                fprintf(hmsg,"Session_key   :\n");
                fprintf(hmsg,"%s",cip_sess_key);
                fprintf(hmsg,"Mic   :   ");
                for (i=0;i<32;i++)
                        fprintf(hmsg,"%c",mic[i]);
        }
    fclose(hmsg);
    unlink("Header");
    fclose(hdr);
}


void Inclu_sec_msg(char*   hdrfile,char*   bodyfile)
{
    FILE   *msg,*hdr,*body;
    char   line[80];
    struct LR ip;

    msg=fopen("bonmessage","wb");
    fprintf(msg,"************************BEGIN   OF   SECURE
    ZONE************************\n\n");
    hdr=fopen(hdrfile,"rb");
    while   (feof(hdr)==0)
        {
                fgets(line,80,hdr);
        fputs(line,msg);
        }
    fclose(hdr);
    unlink("Headermsg");
    fprintf(msg,"\n****************************BEGIN   OF
    MESSAGE****************************\n");
    body=fopen(bodyfile,"rb");
    if   (strcmp(bodyfile,"message")==0)
```

```
        {
                while (feof(body)==0)
                        {
                                if(fgets(line,80,body)!=NULL)
                                fputs(line,msg);
                        }
                fclose(body);
                unlink("message");
        }
    else
        {
                while (fread(&ip,1,sizeof(struct LR),body)!=0)
                      fwrite(&ip,1,sizeof(struct LR),msg);
                fclose(body);
                unlink("mess_enc");
        }
    fprintf(msg,"\n\n*****************************END OF
    MESSAGE*****************************\n");
    fprintf(msg,"*************************END OF SECURE
    ZONE*************************\n");
    fclose(msg);
}


void EAN_sserv_cont_send()
{
    char service[2];
    char text[80],public_key[80],prod_key[80];
    char *serv,*sess_key,*pub_key,*prod,*cip_sess_key,*address_dest;
    FILE *ser,*dst;
    long verif;
    char session_key[size_session_key];
    register long i;

    ser=fopen("Service","rb");
    fscanf(ser,"%s",service);
    fclose(ser);
    if (service[1]=='1')
        {
                dst=fopen("hdnrec","rb");
                fscanf(dst,"%s",text);
                fclose(dst);
                unlink("hdnrec");
                address_dest=text;
                verif=Verif_pkey_recip("/home/witloof/users/ean/rsa_key",
                address_dest);
                if (verif==1)
                {
                      pub_key=Pkeylist_extrac("/home/witloof/users/ean
                      /rsa_key",address_dest);
                      prod=Pkeylist_extrac2("/home/witloof/users/ean
                      /rsa_key",address_dest);
                      i=0;
                      while (*prod)
                      {
                      if (((unsigned long) (*prod)>=48)&&((unsigned long)
                      (*prod)<=57))
                            {
                                  prod_key[i]=*prod;
                                  i++;
```

```
                        }
                    *prod++;
                    }
                    i=0;
                    while  (*pub_key)
                    {
                    if  (((unsigned  long)  (*pub_key)>=48)&&((unsigned
                    long)  (*pub_key)<=57))
                        {
                            public_key[i]=*pub_key;
                            i++;
                        }
                    *pub_key++;
                    }
                    sess_key=Sess_key_generate();
                    strncpy(session_key,sess_key,size_session_key);
                    Encryp_sess_key(session_key,public_key,prod_key,
                    "RSA");
                    serv=&service[1];
                    Bldh_sec_msg(serv,"Headermsg");
                    Encryp_mbody("message","DES",sess_key);
                    Inclu_sec_msg("Headermsg","mess_enc");
                }
                else
                {
                    printf("\nThe recipient of the message is unknow in our
                    SECURE EAN\n");
                    printf("The message will be sent without security
                    service applied on it\n");
                }
        }
if  (service[1]=='2')
        {
            MIC_compute("message","DIGEST");
            serv=&service[1];
            Bldh_sec_msg(serv,"Headermsg");
            Inclu_sec_msg("Headermsg","message");
        }
if  (service[1]=='3')
        {
            dst=fopen("hdnrec","rb");
            fscanf(dst,"%s",text);
            fclose(dst);
            unlink("hdnrec");
            address_dest=text;
            verif=Verif_pkey_recip("/home/witloof/users/ean/rsa_key",
            address_dest);
            if  (verif==1)
            {
                pub_key=Pkeylist_extrac("/home/witloof/users/ean
                /rsa_key",address_dest);
                prod=Pkeylist_extrac2("/home/witloof/users/ean
                /rsa_key",address_dest);
                i=0;
                while  (*prod)
                {
                if  (((unsigned  long)  (*prod)>=48)&&((unsigned  long)
                (*prod)<=57))
                    {
                        prod_key[i]=*prod;
```

```
                        i++;
                    }
                *prod++;
                }
                i=0;
                while  (*pub_key)
                {
                if  (((unsigned  long)  (*pub_key)>=48)&&((unsigned
                long)  (*pub_key)<=57))
                    {
                        public_key[i]=*pub_key;
                        i++;
                    }
                *pub_key++;
                }
                sess_key=Sess_key_generate();
                strncpy(session_key,sess_key,size_session_key);
                Encryp_sess_key(session_key,public_key,prod_key,
                "RSA");
                MIC_compute("message","DIGEST");
                serv=&service[1];
                Bldh_sec_msg(serv,"Headermsg");
                Encryp_mbody("message","DES",sess_key);
                Inclu_sec_msg("Headermsg","mess_enc");
            }
            else
            {
                printf("\nThe recipient of the message is unknow in our
                SECURE EAN\n");
                printf("The message will be sent with only content
                integrity  service\n");
            }
        }
}

void  Ubldh_sec_msg(char  *infile,char  *outfile)
{
    FILE  *in,*out;
    char  *algo1,*algo2,*algo3,*mic,*cip_sess_key,*sec_serv;
    char  line[80];
    char  service[2];
    register  long  i;

    in=fopen(infile,"rb");
    out=fopen(outfile,"wb");
    fgets(line,80,in);
    sec_serv=&line[20];
    fputs(sec_serv,out);
    service[0]=*sec_serv;
    if  (service[0]=='1')
        {
            fgets(line,80,in);
            algo1=&line[18];
            fputs(algo1,out);
            fgets(line,80,in);
            algo2=&line[18];
            fputs(algo2,out);
            fgets(line,80,in);
            fgets(line,80,in);
```

```
                cip_sess_key=&line[0];
                fputs(cip_sess_key,out);
        }
    if (service[0]=='2')
        {
                fgets(line,80,in);
                algo3=&line[14];
                fputs(algo3,out);
                fgets(line,80,in);
                mic=&line[6];
                fputs(mic,out);
        }
    if (service[0]=='3')
        {
                fgets(line,80,in);
                algo1=&line[18];
                fputs(algo1,out);
                fgets(line,80,in);
                algo2=&line[18];
                fputs(algo2,out);
                fgets(line,80,in);
                algo3=&line[14];
                fputs(algo3,out);
                fgets(line,80,in);
                fgets(line,80,in);
                cip_sess_key=&line[0];
                fputs(cip_sess_key,out);
                fgets(line,80,in);
                mic=&line[6];
                fputs(mic,out);
        }
    fclose(in);
    fclose(out);
    unlink("header2");
}

void Extra_mparts(char *infile, char *bout, char *hout)
{
    FILE  *msg,*body,*hdr;
    char  line[80],bon[80],bline[80][80],der[80];
    char  service[2];
    char   car_inter,car_0,car_1;
    long   found1,found2,fin_msg,nblig;
    char   *sec_serv,*inter,*ptr;
    register long i,j;

    msg=fopen(infile,"rb");
    fgets(line,80,msg);
    fgets(line,80,msg);
    found1=0;
    body=fopen(bout,"wb");
    while (found1==0)
        {
            if(strcmp(line,"**************************BEGIN  OF
            MESSAGE**************************")!=0)
                fputs(line,body);
            else
                found1=1;
        }
    fclose(body);
```

```
body=fopen(bout,"rb");
fgets(line,80,body);
sec_serv=&line[20];
service[0]=*sec_serv;
fclose(body);
found2=0;
hdr=fopen(hout,"wb");
if (service[0]=='2')
        {
                j=0;
                while (found2==0)
                        {
                                fgets(line,80,msg);
                                if(strcmp(line,"**************************
                                END OF
MESSAGE**********************")!=0)
                                        {
                                        ptr=&line[0];
                                        i=0;
                                        while (*ptr)
                                                {
                                                if (((unsigned long) (*ptr)>=32)
                                                &&((unsigned long) (*ptr)<=126))
                                                        {
                                                                bline[j][i]=*ptr;
                                                                i++;
                                                        }
                                                *ptr++;
                                                }
                                        bline[j][i]=(char)(13);
                                        bline[j][i+1]='\0';
                                        j++;
                                        }
                                else
                                        found2=1;
                        }
                nblig=j;
                for (j=0;j<=nblig-2;j++)
                if(j!=nblig-2)
                        fprintf(hdr,"%s\n",bline[j]);
                else
                        {
                                i=0;
                                ptr=&bline[j][0];
                                while (*ptr)
                                {
                                        if (((unsigned long) (*ptr)>=32)&&
                                        ((unsigned long) (*ptr)<=126))
                                                {
                                                        der[i]=*ptr;
                                                        i++;
                                                }
                                        *ptr++;
                                }
                                der[i]='\0';
                                fprintf(hdr,"%s",der);
                        }
                fclose(hdr);
        }
 else
```

```
        {
                fin_msg=0;
                i=0;
                while(fin_msg==0)
                {
                        fread(&car_inter,1,sizeof(char),msg);
                        fin_msg=((car_inter=='*')&&(car_1=='*')&&(car_0=='*'));
                        if ((i>=2)&&(fin_msg==0))
                                fwrite(&car_0,1,sizeof(char),hdr);
                        car_0=car_1;
                        car_1=car_inter;
                        i++;
                }
                fclose(hdr);
        }
    fclose(msg);
}


void EAN_sserv_cont_rec()
{
    FILE  *msg2,*hdr,*bdo,*bfin,*flag,*ao,*dest;
    char  text[80],txt[80],adr[80],cip_key[80],algo_cmsg[80],algo_ckey[80];
    char  algo_mic[80],mic[80];
    char  car_inter;
    char  *tex,*sess_key;
    register long  i;
    long  ok,verif;
    char  service[2];
    char  session_key[size_session_key];

    ok=0;
    msg2=fopen("message2","rb");
    fgets(text,80,msg2);
    if (strlen(text)!=73) ok=1;
    if (ok==0)
        for (i=1;i<=25;i++)
            if (text[i]!='*') ok=1;
    if (ok==0)
        for (i=46;i<=70;i++)
            if (text[i]!='*') ok=1;
    if (ok==0)
        {
            tex=&text[26];
            if(strncmp(tex,"BEGIN OF SECURE ZONE",23)!=0) ok=1;
        }
    if (ok==0)
        {
            flag=fopen("flag","w");
            fprintf(flag,"%d",ok);
            fclose(flag);
            fclose(msg2);
            ao=fopen("RECEV","r");
            fscanf(ao,"%s",&txt);
            fclose(ao);
            dest=fopen("Adem","r");
            fscanf(dest,"%s",&adr);
            fclose(dest);
            if  (strcmp(txt,adr)!=0)
                {
```

```
                bfin=fopen("mess_fin","w");
                fprintf(bfin,"YOUR MESSAGE IS A SECURE
MESSAGE                 !!!\n");
                fprintf(bfin,"\n");
                fclose(bfin);
        }
    else
        {
                Extra_mparts("message2","header2","body2");
                bdo=fopen("body2","rb");
                bfin=fopen("mess_fin","wb");
                while  (!feof(bdo))
                {
                        fread(&car_inter,1,sizeof(char),bdo);
                        fwrite(&car_inter,1,sizeof(char),bfin);
                }
                fclose(bfin);
                fclose(bdo);
                Ubldh_sec_msg("header2","header3");
                hdr=fopen("header3","rb");
                fgets(service,80,hdr);
                if  (service[0]=='1')
                {
                        fgets(algo_ckey,80,hdr);
                        fgets(algo_cmsg,80,hdr);
                        fgets(cip_key,80,hdr);
                        fclose(hdr);
                        sess_key =
                        Decryp_sess_key(cip_key,algo_ckey);
                        printf("Sess   %s\n",sess_key);
                        Decryp_mbody
                        ("body2",algo_cmsg,sess_key);
                }
                if  (service[0]=='2')
                {
                        fgets(algo_mic,80,hdr);
                        fgets(mic,80,hdr);
                        fclose(hdr);
                        verif =
                        Verif_mic_compute("body2",mic,algo_mic);
                        if  (verif!=1)
                        printf("\n Attention, the content integrity
                        of your message is not OK !!!\n");
                }
                if  (service[0]=='3')
                {
                        fgets(algo_ckey,80,hdr);
                        fgets(algo_cmsg,80,hdr);
                        fgets(algo_mic,80,hdr);
                        fgets(cip_key,80,hdr);
                        fgets(mic,80,hdr);
                        fclose(hdr);
                        sess_key =
                        Decryp_sess_key(cip_key,algo_ckey);
                        Decryp_mbody
                        ("body2",algo_cmsg,sess_key);
                        verif =
                        Verif_mic_compute("body2",mic,algo_mic);
                        if  (verif!=1)
```

```
                                printf("\n Attention, the content integrity
                                of your message is not OK !!!\n");
                        }
                        unlink("body2");
                        unlink("header3");
                }
        }
    else
        {
                flag=fopen("flag","w");
                fprintf(flag,"%d",ok);
                fclose(flag);
                fclose(msg2);
        }
}
```

MODULE GCF

```
#include <stdio.h>
#include <string.h>

void Encryp_mbody(char *infile,char *id_algo_cip_msg,char *sess_key)
{
    if (strcmp(id_algo_cip_msg,"DES")==0)
        Encryp_DES(sess_key,infile,"mess_enc");
}


void Decryp_mbody(char *infile,char* id_algo_cip_msg,char *sess_key)
{
    if (strcmp(id_algo_cip_msg,"DES")==0)
        Decryp_DES(sess_key,infile,"mess_fin");
}


void MIC_compute(char *infile,char *id_algo_mic)
{
    if (strcmp(id_algo_mic,"DIGEST")==0)
        Digest(infile);
}


long Verif_mic_compute(char *infile,char *mic,char *id_algo_mic)
{
    FILE *hdr;
    char *mic1;
    long verif;

    verif=0;
    if (strcmp(id_algo_mic,"DIGEST")==0)
        {
            Digest(infile);
            hdr=fopen("Header","rb");
            fgets(mic1,80,hdr);
            fclose(hdr);
```

```
                if(strcmp(mic,mic1)==0)
                        verif=1;
                unlink("Header");
        }
    return(verif);
}


void Encryp_sess_key(char *sess_key,char *pub,char *nkey,char *id_algo)
{
    if (strcmp(id_algo,"RSA")==0)
        Encryp_RSA_key(sess_key,pub,nkey);
}


char* Decryp_sess_key(char *s_key,char *id_algo)
{
    char *decip_key;

    if (strcmp(id_algo,"RSA")==0)
        decip_key=Decryp_RSA_key(s_key);
    return(decip_key);
}
```

MODULE DES FONCTIONS DE SECURITE : fonctions cryptographiques

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "global.h"
#include "md5.h"
#include "verylong.h"
#include "des.h"
#define size_session_key 16

void transfo_en_very(char *s,verylong *a)
{
    register long i=0;
    long longueur_s;
    long transfo=1000;

    longueur_s=strlen(s);
    zintoz(0,a);
    for (i=0;i<longueur_s;i++)
        {
            zsadd(*a, (unsigned long) (s[i]),a);
            if (i!=longueur_s-1) zsmul(*a,transfo,a);
        }
}


void vers_very(char *s,verylong *a)
{
    register long i=0;
```

```
        long  longueur_s;
        long  transfo=10;

        longueur_s=strlen(s);
        zintoz(0,a);
        for  (i=0;i<longueur_s;i++)
                {
                        zsadd(*a,  (unsigned  long)  (s[i]-48),a);
                        if  (i!=longueur_s-1)  zsmul(*a,transfo,a);
                }
}


void  transfo_en_ascii(verylong  a,char  *s)
{
        register  long  i=0;
        long  longueur_s,long_car,reste;
        verylong  a_local=0;
        long  diviseur=1000;
        char  car_inter[100];

        i=0;
        zcopy(a,&a_local);
        while  (zsign(a_local))
                {
                        reste=zsdiv(a_local,diviseur,&a_local);
                        car_inter[i]=(char)  ((int)reste);
                        i++;
                }
        longueur_s=i;
        for  (i=0;i<=longueur_s-1;i++)
                s[i]=car_inter[longueur_s-i-1];
        s[longueur_s]='\0';
}


void  Encryp_DES(char*  key,  char*  infile,  char*  outfile)
{
        int  i;
        struct  LR  op,  ip;
        FILE  *fi,  *fo;
        struct  ks  keys[size_session_key];
        int  une;

        une  =  1;
        for  (i  =  0;  i  <  size_session_key;  i++)
                keys[i]  =  KS(i,key);
        if  ((fi  =  fopen(infile,  "rb"))  !=  NULL)
        {
                if  ((fo  =  fopen(outfile,  "wb"))  !=  NULL)
                {
                        while  (fread(&ip,  une,  sizeof(struct  LR),  fi)  !=  0)
                        {
                                int  n;
                                permute(&op.L,  &ip.L,  (long  *)IPtbl,  64);
                                for  (n  =  0;  n  <  size_session_key;  n++)
                                {
                                        ip.L  =  op.R;
                                        ip.R  =  op.L  ^  f(op.R,  keys[n]);
                                        op.R  =  ip.R;
```

```
                        op.L = ip.L;
                    }
                    inverse_permute(&op.L, &ip.L, (long *)IPtbl, 64);
                    fwrite(&op, une, sizeof(struct LR), fo);
                    ip.L = ip.R = 0;
                }
                fclose(fo);
            }
        fclose(fi);
        }
        unlink("message");
}

void Decryp_DES(char* key, char* infile, char* outfile)
{
        int i;
        struct LR op, ip;
        FILE *fi, *fo;
        struct ks keys[size_session_key];
        int une;

        une = 1;
        for (i = 0; i < size_session_key; i++)
                keys[i] = KS(i, key);
        if ((fi = fopen(infile, "rb")) != NULL)
        {
                if ((fo = fopen(outfile, "wb")) != NULL)
                {
                        while (fread(&ip, une, sizeof(struct LR), fi) != 0)
                        {
                                int n;
                                permute(&op.L, &ip.L, (long *)IPtbl, 64);
                                for (n = size_session_key-1;  n >= 0; --n)
                                {
                                        ip.R = op.L;
                                        ip.L = op.R ^ f(op.L, keys[n]);
                                        op.R = ip.R;
                                        op.L = ip.L;
                                }
                                inverse_permute(&op.L, &ip.L, (long *)IPtbl, 64);
                                fwrite(&op, une, sizeof(struct LR), fo);
                                ip.L = ip.R = 0;
                        }
                        fclose(fo);
                }
        fclose(fi);
        }
}

static  void  MD5Print(digest)
        unsigned  char  digest[16];

{
        unsigned  int  i;
        for(i=0;i<16;i++)
                printf("%02x",digest[i]);
}
```

```
void Digest(char  *infile)
{
      MD5_CTX context;
      unsigned  char  verif[16];
      unsigned  int  length_message,i;
      char  car_inter;
      int  count;
      char  message[5000];
      char  service[2];
      FILE  *msg,*inter,*ser;

      msg=fopen(infile,"rb");
      count=0;
      while  (feof(msg)==0)
            {
                  fread(&car_inter,1,sizeof(char),msg);
                  message[count]=car_inter;
                  count++;
            }
      length_message=count-1;
      MD5Init(&context);
      MD5Update(&context,message,length_message);
      MD5Final(verif,&context);
      MD5Print(verif);
      fclose(msg);
      ser=fopen("Service","rb");
      fscanf(ser,"%s",service);
      fclose(ser);
      inter=fopen("Header","ab");
      if  (service[1]=='3')  fprintf(inter,"\n");
      for(i=0;i<16;i++)
            fprintf(inter,"%02x",verif[i]);
      fprintf(inter,"\n");
      fclose(inter);
}


char*  Decryp_RSA_key(char  *cip_sess_key)
{
      FILE  *sec,*ess;
      char  *bsec,*bpro,*bkey,*bitm,*resul;
      char  seckey[80],sec2[80],itm[80],prodkey[80],pro2[80],cip2[80];
      char  decip_key[16];
      verylong  vsec=0;
      verylong  vprod=0;
      verylong  vcip=0;
      verylong  vdecip=0;
      register  long  i;

      sec=fopen("SECKEY","rb");
      fgets(seckey,80,sec);
      fgets(prodkey,80,sec);
      fclose(sec);
      ess=fopen("TRUC","r");
      fscanf(ess,"%s",itm);
      fclose(ess);
      bsec=&seckey[0];
      i=0;
      while  (*bsec)
```

```
        {
                if (((unsigned long) (*bsec)>=48) && ((unsigned long
                (*bsec)<=57))
                        {
                                sec2[i]=*bsec;
                                i++;
                        }
                *bsec++;
        }
    sec2[i]='\0';
    bpro=&prodkey[0];
    i=0;
    while  (*bpro)
        {
                if (((unsigned long) (*bpro)>=48) && ((unsigned long)
                (*bpro)<=57))
                        {
                                pro2[i]=*bpro;
                                i++;
                        }
                *bpro++;
        }
    pro2[i]='\0';
    bitm=&itm[0];
    i=0;
    while  (*bitm)
        {
                if (((unsigned long) (*bitm)>=48) && ((unsigned long)
                (*bitm)<=57))
                        {
                                cip2[i]=*bitm;
                                i++;
                        }
                *bitm++;
        }
    cip2[i]='\0';
    vers_very(sec2,&vsec);
    vers_very(pro2,&vprod);
    vers_very(cip2,&vcip);
    zexpmod(vcip,vsec,vprod,&vdecip);
    transfo_en_ascii(vdecip,decip_key);
    resul=&decip_key[0];
    return(resul);
}

void Encryp_RSA_key(char  *sess_key,char  *pub,char  *nkey)
{
    verylong  session_key=0;
    verylong  pub_key=0;
    verylong  cip_key=0;
    verylong  n_key=0;
    register long  i;
    long  pas_ok;
    char  str_key[80],nk[80],pk[80];
    char  *ptr,*ptr2;
    FILE  *hdr;

    ptr=&nkey[0];
    ptr2=&pub[0];
```

```
        pas_ok=0;
        i=0;
        while  ((*ptr)&&(pas_ok==0))
                {
                        if  (((unsigned  long)(*ptr)>=48)&&((unsigned  long)(*ptr)<=57))
                                {
                                        nk[i]=*ptr;
                                        i++;
                                }
                        else
                                pas_ok=1;
                        *ptr++;
                }
        nk[i]='\0';
        pas_ok=0;
        i=0;
        while  ((*ptr2)&&(pas_ok==0))
                {
                        if  (((unsigned  long)(*ptr2)>=48)&&((unsigned
                        long)(*ptr2)<=57))
                                {
                                        pk[i]=*ptr2;
                                        i++;
                                }
                        else
                                pas_ok=1;
                        *ptr2++;
                }
        pk[i]='\0';
        transfo_en_very(sess_key,&session_key);
        vers_very(pk,&pub_key);
        vers_very(nk,&n_key);
        zexpmod(session_key,pub_key,n_key,&cip_key);
        vers_ascii(cip_key,str_key);
        hdr=fopen("Header","wb");
        fprintf(hdr,"%s",str_key);
        fclose(hdr);
}
```

MODULE DES FONCTIONS DE SECURITE : gestion des clés

```
#include  <stdio.h>
#include   <string.h>
#include   <math.h>
#include  <time.h>
#include   "global.h"
#include   "md5.h"
#include    "verylong.h"
#define  size_session_key  16

void  vers_ascii(verylong  a,char  *s)
{
        register  long  i=0;
        long    longueur_s,long_car,reste;
```

```
        verylong  a_local=0;
        long  diviseur=10;
        char  car_inter[1000];

        i=0;
        zcopy(a,&a_local);
        while  (zsign(a_local))
                {
                        reste=zsdiv(a_local,diviseur,&a_local);
                        car_inter[i]=(char)((int)reste+48);
                        i++;
                }
        longueur_s=i;
        for  (i=0;i<=longueur_s-1;i++)
                s[i]=car_inter[longueur_s-i-1];
        s[longueur_s]='\0';
}


long Seed_generate()
{
        time_t  seconds;
        struct  tm  *gmt_hour;
        struct  tm  hour_gmt;
        long  seed;

        time(&seconds);
        gmt_hour=gmtime(&seconds);
        hour_gmt=*gmt_hour;
        seed=hour_gmt.tm_hour*10000+hour_gmt.tm_min*100+hour_gmt.tm_sec;
        return(seed);
}

char*  Sess_key_generate()
{
        char*  sess_key;
        register  long  i;
        long  seed;
        int  variable;
        double  random_number;
        char  key_inter[size_session_key];

        seed=Seed_generate();
        srand(seed);
        for  (i=0;i<=size_session_key-1;i++)
                {
                        random_number=rand();
                        variable=(int)  (33+(126-33)*(random_number/2147483647));
                        key_inter[i]=(char)variable;
                }
        sess_key=key_inter;
        return(sess_key);
}


void  Public_key_generate()
{
        verylong p = 0;
        verylong q = 0;
        verylong  verif=0;
```

```
        verylong prod_inter = 0;
        verylong ks = 0;
        verylong kp = 0;
        verylong produit = 0;
        verylong  vseed=0;
        long   long_p_inter,long_q_inter,seed;
        long  longueur_p  =102;
        long  longueur_q  =98;
        long  longueur_cle_secrete  =200;
        char   string_secret[100],string_public[100],string_pro[100];
        FILE*  rsa;

        long_p_inter=longueur_p;
        long_q_inter=longueur_q;
        seed=Seed_generate();
        zintoz(seed,&vseed);
        zrstart(vseed);
        while   (!zrandomprime(long_p_inter,10,&p));
              long_p_inter=long_p_inter-1;
        while   ((p==q)||(!zrandomprime(long_q_inter,10,&q)));
              long_q_inter=long_q_inter-1;
        zmul(p,q,&produit);
        zsadd(p,-1,&p);
        zsadd(q,-1,&q);
        zmul(p,q,&prod_inter);
        do
              {
                    zrandoml(longueur_cle_secrete,&ks);
                    zgcd(ks,prod_inter,&verif);
              }
        while   (zinv(ks,prod_inter,&kp));
        vers_ascii(ks,string_secret);
        vers_ascii(kp,string_public);
        vers_ascii(produit,string_pro);
        rsa=fopen("rsa_bis","wb");
        fprintf(rsa,"%s\n%s\n%s\n",string_secret,string_public,string_pro);
        fclose(rsa);
}


long Verif_pkey_recip(char*   fpklist,char   *address)
{
        FILE *rsa;
        char add[80],  pubkey[80],  prodkey[80];
        long  found;

        rsa=fopen(fpklist,"rb");
        found=0;
        rewind(rsa);
        while   ((feof(rsa)==0)&&(found==0))
              {
                    fgets(add,80,rsa);
                    fgets(pubkey,80,rsa);
                    fgets(prodkey,80,rsa);
                    if   (strcmp(add,address)==0)
                          found=1;
              }
        fclose(rsa);
        return(found);
}
```

```
void Store_seckey()
{
      FILE  *rsa_inter,*secfile;
      char  seckey[80],prodkey[80],pubkey[80];

      rsa_inter=fopen("rsa_bis","rb");
      fgets(seckey,80,rsa_inter);
      fgets(pubkey,80,rsa_inter);
      fgets(prodkey,80,rsa_inter);
      fclose(rsa_inter);
      umask(077);
      secfile=fopen("SECKEY","wb");
      fprintf(secfile,"%s%s",seckey,prodkey);
      fclose(secfile);
      umask(000);
}


void Skey_prompt()
{
      FILE  *secfile;
      char  seckey[80],prodkey[80];

      secfile=fopen("SECKEY","wb");
      fscanf(secfile,"%s%s",seckey,prodkey);
      fclose(secfile);
      printf("Your secret key is :\n");
      printf("%s\n",seckey);
      printf("%s\n",prodkey);
}


void Pkeylist_add(char*  fpklist,char  address[80])
{
      FILE  *rsa_inter,*rsa;
      char  seckey[80],pubkey[80],prodkey[80];

      rsa_inter=fopen("rsa_bis","rb");
      fgets(seckey,80,rsa_inter);
      fgets(pubkey,80,rsa_inter);
      fgets(prodkey,80,rsa_inter);
      fclose(rsa_inter);
      unlink("rsa_bis");
      rsa=fopen(fpklist,"ab+");
      fprintf(rsa,"%s\n%s%s",address,pubkey,prodkey);
      fclose(rsa);
}


char*  Pkeylist_extrac(char*  fpklist,char*  address)
{
      long  found;
      FILE  *rsa;
      char*  resul;
      char  pubkey[80],prodkey[80],add[80];

      found=0;
      rsa=fopen(fpklist,"rb");
```

```
        rewind(rsa);
        while  (found==0)
                {
                        fgets(add,80,rsa);
                        fgets(pubkey,80,rsa);
                        fgets(prodkey,80,rsa);
                        if  (strcmp(add,address)==0)  found=1;
                }
        fclose(rsa);
        resul=pubkey;
        return(resul);
}


char*  Pkeylist_extrac2(char*   fpklist,char*   address)
{
        long  found;
        FILE  *rsa;
        char*   resul;
        char  pubkey[80],prodkey[80],add[80];

        found=0;
        rsa=fopen(fpklist,"rb");
        rewind(rsa);
        while  (found==0)
                {
                        fgets(add,80,rsa);
                        fgets(pubkey,80,rsa);
                        fgets(prodkey,80,rsa);
                        if  (strcmp(add,address)==0)  found=1;
                }
        fclose(rsa);
        resul=prodkey;
        return(resul);
}
```

LIEN ENTRE EAN ET LES MODULES DE SECURITE LORS DE L'EMISSION D'UN MESSAGE

```
#include  <stdio.h>
#include   <string.h>
#include   <math.h>
#include    "util/defs.h"
#include    "util/io.h"
#include    "util/time.h"
#include    "util/buf.h"
#include    "util/str.h"
#include    "util/chr.h"
#include    "util/byte.h"
#include     "util/host/frs.h"
#include    "util/host/txf.h"
#include    "util/host/blk.h"
#include    "ccitt/element.h"
#include    "ua/sequence.h"
```

```
#include   "ua/use/message.h"
#include   "ua/folder/folder.h"
#include   "ua/msgset.h"
#include   "ua/use/frs.h"
#include   "ua/p2/p2.h"
#include   "ua/error.h"
#include   "mta/or.h"
#include   "mta/p1.h"
#include   "eansec.h"
#include   "global.h"
#include   "md5.h"
#include   "verylong.h"


ENODE*  Extrac_body(bp,m)
        ENODE*      bp;
        MESSAGE*    m;
{
        FILE   *msg,*cip,*inter,*recip;
        char   *text_body,*sess_key,*text,*texte,*t1,*t2,*t3;
        char   *ajout="@seclib.info.fundp.rtt.be";
        char   alias_pos[40];
        char   session_key[size_session_key];
        char   line[80],der[80],iter[80],der2[80];
        ENODE  *e,*rhdn;
        ENODE**   end_ptr;
        int    l,nb;
        char   c,car;
        long   i;

        e = NULL;  end_ptr = NULL;
        if  (bp)
                {
                        umask(000);
                        text_body=xe_stru(bp);
                        *text_body++;
                        msg=fopen("message","wb");
                        fprintf(msg,"%s",text_body);
                        fclose(msg);
                        cip=fopen("bonmessage","wb");
                        fprintf(cip,"%s",text_body);
                        fclose(cip);
                        rhdn=P2_find_hdng(m->content,P2_RECIPS);
                        text=xe_stru(rhdn);
                        i=0;
                        while   (*text)
                        {
                                if  (((unsigned  long)(*text)>=32)&&((unsigned
                                long)(*text)<=126))
                                        {
                                                der[i]=*text;
                                                i++;
                                        }
                                *text++;
                        }
                        der[i]='\0';
                        t1=&der[0];
                        while   (car!='o')
                                {
                                        car=*t1;
```

```
                        *t1++;
                }
        nb=0;
        while  (strlen(t1)>6)
                {
                        iter[nb]=*t1;
                        *t1++;
                        nb++;
                }
        iter[nb]='\0';
        t2=&iter[0];
        i=0;
        while  (*t2)
                {
                        if  (((unsigned  long)(*t2)>=32)&&((unsigned
                        long)(*t2)<=126))
                                {
                                        der2[i]=*t2;
                                        i++;
                                }
                        *t2++;
                }
        der2[i]='\0';
        t3=&der2[0];
        if  (strcmp(t3,"postmaster")==0)
                {
                        alias_pos[0]='e';
                        alias_pos[1]='a';
                        alias_pos[2]='n';
                        alias_pos[3]='\0';
                        texte=strcat(alias_pos,ajout);
                }
        else
                texte=strcat(t3,ajout);
        recip=fopen("hdnrec","wb");
        fprintf(recip,"%s",texte);
        fclose(recip);

        EAN_sserv_cont_send();

        cip=fopen("bonmessage","rb");
        l = 0;
        while(!feof(cip))
                if(fgets(line,  80,  cip)!=NULL)
                        {
                                l=strlen(line);
                                line[l-1] = '\0';
                                end_ptr = Add_IA5(&e, end_ptr, line);
                        }
        fclose(cip);
        unlink("bonmessage");
    }
return  (e);
}
```

LIEN ENTRE EAN ET LES MODULES DE SECURITE LORS DE LA RECEPTION D'UN MESSAGE

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "util/defs.h"
#include "util/io.h"
#include "util/time.h"
#include "util/host/frs.h"
#include "util/host/txf.h"
#include "util/host/blk.h"
#include "util/byte.h"
#include "util/str.h"
#include "util/buf.h"
#include "ccitt/element.h"
#include "mta/or.h"
#include "mta/p1.h"
#include "ua/sequence.h"
#include "ua/use/message.h"
#include "ua/folder/folder.h"
#include "ua/msgset.h"
#include "ua/use/frs.h"
#include "ua/p2/p2.h"
#include "ua/error.h"
#include "eansec.h"

void Extrac_msg(m)
        MESSAGE*        m;
{

        FILE  *ms,  *hdg,*ao,*dest,*fin;
        char   *text,*text_body,*text_hdng,*texte,*t1,*t2,*t3;
        char   *ajout="@seclib.info.fundp.rtt.be";
        char  alias_pos[40];
        ENODE** end_ptr;
        int  l,nb,i;
        char  line[80],adresse[80],inter[80],der[80],der2[80];
        ENODE   *e,*f,*rhdn,*hdng,*part;
        char  c,car;

        e = (ENODE*)NULL;
        hdng =(ENODE*)NULL;
        part = (ENODE*)NULL;
        f = (ENODE*)NULL;
        end_ptr=NULL;
        umask(000);
        rhdn=P2_find_hdng(m->content,P2_RECIPS);
        text=xe_stru(rhdn);
        i=0;
        while  (*text)
                {
                        if  (((unsigned  long)(*text)>=32)&&((unsigned
                        long)(*text)<=126))
                                {
                                        der[i]=*text;
```

```
                        i++;
                    }
                *text++;
        }
    der[i]='\0';
    t1=&der[0];
    while  (car!='o')
        {
            car=*t1;
            *t1++;
        }
    nb=0;
    while  (strlen(t1)>6)
        {
            inter[nb]=*t1;
            *t1++;
            nb++;
        }
    inter[nb]='\0';
    t2=&inter[0];
    i=0;
    while  (*t2)
        {
            if  (((unsigned  long)(*t2)>=32)&&((unsigned  long)(*t2)<=126))
                {
                    der2[i]=*t2;
                    i++;
                }
            *t2++;
        }
    der2[i]='\0';
    t3=&der2[0];
    if  (strcmp(t3,"postmaster")==0)
        {
            alias_pos[0]='e';
            alias_pos[1]='a';
            alias_pos[2]='n';
            alias_pos[3]='\0';
            texte=strcat(alias_pos,ajout);
        }
    else
        texte=strcat(t3,ajout);
    ao=fopen("RECEV","w");
    fprintf(ao,"%s",texte);
    fclose(ao);
    if ( (e=m->content) && (e=e->constructor) && (e=e->next))
        {
            ms=fopen("message2",  "wb");
            for ( part=e->constructor;  part;  part=part->next  )
                {
                    text_body=xe_stru(part);
                    fprintf(ms,"%s",text_body);
                }
            fclose(ms);
        }

    EAN_sserv_cont_rec();

    unlink("RECEV");
}
```

Table des matières