Le format OpenPGP

Traduit par :

Sébastien Person

personseb@yahoo.fr

Matthieu Hautreux

matthieu.hautreux@insa-rouen.fr

Odile Weyckmans

odile.weyckmans@insa-rouen.fr

Relu et maintenu par :

Yvon Benoist

benoist@insa-rouen.fr

Le format OpenPGP: Traduit par :
par Sébastien Person, Matthieu Hautreux, Odile Weyckmans, **: Relu et maintenu par :** et Yvon Benoist

Ce document est gardé à jour afin de publier toutes les informations nécessaires pour développer des applications utilisant le format OpenPGP. Ce n'est pas une recette, étape par étape pour écrire une application. Il décrit seulement le format et les méthodes nécessaires pour lire, vérifier, générer, et écrire des paquets conformes pour circuler dans tout réseau. Il ne traite pas des questions de stockage et d'implémentation. Par contre, il met en évidence les solutions nécessaires pour éviter les failles de sécurité.

Le logiciel OpenPGP utilise une combinaison de clés publiques très solides et une cryptographie symétrique afin de fournir des services de sécurité pour les communications électroniques et le stockage des données. Ces services couvrent la confidentialité, la gestion de clés, l'authentification, et les signatures numériques. Ce document spécifie les formats des messages utilisés dans OpenPGP.

Table des matières

Liste des tableaux

Statut de ce mémo

Ce document décrit un protocole de piste des standards internet pour la communauté internet, et compte sur des discussions et des suggestions pour l'améliorer. Veuillez vous référer à l'édition actuelle de "Internet Official Protocol Standars" (STD 1) pour le niveau de standardisation et le statut de ce protocole. La distribution de ce mémo n'est pas limitée.

Note IESG

Ce document définit de nombreuses valeurs de tag, mais il ne décrit pas le mécanisme pour ajouter de nouveaux tags (pour de nouvelles fonctionnalités). D'habitude, l'Internet Assigned Number Authority (IANA) gère l'attribution de nouvelles valeurs pour de futurs développements et les RFC définissent la procédure que doit appliquer l'IANA. Cependant, il y a une subtile (mais pas si subtile) interaction dans ce protocole entre les nouvelles fonctions et celles existantes qui mènent à une réduction significative de la sécurité dans son ensemble. C'est pourquoi ce document ne définit pas une procédure d'extension. Par contre, les demandes pour définir de nouvelles valeurs de tags (pour de nouveaux algorithmes de cryptage par exemple) seront transmises aux IESG Security Aera Directors ou au groupe de travail IETF approprié, pour être prises en compte.

Chapitre 1. Introduction

Ce document fournit des informations sur les formats de paquets d'échange de messages utilisés par OpenPGP pour fournir les fonctions de cryptage, décryptage, signature, et de gestion de clés. Son élaboration s'appuie sur les bases fournies dans la RFC 1991 PGP Message Exchange Formats.

1.1. Terminologie

OpenPGP

C'est une définition pour un logiciel de sécurité qui utilise PGP 5.x comme base.

PGP

Pretty Good Privacy. PGP est une famille de systèmes logiciels développée par Philip R. Zimmermann sur laquelle OpenPGP s'est basée.

PGP 2.6.x

Cette version de PGP a beaucoup de variantes d'où l'utilisation du terme PGP 2.6.x. Elle utilise seulement RSA, MD5 et IDEA pour ces opérations de cryptographie. Une RFC informative, RFC n° 1991, a été écrite pour décrire cette version de PGP.

PGP 5.x

Cette version de PGP est initialement connue sous le nom de "PGP 3" au sein de la communauté ainsi que dans le document qui le précède : la RFC 1991. Elle a de nouveaux formats et corrige un certain nombre de problèmes de conception de PGP 2.6.x. On la nomme ici PGP 5.x car ce logiciel a été la première version du code initial de PGP 3.

« This document uses the terms ‹ MUST› ‹ SHOULD› and ‹ MAY› as defined in RFC 2119, along with the negated forms of those terms »

Ce document utilise les termes « MUST », « SHOULD » et « MAY » comme définis dans la RFC 2119, avec les formes négatives de ces termes.

NDT : Dans le document original les termes « MUST », « SHOULD » et « MAY » sont utilisés. Dans la traduction, on utilisera DOIT, DEVRAIT, PEUT. Il faut noter que ces termes indiquent un degré de nécessité de ce qui les précède. Ils ont été cités par ordre d'importance décroissant.

Chapitre 2. Fonctions générales

OpenPGP garantit des fonctionnalités d'intégrité de données pour les messages et les fichiers de données en utilisant les méthodes fondamentales suivantes :

- signature numérique
- cryptage
- compression
- conversion radix-64

En plus, OpenPGP permet de gérer un trousseau de clés et des certificats, mais cela dépasse l'objet de ce document.

2.1. La confidentialité grâce au cryptage

OpenPGP utilise deux méthodes de cryptage pour garantir la confidentialité : le cryptage à clé symétrique et le cryptage à clé publique. Pour le cryptage par clés publiques, cela est réalisé en utilisant un algorithme symétrique de cryptage. Chaque clé symétrique est utilisée seulement une fois. Une nouvelle clé, propre à la session (« session key ») est générée comme un nombre aléatoire à chaque message. Comme elle n'est utilisée qu'une seule fois, la clé est associée au message et transmise avec. Pour protéger la clé, celle-ci est cryptée avec la clé publique du destinataire. La séquence est la suivante :

1. L'émetteur crée un message.
2. OpenPGP (côté émission), génère un nombre aléatoire qui sera utilisé comme clé de session, pour ce message seulement.
3. La clé de session est cryptée en utilisant la/les clé(s) publique(s) de chaque destinataire. Le message commence par cette clé cryptée.
4. OpenPGP (côté émission), crypte le message en utilisant la clé de session qui forme le reste du message. Il est à noter qu'il est d'usage de compresser aussi le message.
5. OpenPGP (côté réception), décrypte la clé de session en utilisant la clé privée du destinataire.
6. OpenPGP (côté réception), décrypte le message en utilisant la clé de session. Si le message était compressé, il sera décompressé.

Avec le cryptage symétrique, un objet doit être crypté avec une clé symétrique dérivée d'une phrase secrète (ou tout autre code secret partagé), ou avec un mécanisme à deux niveaux, similaire à la méthode de la clé publique décrite ci-dessus, dans laquelle une clé de session est elle-même cryptée avec un algorithme symétrique protégé par un code secret.

La signature numérique, les services de confidentialité peuvent être appliqués tous les deux au même message. Premièrement, une signature est générée pour le message et y est attachée. Ensuite, le message et la signature sont cryptés à l'aide d'une clé de session symétrique. Enfin, la clé de session est cryptée grâce à la clé publique de cryptage et le tout est positionné devant le bloc crypté.

2.2. Authentification grâce à la signature numérique

La signature numérique utilise le procédé de hachage de code ou un algorithme de réduction du message et un algorithme de signature par clé publique. La séquence est la suivante :

1. L'émetteur crée le message.
2. Le logiciel d'envoi génère un code haché du message.
3. Le logiciel d'envoi utilise ce code haché ainsi que la clé privée de l'émetteur pour générer une signature.
4. La signature binaire est jointe au message.
5. Le logiciel de réception garde une copie de la signature du message.
6. Le logiciel de réception génère un nouveau code haché pour le message reçu et le vérifie en utilisant la signature du message. Si l'authentification est réussie, le message est considéré comme authentique.

2.3. Compression

L'implémentation d'OpenPGP PEUT compresser le message après avoir appliqué la signature, mais doit le faire avant le cryptage.

2.4. Conversion Radix-64

La représentation spécifique sous-jacente pour les messages cryptés, les certificats de signatures, et les clés sont une suite d'octets arbitraires. Certains systèmes permettent seulement l'utilisation de textes imprimables sous forme de blocs de 7 bits. Pour transporter les octets binaires bruts spécifiques d'OpenPGP par des canaux non sûrs pour des données binaires brutes, un codage imprimable de ces octets binaires est nécessaire. OpenPGP offre la possibilité de convertir des chaînes d'octets binaires bruts de 8 bits en une chaîne de caractères ASCII imprimables, nommée codage Radix-64 ou ASCII Armor.

Les implémentations DEVRAIENT fournir des conversions Radix-64.

Il est utile de rappeler que beaucoup d'applications, plus particulièrement celles de messagerie, demanderont plus d'options évoluées comme cela est décrit dans le document OpenPGP-MIME, RFC 2015. Une application qui implémente OpenPGP DEVRAIT implémenter OpenPGP-MIME.

2.5. Application exclusive de signature

OpenPGP est conçu pour des applications qui utilisent à la fois le cryptage et la signature, mais il y a un certain nombre de problèmes qui peuvent être résolus par une implémentation exclusive de la signature. Bien que cette spécification traite à la fois du cryptage et de la signature, il est raisonnable qu'il y ait des implémentations de sous-ensembles non caractéristiques. Seulement dans le sens où elle laisse de côté le cryptage.

Chapitre 3. Formats des éléments de données

Cette section décrit les éléments de données utilisés par OpenPGP.

3.1. Nombres scalaires

Les nombres scalaires ne sont pas signés, et sont toujours stockés dans le format big-endian. En utilisant n[k] pour désigner le k ième octet à interpréter, la valeur d'un scalaire de deux octets est ((n[0] << 8) + n[1];). La valeur d'un scalaire codé sur 4 octets est ((n[0]; << 24) + (n[1]; << 16) + (n[2]; << 8) + n[3];).

3.2. Entiers multiprécisions

Les entiers à multiprécisions (EPM) sont des entiers non signés utilisés pour stocker de longs entiers comme ceux utilisés dans les calculs cryptographiques.

Un EPM est constitué de deux éléments : un scalaire sur 2 octets qui représente la longueur en bits de l'EPM, suivi par une chaîne d'octets qui contient l'entier en question.

Ces octets forment un nombre big-endian ; un nombre big-endian peut être transformé en un EPM en lui donnant un préfixe de la largeur appropriée.

Exemples :

(tous les nombres sont représentés en hexadécimal)

La suite d'octets [00 01 01]; code un EPM de valeur 1. La chaîne [00 09 01 FF]; code l'EPM de valeur 511

Règles supplémentaires:

La taille d'un EPM est ((EPM.longueur + 7) / 8) + 2 octets.

Le champ de longueur d'un EPM décrit la longueur de celui-ci à partir du bit non nul le plus signifiant. C'est-à-dire que l'EPM [00 02 01] n'est pas formé correctement. Il aurait dû s'écrire [00 01 01].

3.3. Les Key ID (Identificateurs de clés)

Un Key ID est un scalaire de 8 octets qui identifie une clé.

Les implémentations NE DOIVENT PAS considérer que les Key IDs sont uniques. La section « Amélioration des formats de clés » ci-dessous décrit la structure des Key ID.

3.4. Texte

Par défaut, on utilise le format de caractère UTF-8 [RFC2279] qui est l'encodage du code Unicode [ISO10646].

3.5. Champ de temps

Un champ de temps est un nombre non signé sur 4 octets qui contient le nombre de secondes écoulées depuis le 1er janvier 1970 à minuit (UTC).

3.6. Identificateur string-to-key (S2K)

Les indicateurs de string-to-key (S2K) sont utilisés pour convertir des phrases secrètes en clés symétriques de cryptage et de décryptage. Elles sont utilisées actuellement dans deux cas, pour crypter la partie secrète des clés privées dans le trousseau de clés privées, et pour convertir des phrases secrètes en clés de cryptage pour les messages cryptés symétriquement.

3.6.1. Type d'indentificateur string-to-key (S2K)

Il y a trois types d'identificateurs de S2K actuellement compatibles, comme décrit ci-dessous :

3.6.1.1. S2K Simple

Cette méthode effectue un hachage direct de la chaîne pour créer la clé de donnée. Voir ci-dessous comment le hachage est effectué.

```
Octet 0: 0x00
Octet 1: algorithme de hachage
```

La méthode Simple S2K hache la phrase secrète pour créer la clé de session. La manière dont cela est réalisé dépend de la taille de la clé de session (qui dépend elle-même du chiffrage utilisé) et de la taille générée par l'algorithme de hachage. Si la taille du hachage est plus grande ou égale à celle de la clé de session, les octets de poids forts (ceux le plus à gauche) du hachage sont utilisés comme clé.

Si la taille de hachage est moins importante que celle de la clé, plusieurs instances du contexte de hachage sont créées -- suffisamment pour produire la clé de donnée requise. Ces instances sont pré-chargées avec 0, 1, 2 ... octets de zéros (ce qui revient à dire que la première instance n'a pas de pré-chargement, la seconde est pré-chargée avec un octet de zéros, la troisième avec deux octets de zéros, et ainsi de suite).

Une fois qu'une donnée a été hachée, elle est passée indépendamment dans chaque contexte de hachage. Comme les contextes ont été initialisés différemment, ils vont produirent chacun un code haché différent. Une fois que la phrase secrète est hachée, les données issues des différents contextes de hachage sont concaténées, d'abord le hachage le plus à gauche, pour produire la clé de donnée, avec suppression des éventuels octets en trop sur la droite.

3.6.1.2. S2K compliqué (salé)

Cette méthode inclut une donnée "salée" dans l'identificateur S2K -- une donnée arbitraire -- qui est hachée avec la phrase secrète, pour empêcher les attaques basées sur les mots du dictionnaire.

```
Octet 0: 0x01
Octet 1: algorithme de hachage
Octet 2-9: valeur salée sur 8 octets
```

Le S2K salé est exactement comme le S2K simple, à part que l'entrée de(s) fonction(s) de hachage est constituée des 8 octets salés provenant de l'identificateur de S2K, suivie de la phrase secrète.

3.6.1.3. S2K itératif et salé

Cette méthode inclut à la fois une valeur salée et un octet de comptage. La valeur salée est combinée à la phrase secrète, et la valeur résultante est hachée de manière répétitive. Cela augmente la quantité de travail nécessaire pour effectuer une attaque à base des mots du dictionnaire.

```
Octet 0: 0x03
Octet 1: algorithme de hachage
Octet 2-9: valeur salée sur 8 octets
Octet 10: compte, valeur codée sur un octet
```

Le compte est codé sur un octet en utilisant la formule suivante :

```
#define EXPBIAS 6
compte = ((Int32)16 + (c & 15)) «  ((c  » 4) + EXPBIAS);
```

La formule ci-dessus est écrite en C, où « Int32 » est un type pour un entier codé sur 32 bits, et la variable « c » est le compte codé : l'octet 10.

Les S2K salés et itératifs hachent la phrase secrète et la valeur salée plusieurs fois. Le nombre total d'octets à être hachés est indiqué dans le compte codé du spécifieur S2K. Il faut bien voir que la valeur de comptage résultante est un compte d'octets du nombre d'octets à être hachés, et non un comptage itératif.

Au départ, un ou plusieurs contextes de hachages sont configurés comme dans les autres algorithmes S2K, suivant le nombre d'octets que la clé de donnée a besoin. Puis la valeur salée, suivie par la phrase secrète sont hachées de manière répétitive jusqu'à ce que le nombre d'octets, spécifié par l'octet de compte, ait été haché. La seule exception est que si, l'octet de compte est moins important que la taille de la valeur salée additionnée à la phrase secrète, ces deux derniers sont hachés même si la taille est plus grande que l'octet de compte.

Après le hachage, la donnée est sortie du(des) contexte(s) de hachage comme dans les autres algorithmes S2K.

3.6.2. Utilisation des S2K

Les implémentations DEVRAIENT (SHOULD) utiliser les spécifieurs de S2K salés ou salés et itératifs, car les spécifieurs S2K simples résistent moins bien aux attaques de type dictionnaire.

3.6.2.1. clé secrète de cryptage

Un spécifieur S2K peut être enregistré dans le trousseau de clés secrètes pour spécifier la façon de convertir la phrase secrète, en une clé qui débloque la donnée secrète. Les anciennes versions de PGP se contentaient de garder en mémoire un algorithme de codage d'octets précédant la donnée secrète ou 0 pour indiquer que les données secrètes n'étaient pas en-cryptées. Le hachage MD5 était toujours utilisé pour convertir la phrase secrète en une clé pour l'algorithme de codage spécifique.

Pour une question de compatibilité, quand un spécifieur S2K est utilisé, la valeur spéciale 255 est enregistrée à la position où l'octet d'algorithme de hachage aurait été dans l'ancienne structure de données. Cela est immédiatement suivi par un simple octet d'identification de l'algorithme, puis par le spécifieur S2K codé comme décrit ci-dessus.

Cependant, avant les données secrètes, on peut se trouver face à plusieurs possibilitées :

```
0: les données secrètes ne sont pas cryptées
```

3.6.2.2. clé symétrique de cryptage de message

OpenPGP peut créer un paquet contenant une clé de cryptage de session symétrique (ESK) au début du paquet. Cela est utilisé pour autoriser les identifieurs S2K à être utilisés pour la conversion de la phrase secrète ou pour créer des messages qui contiennent un mélange de clés symétriques ESK et de clés publiques ESK. Cela permet d'avoir des messages décryptables à la fois par une clé publique ou une phrase secrète.

PGP 2.x a toujours utilisé IDEA avec une conversion S2K lors de l'encryptage d'un message avec un algorithme symétrique. Cela n'est pas préférable, mais DOIT être utilisé pour garder la compatibilité avec les anciennes versions.

Chapitre 4. Syntaxe des paquets

Cette section décrit les paquets utilisés par OpenPGP.

4.1. Vue générale

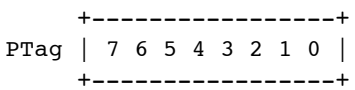
Un message OpenPGP est construit à l'aide d'un nombre d'informations qu'on appelle traditionnellement des paquets. Un paquet est une quantité de données qui possède une signature précisant son sens. Un message OpenPGP, un trousseau de clés, un certificat, etc. sont constitués d'un certain nombre de paquets. Quelques uns de ces paquets peuvent contenir d'autres paquets OpenPGP (par exemple, un paquet de données compressées, quand il est décompressé, contient des paquets OpenPGP).

Chaque paquet est constitué d'un en-tête de paquet, suivi par le corps du paquet. L'en-tête du paquet est de longueur variable.

4.2. En-tête des paquets

Le premier octet d'un en-tête de paquet s'appelle « signature du paquet ». Il détermine le format de l'en-tête et informe sur le contenu du paquet. Le reste de l'en-tête détermine la taille du paquet.

Il faut noter que le bit le plus important est celui le plus à gauche, il est appelé bit 7. Un masque pour ce bit est 0x80 en hexadécimal.



Le bit 7 vaut toujours 1, le bit 6 indique le nouveau format du paquet s'il est positionné.

PGP 2.6.x utilise seulement les paquets dans l'ancien format. Ainsi, les logiciels qui doivent utiliser cette version de PGP ne doivent utiliser que les paquets d'ancien format. Si la compatibilité n'est pas une question importante, l'un ou l'autre format peut être utilisé. Remarquez que les paquets dans l'ancien format possèdent 4 bits de signature du contenu, et que les paquets dans le nouveau format en possèdent 6 ; certaines possibilités ne peuvent pas être utilisées et restent tout de même compatibles.

Paquets dans l'ancien format :

```
      bits 5-2 -- signature de contenu
   bits 1-0 - type de longueur
```

Les paquets dans le nouveau format contiennent :

```
      Bits 5-0 -- signature du contenu
```

4.2.1. Longueur des paquets de l'ancien format

Signification du type de longueur dans les paquets en ancien format.

0

Le paquet a une longueur de 1 octet. L'en-tête fait 2 octets.

1

Le paquet a une longueur de 2 octets. L'en-tête fait 3 octets.

2

Le paquet a une longueur de 4 octets. L'en-tête fait 5 octets.

3

Le paquet est d'une taille indéterminée. L'en-tête fait 1 octet, et la longueur est à déterminer à l'implémentation. Si le paquet est dans un fichier, cela signifie que le paquet s'étend jusqu'à la fin du fichier. En général, l'implémentation NE DOIT PAS utiliser des paquets de tailles indéterminées sauf si la fin des données est proche d'après le contexte, et même dans ce cas il est mieux d'utiliser une taille précise, ou un en-tête dans le nouveau format. L'en-tête du nouveau format décrit ci-dessous possède un mécanisme pour encoder avec précision des données de tailles inconnues.

4.2.2. Longueur des paquets du nouveau format

Les paquets du nouveau format présentent 4 possibilités pour coder la longueur.

1

Un en-tête d'un octet autorise des longueurs de paquets de 191 octets maximum.

2

Un en-tête de deux octets permet d'encoder des paquets de tailles comprises entre 192 et 8383 octets.

3

Un en-tête de 5 octets peut encoder des paquets qui peuvent aller jusqu'à 4,294,967,295 (0xFFFFFFFF) octets en longueur. (Ce qui revient à coder un nombre scalaire de 4 octets).

4

Quand la longueur d'un paquet n'est pas connue à l'avance par celui qui en a besoin, les en-têtes qui renseignent sur des longueurs de corps partiel, codent un paquet de taille indéterminée, en le transformant en une chaîne.

4.2.2.1. Un octet de longueur

Un en-tête d'un octet code une longueur de 0 à 191 octets. Ce type de longueur d'en-tête est reconnu, car la valeur de l'octet est inférieure à 192. La longueur du corps vaut :

```
LongueurCorps = 1er_octet;
```

4.2.2.2. Deux octets de longueur

Un en-tête de deux octets code une longueur comprise entre 192 et 8383 octets. On le reconnaît grâce au premier octet qui est compris entre 192 et 223. La longueur du corps est égale à:

```
LongueurCorps = ((1er_octet - 192) << 8)+ (2eme_octet) + 192
```

4.2.2.3. Cinq octets de longueur

Un en-tête d'une longueur de 5 octets consiste en un simple octet qui vaut 255, suivi par un scalaire de 4 octets. La longueur du corps est égale à :

```
LongueurCorps = (2eme_octet << 24) | (3eme_octet << 16) |
      (4eme_octet << 8)  | 5eme_octet
```

4.2.2.4. Longueur d'un segment de corps

Un en-tête de longueur partielle est mis sur un octet et code seulement la longueur de la partie donnée du paquet. Cette longueur est une puissance de 2, comprise entre 1 et 1 073 741 824 (2 puissance 30). Cet octet est reconnu par le fait qu'il est supérieur ou égal à 224, et inférieur à 255. La longueur partielle est égale à :

```
LongueurPartielleCorps = 1 << (1er_octet & 0x1F);
```

Chaque longueur partielle est suivie par une portion de données du paquet. L'en-tête partiel précise la longueur de la portion. Un autre en-tête de longueur (du type --- un octet, deux octets, ou partiel) suit cette portion. Le dernier en-tête partiel du paquet NE DOIT PAS ÊTRE un en-tête de longueur partiel. Les en-têtes de longueur partielle doivent seulement être utilisés pour les parties non finales du paquet.

4.2.3. Exemple de taille de paquet

Ces exemples montrent l'encodage de la taille des paquets suivant le nouveau format de paquet.

Un paquet avec une taille de 100 peut avoir sa longueur codée sur un octet : 0x64. Cet octet est suivi de 100 autres de données.

Un paquet d'une longueur de 1723 peut avoir une longueur codée sur deux octets : 0xC5, 0xFB. Cet en-tête par exemple est suivi par 1723 octets de données.

Un paquet d'une longueur de 100000 peut avoir sa longueur codée sur 5 octets : 0xFF, 0x00, 0x01, 0x86,0xA0.

Il pourrait aussi être encodé dans le flux d'octets suivants : 0xEF, d'abord 32768 octets de données; 0xE1, suivi de deux octets de données; 0XE0, suivi d'un octet de donnée; 0XF0, suivi de 65536 octets de données; 0xC5, 0xDD, suivi des derniers 1693 octets de données. Ceci n'est qu'un des codages possibles, et beaucoup de

variations sont possibles sur la taille des corps intermédiaires d'en-têtes, du moment qu'une en-tête de longueur de corps normal code la dernière portion des données. Il faut aussi savoir que le dernier en-tête de longueur de corps peut être un en-tête de taille nulle.

Une implémentation PEUT utiliser des longueurs de corps de données partielles pour les paquets de données, qu'ils soient à l'état brut, compressés, ou cryptés. La première longueur partielle doit être au moins de 512 octets. Les longueurs partielles de corps NE DOIVENT être utilisées pour aucun autre type de paquets.

Remarquez que dans toutes ces explications, la longueur totale du paquet est la longueur de l'en-tête plus celle du corps.

4.3. Marqueurs de paquet

La signature de paquet informe sur le type du contenu du corps. Remarquez que pour les paquets d'ancien format, la signature est forcément inférieure à 16, tandis que les paquets de nouveau format d'en-tête peuvent avoir des signatures allant jusqu'à 63. Les marqueurs définis sont les suivants (en décimales) :

- 0 -- réservé - une signature de paquet ne doit pas avoir cette valeur
- 1 -- paquet de clé pour session cryptée par clé publique
- 2 -- signature de paquet
- 3 -- paquet de clé pour session cryptée par clé symétrique
- 4 -- paquet de signature pour un passage
- 5 -- paquet de clé secrète
- 6 -- paquet de clé publique
- 7 -- paquet de sous clé secrète
- 8 -- paquet de données compressées
- 9 -- paquet de données cryptées symétriquement
- 10 -- paquet de marqueur
- 11 -- paquet de données à l'état brut
- 12 -- paquet de confiance
- 13 -- paquet d'identifiant utilisateur
- 14 -- paquet de sous-clé publique
- 60 à 63 -- valeurs privées ou expérimentales

Chapitre 5. Types de paquets

5.1. Paquet de clé de session cryptée par clé publique (marqueur 1)

Un paquet de clé de session crypté par clé publique contient la clé de session utilisée pour crypter le message. Il peut y avoir ou non un ou plusieurs paquets de clés de sessions cryptées (clé publique ou clé symétrique) précédant un paquet de données cryptées symétriquement contenant un message crypté. Le message est crypté avec la clé de session, et la clé de session est elle-même cryptée et enregistrée dans le(s) paquet(s) de clés de sessions cryptées. Le paquet de données cryptées symétriquement est précédé par un paquet de clé de session cryptée par clé publique pour chaque clé OpenPGP avec laquelle le message est crypté. Le receveur du message trouve une clé de session qui est cryptée avec sa clé publique, décrypte la clé de session et l'utilise pour décrypter le message.

Le corps de ce paquet est constitué de :

- Un nombre d'un octet donnant la version du type du paquet. La valeur définie courante pour la version de paquet est 3. Une implémentation devrait accepter, mais non générer une version 2 qui est, pour le reste équivalente à la version 3.
- Un nombre de huit octets qui fournit l'identificateur de la clé publique avec laquelle la clé de session a été crypté.
- Un nombre d'un octet donnant l'algorithme de clé publique utilisée.
- Une chaîne d'octets qui est la clé de session cryptée. Cette chaîne prend en charge le reste du paquet et son contenu dépend de l'algorithme de clé publique utilisée.

Champs spécifiques à l'algorithme pour le cryptage RSA :

- entier multi-précision (MPI) de valeur cryptée RSA m**e mod n.

Champs spécifiques à l'algorithme pour le cryptage Elgamal :

- MPI de valeur Elgamal (Diffie-Hellman) g**k mod p
- MPI de valeur Elgamal (Diffie-Hellman) m * y**k mod p

La valeur « m » dans les formules ci-dessus est obtenue à partir de la clé de session comme décrit ci-après. Dans un premier temps la clé de session est préfixée avec un identifiant d'algorithme d'un octet qui précise l'algorithme de cryptage symétrique utilisé pour crypter le paquet de données cryptées symétriquement qui suit. Ensuite, une somme de contrôle [checksum] de deux octets est rajoutée, checksum qui est égal à la somme des octets de la clé de session précédente, sans inclure l'identifiant d'algorithme, le tout modulo 65536. Cette valeur est ensuite étoffée comme décrit dans PKCS-1 block type 02 [RFC2313] pour former la valeur « m » utilisée dans les formules ci-dessus.

Remarquez que lorsqu'une implémentation forme plusieurs PKESK avec une clé de session, formant un message qui peut être décrypté par plusieurs clés, l'implémentation DOIT faire à nouveau un étoffage PKCS-1 pour chaque clé.

Une implémentation PEUT accepter ou utiliser zéro comme identificateur de clé « joker » ou « supposé ». Dans ce cas, l'implémentation réceptrice essayera toutes les clés privées disponibles, en cherchant une clé de session décryptée valide. Ce format aide à réduire le trafic des analyses de messages.

5.2. Signature de paquets (marqueur 2)

Un paquet de signature décrit les liens entre une clé publique et des données. Les signatures les plus communes sont les signatures de fichier ou d'un bloc de texte, et les signatures certifiant l'identité de l'utilisateur [user ID].

Deux versions de signatures de paquets sont définies. La version 3 fournit des informations de base sur la signature, tandis que la version 4 fournit un format extensible avec des sous-paquets qui peuvent donner plus d'informations au sujet de la signature. PGP 2.6.x accepte seulement les signatures de version 3.

Toutes les implémentations DOIVENT accepter les signatures de version 3. Les implémentations DEVRAIENT pouvoir générer des signatures de version 4. Les implémentations PEUVENT générer une signature de version 3 qui est en mesure d'être vérifiée par PGP 2.6.x.

Remarquons que si une implémentation crée un message crypté et signé, crypté avec une clé V3, il est raisonnable de créer une signature V3.

5.2.1. Types de signatures

Une signature peut avoir plusieurs sens; ce sens est défini dans l'octet du type de la signature et ce dans n'importe quelle signature. Les sens possibles sont les suivants :

0x00

Signature d'un document binaire. La caractéristique de cette signature est qu'elle signifie que le document appartient au signataire, que ce dernier l'a créée ou qu'il certifie qu'il n'a pas été modifié.

0X01

Signature d'un texte de base. La caractéristique de cette signature est qu'elle signifie que le document appartient au signataire, que ce dernier l'a créé ou qu'il certifie qu'il n'a pas été modifié. La signature est calculée sur les données du texte avec conversion en < CR > < LF > des fins de lignes et suppression des blancs de fin de mots.

0x02

Signature autonome. Cette signature signe seulement le contenu de son sous-paquet. Elle est calculée de la même manière qu'une signature pour un document binaire de longueur nulle. Remarquez qu'il est absurde d'avoir une signature autonome de version 3.

0x10

Certification générique d'un identifiant utilisateur et d'un paquet de clé publique. L'utilisateur de cette certification ne donne aucune assurance particulière qu'une vérification sérieuse a été faite pour voir si le propriétaire de la clé est bien la personne décrite dans l'identifiant utilisateur [user ID]. Remarquez que toutes « les signatures de clés » PGP possèdent ce type de certification.

0x11

Certification casuelle d'un ID utilisateur et d'un paquet de clé publique. L'utilisateur de cette certification n'a effectué aucune vérification de l'affirmation donnant le possesseur de cette clé comme l'ID utilisateur spécifié.

0x12

Certification casuelle de l'ID utilisateur et du paquet de clé publique. L'utilisateur de cette certification a effectué quelques vérifications casuelle de la déclaration d'identité.

0x13

Certification positive de l'ID utilisateur et du paquet de clés publiques. L'utilisateur de cette certification a effectué des vérifications substantielles de la déclaration d'identité.

0x18 : Signature correspondant à la sous-clé

La signature est un signalement fait par la clé de haut niveau signataire indiquant que la sous-clé lui appartient. Cette signature est calculée directement sur la sous clé elle-même, et non sur aucun autre ID utilisateur ou autre sous-paquets.

0x1F : Signature directe sur une clé

Cette signature est calculée directement sur une clé. Elle fait correspondre les informations contenues dans le sous-paquets de clé avec la clé, et elle s'utilise de manière appropriée pour les sous-paquets qui fournissent des informations à propos de la clé, comme les sous-paquets de clé de révocation. Son utilisation est également appropriée pour les statuts que les certificateurs veulent effectuer au sujet de la clé elle-même, au lieu de la correspondance entre une clé et un nom.

0x20 : Signature de révocation de clé

La signature est calculée directement sur la clé en train d'être révoquée. Une clé révoquée n'est pas supposée être utilisée. Seules les signatures de révocations de la clé en train d'être révoquée, ou pour une révocation de clé autorisée, peuvent être considérées comme des signatures de révocation valides.

0x28 : Signature de révocation de sous-clé

La signature est calculée directement sur la sous-clé en cours de révocation. Une sous-clé révoquée ne doit pas être utilisée. Seules les signatures de révocation par les clés de signatures de haut niveau qui sont attachées à cette sous-clé, ou par une clé de révocation autorisée, doivent être considérées comme des signatures de révocation valide.

0x30 : Signature de révocation de sous-clé

La signature révoque une signature de certification d'ID utilisateur précédente (les signatures de la classe 0x10 à 0X13). Elle doit être issue de la même clé que celle qui est issue de la signature de révocation ou d'une clé de révocation autorisée. La signature doit avoir une date de création plus ancienne que celle de la signature qu'elle révoque.

0x40 : Signature de temps

Cette signature n'a de sens que pour la référence de temps qu'elle contient.

5.2.2. Format d'un paquet de signature version 3

Le corps d'un paquet de signature version 3 contient :

- Un numéro de version sur un octet (3)
- Un octet de longueur pour les éléments hachés décrit ci-dessous. DOIT être 5.
  - Un type de signature sur un octet.
  - Un temps de création sur quatre octets.

- huit octets identificateur de clé du signataire.
- Un octet pour l'algorithme de clé publique.
- Un octet pour l'algorithme de hachage.
- Un champs de deux octets contenant les 16 bits de gauche de la valeur hachée et signée.
- Un ou plusieurs entiers multi-précision comprenant la signature. Cette portion est spécifique à l'algorithme comme décrit ci-dessous.

  La donnée à signer est hachée, puis le type de signature et la date de création à partir du paquet de signature sont hachés (5 octets supplémentaires). On utilise la valeur hachée qui en résulte dans l'algorithme de signature. Les 16 bits de poids forts (deux premiers octets) du hachage sont inclus dans le paquet de signature pour fournir un test rapide pour rejeter les signatures invalides.

Champs spécifiques à l'algorithme pour les signatures de type RSA :

- entier multi-précision (MPI) ou valeur de signature RSA m**d.

Champs spécifiques à l'algorithme pour les signatures de type DSA :

- MPI de valeur DSA r.
- MPI de valeur DSA s.

Le calcul de la signature est basé sur un hachage des données signées, comme décrit ci-dessous. Les détails des calculs pour les signatures DSA sont différents de ceux des RSA.

Pour les signatures RSA, la valeur hachée est codée comme décrit dans la section 10.1.2 de PKCS-1 nommée « Codage des données », en produisant une valeur ASN.1 de type DigestInfo, et ensuite étoffée en utilisant des blocs PKCS-1 de type 01 [RFC2313]. Cela nécessite l'insertion de la valeur hachée comme chaîne d'octets dans une structure ASN.1. L'identifiant d'objet pour le type de hachage utilisé est inclus dans la structure. Les représentations hexadécimales pour les algorithmes de hachage définis ici sont :

- MD2: 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x02, 0x02
- MD5: 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x02, 0x05
- RIPEMD-160: 0x2B, 0x24, 0x03, 0x02, 0x01
- SHA-1: 0x2B, 0x0E, 0x03, 0x02, 0x1A

Les OID ASN.1 sont :

- MD2: 1.2.840.113549.2.2
- MD5: 1.2.840.113549.2.5
- RIPEMD-160: 1.3.36.3.2.1
- SHA-1: 1.3.14.3.2.26
- MD5: 1.2.840.113549.2.5
- RIPEMD-160: 1.3.36.3.2.1
- SHA-1: 1.3.14.3.2.26

Les préfixes complets de hachage pour ces derniers sont :

MD2:

> 0x30, 0x20, 0x30, 0x0C, 0x06, 0x08, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x02, 0x02, 0x05, 0x00, 0x04, 0x10

MD5:

> 0x30, 0x20, 0x30, 0x0C, 0x06, 0x08, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x02, 0x05, 0x05, 0x00, 0x04, 0x10

RIPEMD-160:

> 0x30, 0x21, 0x30, 0x09, 0x06, 0x05, 0x2B, 0x24, 0x03, 0x02, 0x01, 0x05, 0x00, 0x04, 0x14

SHA-1:

> 0x30, 0x21, 0x30, 0x09, 0x06, 0x05, 0x2b, 0x0E, 0x03, 0x02, 0x1A, 0x05, 0x00, 0x04, 0x14

Les signatures DSA DOIVENT utiliser des hachages d'une taille de 160 bits, pour correspondre à q, la taille du groupe généré par la valeur de générateur de la clé DSA. Le résultat de la fonction de hachage est traitée comme un nombre de 160 bits et utilisé directement dans l'algorithme de signature DSA.

5.2.3. Format de la version 4 de la signature de paquet

Le corps d'un paquet de signature version 4 contient :

- Un numéro de version d'un octet(4).
- Un type de signature d'un octet.
- Un algorithme de clé publique sur un octet.
- Un algorithme de hachage sur un octet.
- Un compteur d'octets scalaires de deux octets pour les données de sous-paquets hachées qui suivent. Remarquez que c'est la longueur en octets de tous les sous-paquets hachés; un pointeur incrémenté par ce nombre sautera les sous-paquets hachés.
- Les données de sous-paquets hachées. (zéro sous-paquets ou plus)
- Un compteur d'octets scalaires de deux octets pour les sous-paquets de données non hachées. Remarquez que c'est la longueur en octets de tous les sous-paquets non hachés; un pointeur incrémenté par ce nombre sautera les sous-paquets non hachés.
- Données de sous-paquets non hachés. (zéros sous paquets ou plus)
- Champ de deux octets contenant les 16 bits gauche de la valeur hachée signée.
- Un ou plusieurs entiers multi-précision comprenant la signature. Cette portion est spécifique à l'algorithme, comme décrit ci-dessus.

Les données en train d'être hachées sont signées, puis les données de signature depuis le numéro de version jusqu'au sous-paquet de données hachées (inclus) sont hachées. La valeur de hachage qui en résulte est signée.

Les 16 bits de gauche du hachage sont inclus dans le paquet de signature pour fournir un test rapide pour rejeter les signatures invalides.

Il y a deux champs constitués de sous-paquets de signature. Le premier champs est haché et contient le reste des données de signature, tandis que le second n'est pas haché. Le second ensemble de sous-paquets n'est pas protégé de manière cryptographique par la signature et ne doit inclure que des renseignements.

Les algorithmes pour la conversion en signature du résultat de la fonction de hachage sont décrits un peu plus bas.

5.2.3.1. Spécifications des sous-paquets de signature

Les champs de sous-paquets consistent en aucun ou plusieurs sous-paquets de signature. Chaque ensemble de sous-paquets est précédé par un compteur scalaire de deux octets de la grandeur de l'ensemble des sous-paquets.

Chaque sous-paquet comporte un en-tête de sous-paquet et un corps. L'en-tête comporte :

- La longueur de sous-paquet (1, 2 ou 5 octets)
- Le type de sous-paquet (1 octet)

et est suivie par les données spécifiques au sous-paquet.

La longueur inclut l'octet de type mais pas cette longueur. Son format est similaire aux longueurs d'en-tête de paquet dans le « nouveau » format, mais ne peut pas avoir de longueurs de corps partielles. C'est à dire :

```
si le premier octet < 192, alors
longueurDeLongueur = 1
LongueurDeSousPaquet = premier_octet

si le premier octet >= 192 et < 255, alors
longueurDeLongueur = 2
LongueurDeSousPaquet = ((premier_octet - 192) << 8) + (second_octet) + 192

si le premier octet = 255, alors
longueurDeLongueur = 5
LongueurDeSousPaquet = [scalaire de quatre octets commençant au second_octet]
```

La valeur de l'octet de type de sous-paquet peut être :

- 2 = heure de la création de la signature
- 3 = heure d'expiration de la signature
- 4 = certification exportable
- 5 = signature de confiance
- 6 = expression normale
- 7 = révocable
- 9 = temps d'expiration de clé
- 10 = maintien de position pour rester compatible avec les anciennes versions
- 11 = algorithmes symétriques préférés
- 12 = clé de révocation

- 16 = Identifiant de la clé de l'expéditeur/ de l'émetteur
- 20 = données de notation
- 21 = algorithmes de hachage préférés
- 22 = algorithmes de compression préférés
- 23 = préférences du serveur de clés
- 24 = serveur de clés préféré
- 25 = identifiant utilisateur primaire
- 26 = politique de gestion des URL
- 27 = paramètres de clé
- 28 = identifiant utilisateur du signataire
- 29 = raison de la révocation
- 100 à 110 = interne ou défini par l'utilisateur

Une implémentation DOIT ignorer tous les sous-paquets de type inconnu.

Le bit 7 du type de sous-paquet est le bit « critique ». S'il est positionné, il informe que le sous-paquet fait partie de ceux qu'il est primordial pour l'évaluateur de la signature de reconnaître. Si un sous-paquet signalé comme critique se présente mais n'est pas reconnu par le programme d'évaluation, l'évaluateur DOIT considérer la signature comme erronée.

Un évaluateur peut « reconnaître » un sous paquet , mais ne pas l'implémenter. Le but du bit critique est de permettre au signataire d'avertir l'évaluateur qu'il préférerait une autre caractéristique inconnu pour générer une erreur que d'être ignorée.

Les implémentations DOIVENT implémenter les « préférences ».

5.2.3.2. Types de sous paquets de signatures

Un nombre de sous-paquets est couramment défini. Certains sous-paquets s'appliquent à la signature elle-même et d'autres sont des attributs de la clé. Les sous-paquets trouvés sur une auto-signature sont placés sur une certification d'identifiant utilisateur effectuée par la clé elle-même. Remarquez qu'une clé peut avoir plus d'un identifiant utilisateur, et dans ce cas peut avoir plus d'une auto-signature, et des sous-paquets différents.

Une auto-signature est une signature de liaison faite par la clé à laquelle la signature se réfère. Il y a trois types d'auto-signatures, les signatures de certification (types 0x10-0x13), les signatures de clés direct (type 0x1F), et les signatures de liaison de sous clé (type 0x18). Pour les auto-signatures de certification, chaque identifiant utilisateur peut posséder une auto-signature, et donc des sous-paquets différents dans ces auto-signatures. Pour les signatures de liaison de sous-clé, chaque sous-clé possède, en fait, une auto-signature. Les sous-paquets qui apparaissent dans une auto-signature de certificat s'applique au nom d'utilisateur, et les sous-paquets qui apparaissent dans l'auto-signature de la sous-clé s'applique à la sous-clé. Enfin, les sous-paquets sur la signature de clé directe s'applique à la clé entière.

Les programmes implémentés doivent interpréter les sous-paquets de préférences d'auto-signature aussi rigoureusement que possible. Par exemple, supposons qu'une clé a deux noms d'utilisateurs, Alice et Bob. Supposons qu'Alice préfère l'algorithme symétrique CAST5, et Bob préfère l'IDEA ou le Tripple-DES. Si le programme repère cette clé par le biais du nom d'Alice, dans ce cas l'algorithme préféré est CAST5, si le programme trouve la clé grâce au nom de Bob, alors l'algorithme préféré est IDEA. Si la clé est repérée par l'identifiant de clé, l'algorithme symétrique par défaut est celui de l'identifiant d'utilisateur par défaut.

On peut trouver un sous paquet aussi bien dans une section de sous-paquet haché que non haché d'une signature. Si un sous-paquet n'est pas haché, l'information contenue par ce dernier ne peut pas être considérée définitive car il ne fait pas partie de la signature elle-même.

5.2.3.3. Temps de création de la signature

(champs de temps de 4 octets)

La date à laquelle la signature à été fabriquée.

DOIT être présente dans la partie hachée.

5.2.3.4. Emetteur

(identifiant de clef de 8 octets)

L'identifiant de clef OpenPGP de la clef émettant la signature.

5.2.3.5. Date d'expiration des clés

(Champs de date de 4 octets)

La période de validité de la clef. C'est le nombre de secondes après l'heure de création de la clef après lesquelles la clef expire. Si elle n'est pas présente ou que sa valeur est zéro, la clef n'expire jamais. Ce champ n'est présent que dans les auto-signatures.

5.2.3.6. Algorithmes symétriques préférés

(Séquence de valeurs d'un octet)

Nombres d'algorithme symétriques qui indiquent quels algorithmes le détenteur de la clef préfère utiliser. Le corps de sous paquet est une liste d'octets ordonnées avec les préférés en tête de liste. On suppose que seuls les algorithmes listés sont supportés par le logiciel du destinataire. Les nombres d'algorithmes sont présentés à la section 9. Ce champs n'est présent que dans les auto-signature.

5.2.3.7. Algorithmes de hachage préférés

(tableau de valeurs d'un octet)

Nombres d'algorithme de résumé de message qui indiquent quels algorithmes le détenteur de clef préfère recevoir. Comme les algorithmes symétriques préférés, la liste est ordonnée. Les nombres d'algorithme se trouve dans la section 6. Ce champ n'est présent que dans les auto-signatures.

5.2.3.8. Algorithmes de compression préférés

(tableau de valeurs d'un octet)

Nombres d'algorithme de compression qui indiquent quel algorithme le détenteur de la clef préfère utiliser. Comme les algorithmes symétriques préférés, la liste est ordonnée. Les nombres d'algorithme se trouvent dans la section 6. Si ce sous paquet n'est pas inclus, ZIP est privilégié. Un zéro indique que des données non compressées sont préférées; il se pourrait que le logiciel du détenteur de clef n'ait pas de logiciel de compression dans cette implémentation. Ce champ n'est présent que dans les auto-signatures.

5.2.3.9. Délai d'expiration de la signature

(champs de date de 4 octets)

Période de validité de la signature. C'est le nombre de secondes après la date de création de la signature où la signature expire. Si ce champ n'est pas présent ou si sa valeur est zéro, elle n'expire jamais.

5.2.3.10. Certifications exportables

(1 octet d'exportabilité, 0 pour non, 1 pour exportable)

Ce sous paquet indique si une signature de certification est « exportable » ou non, pour être utilisé par d'autres utilisateurs que celui qui a émis la signature. Le corps de paquet contient un drapeau booléen indiquant si la signature est exportable. Si ce paquet n'est pas présent, la signature est exportable; c'est l'équivalent du drapeau contenant 1.

Les certifications non exportables, ou « local » sont des signatures faites par un utilisateur pour marquer une clef comme valide seulement au sein de l'implémentation de cet utilisateur. Ainsi, lorsqu'une implémentation prépare une copie de clef d'utilisateur pour la transporter à un autre utilisateur (c'est le processus « d'exportation » de la clef), toutes les signatures de certification local sont supprimées de la clef.

Le receveur d'une clef transportée « l'importe » et, de la même façon, supprime toutes certifications locales. Dans le déroulement normal des choses, il n'y en aura pas, en admettant que l'importation est faite sur une clef exportée. Cependant, il y a des cas ou cela peut raisonnablement se produire. Par exemple, si une implémentation autorise l'importation de clefs depuis une base de donnée de clefs en plus d'une clef exportée, dans ce cas cette situation peut se produire.

Quelques implémentations ne représentent pas l'intérêt d'un utilisateur unique (par exemple, un serveur de clefs). Ces implémentations supprimeent toujours les certifications locales de toutes les clefs qu'elles gèrent.

5.2.3.11. Révocable

(1 octet de révocabilité, 0 pour non, 1 pour révocable)

État de la révocabilité de la signature. Le corps de paquet contient un drapeau booléen indiquant si la signature est révocable. Les signatures qui ne sont pas révocables ignorent toutes les signatures de révocation ultérieures. Elles représentent un engagement du signataire comme quoi qu'il ne peut pas révoquer sa signature pour la durée de vie de sa clef. Si ce paquet n'est pas présent, la signature est révocable.

5.2.3.12. Signature de confiance

(« niveau » d'1 octet (profondeur), 1 octet de taux de confiance)

Le signeur annonce que la clef n'est pas seulement valide, mais aussi fiable, au niveau spécifié. Le niveau 0 a le même sens qu'une signature de validité ordinaire. Le niveau 1 signifie que la clef signée est certifiée, valide et fiable, avec le second octet du corps précisant le degré de confiance. Le niveau 2 signifie que la clef signée est certifiée fiable pour l'émission de signatures de confiance d'un niveau 1, cad que c'est un « meta laisser-passer ». Généralement, un niveau n de signature de confiance assure que la clef est fiable pour l'émission de signatures d'un niveau de confiance n-1. Le taux de confiance est compris entre 0 et 255, et est interprétée tel que les niveau inférieur à 120 indique une confiance partielle et les valeurs de 120 et plus indique une confiance totale. Les implémentations DEVRAIENT émettre des valeurs de 60 pour une confiance partielle et de 120 pour une confiance totale.

5.2.3.13. Expression régulière

(expression régulière terminée par un caractère nul)

Elles sont utilisées conjointement avec les paquets de signature de confiance (d'un niveau > 0) pour limiter l'étendue de confiance que l'on veut élargir. Seules les signatures des clefs cibles sur les identifiants d'utilisateur qui correspondent avec l'expression régulière dans le corps du paquet ont des confiances élargies par le sous-paquet de signature de confiance. L'expression régulière utilise la même syntaxe que le package "presque dans le domaine public" d'expression régulière d'Henry Spencer. Une description de la syntaxe se trouve dans une section plus bas.

5.2.3.14. Clé de révocation

(1 octet de classe, 1 octet d'ID d'algorithme, 20 octets d'empreintes digitales)

Autorise la clef spécifiée à utiliser les signatures de révocation pour cette clef. L'octet de classe doit avoir le bit 0x80 positionné. Si le bit 0x40 est positionné, alors cela signifie que l'information de révocation est sensible. Les autres bits sont réservés pour une utilisation future pour d'autres types d'autorisations. Ce champ n'est présent que dans les auto-signatures.

Si le drapeau « sensible » est positionné, le détenteur de clef sent que ce sous paquet contient de l'information de confiance privée qui décrit une relation sensible réelle. Si ce drapeau est positionné, les implémentations NE DOIVENT PAS exporter cette signature vers d'autres utilisateurs sauf dans les cas ou les données doivent êtres disponibles: quand la signature est en train d'être envoyée au révoqueur désigné, ou quand elle est accompagnée par une signature de révocation de ce révoqueur. Notez qu'il peut être approprié d'isoler ce sous-paquet dans une signature séparée de manière à ce qu'il ne soit pas associé a d'autres sous-paquets qui doivent être exportés.

5.2.3.15. Données de notation

(4 octets de drapeaux, 2 octets de longueur du nom (M), 2 octets de la longueur de valeur (N), M octets de donnée pour le nom, N octets de donnée de la valeur)

Ce sous-paquet décrit une « notation » sur la signature que l'émetteur souhaite faire. La notation possède un nom et une valeur, qui sont des chaînes d'octets. Il peut y avoir plus d'une notation dans une signature. Les notations peuvent être utilisés pour n'importe quelle extension l'émetteur de la signature souhaite faire. Le champ « drapeau » contient 4 octets de drapeaux.

Tous les drapeaux indéfinis DOIVENT IMPERATIVEMENT être à zéro. Les drapeaux définis sont :

Premier octet

> 0x80 = version humaine. Cette note est du texte, une note d'une personne à l'autre, et n'a pas de sens pour le logiciel.

Autres octets

> Rien.

5.2.3.16. Préférences du serveur de clé

(N octets de drapeaux)

C'est une liste de drapeaux qui indiquent les préférences que le détenteur de clés a par rapport à la manière dont la clé est prise en charge par le serveur. Tous les drapeaux indéfinis DOIVENT IMPERATIVEMENT être zero.

Premier octet

0x80 = ne modifie pas. le détenteur de clé demande que cette clé ne soit modifiée ou mise à jour que par le détenteur de clef ou un administrateur du serveur de clé.

Ce champ n'est présent que dans les auto-signatures.

5.2.3.17. Serveur de clés préféré

(chaîne de caractères)

C'est une URL du serveur de clefs que le détenteur de clefs préfère pour les mises à jour. Notez que les clefs avec plusieurs identifiants d'utilisateurs peuvent avoir un serveur de clef préféré pour chaque identifiant d'utilisateur. Notez également que comme c'est une URL, le serveur de clef peut être une copie d'une clef récupérée par ftp, http, finger, etc.

5.2.3.18. ID utilisateur primaire

(1 octet, booléen)

C'est un drapeau dans une auto signature d'identifiant utilisateur qui précise si l'identifiant utilisateur est l'identifiant utilisateur principal pour cette clef. Il est raisonnable pour une implémentation de résoudre les ambiguïtés dans les préférences, etc. pour un renvoi à l'identifiant d'utilisateur primaire. Si ce drapeau est absent, sa valeur est zéro. Si plus d'un identifiant utilisateur est marqué primaire dans une clef, l'implémentation peut résoudre l'ambiguïté de n'importe quelle façon qu'elle juge bonne.

5.2.3.19. URL énonçant les règles

(chaîne de caractères)

Ce sous-paquet contient l'URL d'un document qui décrit les règles selon lesquelles la signature a été émise.

5.2.3.20. Drapeaux de clés

(chaîne d'octets)

Ce sous-paquet contient une liste de drapeaux binaires qui contiennent l'information au sujet de la clef. C'est une chaîne d'octets, et une implémentation NE DOIT ABSOLUMENT PAS adopter une taille fixe, ceci afin qu'elle puisse grandir avec le temps. Si une liste est plus courte qu'une implémentation ne le prévoit, les drapeaux inconnus sont considérés comme étant à zéro. Les drapeaux définis sont :

Premier octet

0x01 - Cette clef peut être utilisée pour certifier d'autre clefs.

0x02 - Cette clef peut être utilisée pour signer des données.

0x04 - Cette clef peut être utilisée pour crypter des communications.

0x08 - Cette clef peut être utilisée pour crypter un enregistrement.

0x10 - Le composant privé de cette clef peut avoir été fractionné par un mécanisme de partage de secrets.

0x80 - Le composant privé de cette clef peut être en possession de plus d'une personne.

Notes d'utilisation :

Les drapeaux dans ce paquet peuvent apparaître dans les auto-signatures ou dans les signatures de certification. Ils expriment différentes choses suivant la personne qui émet l'instruction -- par exemple, une signature de certification qui possède le drapeau « signer les données » signale que la certification est dans ce but. Par contre, le drapeau « cryptage de communication » dans une auto-signature indique la préférence pour qu'une clef donnée soit utilisée pour les communications. Notez cependant, que c'est un autre problème que de déterminer ce qu'est « une communication » et ce qu'est un « enregistrement ». Cette décision est laissée entièrement à l'implémentation; les auteurs de ce document ne prétendent pas avoir d'avis particulier sur la question, et se rendent compte que l'opinion généralement répandue peut changer.

Les drapeaux de « clef découpée » (0x10) et de « clef de groupe » (0x80) sont placés seulement sur les auto-signatures; elle n'ont aucun sens pour les signatures de certification. Elles DOIVENT seulement être placés sur les signatures de clefs directes (type 0x1f) ou les signatures de sous clef (type 0x18), une qui renvoie à la clef à laquelle le drapeau s'applique.

5.2.3.21. ID utilisateur du signataire

Ce sous-paquet autorise un détenteur de clef à déclarer quel identifiant d'utilisateur est responsable de la signature. Beaucoup de détenteurs de clef utilisent une seule clef pour différentes applications, comme les communications professionnelles aussi bien que les communications personnelles. Ce sous-paquet autorise ce type de détenteur de clefs à déclarer à quel titre il fait une signature.

5.2.3.22. Raisons de révocation

(1 octet de code de révocation, N octets de chaînes de caractères pour la raison)

Ce sous paquet est utilisé seulement dans les révocations de clefs et dans les signatures de révocation de certification. Il décrit la raison de la révocation de la clef ou du certificat.

Le premier octet contient un code lisible par la machine qui indique la raison de la révocation :

0x00

Aucune raison spécifiée (révocation de clef ou de certificat)

0x01

La clef est périmée (révocations de clef)

0x02

La clef matérielle a été compromise (révocation de clef)

0x03

La clef n'est plus utilisée (révocations de clef)

0x20

L'information d'identifiant utilisateur n'est plus valide (révocations de certificat)

Le code de révocation est suivi d'une chaîne d'octets qui fournit des informations au sujet de la raison de la révocation en langage humain (UTF-8). La chaîne peut être nulle, c'est à dire, de longueur nulle. La longueur du sous paquet est la longueur de la chaîne donnant la raison plus un.

5.2.4. Créer les signatures

Toutes les signatures sont formées en effectuant un hachage sur la donnée de signature, puis en utilisant dans l'algorithme de signature le hachage qui en résulte.

Les données de signature sont simples à produire pour les signatures de document (type 0x00 et 0x01), pour lesquelles le document lui même représente les données. Pour les signatures seules, c'est une chaîne vide.

Quand une signature est crée par rapport à une clef, les données hachées débutent avec l'octet 0x99, suivi par deux octets de longueurs de clef, puis le corps du paquet de clef. (Notez que ceci est un style ancien d'en-tête de paquet pour un paquet de clef de deux octets de longueur.) Une signature de sous-clef (type 0x18) hache ensuite la sous-clef, en utilisant le même format que pour la clef principale. Les signatures de révocation de clef (type 0x20 et 0x28) hache seulement la clef en cours de révocation.

5.2.4.1. Conseils pour les sous-paquets

Une implémentation DOIT mettre les deux sous-paquets obligatoires, heure de création et émetteur, comme premiers sous-paquets dans la liste des sous-paquets, simplement pour que l'implémenteur les trouve plus facilement.

Il est certainement possible qu'une signature contienne des informations contradictoires dans les sous-paquets. Par exemple, une signature peut contenir plusieurs copies d'une préférence ou des temps d'expiration multiples. Dans la plupart des cas, une implémentation DEVRAIT utiliser le dernier sous-paquet de la signature, mais PEUT utiliser tout schéma de résolution de conflit pouvant être sensé. Veuillez noter que nous laissons intentionnellement la résolution de conflit à l'implémenteur ; la plupart des conflits sont simplement des erreurs de syntaxe, et les propos un peu flou que l'on tient ici permettent au récepteur d'être généreux sur ce qu'il accepte, tout en faisant pression sur le créateur pour qu'il soit radin sur ce qu'il génère.

Quelques conflits apparents peuvent en fait se comprendre -- par exemple, supposons qu'un détenteur de clef possède une clef V3 et une clef V4 qui partagent la même clef RSA de base. L'une ou l'autre de ces clés peut vérifier une signature créée par l'autre, et il peut être raisonnable pour une signature de contenir un sous-paquet émetteur pour chaque clef, de manière à attacher explicitement ces clefs à la signature.

5.3. Paquets de clés de session cryptés avec clé symétrique (Tag 3)

Le paquet de clef de session crypté par clef symétrique contient le cryptage par clef symétrique par clef symétrique de la clef de session utilisée pour crypter le message. Un ou plusieurs paquets (il peut aussi y en avoir aucun) de clef de session cryptés ou non par clef symétrique précèdent un paquet de données cryptées symétriquement qui contient un message crypté. Le message est crypté avec une clef de session, et la clef de session est elle-même cryptée et sauvegardée dans le paquet de clef de session cryptée ou dans le paquet de clef de session cryptée par clef symétrique.

Si le paquet de données cryptée symétriquement est précédé par au moins un paquet de clef de session cryptée par clef symétrique , chacun indique une phrase mot de passe qui sera utilisée pour décrypter le message. Cela permet d'avoir un message crypté par un nombre de clefs publiques, et également par une ou plusieurs phrase mot de passe. Ce type de paquet est nouveau, et n'est pas généré par PGP 2.x ou PGP 5.0.

Le corps de ce paquet comprends :

- Un numéro de version sur un octet. La seule version définit couramment est la 4

- Un nombre d'un octet décrivant l'algorithme symmétrique utilisé.
- Un identificateur de string-to-key (S2K), dont la longueur est spécifié ci dessus.
- (facultatif) la clef de session encryptée elle-même, qui est décryptée grâce à l'objet string-to-key.

Au cas où la clef de session cryptée n'est pas présente (ce qui peut être détecté d'après la longueur de paquets et les tailles d'identifieur S2K), l'algorithme S2K appliqué à la phrase secrète produit la clef de session pour décrypter le fichier, en utilisant l'algorithme de chiffrage symétrique venant du paquet de clef de session cryptée par clef symétrique.

Si la clef de session est présente, le résultat de l'application de l'algorithme S2K sur la phrase mot de passe est utilisé pour décrypter seulement le champs de clef de session cryptée en utilisant le mode CFB avec un Vecteur Initial (IV) tout en zéros. Le résultat du décryptage consiste en un identifieur d'algorithme d'un octet qui spécifie l'algorithme de cryptage à clef symétrique utilisé pour crypter le paquet de données cryptées symétriquement suivant, suivi par les octets de clef de session eux-mêmes.

Note : comme on utilise un IV de zéros pour ce décryptage, le spécifieur S2K DOIT ABSOLUMENT utiliser une valeur salée, soit un S2K salé soit un S2K salé itératif. La valeur salée fera en sorte que la clef de décryptage n'est pas répétée même si la phrase mot de passe est réutilisée.

5.4. Paquets de signature à une passe (Tag 4)

Le paquet de signature une passe précède les données signées et contient assez d'informations pour permettre au recepteur de commencer à calculer un des hachages nécessaires pour vérifier la signature. Il permet de placer le paquet de signature à la fin du message, de manière à ce que le signataire puisse fabriquer la totalité du message signé en une passe.

Une signature une passe n'intéragit pas avec PGP 2.6.x ou les versions antérieures.

Le corps de ce paquet comprend :

- Un numéro de version sur un octet. La version courante est 3.
- Un type de signature d'un octet. Les types de signatures sont décrits dans la section 5.2.1.
- Un nombre d'un octet précisant l'algorithme de hachage utilisé.
- Un nombre d'un octet précisant l'algorithme de clef publique utilisé.
- Un nombre de huit octets qui détient l'identifiant de clef de la clef signante.
- Un nombre d'un octet contenant un drapeau précisant si la signature est encapsulé. Une valeur de zéro indique que le paquet suivant est un autre paquet de signature une passe qui décrit une autre signature à appliquer aux mêmes données du message.

Notez que dans le cas où un message contient plus d'une signature en une passe, les paquets de signature entourent le message; c'est-à-dire: le premier paquet de signature suivant le message correspond au dernier paquet une passe et le paquet de signature final correspond au premier paquet une passe.

5.5. Paquet d'information sur les clefs

Un paquet d'information sur les clefs contient toutes les informations au sujet d'une clef privée ou publique. Il y a quatre variantes de ce type de paquet, et deux versions majeures. Par conséquent, cette section est complexe.

5.5.1. Variantes de paquets sur les clefs

5.5.1.1. Paquet sur les clefs publiques (Tag 6)

Un paquet sur les clefs publiques commence par une série de paquets qui forme une clef OpenPGP (parfois appelée un certificat OpenPGP).

5.5.1.2. Paquet sur les sous-clef publique (Tag 14)

Un paquet sur les sous clefs publiques (Tag 14) possède exactement le même format qu'un paquet sur les clefs publiques, mais signale une sous-clef. Une ou plusieurs sous-clefs peuvent être associées avec une clef de niveau supérieur. Par convention, la clef de niveau supérieur fournit les services de signatures, et les sous-clefs fournissent les services de cryptage.

Note: dans PGP 2.6.x, le tag 14 a été conçu pour indiquer un paquet de commentaire. Ce tag a été sélectionné pour réutilisation car aucune version précédente de PGP n'a jamais émis de paquets de commentaire mais au contraire elles les ignoraient carrément. Les paquets sur les sous-clefs publiques sont ignorés par PGP 2.6.x et ne le font pas planter, fournissant un peu de compatibilité descendante.

5.5.1.3. Paquet sur les clefs secrètes (Tag 5)

Un paquet sur les clefs secrètes contient toutes les informations trouvées dans un paquet sur les clefs publiques, y compris les informations sur les clefs publiques, mais il comprend aussi les informations sur les clefs secrètes, qui se trouvent après tous les champs de clefs publiques. secrète après tous les champs de clef publique.

5.5.1.4. Paquet sur les sous-clefs secrètes (Tag 7)

Un paquet sur les sous-clefs secrètes (tag 7) est l'équivalent pour les sous-clefs du paquet et a exactement le même format.

5.5.2. Formats du paquet sur les clefs publiques

Il y a deux versions de paquets d'information sur les clefs. Les paquets de Version 3 ont été générés à l'origine par PGP 2.6. Les paquets de Version 2 sont de formats identiques aux paquets de Version 3, mais sont générés par PGP 2.5 ou antérieur. Les paquets V2 sont a rejeter et ne DOIVENT ABSOLUMENT PAS être générés. PGP 5.0 à introduit les paquets de Version 4, avec de nouveaux champs et de nouvelles sémantiques. PGP 2.6.x n'acceptera pas les paquets d'information sur les clefs de version supérieure à 3.

Les implémentations de OpenPGP DOIVENT créer des clefs avec le format de la version 4. Une implémentation PEUT générer une clef V3 pour assurer l'interropérabilité avec les anciens logiciels; notez, cependant, que les clefs V4 corrigent quelques défaillance de sécurité des clefs V3. Ces défaillances sont décrites ci-dessous. Une implémentation ne DOIT EN AUCUN CAS créer une clef V3 avec un algorithme à clef publique autre que RSA.

Un paquet Version 3 sur une clef publique ou sur une sous-clef publique contient :

- Un numéro de version (3) sur un octet.
- Un nombre de quatre octets indiquant la date de création de la clef.
- Un nombre de deux octets indiquant la durée de validité en jours de la clef. Si ce nombre est zéro, la clef n'expire pas.

- Un nombre d'un octet indiquant l'algorithme de clef publique de cette clef.
- Une série d'entiers multi-précision comprenant les informations suivantes sur les clefs :
  - un entier multi-précision (MPI ndt:Multi Precision Integer) de modulo publique RSA n;
  - un MPI de cryptage publique RSA exposant e.

Les clefs V3 DOIVENT seulement être utilisées pour des questions de compatibilité antérieure car elle comporte trois points faibles. Premièrement, il est relativement facile de construire une clef V3 qui possède le même identifiant de clef qu'une autre clef car l'identifiant de clef est simplement constitué des 64 bits de poids faible du modulo publique. Deuxièmement, car l'empreinte d'une clef V3 hache les informations sur la clef, mais pas sa longueur, ce qui accroit les possibilités de conflits d'empreintes. Troisièmement, il y a quelques petites lacunes dans l'algorithme de hachage MD5 qui font que les développeurs préfèrent les autres algorithmes. Vous trouverez ci-dessous pour une discussion plus complète des identifiants de clefs et des empreintes.

Le format de la version 4 est similaire à celui de la version 3 excepté en ce qui concerne l'absence de la période de validité. Elle a été déplacée dans le paquet de signature. De plus, les empreintes des clefs de version 4 sont calculées différemment de celles de la version 3, comme décrit dans la section "Formats de clefs amélioré".

Un paquet de version 4 contient:

- Un numéro de version (4) d'un octet.
- Un nombre de quatre octets indiquant la date de création de la clef.
- Un nombre d'un octet indiquant l'algorithme de clef publique de la clef.
- Une série d'entiers multiprécision comprenant les informations de clef. La portion spécifique à l'algorithme est :

  Les champs spécifiques à l'algorithme pour des clefs publiques RSA :

  - entier multi-précision (MPI) de public RSA modulo n;
  - MPI de cryptage publique RSA d'exposant e.

  Les champs spécifiques à l'algorithme pour les clefs publiques DSA :

  - un MPI de nombre premier DSA p;
  - un MPI d'ordre de groupe DSA q (q est un diviseur du nombre premier p-1);
  - un MPI de générateur de groupe DSA g;
  - un MPI de valeur de clef publique DSA y (= g**x où x est secret).

  Les champs spécifiques à l'algorithme pour des clefs publiques Elgamal :

  - un MPI de nombre premier Elgamal p;
  - un MPI de groupe générateur Elgamal g;
  - un MPI Elgamal de valeur de clef publique y (= g**x où x est secret).

5.5.3. Formats de paquet sur les clefs secrètes

Ces paquets sur une clef secrète et sur une sous-clef secrète contiennent toutes les données et la paquets de clef publique de sous-clef publique, auxquelles sont ajoutés des données de clef secrète supplémentaire spécifiques à l'algorithme, sous forme cryptée. [XXXXXXXXXXXXXXXXX]

Le paquet contient:

- un paquet sur la clef publique et la sous-clef publique, comme décrit ci dessus
- Un octet indiquant les conventions d'usage des string-to-key. 0 indique que les données de clef secrète ne sont pas cryptées. 255 indique qu'un identifiieur string-to-key est donnée. Toutes les autres valeurs, sont des identifieurs d'algorithme de cryptage à clef symétrique.
- [Optionnel] Si l'octet d'usage string-to-key était 255, un algorithme de cryptage symétrique d'un octet.
- [Optionnel] Si l'octet d'usage string-to-key était 255, un identifieur string-to-key. La longueur de l'identifieur est induite par son type, comme décrit ci dessus.
- [Optionnel] Si la donnée secrète est cryptée, un Vecteur Initial de huit octets (IV).
- Des entiers multi-précision cryptés comprenant les données de la clef secrète. Ces champs spécifiques à l'algorithme sont décrits si dessous.
- Un checksum de deux octets du texte brut de la partie spécifique à l'algorithme (la somme de tous les octets, mod 65536).

  Des champs spécifiques à l'algorithme pour les clefs secrètes RSA:

  - un entier multiprécision (MPI) d'exposant secret RSA d.
  - MPI de valeur secrète de nombre premier RSA p.
  - MPI de valeur secrète de nombre premier RSA q (p < q).
  - MPI de u, l'inverse multiplicatif de p, module q.

  Des champs spécifiques à l'algorithme pour les clefs secrètes DSA:

  - Un MPI d'exposant secret DSA x.

  Champs spécifiques à l'algorithme pour les clefs secrètes Elgamal:

  - Un MPI d'exposant secret Elgamal x.

Les valeurs MPI secrètes peuvent être cryptées en utilisant une phrase mot de passe. Si un identifieur string-to-key est donné, celui-ci décrit l'algorithme pour convertir la phrase secrète en une clef, sinon un simple hachage MD5 de la phrase secrète est utilisé. Les implémentations DOIVENT utiliser un identifieur string-to-key; l'utilisation d'un hachage simple sert a garder la compatiblité antérieure. Le chiffrage pour crypter les MPI est spécifié dans le paquet de clef secrète.

Le cryptage/décryptage des données secrètes est faite en mode CFB en utilisant la clef créée à partir de la phrase secrète et le Vecteur Initial venant du paquet. Un mode différent est utilisé avec les clefs V3 (qui ne sont que de type RSA) par rapport aux autres formats de clefs. Avec les clefs V3, le préfixe de comptage de bit MPI (i.e., les deux premiers octets) n'est pas crypté. Seules les données non préfixés MPI sont cryptées. De plus, l'état CFB

est resynchronisé au début de chaque nouvelle valeur MPI, de manière à ce que la limite de bloc CFB soit aligné avec le début des données MPI.

Avec les clefs V4, une méthode plus simple est utilisée. Toutes les valeurs secrètes MPI sont cryptées en mode CFB, y compris le préfixe de comptage de bit MPI.

Le checksum 16-bit qui suit la partie spécifique à l'algorithme est la somme algébrique, mod 65536, du texte brut de tous les octets spécifiques d'algorithme (incluant le préfixe et les données MPI). Avec les clefs V3, le checksum est stocké en clair. Avec les clefs V4, le checksum est crypté comme les données spécifiques à l'algorithme. Cette valeur est utilisée pour vérifier que la phrase mot de passe est correcte.

5.6. Paquet des données compressées (Tag 8)

Les paquets de données contiennent les données compressées. Typiquement, ce paquet est trouvé comme le contenu d'un paquet crypté, ou à la suite d'une signature ou d'un paquet de signature une passe, et contient des paquets de données littéraires.

Le corps du paquet comprend:

- Un octet qui donne l'algorithme utilisé pour compresser le paquet.
- Le reste du paquet comprend des données compressées.

Le corps du paquet de données compressées contient un bloc qui compresse un ensemble de paquets. Voir la section « Composition de paquet » pour les détails sur la façon dont les messages sont formés.

Les paquets compressés ZIP sont compressés avec des blocs brut RFC 1951 DEFLATE. Notez que PGP V2.6 utilise 13 bits de compression. Si une implementation utilise plus de bits de compression, PGP V2.6 ne peut pas le décompresser.

Les paquets compressés ZLIB sont compressés avec les blocs du style ZLIB de la RFC 1950.

5.7. Paquet des données cryptées symétriquement (Tag 9)

Les paquets de données cryptées symétriquement contiennent des données cryptées avec un algorithme à clef symétrique. Une fois décrypté, il contient typiquement d'autres paquets (souvent des paquets de données littéraires ou des paquets de données compressées).

Le corps de ce paquet consiste en :

- Des données cryptées, la sortie du chiffrage à clef symétrique sélectionné opérant dans la variante PGP du mode de Retour d'information de chiffrage (CFB) [Cipher Feedback].

Le chiffrage symétrique utilisé peut être spécifié dans un paquet publique de clef de session crypté par clef symétrique qui précède le paquet de données cryptées symétriquement. Dans ce cas, l'octet d'algorithme de chiffrage est mis en préfixe à la clef de session avant le cryptage. Si aucun paquet de ce type ne précède les données cryptées, l'algorithme IDEA est utilisé avec la session de clef calculée comme hachage MD5 de la phrase mot de passe.

Les données sont cryptées en mode CFB, avec une taille de décalage CFB égale à la taille du bloc de chiffrage. Le vecteur initial (IV) est spécifié étant tout en zéros. Au lieu d'utiliser un IV, OpenPGP met une chaine de 10 octets devant les données avant cryptage. Les huit premiers octets sont aléatoires, et les 9ème et 10ème octets sont des copies des 7ème et 8ème octets, respectivement. Après avoir décrypté les 10 premiers octets, l'état CFB

est resynchronisé si la taille de bloc de chiffrage est de 8 octets ou moins. Les 8 derniers octets du texte de chiffrage sont passés a travers le chiffrage et l'enveloppe de bloc est réinitialisée.

La répétition des 16 bits dans les 80 bits de données aléatoires préfixées au message permet au récepteur de vérifier immédiatement si la clef de session est incorrecte.

5.8. Paquet de marqueur (Ancien paquet de libellé) (Tag 10)

Une version expérimentale de PGP a utilisé ce paquet comme un paquet littéral, mais aucune version paru de PGP a généré des paquets Littéral avec ce tag. Avec PGP 5.x, ce paquet a été réassigné et est réservé à l'utilisation pour les paquets de marqueurs.

Le corps de ce paquet consiste en :

- Les trois octets 0x50, 0x47, 0x50 (qui s'épelle "PGP" dans UTF-8).

Un tel paquet DOIT ABSOLUMENT être ignoré quand il est reçu. Il doit être placé au début d'un message qui utilise des propriétés non disponible dans PGP 2.6.x de manière à permettre à cette version d'avertir qu'un nouveau logiciel est nécessaire pour traiter le message.

5.9. Paquet de données littérales (Tag 11)

Un paquet de données littéral contient le corps d'un message; données qui n'a lieu d'être interprété après.

Le corps de ce paquet consiste en:

- Un champs d'un octet qui décrit comment les données sont formatés.

Si c'est un 'b' (0x62), alors le paquet littéral contient des données binaires. Si c'est un 't' (0x74), alors il contient des données texte, et dans ce cas il doit y avoir besoin de convertir les fins de ligne sous la forme locale, ou d'effectuer d'autres transformations de texte. La RFC 1991 définie également une valeur 'l' comme un mode 'local' pour les conversions machine-locale. Cette utilisation est maintenant rejeté.

- Un nom de fichier sous forme de chaîne de caractère (un octet de longueur, suivi du nom de fichier), si la donnée cryptée peut être sauvé sous forme de fichier.

Si le nom spécial « _CONSOLE » est utilisé, le message est considéré être « seulement pour les yeux ». Cela avertit que les données du message est inhabituellement sensible, et que le programme de réception doit la traiter avec plus de précaution, peut être en évitant de sauvegarder les données reçues sur le disque, par exemple.

- Un nombre de quatres octets qui indique la date de modiffication du fichier, ou l'heure de création du paquet, ou encore l'heure actuelle.
- Le contenu du paquet sont des données littéraire .

Les données textes sont stockées avec des fins de texte < CR >< CF > (i.e. fins de lignes normales des réseaux). Cela doit être converti en fins de lignes natives par le logiciel récepteur.

5.10. Paquet de confiance (Tag 12)

Le paquet de confiance est utilisé seulement dans l'espace sécurisée des trousseaux de clefs et n'est normalement pas exporté. Les paquets de confiance contiennent des données qui enregistrent les caractéristiques de l'utilisateur, caractéristiques qui peuvent être introduite par les détenteurs de clefs parce qu'ils sont dignes de confiance, ainsi que d'autres informations que les logiciels d'implémentation utilisent pour les informations de confiance.

Les paquets de confiance NE DOIVENT PAS être émis dans des chaînes de sortie qui sont transférées à d'autres utilisateurs, et il DOIVENT être ignoré à n'importe quelle entrée autre que les fichiers de trousseau de clef locale.

5.11. Paquet d'identification de l'utilisateur (Tag 13)

Un paquet d'identifiant d'utilisateur consiste en des données qui sont censée représenter le nom et l'adresse e-mail du détenteur de clef. Par convention, il inclut un nom de courrier RFC 822, mais il n'y pas pas de restrictions sur son contenu. La longueur du paquet dans l'en-tête spécifie la longueur de l'identifiant utilisateur. Si c'est du texte, il est encodé en UTF-8.

Chapitre 6. Conversions Radix-64

Comme il a été dit dans l'introduction, la représentation sous-jacente native des objets OpenPGP est un flux d'octets aléatoires. Certains systèmes requièrent que ces objets soient à l'abri de tout dommage issu de conversion de jeux de caractères, de données, etc.

En théorie, tout schéma d'encodage des données correspondant aux exigences des canaux non sûrs serait suffisant, puisqu'il ne modifierait pas le flux de bit sous-jacent des structures de donnée natives d'OpenPGP. La norme d'OpenPGP spécifie un schéma d'encodage de ce type pour assurer l'interopérabilité.

L'encodage Radix-64 d'OpenPGP est constitué de deux parties: l'encodage des données binaires en base64 et un checksum. L'encodage en base64 est identique au content-transfer-encoding (contenu-transfert-encodage) en base64 MIME [RFC2231,Section 6.8]. Une implémentation OpenPGP peut utiliser une armure ASCII pour protéger les données binaires brutes.

Le checksum est le résultat d'un CRC sur 24 bits converti en quatre caractères d'encodage radix-64 encodés par le même procédé MIME base64, et précédé d'un signe égal (=). Le CRC est calculé en utilisant le générateur 0x864CFB et une initialisation à 0xB704CE. La somme est effectuée sur les données avant leur conversion en radix-64 plutôt qu'après. Un exemple d'implémentation de cet algorithme est disponible dans la prochaine section.

On peut faire apparaitre le résultat du checksum, précédé du signe égal, sur la première ligne suivant les données encodées.

Le CRC-24 repose sur la logique suivante: une taille de 24 bits lui permet de s'adapter parfaitement à une base64 imprimable. L'initialisation non nul permet de détecter plus d'erreur qu'une initialisation à zéro.

6.1. Une implémentation du CRC-24 en « C »

```
#define CRC24_INIT 0xb704ceL
       #define CRC24_POLY 0x1864cfbL

       typedef long crc24;
       crc24 crc_octets(unsigned char *octets, size_t len)
       {
           crc24 crc = CRC24_INIT;
           int i;

           while (len--) {
               crc ^= (*octets++) << 16;
               for (i = 0; i < 8; i++) {
                   crc <<= 1;
                   if (crc & 0x1000000)
                       crc ^= CRC24_POLY;
               }
           }
           return crc & 0xffffffL;
       }
```

6.2. Création de l'armure ASCII

Lorsqu' OpenPGP encode des données dans une armure ASCII, il encapsule les données grâce à des entêtes spécifiques, dans l'optique de leur reconstruction ultérieure. OpenPGP informe l'utilisateur du type de donnée encodé dans l'armure ASCII par l'intermédiaire de ces entêtes.

La concaténation des données suivantes permet de créer l'armure ASCII :

- Une ligne d'en-tête d'armure appropriée au type de donnée à encoder,
- Les en-têtes de l'armure,
- Une ligne vide (de taille nulle ou ne contenant que des espacements),
- Les données encodées,
- Un checksum spécifique à l'armure,
- La ligne de fin d'armure, dépendant de la première ligne.

La ligne d'en-tête d'armure se débute et se termine par cinq tirets ('-', 0x2D) encadrant le contenu approprié. Elle est choisie en fonction du type de donnée à encoder et de la méthode d'encodage.

Elle contient les chaînes de caractères suivantes :

BEGIN PGP MESSAGE

Utilisé pour les fichiers signés, cryptés ou compressés,

BEGIN PGP PUBLIC KEY BLOCK

Utilisé pour protéger par l'armure les clés publiques,

BEGIN PGP PRIVATE KEY BLOCK

Utilisé pour protéger par l'armure les clés privées,

BEGIN PGP MESSAGE, PART X/Y

Utilisé pour les messages composites, lorsque l'armure est séparée en Y morceaux et que le fichier est le Xième sur Y,

BEGIN PGP MESSAGE, PART X

Utilisé pour les messages composites, lorsque l'armure est séparée en un nombre de morceaux inconnu et que le fichier est le Xième. Nécessite l'en-tête d'armure MESSAGE-ID pour être utilisé,

BEGIN PGP SIGNATURE

Utilisé pour les signatures détachées, les signatures OpenPGP/MIME, et les signatures qui suivent des messages signés en clair. Remarque: PGP en version 2.x utilise BEGIN PGP MESSAGE pour les signatures détachées.

Les entêtes de l'armure sont des pairs de chaînes de caractère permettant à l'utilisateur ou à l'implémentation OpenPGP de réception d'obtenir des renseignements sur la manière de décoder ou d'utiliser le message. Ces entêtes font partie de l'armure et non pas du message. Par conséquent, elle ne sont protégées par aucune des signatures appliquées au message.

Le format d'une entête d'armure est est constitué d'une paire clé-valeur. La clé et sa valeur sont séparés par ': '(0x38) et le caractère d'espacement simple ' ' (0x20). OpenPGP considérera une entête d'armure mal formatée comme une altération de l'armure ASCII. Un cas de clé non reconnue sera signalé à l'utilisateur sans que le processus OpenPGP appliqué au message soit arrêté.

Les clés actuellement définies d'une entête d'armure sont :

- "Version", qui précise la version d'OpenPGP utilisé pour encoder le message,
- "Comment", commentaire pouvant être défini par l'utilisateur,
- "MessageID", chaîne composée de 32 caractères imprimables. Elle doit être identique pour tous les morceaux d'un message composite utilisant l'entête d'armure "PART X". Elle doit être unique (dans la mesure du possible) pour que le bénéficiaire du message puisse associer les différentes parties du message à sa réception. Un checksum fiable ou une fonction de hachage cryptographique suffisent,
- "Hash", listes d'algorithmes de hachage séparés par des virgules qui sont utilisés dans le message. Utilisé seulement dans les messages non signé,
- "Charset", description du jeu de caractères utilisé dans le texte non codé.

Il est à noter qu'OpenPGP défini par défaut le texte comme étant en UTF-8. Le résultat d'un codage sera meilleur si les données sont au préalable traduites en UTF-8 puis retranscrite après décodage. Cependant, il est possible que cela soit plus facile à dire quà faire. Par ailleurs, certains groupes d'utilisateurs n'ont pas besoin de UTF-8 parce qu'un jeu de caractère japonais ou encore ISO Latin-5 leur suffit. Dans ce cas, l'implémentation peut redéfinir la description du jeu de caractères en utilisant cette clé. Une implémentation peut utilisée cette clé et tous les jeux de caractère à sa guise; ou encore l'ignorer et supposer que le texte suit la norme UTF-8.

La clé MessageID NE DOIT PAS apparaître à moins que le message soit un message composite. Au cas où il apparaîtrait, il DOIT IMPERATIVEMENT être calculé à partir du message terminé(crypté, signé, etc) et non pas initialisé à une valeur purement aléatoire. Ceci permet de prouver au bénéficiaire légitime du message que ce champ de donnée n'est pas un moyen caché de provoquer des fuites d'information, permettant d'accéder à des clés cryptographiques.

La ligne de fin d'armure est construite de la même manière que la ligne d'en-tête d'armure à l'exception de la chaîne "BEGIN" qui est remplacée par "END".

6.3. Encoder des données binaires en Radix-64

Le processus d'encodage accepte en entrée des groupes de 24 bits et renvoie en sortie des chaînes de quatre caractères encodés. Procédant de gauche à droite, un groupe de 24 bits est formé par concaténation de 3 groupes de 8 bits. Ces 24 bits sont ensuite traités comme étant 4 groupes de 6 bits, chacun d'entre eux étant associé à un unique signe de l'alphabet Radix-64. Lorsque l'on encode un flux de bit à l'aide de l'encodage Radix-64, le flux doit être ordonné de manière à ce que les bits de poids fort soient en premier. Cela signifie que le premier bit du flux doit être le bit de poids le plus fort du premier octet, le huitième, étant le bit de poids le plus faible du premier octet, et ansi de suite.

+-- 1er octet --+- 2eme octet -+-- 3eme octet --+

|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|

+-----------+---+-------+-------+---+-----------+

|5 4 3 2 1 0|5 4 3 2 1 0|5 4 3 2 1 0|5 4 3 2 1 0|

+- 1.index -+- 2.index -+- 3.index -+- 4.index -+

Chaque groupe de six bits est utilisé comme un index dans un tableau de 64 caractères imprimables (cf tableau suivant). Le caractère référencé par l'index est placé dans la chaîne de sortie.

Tableau 6-1. Valeurs encodées

Valeurs encodées			
0 A	17 R	34 i	51 z
1 B	18 S	35 j	52 0
2 C	19 T	36 k	53 1
3 D	20 U	37 l	54 2
4 E	21 V	38 m	55 3
5 F	22 W	39 n	56 4
6 G	23 X	40 o	57 5
7 H	24 Y	41 p	58 6
8 I	25 Z	42 q	59 7
9 J	26 a	43 r	60 8
10 K	27 b	44 s	61 9
11 L	28 c	45 t	62 +
12 M	29 d	46 u	63 /
13 N	30 e	47 v	
14 O	31 f	48 w	(caractère de remplissage)=
15 P	32 g	49 x	
16 Q	33 h	50 y	

Le flux de caractères encodés en sortie doit être représenté par des lignes de moins de 76 caractères chacune. Un processus particulier est appliqué au dernier groupement de bits lorsque celui-ci est composé de moins de 24 bits. Il y a donc trois possibilités :

- S'il est composé de 24 bitss (3 octets), aucun processus particulier n'est requis,
- S'il est composé de 16 bits (2 octets), les deux premiers groupes de six caractères sont traités comme précédemment. Le troisième groupe de donnée (incomplet) se voit attribuer 2 bits nuls suplémentaires avant traitement. Un caractère supplémentaire est ajouté à la sortie (=),
- S'il est composé de 8 bits (1 octets), le premier groupe de six caractères est traité comme précédemment. Le deuxième groupe de donnée (incomplet) se voit attribuer 4 bits nuls suplémentaires avant traitement. Deux caractères supplémentaires sont ajoutés à la sortie (==).

6.4. Décoder du Radix-64

Tout caractère ne figurant pas dans l'alphabet Radix-64 est ignoré dans les données Radix-64. Le programme de décodage doit ignorer tous les retour chariot ou autres caractères non présent dans la table ci-dessus.

Dans des données en Radix-64, tout caractère n'appartenant pas à la table, retour chariot ou autres espacements, indique qu'il y a probablement eu une erreur durant la transmission. Un message d'avertissement ou un rejet du

message peuvent donc être appropriés dans certaines circonstances.

Du fait qu'il soit utilisé seulement comme caractère de remplissage à la fin des données, toute apparition du symbole égal peut être considérée comme le signe de la fin des données à traiter (sans découpe durant la transmission). Une telle certitude est cependant impossible lorsque le nombre d'octets transmis est un multiple de trois et qu'aucun signe « = » n'a été détecté.

6.5. Exemples de Radix-64

```
données en entrée:  0x14fb9c03d97e
Hex:     1   4    f   b    9   c     | 0   3    d   9    7   e
8-bit:   00010100 11111011 10011100  | 00000011 11011001 11111110
6-bit:   000101 001111 101110 011100 | 000000 111101 100111111110
Décimal: 5      15     46     28       0      61     37     62
Sortie:  F      P      u      c        A      9      l      +

données en entrée:  0x14fb9c03d9
Hex:     1   4    f   b    9   c     | 0   3    d   9
8-bit:   00010100 11111011 10011100  | 00000011 11011001
                                           complété avec 00
6-bit:   000101 001111 101110 011100 | 000000 111101 100100
Décimal: 5      15     46     28       0      61     36
                                               complété avec =
Sortie:  F      P      u      c        A      9      k      =

données en entrée:  0x14fb9c03
Hex:     1   4    f   b    9   c     | 0   3
8-bit:   00010100 11111011 10011100  | 00000011
                                     complété avec 0000
6-bit:   000101 001111 101110 011100 | 000000 110000
Décimal: 5      15     46     28       0      48
                                     complété avec =      =
Sortie:  F      P      u      c        A      w      =      =
```

6.6. Exemple d'un message ASCII de type Armor

```
-----BEGIN PGP MESSAGE-----
Version: OpenPrivacy 0.99

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END PGP MESSAGE-----
```

Chapitre 7. Structure de signature de texte en clair

Il est souhaitable de signer un flux d'octet textuel sans le placer dans une armure ASCII, ainsi le texte signé est encore lisible sans logiciel particulier. Afin de lier une signature à un tel texte en clair, cette structure est employée. (noter que RFC 2015 définit une autre manière de signer en clair les messages pour les environnements qui supportent le MIME.)

Le message correspondant au texte en clair signé est constitué de :

- L'entête de texte en clair '-----BEGIN PGP SIGNED MESSAGE-----' sur une ligne unique,

  Une ou plusieurs entêtes d'armure « Hash »,

  Exactement un ligne vide non incluse dans le sommaire du message,

  Le texte en clair inclus dans le sommaire du message avec un processus de distinction des tirets de début de ligne,

  La(es) signature(s) avec armure ASCII incluant la ligne d'en-tête et celle de fin d'armure '-----BEGIN PGP SIGNATURE-----'.

Si l'en-tête d'armure « Hash » est donnée, l'algorithme de sommaire de message indiqué est employé pour la signature. Si il n'y a pas de pareilles entêtes, MD5 est employé, une exécution PEUT les omettre pour assurer la compatibilité avec les versions V2.x. Si plus d'un sommaire de message est employé dans la signature, l'en-tête d'armure « Hash » contient la liste des sommaires de message utilisés, délimités par des virgules.

Les noms actuels des sommaire de message sont décrits plus bas avec les ID des algorithmes.

7.1. Processus de distinction des tirets de début de ligne

Il est nécessaire de marquer la présence de certains tirets dans le contenu du texte en clair.

Ainsi toute ligne de texte en clair debutant par un tiret (0x2D) est préfixée par une séquence composée d'un tiret ' - ' (0x2D) et d'un espace ' ' (0x20). Ceci permet d'éviter que l'analyseur syntaxique ne reconnaisse les entêtes d'armure du texte en clair lui-même. Le sommaire de message est traité en utilisant le texte en clair lui-même sans ces préfixes.

Comme pour les signatures binaires sur les documents textuels, une signature de texte en clair est calculée sur le texte en utilisant des fins de ligne canonique . La fin de ligne (c'est à dire) précédent la ligne '-----BEGIN PGP SIGNATURE-----' et mettant fin au texte signé n'est pas considérée comme appartenant au texte signé.

De plus, tout espacement de décalage (espacements et tabulations , 0x09) en fin de ligne est ignoré lorsque la signature du texte en clair est calculée.

Chapitre 8. Expressions régulières

Une expression régulière est composée de zéro branche ou plus, séparés par le caractère ' | '. Elle correspond à tout ce qui peut correspondre avec l'une des branches.

Une branche est composé de zéro partie ou plus, concaténées. Elle correspond à une concordance avec la première partie, suivi d'une concordance avec la deuxième,...

Une partie est un atome suivi éventuellement par ' * ', ' + ' ou ' ? '. Un atome suivi de ' * ' correspond à une séquence de 0 concordance avec l'atome ou plus. Un atome suivi de ' + ' correspond à une séquence de 1 concordance avec l'atome ou plus. Un atome suivi de ' ? ' correspond à une concordance avec l'atome ou avec une chaîne vide.

Un atome peut être :

- Une expression régulière entre parenthèses (correspondant à une concordance pour l'expression régulière),
- Un intervalle (cf ci-dessous),
- Le caractère ' . ' (correspondant à tout caractère unique),
- Le caractère ' ^ ' (correspondant à la chaîne vide au début de la chaîne entrée),
- Le caractère ' $ ' (correspondant à la chaîne vide à la fin de la chaîne entrée),
- Le caractère ' \ ' suivi par un caractère unique (correspondant à ce caractère unique),
- Un caractère unique sans sens particulier (correspondant à ce caractère).

Un intervalle est une séquence de caractères inclus entre les caractères ' [' et '] '. Il correspond normalement à tout caractère appartenant à la séquence. Si la séquence commence par ' ^ ', il correspond à tout caractère unique n'appartenant pas au reste de la séquence. Si deux caractères de la séquence sont séparés par un ' - ', il correspond à n'importe quel caractère ASCII compris entre les deux caractères reliés, ceux-ci inclus (par exemple ' [0-9] ' correspond à n'importe quel chiffre décimal). Pour inclure le caractère '] ' dans la séquence, il faut le placer en première position (ou éventuellement en deuxième sil suit ' ^ '). Pour inclure le caractère ' - ', il faut le placer en première ou en dernière position dans la séquence.

Chapitre 9. Constantes

Cette section décrit les constantes utilisées dans OpenPGP.

Il est à noter que ces tables ne sont pas des listes exhaustives. Une implémentation PEUT implémenter un algorithme n'appartenant pas à ces listes.

Se référer à la section « Notes sur les algorithmes », chapitre 12, pour plus d'information sur les algorithmes.

9.1. Algorithmes à clé publique

Tableau 9-1. ID des algorithmes

ID	Algorithme
1	RSA (cryptage ou signature)
2	RSA seulement cryptage
3	RSA seulement signature
16	Elgamal (seulement cryptage), cf [ELGAMAL]
17	DSA (standard de signature digitale)
18	Réservé pour les courbes elliptiques
19	Réservé pour ECDSA
20	Elgamal (cryptage ou signature)
21	Réservé pour Diffie-Hellman (X9.42, comme défini pour IETF-S/MIME)
100 à 110	Algorithme personnel/expérimental

Les implémentations DOIVENT implémenter l' algorithme DSA pour la signature et l' Elgamal pour le cryptage. Il FAUT qu'elles implémentent des clés RSA. Les implémentations PEUVENT implémenter n'importe quel autre algorithme.

9.2. Algorithmes à clé symétrique

Tableau 9-2. ID des algorithmes

ID	Algorithme
0	Texte ou donnée non codés
1	IDEA [IDEA]
2	Triple-DES (DES-EDE, conforme aux spécifications pour les clé de 168 bits dérivées des 192)
3	CAST5 (clé de 128 bits, cf RFC 2144)
4	Blowfish (clé de 128 bit, 16 periodes) [BLOWFISH]
5	SAFER-SK128 (13 périodes) [SAFER]
6	Réservé pour DES/SK

ID	Algorithme
7	Réservé pour AES à clé de 128 bits
8	Réservé pour AES à clé de 192 bits
9	Réservé pour AES à clé de 256 bits
100 à 110	Algorithme personnel/expérimental

Les implémentations DOIVENT implémenter l' algorithme Triple-DES. Il FAUT qu'elles implémentent IDEA and CAST5. Les implémentations PEUVENT implémenter n'importe quel autre algorithme.

9.3. Algorithmes de compression

Tableau 9-3. ID des algorithmes

ID	Algorithme
0	Non compressé
1	ZIP (RFC 1951)
2	ZLIB (RFC 1950)
100 à 110	Algorithme personnel/expérimental

Les implémentations DOIVENT implémenter les données non compressées. Il FAUT qu'elles implémentent ZIP. Les implémentations PEUVENT implémenter ZLIB.

9.4. Algorithmes de hachage

Tableau 9-4. ID des algorithmes

ID	Algorithme	Nom du texte
1	MD5	« MD5 »
2	SHA-1	« SHA1 »
3	RIPE-MD/160	« RIPEMD160 »
4	Réservé pour le SHA à longueur double(expérimental)	
5	MD2	« MD2 »
6	Réservé pour TIGER/192	« TIGER192 »
7	Réservé pour HAVAL (5 pass, 160-bit)	« HAVAL-5-160 »
100 à 110	Algorithme personnel/expérimental	

Les implémentations DOIVENT implémenter SHA-1. Il FAUT qu'elles implémentent MD5.

Chapitre 10. Arrangement des paquets

Les paquets OpenPGP sont assemblés en séquence dans le but de créer les messages et de transférer les clés. Toutes les combinaisons possibles de paquets ne forment pas de séquences significatives et correctes. Cette section décrit les règles spécifiant comment les paquets doivent être placés dans les séquences.

10.1. Clés publiques transférables

Les utilisateurs d'OpenPGP peuvent transférer des clés publiques. Les éléments essentiels d'une clé publique transférable sont :

- Un paquet contenant la clé publique,
- Eventuellement une ou plusieurs signatures de révocation,
- Un ou plusieurs paquets correspondant à un identifiant utilisateur,
- Après chacun de ces paquets, éventuellement une ou plusieurs signatures de paquet,
- Eventuellement un ou plusieurs paquets de sous-clé,
- Après chaque paquet de sous-clé, un paquet de signature, optionellement une révocation.

Le paquet contenant la clé publique se présente en premier. L'identité du propriétaire de la clé publique est fournie par chaque paquet d'identifiant utilisateur suivant le paquet de clé publique. Si il y a plusieurs paquets d'identifiant utilisateur, cela correspond à plusieurs moyens d'identification d'un unique utilisateur individuel. Par exemple, un utilisateur peut posséder plus d'une adresse e-mail et construire un identifiant utilisateur pour chacune d'elles.

Immédiatement après chaque paquet d'identifiant utilisateur, il peut éventuellement y avoir un ou plusieur paquet de signature. Chacun de ces paquets de signature est calculé à partir du paquet d'identifiant utilisateur le précédent et du premier paquet contenant la clé publique. La signature sert à certifier la clé publique et l'identifiant utilisateur correspondant. Par conséquent, le signataire fait le serment que cette clé publique appartient à l'utilisateur correspondant à l'identifiant utilisateur.

Après les paquets d'identifiant utilisateur, il peut y avoir un ou plusieur paquets de sous-clé. En général, les sous-clé sont fournies lorsque la clé publique de niveau supérieur est seulement une clé de signature. Cependant, toute clé V4 possède des sous-clés pouvant être des clés dédiées au cryptage, dédiées à la signature ou encore à but global.

Chaque paquet de sous-clé doit être suivi par un paquet de signature, devant être une signature liée à la sous-clé et provenant de la clé de niveau supérieure.

Les paquets de sous-clé et de clé peuvent chacun être suivi par un paquet de révocation de signature indiquant que la clé est révoquée. Les révocations de signatures sont acceptées à la condition qu'elles proviennent de la clé elle-même ou d'une clé étant autorisée à émettre des révocations par la clé de niveau supérieure via un sous-paquet de révocation de clé dans une auto-signature.

Les séquences de paquet de clé publique transférable peuvent être concaténées pour permettre la transmission de clé publique multiple en une seule opération.

10.2. Messages OpenPGP

Un message OpenPGP est un paquet ou une séquence de paquets correspondant aux règles grammaticales suivantes (une virgule représente une composition séquentielle, une barre verticale sépare plusieurs alternatives):

Message OpenPGP :- Message crypté | Message signé | Message compressé | Message littéral

Message compressé :- paquet de données compressées

Message littéral :- paquet de donnée littéral

ESK :- Paquet de clé de session crypté par clé publique | Paquet de clé de session crypté par clé symétrique séquence ESK :- ESK | séquence ESK , ESK

Message crypté :- Paquet de donnée crypté symétriquement | sequence ESK , Paquet de donnée crypté symétriquement

Message signé en une passe :- Paquet de signature à une passe , Message OpenPGP , Paquet de signature correspondant

Message signé :- Paquet de signature , Message OpenPGP | Message signé en une passe

De plus, le décryptage d'un paquet de données cryptés symétriquement et la décompression d'un paquet de données compressées doit produire un message OpenPGP valide.

10.3. Signatures détachées

Certaines applications OpenPGP utilisent des signatures dites 'détachées'. Par exemple, un paquetage peut contenir deux fichiers tel que le deuxième soit la signature détachée du premier. Ces signatures détachées sont simplement des paquets de signature stockés séparément des données auquelles elles correspondent.

Chapitre 11. Formats de clés améliorés

11.1. Structure de clé

Le format d'une clé OpenPGP V3 est le suivant(les entrées entre parenthèses sont optionelles et les points de suspensions représentent des répétitions):

Clé publique RSA

- [auto-signature de révocation]
- Identifiant utilisateur [Signature ...]
- [Identifiant utilisateur [Signature ...] ...]

Chaque signature certifie la clé publique RSA ainsi que l'identifiant utilisateur la précédent. La clé public RSA peut avoir plusieurs identifiants utilisateurs et chaque identifiant peut avoir plusieurs signatures.

Le format d'une clé OpenPGP V4 utilisant deux clés publiques est similaire à l'exception que les autres clés sont ajoutées à la fin en tant que sous-clé de la clé primaire.

Clé primaire

- [auto-signature de révocation]
- [auto-signature de clé franche ...]
- Identifiant utilisateur [Signature ...]
- [Identifiant utilisateur [Signature ...] ...]
- [[sous-clé [Révocation de signature liante]signature liée à la clé primaire] ...]

Une sous-clé est toujours suivi de sa signature correspondante, établie à partir de la clé primaire dans le but de lier les deux clés ensembles. Cette signature liante peut être de format V3 ou V4, ce dernier format de clé étant bien évidemment privilégié.

Dans le schéma ci-dessus, si la signature liante d'une sous-clé a été révoquée, celle-ci peut être enlever, laissant seulement une signature.

Dans une clé possédant une clé principale et des sous-clés, la clé primaire DOIT être une clé capable de signer. Les sous-clés peuvent être de tout type. D'autres constructions de clé V4 sont également possibles, par exemple une clé RSA au format V4 à clé unique, une clé primaire DSA avec une clé de cryptage RSA, une clé primaire RSA avec des sous-clés Elgamal,..

Il est aussi possible d'avoir une sous-clé dédiée à la signature. Cela permet l'utilisation d'une clé primaire certifiant seulement les sous-clés, celle-ci étant utilisées pour le cryptage et les signatures.

11.2. Identification des clés et empreintes digitales

Pour une clé V3, l'identifiant clé de 8 octets se compose des 64 bits de poids faible du modulo public de la clé RSA.

L'empreinte digitale d'une clé V3 est créée en hachant le corps (sans le champ longueur de 2 octets) du MPI formant la clé (modulo public n, suivi par l'exposant e)avec MD5.

L'empreinte digitale d'une clé V4, codé sur 160 bits, est le résultat du hachage par la méthode SHA-1 de l'étiquette de 1 octet du paquet, suivi par la taille du paquet codée sur 2 octets et par l'intégralité du paquet de clé publique commençant au champ version. L'identifiant clé correspond au 64 bits de poids faible de l'empreinte digitale. Les champs à renseigner pour la fonction de hachage sont indiqués ci dessous, via l'exemple pour une clé DSA.

1.

    1. 0x99 (1 octet),
    2. 1 octet correspondant aux bits de poids fort de (b)-(f),
    3. 2 octet correspondant aux bits de poids fort de (b)-(f),

2. Numéro de version = 4 (1 octet),
3. Champ renseignant l'heure de création de la clé (4 octets),
4. Algorithme (1 octet): 17 = DSA (exemple),
5. Champs spécifiques à l'algorithme.

   Champs spécifiques à l'algorithme pour une clé DSA (exemple) :

6.

    1. MPI du principal p DSA,
    2. MPI du groupement DSA d'ordre q (q est le plus grand diviseur de p-1),
    3. MPI du groupement DSA correspondant au générateur g,
    4. MPI de la valeur de la clé publique DSA y (y = g**x où x est une valeur cachée).

Il est à noter que des conflits d' identifiant clé sont possibles, c'est à dire deux clés différentes avec le même identifiant. Il est à noter qu'il existe une probabilité, certes très faible, mais présente, que deux clés différentes aient la même empreinte digitale.

Il est aussi à noter que si deux clés de format V3 et V4 partagent le même contenu de clé RSA, elles auront des identifiants clés et des empreintes digitales différents.

Chapitre 12. Note sur les algorithmes

12.1. Préférences des algorithmes symétriques

La préférence dun algorithme symétrique est une liste ordonnée dalgorithmes acceptés par le propriétaire de clé. Puisquelle est fournie avec une auto-signature, il est possible pour un propriétaire de clé de posséder différentes préférences. Par exemple, Alice peut avoir le TripleDES spécifié seulement pour « alice@work.com » mais le CAST5, le BlowFish et le TripleDES de spécifiés pour « alice@home.org ».

Noter quil est aussi possible que des préférences de se situent dans la signature liante dune sous-clé.

Puisque le TripleDES est lalgorithme devant être implémenter obligatoirement, si celui-ci nest pas explicitement situé dans la liste, il y sera ajouté à la fin de manière implicite. Noter aussi quune implémentation qui nimplémentent pas la notion de préférence implémentera uniquement le TripleDES de manière implicite.

Une implémentation NE DOIT PAS utiliser un algorithme symétrique nétant pas dans la liste de préférence du destinataire. Lors dun chiffrage pour plus dun destinataire, limplémentation fournira un algorithme adéquat en prenant lintersection des préférences de ces destinataires. Noter que lalgorithme devant être implémenté obligatoirement, TripleDES , assure une intersection non nulle. Limplémentation peut utiliser nimporte quel mécanisme pour choisir un algorithme dans lintersection.

Si une implémentation peut décrypter un message que le propriétaire de clé ne possède pas dans ses préférences, il serait souhaitable que limplémentation décrypte tout de même le message, mais elle DEVRA alors OBLIGATOIREMENT avertir le propriétaire que le protocole a été violé. (Par exemple, supposons quAlice, mentionné plus haut, possède un logiciel implémentant tous les algorithmes de cette description. Néanmoins, elle préfère des sous-ensembles correspondants à son travail ou à sa résidence. Si elle reçoit un message chiffré par IDEA, qui nest pas dans ses préférences, le logiciel lavertira que quelquun lui a envoyé un message crypté par IDEA, mais dans lidéal, il le décrypterait.)

Une implémentation sefforçant de respecter la compatibilité descendante pourra considérer une clé V3 avec une auto-signature V3 comme une préférence implicite pour l IDEA, sans aucune compétence pour faire du TripleDES. Cela ne respecte pas les règles techniquement mais une implémentation PEUT violer la règle précédente dans cet unique cas et utiliser IDEA pour crypter le message, à condition que le créateur du message soit prévenu. Pourtant, dans lidéal, limplémentation suivrait la règle en générant en fait deux messages, car il est possible que limplémentation de lutilisateur dOpenPGP ne possède pas IDEA et quil ne puisse ainsi pas lire le message. Par conséquent, une implémentation PEUT, même si ce nest pas souhaitable, utiliser IDEA dans un conflit dalgorithme avec une clé V3.

12.2. Préférences des autres algorithmes

Les préférences des autres algorithmes fonctionnent de la même manière que les préférences des algorithmes symétriques dans la mesure où elles précisent quels algorithmes le propriétaire de clé accepte. Cependant, il y a deux autres cas intéressants qui appellent quelques commentaires supplémentaires, à savoir les préférences de compression et les préférences de hachage.

12.2.1. Préférences de compression

La compression est partie intégrante de PGP depuis ses débuts. OpenPGP ainsi que toutes les versions précédentes de PGP fournit la compression. Et dans cette spécification, les messages sont compressés par défaut, bien quaucune implémentation ne soit nécessaire pour cela. En conséquence, la préférence de compression

permet à un propriétaire de clé de demander à ce que le message ne soit pas compressé, vraisemblablement parce quils utilisent une implémentation minimale qui ninclut pas la compression. En outre, cela permet à un propriétaire de clé de déclarer quil peut supporter des algorithmes alternatifs.

Tout comme les préférences dalgorithme, une implémentation NE DOIT PAS utiliser un algorithme qui nest pas dans le vecteur des préférences. Si les préférences ne sont pas présentes, alors elles sont supposées être [ZIP(1), non compressé(0)].

12.2.2. Préférences des algorithmes de hachage

Généralement, le choix dun algorithme de hachage est une chose effectuée par le signataire, plutôt que par le vérificateur, car il ne sait généralement pas qui vérifiera la signature. Cependant, cette préférence autorise un protocole basé sur une facilité de signatures numériques dans les négociations.

Ainsi, si Alice sauthentifie auprès de Bob par une signature, il est normal quelle se serve dun algorithme de hachage utilisé par le logiciel de Bob. Cette préférence permet à Bob de mentionner dans sa clé quels algorithmes Alice peut utiliser.

12.3. Texte non codé

Lalgorithme 0, texte non codé, peut seulement être utilisé pour indiquer les clés secrètes qui sont stockées en clair. Les implémentations ne doivent pas utiliser le texte non codé dans les paquets de données cryptées symétriquement ; elles doivent utiliser des paquets de données littérales pour coder des données non cryptées ou littérales.

12.4. RSA

Il y a des types dalgorithmes spécifiques aux clés dédiées seulement à la signature RSA ou au cryptage RSA. Ces types sont fortement critiqués. Le sous-paquet des drapeaux de clé dans une signature est une manière bien meilleure dexprimer la même idée, et il la généralise à tous les algorithmes. Il NEST PAS SOUHAITABLE quune implémentation puisse créer une telle clé, mais elle PEUT linterpréter.

IL NEST PAS SOUHAITABLE quune implémentation implémente des clés RSA dune taille inférieure à 768 bits.

Il est permis à une implémentation de supporter RSA simplement pour assurer la compatibilité descendante ; par exemple, une telle implémentation supporterait les clés V3 à cryptographie symétrique IDEA. Noter que cest une exception par rapport aux autres règles relatives à limplémentation obligatoire. Une implémentation qui supporte le RSA dans les clés V4 DOIT OBLIGATOIREMENT implémenter les caractéristiques devant être obligatoirement implémentées.

12.5. Elgamal

Si une clé Elgamal doit servir à la fois à la signature et au cryptage, une attention particulière doit être portée lors de sa création.

Une clé Elgamal consiste en un générateur g, un modulo premier p, un exposant secret x, et une valeur publique y=g^x mod p.

Le générateur et le modulo doivent être choisis de manière à ce que la résolution du problème logarithmique discret soit insoluble. Le groupe g devrait générer le groupe multiplicatif mod p-1 ou un grand sous-groupe lui correspondant et lordre de g devrait avoir au moins un facteur premier élevé. Un bon choix serait dutiliser un nombre premier de Sophie-Germain « fort » pour choisir p, de telle sorte que p et (p-1)/2 soient premiers. En effet, ce choix est si judicieux quil SERAIT SOUHAITABLE que les implémentations le respectent, puisquil évite lattaque des petits sous-groupes.

De plus, un résultat de Bleichenbacher [BLEICHENBACHER] montre que si le générateur g possède seulement de petits facteurs premiers, et si g divise lordre du groupe quil génère, alors les signatures peuvent être falsifiées. En particulier, choisir g=2 est un mauvais choix si lordre du groupe peut être pair. Dun autre côté, un générateur de 2 est un bon choix pour une clé dédiée seulement au cryptage puisquelle augmentera sa vitesse de réalisation.

Lors de la vérification des signatures Elgamal, noter quil est important de vérifier que r et s son inférieur à p. Si cela nest pas fait, alors les signatures peuvent être facilement falsifier en utilisant des valeurs élevées de r dune longueur de deux fois environ la longueur de p. Cette attaque est aussi expliquée dans larticle de BleinBacher.

On trouvera des informations sur lutilisation sécurisée de signatures Elgamal dans [MENEZES], qui traite toutes les faiblesses décrites plus haut.

Si une implémentation autorise les signatures Elgamal, alors elle DOIT OBLIGATOIREMENT utiliser lidentifiant dalgorithme 20 pour une clé publique dElgamal pouvant signer.

Il NEST PAS SOUHAITABLE quune implémentation implémente des clés Elgamal dune taille inférieure à 768 bits. Pour une sécurité à long terme, les clés Elgamal devraient être dune longueur de 1024 bits ou plus.

12.6. DSA

Il EST SOUHAITABLE quune implémentation nimplémente pas des clés DSA dune longueur inférieur à 768. Noter que le DSA actuel est limité à une longueur maximal de clé de 1024 bits, longueur recommandée pour une utilisation sur le long terme.

12.7. Nombres réservés aux algorithmes

Un nombre d'identifiants [ID] d'algorithmes ont été réservés pour des algorithmes qui seraient utiles à utiliser dans une implémentation OpenPGP, pourtant il y a des problèmes qui empêchent l'implémenteur d'implémenter réellement l'algorithme. Ils sont signalés dans la section des algorithmes publics comme « (reserved for) ».

Avec les algorithmes réservés à clé publique, Elliptic Curve (18), ECDSA (19), et X9.4 2 (21), les paramètres nécessaires, l'ordre des paramètres ou des sémantiques ne sont pas définis.

Avec l'algorithme réservé à clé symétrique, DES/SK (6), il n'y a pas de sémantiques définies.

Les algorithmes réservés de hachage, TIGER192 (6) et HAVAL-5-160 (7), n'ont pas d'OIDs. L'algorithme réservé numéro 4, réservé pour les variantes de taille double de SHA1, n'est pas actuellement défini.

Nous avons réservé trois identifiants d'algorithme pour le standard de cryptage avancé de US NIST. Cet algorithme fonctionnera avec (au moins) des clés de 128, 192, et 256 bits. Nous prévoyons que cet algorithme sélectionné parmi les algorithmes candidats de l'année 2000.

12.8. Le mode CFB de OpenPGP

OpenPGP fait du cryptage symétrique en utilisant une variante de Cipher Feedback Mode (CFB mode). Cette section décrit la procédure qu'elle utilise en détails. Ce mode est celui qui est utilisé pour l'encryptage symétrique de paquets de données; le mécanisme utilisé pour en-crypter la clé secrète matérielle est similaire, mais décrit dans le sections ci-dessous.

Le mode CFB de OpenPGP utilise un vecteur d'initialisation (IV) constitué de zéros, et prévalue le texte en clair avec 10 octets de données aléatoires, de manière a ce que les octets 9 et 10 correspondent aux octets 7 et 8. Il effectue une CFB « resynchronisation » après avoir cryptés ces 10 octets.

Remarquez que pour un algorithme qui possède une taille de blocs supérieures a 64 bits, la fonction équivalente sera faite avec cet entière bloc. Par exemple, un algorithme de blocs de 16 octets travaillera sur 16 octets, et produira alors deux octets de vérification, et travaillera alors sur des blocs de 16 octets.

Voici la procédure étape par étape :

1. Le registre de retour (FR [Feedback Register]) est mis au IV, et ne contient que des zéros.
2. FR est encrypté pour former FRE (FR Crypté). C'est le cryptage d'une valeur constituée de zéros.
3. On fait un ou exclusif [xor] entre FRE et les 8 premiers octets de données aléatoires pré-valués dans le texte en clair pour former C1-C8, les 8 premiers octets du texte chiffré.
4. FR est chargé avec C1-C8.
5. FR est crypté pour former FRE, le cryptage des 8 premiers octets du texte chiffré.
6. On fait un ou exclusif entre les deux octets de gauche de FRE et les deux octets de données suivants qui ont été pré-fixés au texte en clair. Cela crée C9-C10, les deux octets suivants du texte chiffré.
7. (L'étape de resynchronisation) FR est chargé avec C3-C10.
8. FR est crypté pour former FRE.
9. On fait un ou exclusif entre FRE et les 8 premiers octets du texte en clair donné, maintenant que nous avons fini le cryptage des 10 octets de données pré-valués. Cela produit C11-C18, les 8 prochains octets du texte chiffré.
10. FR est chargé avec C11-C18.
11. FR est crypté pour produire FRE.
12. On effectue un ou exclusif entre FRE et les 8 prochains octets de texte en clair, pour créer les 8 prochains octets du texte chiffré. Ceux-ci sont chargés dans FR et la procédure est répétée jusqu'à ce que le texte en clair soit terminé.

Chapitre 13. Considérations sur la sécurité

Comme toutes les technologies touchant la cryptographie, vous devez vérifier la litérature courante pour déterminer si aucun algorithme utilisé ici n'a été détecté comme étant vulnérable aux attaques.

Cette spécification utilise des technologies de cryptographie à base de clé publique. On considère que le ou les personnes habilitées contrôlent qui est en possission de la portion de clé privée d'une paire de clés publique-privée.

Certaines opérations dans cette spécification implique l'utilisation de nombres aléatoires. Une source d'entropie appropriée doit être utilisée pour générer ces nombres. Voir la RFC 1750.

L'algorithme de hachage MD5 a été montré comme ayant des faiblesses (pseudo collisions dans la fonction de compression) qui font que certaines personnes déconseillent son utilisation. Elles considèrent l'algorithme SHA-1 meilleur.

Beaucoup de concepteurs de protocoles de sécurité pensent que c'est une mauvaise idée d'utiliser une seule clé pour à la fois la confidentialité (cryptage) et l'intégrité (signatures). En fait, ceci est une des motivations de l'origine du format des clés V4 qui sépare les clés de signature et les clés de cryptage. Si vous, en tant qu'implémenteur vous préconisez l'utilisation des doubles clefs, vous devez au moins être au courant de cette controverse.

L'algorithme DSA fonctionnera avec n'importe quel hachage de 160 bits, mais il est sensible à la qualité de l'algorithme de hachage, si l'algorithme de hachage est cassé, il peut provoquer la divulgation de la clé secrète. Le standard de signature numérique (DSS [Digital Signature Standard]) spécifie que DSA doit être utilisé avec SHA-1. RIPEMD-160 est considéré par beaucoup de cryptographes comme assez solide. Une implémentation doit faire attention aux algorithmes de hachages utilisés avec DSA, car un faible hachage peut non seulement permettre la contrefaçon de la signature mais peut également provoquer la divulgation de la clé secrète. Les mêmes remarques concernant la qualité de l'algorithme de hachage s'appliquent aux signatures Elgamal.

Si vous construisez un système d'authentification, le destinataire doit spécifier un algorithme de signature préféré. Cependant, le signataire serait idiot d'utiliser un algorithme faible simplement parce que le destinataire le demande.

Quelques-uns des algorithmes de cryptage mentionnés dans ce document ont été moins analysés que d'autres. Par exemple, bien que CAST5 soit actuellement considéré comme solide, il a été moins analysé que Triple-DES. Les autres algorithmes doivent être l'objet d'autres controverses.

Quelques-unes des technologies mentionnées ici sont soumises à un contrôle du gouvernement dans certains pays.

Chapitre 14. Nits d'implémentation

Cette partie est un ensemble de commentaires pour aider un implémenteur, particulièrement avec des visées sur la compatibilité arrière. Les précédentes implémentations de PGP ne sont pas compatible avec OpenPGP. Souvent les différences sont petites, mais de petites différences sont souvent plus contrariantes que de grandes différences. D'où, cette liste de problèmes et de pièges potentiels est pour un développeur qui essaie d'être compatible vers l'arrière.

- PGP 5.x n'accepte les signatures V4 que pour les clés matérielles.
- PGP 5.x ne reconnaît pas les longueurs de cinq octets dans les nouveaux formats d'en-tête ou les longueurs des sous paquets de signature.
- PGP 5.0 rejette une clé de session cryptée si la longueur de la clé diffère de l'algorithme symétrique S2K. Cela constitue un bug dans sa fonction de validation.
- PGP 5.0 ne gère pas les multiples en-têtes et pied des signatures à un passage. Le fait d'en signer une compressera le libellé signé à un passage et mettra en-tête une signature V3 au lieu de faire une signature à un passage de sous programmes imbriqués.
- En exportant une clé privée, PGP 2.x génère l'en-tête « début du bloc de clé secrète PGP » [BEGIN PGP SECRET KEY BLOCK] au lieu de « début du bloc de clé privée PGP » [Begin PGP private key block]. Toues les versions précédentes ignorent le type de données concernées, et regardent directement le type des données du paquet.
- Dans une signature signée en clair, PGP 5.0 trouvera l'algorithme correct de hachage s'il n'y a pas l'en-tête « Hash: », mais il rejettera une disparité entre l'en-tête et l'algorithme effectivement utilisé. La version « standard » (i.e. Zimmermann/Finney/ et al.) de PGP 2.x rejette l'en-tête « Hash: » et utilise MD5. Il y a un nombre de variantes améliorées de PGP 2.6.x qui ont été modifiées pour les signatures SHA-1.
- PGP 5.0 peut lire une clé RSA dans un format V4, mais peut seulement la reconnaître avec un identifiant de clé V3, et ne peut utiliser correctement qu'une clé RSA au format V3.
- Ni PGP 5.x ni PGP 6.0 ne reconnaissent les clés de signature et cryptage Elgamal. Ils manipulent seulement des clés limitées au cryptage Elgamal.
- Il y a beaucoup de façons possibles pour deux clés d'avoir le même contenu de clé, mais des empreintes différentes (et donc des identifiants de clés). Peut-être que le plus intéressant est une clé RSA qui a été mise à niveau au format V4, mais comme une empreinte V4 est construite en hachant le temps de création de la clé ainsi que d'autres choses, deux clés V4 crées à des moments différents, avec cependant le même contenu de clé, auront différentes empreintes.
- Si une implémentation utilise zlib pour intéragir avec PGP 2.x, alors le paramètre « windowBits » devra être mis à -13.

Chapitre 15. Auteurs et président du groupe de travail

Le président peut être contacté via le siège actuel : John W. Noerenberg, II Qualcomm, Inc 6455 Lusk Blvd San Diego, CA 92131 USA Téléphone : +1 619-658-3510 Mail : jwn2@qualcomm.com

Les principaux auteurs de ce mémo sont : Jon Callas Network Associates, Inc. 3965 Freedom Circle Santa Clara, CA 95054, USA Téléphone : +1 408-346-5860 Mail: jon@pgp.com, jcallas@nai.com Lutz Donnerhacke IKS GmbH Wildenbruchstr. 15 07745 Jena, Germany Téléphone : +49-3641-675642 Mail: lutz@iks-jena.de Hal Finney Network Associates, Inc. 3965 Freedom Circle Santa Clara, CA 95054, USA Mail: hal@pgp.com Rodney Thayer EIS Corporation Clearwater, FL 33767, USA Mail: rodney@unitran.com

Ce mémo s'appuie également sur beaucoup de travaux antérieurs d'un certain nombre d'autres auteurs dont : Derek Atkins, Charles Breed, Dave Del Torto, Marc Dyksterhouse, Gail Haspert, Gene Hoffman, Paul Hoffman, Raph Levien, Colin Plumb, Will Price, William Stallings, Mark Weaver, et Philip R. Zimmermann.

Chapitre 16. Références

Liste des références

Nom

`Liste des références` — Toutes les références

BLEICHENBACHER

Bleichenbacher, Daniel, "Generating ElGamal signatures without knowing the secret key", Eurocrypt 96.

Noter que la version en cours contient une erreur. Une version corrigée est disponible, au moment où nous publions, à l'endroit suivant : ftp://ftp.inf.ethz.ch/pub/publications/papers/ti/isc/elgamal.ps

BLOWFISH

Schneier, B. "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)" logiciel de cryptage rapide, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp191-204 http://www.counterpane.com/bfsverlag.html

DONNERHACKE

Donnerhacke, L., et. al, "PGP263in - an improved international version of PGP", ftp://ftp.iks-jena.de/mitarb/lutz/crypt/software/pgp/

ELGAMAL

T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985, pp. 469-472.

IDEA

Lai, X, "On the design and security of block ciphers", Séries ETH dans le traitement de l'information, J.L. Massey (editor), Vol. 1, Hartung-Gorre Verlag Knostanz, Technische Hochschule (Zurich), 1992

ISO-10646

ISO/IEC 10646-1:1993. International Standard -- Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Partie 1: Basic Multilingual Plane. UTF-8 est décrit dans l'annexe R, adopté mais pas encore publié. UTF-16 est décrit dans l'annexe Q, adopté mais pas encore publié.

MENEZES

Alfred Menezes, Paul van Oorschot, et Scott Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996.

RFC822

Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, août 1982.

RFC1423

Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, octobre 1993.

RFC1641

Goldsmith, D. and M. Davis, "Using Unicode with MIME", RFC 1641, July 1994.

RFC1750

Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

RFC1951

Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3.", RFC 1951, May 1996.

RFC1983

Malkin, G., "Internet Users' Glossary", FYI 18, RFC 1983, August 1996.

RFC1991

Atkins, D., Stallings, W. and P. Zimmermann, PGP Message Exchange Formats", RFC 1991, August 1996.

RFC2015

Elkins, M., "MIME Security with Pretty Good Privacy (PGP)", RFC 2015, October 1996.

RFC2231

Borenstein, N. and N. Freed, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.", RFC 2231, November 1996.

RFC2119

Bradner, S., "Key words for use in RFCs to Indicate Requirement Level", BCP 14, RFC 2119, March 1997.

RFC2144

Adams, C., "The CAST-128 Encryption Algorithm", RFC 2144, May 1997.

RFC2279

Yergeau., F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2279, January 1998.

RFC2313

Kaliski, B., "PKCS #1: RSA Encryption Standard version 1.5", RFC 2313, March 1998.

SAFER

Massey, J.L. "SAFER K-64: One Year Later", B. Preneel, editor, Fast Software Encryption, Second International Workshop (LNCS 1008) pp212-241, Springer-Verlag 1995

Chapitre 17. Notice complète du copyright