Groupe de travail Réseau
Request for Comments : 4880
RFC rendues obsolètes : 1991, 2440
Catégorie : sur la voie de la normalisation

Traduction Claude Brière de L'Isle

J. Callas, PGP Corporation
L. Donnerhacke, IKS GmbH
H. Finney, PGP Corporation
D. Shaw
R. Thayer
novembre 2007

Format du message OpenPGP

Statut de ce mémoire
Le présent document spécifie un protocole en cours de normalisation de l'Internet pour la communauté de l'Internet, et appelle à des discussions et suggestions pour son amélioration. Prière de se référer à l'édition en cours des "Normes officielles des protocoles de l'Internet" (STD 1) pour connaître l'état de la normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.
(La présente traduction incorpore les errata 2270, 2271, 3298, 2242, 2199, 2200, 2206, 2208, 2214, 2216, 2219, 2222, 2226, 2234, 2235, 2236, 2238, 2240, 2243, et 5491)

Résumé
Le présent document est destiné à publier toutes les informations nécessaires pour le développement d'applications interopérables sur la base du format OpenPGP. Ce n'est pas un livre de recettes décrivant pas à pas la rédaction d'une application. Il décrit seulement le format et les méthodes nécessaires pour lire, vérifier, générer, et écrire des paquets conformes pour traverser n'importe quel réseau. Il ne traite pas de la mémorisation ni des questions de mise en œuvre. Il discute cependant des questions de mise en œuvre nécessaires pour éviter des fautes de sécurité.

Le logiciel OpenPGP utilise une combinaison de forte clé publique et de chiffrement symétrique pour fournir des services de sécurité pour les communications électroniques et la mémorisation de données. Ces services incluent la confidentialité, la gestion de clés, l'authentification, et les signatures numériques. Le présent document spécifie les formats de message utilisés dans OpenPGP.

Table des matières

1. Introduction

Le présent document donne des informations sur les formats de paquet d'échange de message utilisés par OpenPGP pour fournir les fonctions de chiffrement, déchiffrement, signature, et de gestion de clés. C'est une révision de la RFC 2440, "Format de message OpenPGP", qui elle-même remplace la [RFC1991], "Formats d'échange de message PGP".

1.1 Termes

* OpenPGP : terme pour le logiciel de sécurité qui utilise PGP 5.x comme base, formalisé dans la RFC 2440 et le présent document.

* PGP *(Pretty Good Privacy)* : PGP est une famille de systèmes de logiciels développée par Philip R. Zimmermann sur laquelle se fonde OpenPGP.

* PGP 2.6.x : cette version de PGP a de nombreuses variantes, d'où le terme PGP 2.6.x. Elle utilise seulement RSA, MD5, et IDEA pour ses transformations cryptographiques. Une RFC d'information, la RFC 1991, décrit cette version de PGP.

* PGP 5.x : cette version de PGP était appelée "PGP 3" dans la communauté et est aussi le prédécesseur du présent document, la RFC 1991. Elle a de nouveaux formats et corrige un certain nombre de problèmes de la conception de PGP 2.6.x. Elle est appelée ici PGP 5.x parce que ce logiciel était la première livraison du code de base "PGP 3".

* GnuPG *(GNU Privacy Guard)* aussi appelé GPG : GnuPG est une mise en œuvre de OpenPGP qui évite tous les algorithmes compliqués. Par conséquent, les premières versions de GnuPG n'incluaient pas les clés publiques RSA. GnuPG peut ou non (suivant la version) prendre en charge IDEA ou d'autres algorithmes compliqués.

"PGP", "Pretty Good", et "Pretty Good Privacy" sont des marques commerciales de la PGP Corporation et sont utilisées avec sa permission. Le terme "OpenPGP" se réfère au protocole décrit dans le présent document et ceux qui s'y rapportent.

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

Les mots clés "utilisation privée", "allocation hiérarchique", "premier arrivé premier servi", "revue d'expert", "spécification exigée", "approbation de l'IESG", "consensus de l'IETF", et "action de normalisation" qui apparaissent dans ce document quand ils sont utilisés pour décrire l'allocation d'espace de noms sont à interpréter comme décrit dans la [RFC2434].

2. Fonctions générales

OpenPGP fournit des services d'intégrité des données pour les messages et fichiers de données en utilisant les technologies centrales suivantes :
- signatures numériques
- chiffrement
- compression
- conversion Radix-64

De plus, OpenPGP fournit des services de gestion de clés et de certificats, mais beaucoup de ces services sortent du domaine d'application du présent document.

2.1 Confidentialité par chiffrement

OpenPGP combine le chiffrement de clé symétrique et le chiffrement à clé publique pour assurer la confidentialité. Quand il est rendu confidentiel, l'objet est d'abord chiffré en utilisant un algorithme de chiffrement symétrique. Chaque clé symétrique est utilisée une seule fois, pour un seul objet. Une nouvelle "clé de session" est générée comme un nombre aléatoire pour chaque objet (parfois appelé une session). Comme elle est utilisé une seule fois, la clé de session est liée au message et transmise avec lui. Pour protéger la clé, elle est chiffrée avec la clé publique du receveur. La séquence est la suivante :

1. L'envoyeur crée un message.
2. L'OpenPGP envoyeur génère un nombre aléatoire à utiliser comme clé de session pour ce seul message.
3. La clé de session est chiffrée en utilisant chaque clé publique du receveur. Ces "clés de session chiffrées" commencent le message.
4. L'OpenPGP envoyeur chiffre le message en utilisant la clé de session, qui forme le reste du message. Noter que le message est aussi généralement compressé.
5. L'OpenPGP receveur déchiffre la clé de session en utilisant la clé privée du receveur.
6. L'OpenPGP receveur déchiffre le message en utilisant la clé de session. Si le message a été compressé, il va être décompressé.

Avec le chiffrement à clé symétrique, un objet peut être chiffré avec une clé symétrique déduite d'un mot de passe (ou autre secret partagé) ou un mécanisme en deux étapes similaire à la méthode de clé publique décrite ci-dessus, dans laquelle une clé de session est elle-même chiffrée avec un algorithme symérique à partir d'un secret partagé.

Des services de signature numérique et de confidentialité peuvent être appliqués au même message. D'abord, une signature est générée pour le message et attachée au message. Ensuite le message plus la signature est chiffré en utilisant une clé de session symétrique. Finalement, la clé de session est chiffrée en utilisant le chiffrement à clé publique et elle est ajoutée en préfixe au bloc chiffré.

2.2 Authentification par signature numérique

La signature numérique utilise un algorithme de code de hachage ou de résumé de message, et un algorithme de signature à clé publique. La séquence est la suivante :

1. L'envoyeur crée un message.
2. Le logiciel d'envoi génère un code de hachage du message.
3. Le logiciel d'envoi génère une signature à partir du code de hachage en utilisant la clé privée de l'envoyeur.
4. La signature numérique est attachée au message.
5. Le logiciel receveur garde une copie de la signature du message.
6. Le logiciel receveur génère un nouveau code de hachage pour le message reçu et le vérifie en utilisant la signature du message. Si la vérification réussit, le message est accepté comme authentique.

2.3 Compression

Les mises en œuvre de OpenPGP DEVRAIENT compresser le message après l'application de la signature mais avant le chiffrement.

Si une mise en œuvre ne met pas en œuvre la compression, ses auteurs devrait être conscients que la plupart des messages OpenPGP dans le monde sont compressés. Donc, il peut être avisé pour une mise en œuvre même restreinte en espace de mettre en œuvre la décompression, mais pas la compression.

De plus, la compression a l'effet collatéral supplémentaire que certains types d'attaques peuvent être déjouées par le fait que des données compressées légèrement altérées, sont rarement décompressées sans de sévères erreurs. Ceci n'est pas très rigoureux, mais cela fonctionne. Ces attaques peuvent être strictement empêchées en mettant en œuvre et en utilisant les codes de détection des modifications (MDC) comme décrit dans les paragraphes qui suivent.

2.4 Conversion en Radix-64

La représentation native sous-jacente de OpenPGP pour les messages chiffrés, les certificats de signature, et les clés est un flux d'octets arbitraire. Certains systèmes permettent seulement l'utilisation de blocs consistant en texte imprimable à sept bits. Pour transporter les octets binaires bruts natifs de OpenPGP sur des canaux qui ne sont pas sûrs pour les données binaires brutes, un codage imprimable de ces octets binaires est nécessaire. OpenPGP fournit le service de conversion du flux d'octets binaires bruts de 8 bits en un flux de caractères ASCII imprimables, appelé codage Radix-64 ou armure ASCII.

Les mises en œuvre DEVRAIT fournir la conversion Radix-64.

2.5 Applications de signature seule

OpenPGP est conçu pour les applications qui utilisent à la fois le chiffrement et les signatures, mais un certain nombre de problèmes sont résolus par une mise en œuvre de signature seule. Bien que la présente spécification exige le chiffrement et les signatures, il est raisonnable qu'il y ait un sous ensemble de mises en œuvre qui ne sont non conformes que si elles omettent le chiffrement.

3. Formats des éléments de données

Cette section décrit les éléments de données utilisés par OpenPGP.

3.1 Nombres scalaires

Les nombres scalaires sont non signés et sont toujours mémorisés en format gros boutien. En utilisant n[k] pour se référer au kième octet à interpréter, la valeur d'un scalaire de deux octets est ((n[0] << 8) + n[1]). La valeur d'un scalaire de quatre octets est ((n[0] << 24) + (n[1] << 16) + (n[2] << 8) + n[3]).

3.2 Entiers multiprécision

Les entiers multi précisions (MPI, *Multiprecision integer*) sont des entiers non signés utilisés pour contenir de grands entiers comme ceux utilisés dans les calculs cryptographiques.

Un MPI comporte deux parties : un scalaire de deux octets qui est la longueur du MPI en bits, suivi par une chaîne d'octets qui contient l'entier réel.

Ces octets forment un nombre gros boutien ; un nombre gros boutien peut être transformé en un MPI en y ajoutant en préfixe la longueur appropriée.

Exemples : (tous les nombres sont en hexadécimal)
La chaîne d'octets [00 01 01] forme un MPI avec la valeur 1.
La chaîne [00 09 01 FF] forme un MPI avec la valeur 511.

Règles supplémentaires :
La taille d'un MPI est ((longueurMPI + 7) / 8) + 2 octets.
Le champ Longueur d'un MPI décrit la longueur en commençant par son bit de poids fort non zéro. Donc, le MPI [00 02 01] n'est pas formé correctement. Il devrait être [00 01 01].
Les bits inutilisés d'un MPI DOIVENT être à zéro.

Noter aussi que quand un MPI est chiffré, la longueur se réfère au MPI en clair. Il peut être mal formé dans son texte chiffré.

3.3 Identifiants de clé

Un identifiant de clé (KEY_ID) est un scalaire de huit octets qui identifie une clé. Les mises en œuvre NE DEVRAIENT PAS supposer que les identifiants de clé sont uniques. La Section 12 "Formats de clé améliorés" décrit comment les identifiants de clé sont formés.

3.4 Texte

Sauf spécification contraire, le jeu de caractères pour le texte est le codage UTF-8 [RFC3629] de Unicode [ISO10646].

3.5 Champs d'heure

Un champ Heure est un nombre non signé de quatre octets contenant le nombre de secondes écoulées depuis le 1er janvier 1970 à minuit, UTC, en ignorant les secondes sautées.

3.6 Anneaux de clés

Un anneau de clés est une collection d'une ou plusieurs clés dans un fichier ou base de données. Traditionnellement, un anneau de clés est simplement une liste d'une suite de clés, mais peut être toute base de données convenable. Il sort du domaine d'application de la présente norme de discuter les détails des anneaux de clés ou autres bases de données.

3.7 Spécificateurs de chaîne à clé (S2K)

Les spécificateurs de chaîne à clé (S2K, *String-to-key*) sont utilisés pour convertir les chaînes de mot de passe en clés de chiffrement/déchiffrement de clé symétrique. Ils sont utilisés dans deux endroits actuellement : pour chiffrer la partie secrète des clés privées dans le chiffrement privé, et pour convertir les mots de passe en clés de chiffrement pour les messages à chiffrement symétrique.

3.7.1 Types de spécificateurs de chaîne à clé

Trois types de spécificateurs S2K sont actuellement pris en charge, et quelques valeurs sont réservées :

ID	Type de S2K
0	S2K simple
1	S2K salé
2	valeur réservée
3	S2K itéré et salé
100 à 110	S2K privé/experimental

Ils sont décrits dans les paragraphes 3.7.1.1 à 3.7.1.3.

3.7.1.1 S2K simple

Il hache directement la chaîne pour produire les données de clé. Voir ci-dessous comment ce hachage est fait.

Octet 0 : 0x00
Octet 1 : algorithme de hachage

Le S2K simple hache le mot de passe pour produire la clé de session. La manière dont cela est fait dépend de la taille de la clé de session (qui va dépendre du chiffrement utilisé) et de la taille du résultat de l'algorithme de hachage. Si la taille du hachage est supérieure à celle de la clé de session, les octets de poids fort (les plus à gauche) du hachage sont utilisés comme clé.

Si la taille du hachage est inférieure à la taille de la clé, plusieurs instances du contexte de hachage sont créées -- assez pour produire les données de clé requises. Ces instances sont préchargées avec 0, 1, 2, ... octets de zéros (c'est-à-dire, la première instance n'a pas de préchargement, la seconde est préchargée avec 1 octet de zéro, la troisième avec deux, et ainsi de suite).

Lorsque les données sont hachées, elles sont données indépendamment à chaque contexte de hachage. Comme les contextes ont été initialisés différemment, ils vont chacun produire un résultat de hachage différent. Une fois le mot de passe haché, les données de résultat des multiples hachages sont enchaînées, le premier hachage le plus à gauche, pour produire les données de clé, en éliminant tous les octets en excès sur la droite.

3.7.1.2 S2K salé

Cela inclut une valeur de "sel" dans le spécificateur S2K -- des données arbitraires -- qui est hachée avec la chaîne de mot de passe, pour aider à empêcher les attaques de dictionnaire.

Octet 0 : 0x01
Octet 1 : algorithme de hachage
Octets 2 à 9 : valeur de sel de 8 octets

Le S2K salé est exactement comme le S2K simple, sauf que l'entrée à la ou aux fonctions de hachage consiste en les 8 octets de sel provenant du spécificateur S2K, suivis par le mot de passe.

3.7.1.3 S2K itéré et salé

Cela inclut à la fois un sel et un compte d'octets. Le sel est combiné avec le mot de passe et la valeur résultante est hachée de façon répétée. Cela augmente encore la quantité de travail qu'un attaquant doit effectuer pour essayer des attaques de dictionnaire.

Octet 0 : 0x03
Octet 1 : algorithme de hachage
Octets 2 à 9 : valeur de sel de 8 octets
Octet 10 : compte, valeur codée de un octet

Le compte est codé dans un nombre de un octet en utilisant la formule suivante :

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

La formule ci-dessus est en langage C, où "Int32" est un type pour un entier de 32 bits, et la variable "c" est le compte codé, de l'octet 10.

S2K itéré et salé hache le mot de passe et les données de sel plusieurs fois. Le nombre total d'octets à hacher est spécifié dans le compte codé dans le spécificateur S2K. Noter que la valeur résultante du compte est un compte d'octets de combien d'octets vont être hachés, pas un compte d'itérations.

Initialement, un ou plusieurs contextes de hachage sont établis comme avec les autres algorithmes S2K, selon le nombre d'octets de données de clé nécessaires. Ensuite, l'enchaînement du sel et des données du mot de passe est répété suffisamment souvent et enchaîné. L'enchaînement est tronqué au nombre d'octets spécifié par le compte d'octets. L'enchaînement tronqué est haché. La seule exception est que si le compte d'octets est inférieur à la taille du sel plus le mot de passe, le sel complet plus le mot de passe vont être hachés même si c'est plus grand que le compte d'octets. Après le hachage, les données sont déchargées du ou des contextes de hachage, comme avec les autres algorithmes S2K.

3.7.2 Usage de chaîne à clé

Les mises en œuvre DEVRAIT utiliser des spécificateurs S2K salés ou itérés et salés, car les simples spécificateurs S2K sont plus vulnérables aux attaques de dictionnaire.

3.7.2.1 Chiffrement de clé secrète

Un spécificateur S2K peut être mémorisé dans le chiffrement secret pour spécifier comment convertir le mot de passe en une clé qui déverrouille les données secrètes. Les anciennes versions de PGP mémorisaient juste l'octet d'algorithme précédant les données secrètes ou un zéro pour indiquer que les données secrètes n'étaient pas chiffrées. La fonction de hachage MD5 était toujours utilisée pour convertir le mot de passe en une clé pour l'algorithme de chiffrement spécifié.

Pour la compatibilité, quand un spécificateur S2K est utilisé, les valeurs spéciales 254 ou 255 sont mémorisées dans la position où l'octet d'algorithme de chiffrement aurait été dans l'ancienne structure de données. Ceci est alors suivi immédiatement par un identifiant d'algorithme de chiffrement de un octet, et ensuite par le spécificateur S2K comme codé ci-dessus.

Donc, précédant les données secrètes, il va y avoir une des possibilités suivantes :
0 : les données secrètes ne sont pas chiffrées (pas de mot de passe)
255 ou 254 : suivi par l'octet d'algorithme et le spécificateur S2K
Cipher alg : utilise l'algorithme S2K simple qui utilise le hachage MD5

Cette dernière possibilité, le numéro d'algorithme de chiffrement avec une utilisation implicite de MD5, est fournie pour la rétro compatibilité ; elle PEUT être comprise, mais NE DEVRAIT PAS être générée, et est déconseillée.

Ceci est suivi par un vecteur initial de la même longueur que la taille de bloc du chiffrement pour le déchiffrement des valeurs secrètes, si elles sont chiffrées, et ensuite, les valeurs de clés secrètes elles-mêmes.

3.7.2.2 Chiffrement de message à clés symétriques

OpenPGP peut créer un paquet de clé de session chiffrée (ESK, *Encrypted Session Key*) à clé symétrique devant un message. C'est utilisé pour permettre que les spécificateurs S2K soient utilisés pour la conversion du mot de passe ou pour créer des messages avec un mélange d'ESK à clé symétrique et d'ESK à clé publique. Cela permet qu'un message soit déchiffré avec un mot de passe ou une paire de clés publiques.

PGP 2.X a toujours utilisé IDEA avec une simple conversion de chaîne à clé quand on chiffre un message avec un algorithme symétrique. Ceci est déconseillé, mais PEUT être utilisé pour la rétro compatibilité.

4. Syntaxe de paquet

Cette Section décrit les paquets utilisés par OpenPGP.

4.1 Généralités

Un message OpenPGP est construit à partir d'un certain nombre d'enregistrements qui sont traditionnellement appelés des paquets. Un paquet est un tronçon de données qui a une étiquette spécifiant sa signification. Un message OpenPGP, un anneau de clés, un certificat, et ainsi de suite, consiste en un certain nombre de paquets. Certains 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 consiste en un en-tête de paquet, suivi par le corps de paquet. L'en-tête de paquet est de longueur variable.

4.2 En-tête de paquet

Le premier octet de l'en-tête de paquet code le format de paquet et "l'étiquette de paquet". Il détermine le format de l'en-tête et note le contenu du paquet. Le reste de l'en-tête de paquet est la longueur du corps du paquet.

Noter que le bit de poids fort est le bit de gauche, appelé le bit 7. Un gabarit pour ce bit est 0x80 en hexadécimal.

```
     +---------------+
PTag |7 6 5 4 3 2 1 0|
     +---------------+
```

Bit 7 -- Toujours un
Bit 6 -- Format du nouveau paquet si il est à un

PGP 2.6.x utilise seulement des paquets d'ancien format. Donc, le logiciel qui interopère avec ces versions de PGP doit seulement utiliser des paquets au vieux format. Si l'interopérabilité n'est pas un problème, le nouveau format de paquet est RECOMMANDÉ. Noter que les paquets de l'ancien format ont des étiquettes de paquet de quatre bits, et les paquets au nouveau format en ont six ; certaines caractéristiques ne peuvent pas être utilisées et sont quand même rétro compatibles.

Noter aussi que les paquets avec une étiquette supérieure ou égale à 16 DOIVENT utiliser des paquets au nouveau format. Les paquets à l'ancien format peuvent seulement exprimer des étiquettes égales ou inférieures à 15.

Les paquets à l'ancien format contiennent :
Bits 5-2 -- étiquette de paquet
Bits 1-0 -- longueur-type

Les paquets au nouveau format contiennent :
Bits 5-0 -- étiquette de paquet

4.2.1 Longueurs de paquet de vieux format

La signification de longueur-type dans le vieux format de paquets est :
0 -le corps du paquet est long d'un octet. L'en-tête est de 2 octets.
1 - le corps du paquet est long de deux octets. L'en-tête est de 3 octets.
2 - le corps du paquet est long de quatre octets. L'en-tête est de 5 octets.
3 - le paquet est de longueur indéterminée. L'en-tête est de 1 octet, et la mise en œuvre doit déterminer la longueur du paquet. Si le paquet est dans un fichier, cela signifie que le paquet s'étend jusqu'à la fin du fichier. En général, une mise en œuvre NE DEVRAIT PAS utiliser des paquets de longueur indéterminée sauf quand la fin des données va être claire d'après le contexte, et même alors il est préférable d'utiliser une longueur définie, ou un en-tête au nouveau format. Les en-têtes au nouveau format décrits ci-dessous ont un mécanisme pour coder avec précision les données de longueur indéterminée.

4.2.2 Longueurs de paquet de nouveau format

Il ya quatre façons possibles de coder la longueur de paquets au nouveau format :
1. Un en-tête Longueur de corps de un octet code les longueurs de corps de paquet jusqu'à 191 octets.
2. Un en-tête Longueur de corps de deux octets code les longueurs de corps de paquet de 192 à 8383 octets.
3. Un en-tête Longueur de corps de cinq octets code les longueurs de corps de paquet jusqu'à 4 294 967 295 (0xFFFFFFFF) octets de long. (Cela code en fait un nombre scalaire de quatre octets.)
4. Quand la longueur du corps de paquet n'est pas connue à l'avance par l'émetteur, des en-têtes Longueur partielle de corps codent un paquet de longueur indéterminée, en faisant effectivement un flux.

4.2.2.1 Longueurs d'un octet

Un en-tête Longueur de corps de un octet code une longueur de 0 à 191 octets. Ce type d'en-tête de longueur est reconnu parce que la valeur de un octet est inférieure à 192. La longueur de corps est égale à : bodyLen = 1st_octet;

4.2.2.2 Longueurs de deux octets

Un en-tête Longueur de corps de deux octets code une longueur de 192 à 8383 octets. Elle est reconnue parce que son premier octet est dans la gamme 192 à 223. La longueur de corps est égale à :

bodyLen = ((1st_octet - 192) << 8) + (2nd_octet) + 192

4.2.2.3 Longueurs de cinq octets

Un en-tête Longueur de corps de cinq octets consiste en un seul octet contenant la valeur 255, suivie par un scalaire de quatre octets. La longueur de corps est égale à :

bodyLen = (2nd_octet << 24) | (3rd_octet << 16) | (4th_octet << 8) | 5th_octet

Cet ensemble de base de longueurs de un, deux, et cinq octets est aussi utilisé à l'intérieur de certains paquets.

4.2.2.4 Longueurs de corps partiel

Un en-tête Longueur de corps partielle est long d'un octet et code la longueur d'une partie du paquet de données. Cette longueur est une puissance de 2, de 1 à 1 073 741 824 (2 à la puissance 30). Il est reconnu par sa valeur de un octet qui est supérieure ou égale à 224, et inférieure à 255. La longueur de corps partielle est égale à :

partialBodyLen = 1 << (1st_octet & 0x1F);

Chaque en-tête Longueur de corps partielle est suivi par une portion des données du corps de paquet. L'en-tête Longueur de corps partielle spécifie la longueur de cette portion. Un autre en-tête Longueur (un octet, deux octets, cinq octets, ou partiel) suit cette portion. Le dernier en-tête Longueur dans le paquet NE DOIT PAS être un en-tête Longueur de corps partielle. Les en-têtes Longueur de corps partielle peuvent seulement être utilisés pour les parties non finales du paquet.

Noter aussi que le dernier en-tête Longueur de corps peut être un en-tête de longueur zéro.

Une mise en œuvre PEUT utiliser des longueurs de corps partielles pour des paquets de données, qu'ils soient littéraux, compressés, ou chiffrés. La première longueur partielle DOIT être d'au moins 512 octets. Les longueurs de corps partielles NE DOIVENT PAS être utilisées pour d'autre type de paquet.

4.2.3 Exemples de longueur de paquet

Ces exemples montrent les façons dont les paquets au nouveau format peuvent coder les longueurs de paquet.

Un paquet de longueur 100 peut avoir sa longueur codée dans un octet : 0x64. Ceci est suivi par 100 octets de données.

Un paquet de longueur 1723 peut avoir sa longueur codée dans deux octets : 0xC5, 0xFB. Cet en-tête est suivi par les 1723 octets de données.

Un paquet de longueur 100000 peut avoir sa longueur codée dans cinq octets : 0xFF, 0x00, 0x01, 0x86, 0xA0.

Il pourrait aussi être codé dans le flux d'octets suivant : 0xEF, d'abord 32768 octets de données ; 0xE1, ensuite deux octets de données ; 0xE0, ensuite un octet de données ; 0xF0, ensuite 65536 octets de données ; 0xC5, 0xDD, enfin 1693 octets de données. Ceci est juste un codage possible, et de nombreuses variations sur la taille des en-têtes Longueur parteille de corps sont possibles, tant qu'un en-tête régulier Longueur de corps code la dernière portion des données.

Noter que dans toutes ces explications, la longueur totale du paquet est la longueur du ou des en-têtes plus la longueur du corps.

4.3 Étiquettes de paquet

L'étiquette de paquet note quel type de paquet contient le corps. Noter que les en-têtes à l'ancien format peuvent seulement avoir des étiquettes de moins de 16, tandis que les en-têtes au nouveau format peuvent avoir des étiquettes allant jusqu'à 63. Les étiquettes définies (en décimal) sont les suivantes :

0 - Réservé - une étiquette de paquet NE DOIT PAS avoir cette valeur
1 - Paquet Clé de session chiffrée à clé publique
2 - Paquet Signature
3 - Paquet Clé de session chiffrée à clé symétrique

4 - Paquet Signature à une passe
5 - Paquet Clé secrète
6 - Paquet Clé publique
7 - Paquet Sous clé secrète
8 - Paquet Données compressées
9 - paquet Données chiffrées symétriquement
10 - Paquet Marqueur
11 - Paquet Données littérales
12 - Paquet Confiance
13 - Paquet Identifiant d'utilisateur
14 - Paquet Sous clé publique
17 - Paquet Attribut d'utilisateur
18 - Paquet Données à chiffrement symétrique et protégées en intégrité
19 - Paquet Code de détection de modification
60 à 63 - Valeurs privées ou expérimentales

5. Types de paquet

5.1 Paquets Clé de session chiffrée à clé publique (étiquette 1)

Un paquet Clé de session chiffrée à clé publique (PKESK) détient la clé de session utilisée pour chiffrer un message. Zéro, un ou plusieurs paquets Clé de session chiffrée à clé publique et/ou paquets Clé de session chiffrée à clé symétrique peuvent précéder un paquet Données chiffrées symétriquement, qui contient un message chiffré. Le message est chiffré avec la clé de session, et la clé de session est elle-même chiffrée et mémorisée dans le ou les paquets Clé de session chiffrée. Le paquet Données chiffrées symétriquement est précédé par un paquet Clé de session chiffrée à clé publique pour chaque clé OpenPGP avec laquelle le message est chiffré. Le receveur du message trouve un paquet Clé de session chiffrée à clé publique qui contient la clé de session chiffrée avec la clé publique du receveur, déchiffre la clé de session, et ensuite utilise la clé de session pour déchiffrer le message.

Le corps de ce paquet consiste en :
- Un nombre d'un octet du numéro de version du type de paquet. La valeur actuellement définie pour la version de paquet est 3.
- Un nombre de huit octets qui donne l'identifiant de clé de la clé publique avec laquelle la clé de session est chiffrée. Si la clé de session est chiffrée avec une sous clé, alors l'identifiant de clé de cette sous clé est utilisé à la place de l'identifiant de clé de la clé principale.
- Un nombre d'un octet qui donne l'algorithme de clé publique utilisé.
- Une chaîne d'octets qui est la clé de session chiffrée. Cette chaîne prend le reste du paquet, et son contenu dépend de l'algorithme de clé publique utilisé.

Champs spécifiques de l'algorithme pour le chiffrement RSA
- entier multi précisions (MPI) de la valeur RSA chiffrée m**e mod n.

Champs spécifiques de l'algorithme pour le chiffrement Elgamal :
- MPI de la valeur Elgamal (Diffie-Hellman) g**k mod p.
- MPI de la valeur Elgamal (Diffie-Hellman) m * y**k mod p.

La valeur "m" dans les formules ci-dessus est déduite de la clé de session comme suit. D'abord, la clé de session est munie en préfixe d'un identifiant d'algorithme d'un octet qui spécifie l'algorithme de chiffrement symétrique utilisé pour chiffrer le paquet de données chiffrées symétriquement suivant. Ensuite une somme de contrôle de deux octets est ajoutée, qui est égale à la somme des octets de la clé de session précédente, non inclus l'identifiant d'algorithme, modulo 65536. Cette valeur est alors codée comme décrit dans le codage de bloc PKCS n° 1 EME-PKCS1-v1_5 au paragraphe 7.2.1 de la [RFC3447] pour former la valeur "m" utilisée dans les formules ci-dessus. Voir au paragraphe 13.1 du présent document des notes sur l'utilisation de PKCS n° 1 dans OpenPGP.

Noter que quand une mise en œuvre forme plusieurs paquets Clé de session chiffrée à clé publique avec une seule clé de session, formant un message qui peut être déchiffré par plusieurs clés, la mise en œuvre DOIT faire un nouveau codage PKCS n° 1 pour chaque clé.

Une mise en œuvre PEUT accepter ou utiliser un identifiant de clé de zéro comme un identifiant de clé à "caractère générique" ou "spéculatif". Dans ce cas, la mise en œuvre receveuse va essayer toutes les clés privées disponibles, à la recherche d'une clé de session déchiffrée valide. Ce format aide à réduire l'analyse de trafic des messages.

5.2 Paquet Signature (étiquette 2)

Un paquet Signature décrit un lien entre une clé publique et des données. Les signatures les plus courantes sont la signature d'un fichier ou d'un bloc de texte, et une signature qui est la certification d'un identifiant d'utilisateur.

Deux versions de paquet Signature sont définies. La version 3 (V3) fournit les informations de base de signature, tandis que la version 4 (V4) fournit un format extensible avec des sous paquets qui peuvent spécifier plus d'informations sur la signature. PGP 2.6.x accepte seulement les signatures version 3.

Les mises en œuvre DEVRAIENT accepter les signatures V3. Les mises en œuvre DEVRAIT générer des signatures V4.

Noter que si une mise en œuvre crée un message chiffré et signé qui est chiffré avec une clé V3, il est raisonnable de créer une signature V3.

5.2.1 Types de signature

Il y a un certain nombre de significations possibles à une signature, qui sont indiquées dans un octet Type de signature dans toute signature. Noter que le vague de ces significations n'est pas une erreur, mais une caractéristique du système. Comme OpenPGP donne l'autorité finale sur la validité au receveur d'une signature, il se peut que l'action d'un signataire soit moins rigoureuse que l'action positive d'une autre autorité. Voir au paragraphe 5.2.4, "Calcul de signatures", des informations détaillées sur la façon de calculer et vérifier les signatures de chaque type.

Ces significations sont les suivantes :

0x00 : Signature d'un document binaire. Cela signifie que le signataire le possède, l'a créé, ou certifie qu'il n'a pas été modifié.

0x01 : Signature d'un document de texte canonique. Cela signifie que le signataire le possède, l'a créé, ou certifie qu'il n'a pas été modifié. La signature est calculée sur les données du texte avec ses terminaisons de ligne converties en <CR><LF>.

0x02 : Signature autonome. Cette signature est une signature du seul contenu de son propre sous paquet. Elle est calculée comme une signature sur un document binaire de longueur zéro. Noter qu'une signature V3 autonome n'a pas de sens.

0x10 : Certification générique d'un identifiant d'utilisateur et paquet de clé publique. Le producteur de cette certification ne fait aucune assertion particulière sur la qualité de la vérification de celui qui certifie que le possesseur de la clé est en fait la personne décrite dans l'identifiant d'utilisateur.

0x11 : Certification personnelle d'un identifiant d'utilisateur et de paquet de clé publique. Le producteur de cette certification n'a pas fait de vérification de la prétention que le possesseur de cette clé est l'identifiant d'utilisateur spécifié.

0x12 : Certification fortuite d'un identifiant d'utilisateur et de paquet de clé publique. Le producteur de cette certification a fait une vérification fortuite de la revendication d'identité.

0x13 : Certification positive d'un identifiant d'utilisateur et de paquet de clé publique. Le producteur de cette certification a fait une vérification substancielle de la revendication d'identité.

La plupart des mises en œuvreOpenPGP font leurs "signatures de clé" comme des certifications 0x10. Certaines mises en œuvre peuvent produire des certifications 0x11-0x13, mais peu d'entre elles différencient les types.

0x18 : Signature de lien de sous clé. Cette signature est une déclaration par la clé de signature de niveau supérieur qui indique qu'elle possède la sous clé. Cette signature est calculée directement sur la clé et la sous clé principales, et non sur un identifiant d'utilisateur ou autres paquets. Une signature qui lie une sous clé de signature DOIT avoir un sous paquet Signature incorporé dans cette signature de lien qui contient une signature 0x19 faite par la sous clé de signature sur la clé et sous clé principales.

0x19 : Signature de lien de clé principale. Cette signature est une déclaration par une sous clé de signature, qui indique qu'elle est possédée par la clé principale. Cette signature est calculée de la même façon qu'une signature 0x18 : directement sur la clé et sous clé principales, et non sur un identifiant d'utilisateur ou autres paquets.

0x1F : Signature directement sur une clé. Cette signature est calculée directement sur une clé. Elle lie les informations dans les sous paquets Signature à la clé, et est appropriée pour être utilisée pour les sous paquets qui donnent des informations sur la clé, comme le sous paquet Clé de révocation. Elle est aussi appropriée pour des déclarations que des certificateurs non auto proclamés veulent faire sur la clé elle-même, plutôt que sur le lien entre une clé et un nom.

0x20 : Signature de révocation de clé. La signature est calculée directement sur la clé révoquée. Une clé révoquée n'est pas à utiliser. Seules les signatures de révocation par la clé révoquée, ou par une clé de révocation autorisée, devraient être considérées comme des signatures de révocation valides.

0x28 : Signature de révocation de sous clé. La signature est calculée directement sur la sous clé révoquée. Une sous clé révoquée n'est pas à utiliser. Seules les signatures de révocation par la clé de signature de niveau supérieur qui est liée à cette sous clé, ou par une clé de révocation autorisée, devraient être considérées comme des signatures de révocation valides.

0x30 : Signature de révocation de certification. Cette signature révoque une signature de certification d'identifiant d'utilisateur antérieure (classe de signature de 0x10 à 0x13) ou une signature de clé directe (0x1F). Elle devrait être produite par la même clé qui a produit la signature révoquée ou une clé de révocation autorisée. La signature est calculée sur les mêmes données que le certificat qu'elle révoque, et devrait avoir une date de création postérieure à celle de ce certificat.

0x40 : Signature d'horodatage. Cette signature n'a de signification que pour l'horodatage qui y est contenu.

0x50 : Signature de confirmation de tiers. Cette signature est une signature sur un autre paquet Signature OpenPGP. Elle est analogue à un sceau notarié sur les données signées. Une signature de tiers DEVRAIT inclure un ou des sous paquets Cible de signature pour fournir une identification aisée. Noter qu'on dit bien DEVRAIT. Il y a des utilisations plausibles pour cela (comme une partie aveugle qui voit seulement la signature, et non la clé ou le document source) qui ne peuvent pas inclure de sous paquet cible.

5.2.2 Format de paquet Signature version 3

Le corps d'un paquet Signature version 3 contient :
- Le numéro de version (3) d'un octet.
- Six octets avec les trois éléments suivants :
  - Un octet de longueur des deux éléments restants qui sont inclus dans le hachage. DOIT être 5.
  - Identifiant de clé de huit octets du signataire.
  - Un octet d'algorithme de clé publique.
- Un octet d'algorithme de hachage.
- Un champ de deux octets contenant les 16 premiers bits de la valeur du hachage signé.
- Un ou plusieurs entiers multi précisions constituant la signature. Cette portion est spécifique de l'algorithme, comme décrit ci-dessous.

L'enchaînement des données à signer, du type de signature, et l'heure de création du paquet Signature (5 octets supplémentaires) est haché. La valeur de hachage résultante est utilisée dans l'algorithme de signature. Les 16 bits de poids fort (deux premiers octets) du hachage sont inclus dans le paquet Signature pour fournir une vérification rapide pour rejeter des signatures invalides.

Champs spécifiques de l'algorithme pour les signatures RSA :
- entier multi précision (MPI) de la valeur de signature RSA m**d mod n.

Champs spécifiques de l'algorithme pour les signatures DSA :
- MPI de la valeur DSA r.
- MPI de la valeur DSA s.

Le calcul de signature se fonde sur un hachage des données signées, comme décrit ci-dessus. Les détails du calcul sont différents pour les signatures DSA et pour les signatures RSA.

Avec les signatures RSA, la valeur du hachage est codée en utilisant le type de codage PKCS n° 1 EMSA-PKCS1-v1_5 comme décrit au paragraphe 9.2 de la RFC 3447. Cela exige d'insérer la valeur du hachage comme une 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 des algorithmes de hachage actuellement définis sont les suivantes :
- MD5 : 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x02, 0x05
- RIPEMD-160 : 0x2B, 0x24, 0x03, 0x02, 0x01

- SHA-1 : 0x2B, 0x0E, 0x03, 0x02, 0x1A
- SHA224 : 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x04
- SHA256 : 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x01
- SHA384 : 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x02
- SHA512 : 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x03

Les identifiants d'objet (OID, *Object Identifier*) ASN.1 sont les suivants :
- MD5 : 1.2.840.113549.2.5
- RIPEMD-160 : 1.3.36.3.2.1
- SHA-1 : 1.3.14.3.2.26
- SHA224 : 2.16.840.1.101.3.4.2.4
- SHA256 : 2.16.840.1.101.3.4.2.1
- SHA384 : 2.16.840.1.101.3.4.2.2
- SHA512 : 2.16.840.1.101.3.4.2.3

Les préfixes de hachage complets pour eux sont les suivants :
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
SHA224 : 0x30, 0x2d, 0x30, 0x0d, 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x04, 0x05, 0x00, 0x04, 0x1C
SHA256 : 0x30, 0x31, 0x30, 0x0d, 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x01, 0x05, 0x00, 0x04, 0x20
SHA384 : 0x30, 0x41, 0x30, 0x0d, 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x02, 0x05, 0x00, 0x04, 0x30
SHA512 : 0x30, 0x51, 0x30, 0x0d, 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x03, 0x05, 0x00, 0x04, 0x40

Les signatures DSA DOIVENT utiliser des hachages égaux en taille au nombre de bits de q, le groupe généré par la valeur de générateur de la clé DSA.

Si la taille du résultat du hachage choisi est supérieure au nombre de bits de q, le résultat du hachage est tronqué pour tenir dans ce nombre de bits, en prenant le nombre de bits les plus à gauche égal au nombre de bits de q. Ce résultat de fonction de hachage (éventuellement tronqué) est traité comme un nombre et est utilisé directement dans l'algorithme de signature DSA.

5.2.3 Format de paquet Signature version 4

Le corps d'un paquet Signature version 4 contient :
- Un octet de numéro de version (4).
- Un octet de type de signature.
- Un octet d'algorithme de clé publique.
- Un octet d'algorithme de hachage.
- Deux octets de compte d'octet scalaire pour les données du sous paquet haché suivant. Noter que ceci est la longueur en octets de tous les sous paquets hachés ; un pointeur incrémenté par ce nombre va sauter les sous paquets hachés.
- Ensemble de données de sous paquet haché (zéro un ou plusieurs sous paquets).
- Deux octets de compte d'octet scalaire pour les données du sous paquet non haché suivant. Noter que ceci est la longueur en octets de tous les sous paquets non hachés ; un pointeur incrémenté par ce nombre va sauter les sous paquets non hachés.
- Ensemble de données de sous paquet non haché (zéro un ou plusieurs sous paquets).
- Champ de deux octets contenant les 16 bits de poids fort de la valeur du hachage signé.
- Un ou plusieurs entiers multi précisions comprenant la signature.

Cette portion est spécifique de l'algorithme, comme décrit ci-dessus.

L'enchaînement des données signées et des données de signature depuis le numéro de version jusqu'aux données de sous paquet haché (inclus) plus en en-queue de six octets (voir le paragraphe 5.2.4) est haché. La valeur de hachage résultante est convertie en la signature. Les 16 bits de gauche du hachage sont inclus dans le paquet Signature pour fournir un essai rapide pour rejeter des signatures invalides.

Il y a deux champs qui constituent les sous paquets Signature. Le premier champ (avec les parties précédentes de la signature) est inclus dans le hachage, tandis que le second ne l'est pas. Le second ensemble de sous paquets n'est pas protégé cryptographiquement par la signature et devrait inclure seulement des informations pour avis.

Les algorithmes pour convertir le résulat de la fonction de hachage en une signature sont décrits plus loin.

5.2.3.1 Spécification du sous paquet Signature

Un ensemble de données de sous paquet consiste en zéro, un ou plusieurs sous paquets Signature. Dans les paquets Signature, l'ensemble de données de sous paquet est précédé par un compte scalaire de deux octets de la longueur en octets de tous les sous paquets. Un pointeur incrémenté de ce nombre va sauter l'ensemble de données de sous paquet.

Chaque sous paquet consiste en un en-tête de sous paquet et un corps. L'en-tête consiste en :
- la longueur du sous paquet (1, 2, ou 5 octets),
- le type de sous paquet (1 octet),

et est suivi par les données spécifiques du sous paquet.

La longueur inclut l'octet de type mais pas sa longueur. Son format est similaire au "nouveau" format d'en-tête de paquet Longueur, mais ne peut pas avoir de longueur de corps partielle. C'est-à-dire :

si le premier octet < 192, alors
    longueurDeLongueur = 1
    longueurDeSousPaquet = 1er_octet
si le 1er octet ≥ 192 et < 255, alors
    longueurDeLongueur = 2
    longueurDeSousPaquet = ((1er_octet - 192) << 8) + (2nd_octet) + 192
si le 1er octet = 255, alors
    longueurDeLongueur = 5
    longueurDeSousPaquet = [scalaire de quatre octets commençant au 2nd_octet]

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

0 = Réservé
1 = Réservé
2 = Heure de création de signature
3 = Heure d'expiration de signature
4 = Certification exportable
5 = Signature de confiance
6 = Expression régulière
7 = Révocable
8 = Réservé
9 = Heure d'expiration de clé
10 = Fourre-tout pour la rétro compatibilité
11 = Algorithmes symétriques préférés
12 = Clé de révocation
13 = Réservé
14 = Réservé
15 = Réservé
16 = Producteur
17 = Réservé
18 = Réservé
19 = Réservé
20 = Données de notation
21 = Algorithmes de hachage préférés
22 = Algorithmes de compression préférés
23 = Préférences de serveur de clés
24 = Serveur de clés préféré
25 = Identifiant d'utilisateur principal
26 = URI de politique
27 = Fanions de clé
28 = Identifiant d'utilisateur du signataire
29 = Raison de la révocation
30 = Caractéristiques
31 = Cible de signature
32 = Signature incorporée
100 à 110 = Privé ou expérimental

Le bit 7 du type de sous paquet est le bit "critique". Si il est établi, il note que le sous paquet est critique pour l'évaluateur de la signature. Si un sous paquet est marqué critique mais n'est pas reconnu par le logicel d'évaluation, l'évaluateur DEVRAIT considérer que la signature est erronée.

Une mise en œuvre DEVRAIT ignorer tout sous paquet d'un type non reconnu.

Un évaluateur peut "reconnaître" un sous paquet, mais ne pas le mettre en œuvre. L'objet du bit critique est de permettre au signataire de dire à l'évaluateur qu'il préfèrait une nouvelle caractéristique, inconnue, pour générer une erreur plutôt que d'être ignoré.

Les mises en œuvre DEVRAIT mettre en œuvre les trois sous paquets d'algorithme préféré (11, 21, et 22), ainsi que le sous paquet "Raison de révocation". Noter, cependant, que si une mise en œuvre choisit de ne pas mettre en œuvre certaines des préférences, il est exigé qu'elle se comporte d'une manière polie pour respecter les souhaits des utilisateurs qui mettent en œuvre ces préférences.

5.2.3.2 Types de sous paquet Signature

Un certain nombre de sous paquets sont actuellement définis. Certains sous paquets s'appliquent à la signature elle-même et certains sont des attributs de la clé. Les sous paquets qui sont trouvés sur une auto signature sont placés sur un certificat fait par la clé elle-même. Noter qu'une clé peut avoir plus d'un identifiant d'utilisateur, et donc peut avoir plus d'une auto signature, et différents sous paquets.

Un sous paquet peut être trouvé sans les sections de sous paquet hachées ou non hachées d'une signature. Si un sous paquet n'est pas haché, alors les informations qu'il contient ne peuvent pas être considérées comme définitives parce que il ne fait pas partie de la signature proprement dite.

5.2.3.3 Notes sur les auto signatures

Une auto signature est un lien de signature fait par la clé à laquelle se réfère la signature. Il y a trois types d'auto signatures, les signatures de certification (types 0x10-0x13) la signature de clé directe (type 0x1F) et la signature de lien de sous clé (type 0x18). Pour les auto signatures de certification, chaque identifiant d'utilisateur peut avoir une auto signature, et donc différents sous paquets dans ces auto signatures. Pour les signatures de lien de sous clé, chaque sous clé a en fait une auto signature. Les sous paquets qui apparaissent dans une auto signature de certification s'appliquent à l'identifiant d'utilisateur, et les sous paquets qui apparaissent dans l'auto signature de sous clé s'appliquent à la sous clé. Enfin, les sous paquets sur la signature de clé directe s'appliquent à la clé entière.

Le logiciel de mise en œuvre devrait interpréter les sous paquets de préférence d'auto signature aussi étroitement que possible. Par exemple, si on suppose qu'une clé a deux noms d'utilisateur, Alice et Bob. On suppose que Alice préfère l'algorithme symétrique CAST5, et que Bob préfère IDEA ou TripleDES. Si le logiciel localise cette clé via le nom d'Alice, alors l'algorithme préféré sera CAST5 ; si le logiciel localise la clé via le nom de Bob, alors l'algorithme préféré sera IDEA. Si la clé est localisée par l'identifiant de clé, l'algorithme l'identifiant d'utilisateur principal de la clé fournit l'algorithme symétrique préféré.

Révoquer une auto signature ou lui permettre d'arriver à expiration une signification qui varie avec le type de signature. Révoquer l'auto signature sur un identifiant d'utilisateur retire effectivement ce nom d'utilisateur. L'auto signature est une déclaration, "Mon nom X est lié à ma clé de signature K" et est corroborée par d'autres certificats de l'utilisateur. Si un autre utilisateur révoque ses certificats, cela dit effectivement qu'il ne croit plus que ce nom et cette clé sont liés. De même, si les utilisateurs eux-mêmes révoquent leur auto signature, alors les utilisateurs ne vont plus aller avec ce nom, ne plus avoir cette adresse de messagerie, etc. Révoquer une signature de lien retire effectivement cette sous clé. Révoquer une signature de clé directe annule cette signature. Voir les détails pertinents au paragraphe 5.2.3.23, sous paquet Raison de révocation.

Comme une auto signature contient des informations importantes sur l'utilisation de la clé, une mise en œuvre DEVRAIT permettre à l'utilisateur de réécrire l'auto signature, et les informations importantes qu'elle contient, comme les préferences et le moment d'expiration de la clé.

Il est de bonne pratique de vérifier qu'une auto signature importée dans une mise en œuvre n'annonce pas des caractéristiques que la mise en œuvre ne prend pas en charge, en réécrivant la signature si c'est approprié.

Une mise en œuvre qui rencontre plusieurs auto signatures sur le même objet peut résoudre l'ambiguïté de toute façon qui paraît convenir , mais il est RECOMMANDÉ que la priorité soit donnée à la plus récente auto signature.

5.2.3.4 Heure de création de signature

(Champ d'heure de 4 octets). L'heure de la signature DOIT être présente dans la zone hachée.

5.2.3.5 Producteur

(Identifiant de clé de 8 octets). L'identifiant de clé OpenPGP de la clé produisant la signature.

5.2.3.6 Heure d'expiration de clé

(Champ d'heure de 4 octets). La période de validité de la clé. C'est le nombre de secondes après l'heure de création de la clé où la clé expire. Si ce champ n'est pas présent ou a une valeur de zéro, la clé n'expire jamais. Ceci ne se trouve que sur une auto signature.

5.2.3.7 Algorithmes symétriques préférés

(Dispositif de valeurs de un octet). Les numéros d'algorithmes symétriques qui indiquent quels algorithmes le détenteur de la clé préfère utiliser. Le corps de sous paquet est une liste ordonnée d'octets avec le préféré en premier. On suppose que seuls les algorithmes de la liste sont pris en charge par le logiciel du receveur. Les numéros d'algorithmes sont à la Section 9. Cela ne se trouve que sur une auto signature.

5.2.3.8 Algorithmes de hachage préférés

(Dispositif de valeurs de un octet). Les numéros d'algorithmes de résumé de message qui indiquent quels algorithmes préfère recevoir le détenteur de la clé. Comme les algorithmes symétriques préférés, la liste est ordonnée. Les numéros d'algorithmes sont à la Section 9. Cela ne se trouve que sur une auto signature.

5.2.3.9 Algorithmes de compression préférés

(Dispositif de valeurs de un octet). Les numéros d'algorithmes de compression qui indiquent quels algorithmes préfère utiliser le détenteur de la clé. Comme les algorithmes symétriques préférés, la liste est ordonnée. Les numéros d'algorithmes sont à la Section 9. Si ce sous paquet n'est pas inclus, ZIP est préféré. Un zéro note que des données non compressées sont préférées ; le logiciel du détenteur de la clé pourrait ne pas avoir de logiciel de compression dans cette mise en œuvre. Cela ne se trouve que sur une auto signature.

5.2.3.10 Heure d'expiration de signature

(Champ d'heure de 4 octets). La période de validité de la signature. C'est le nombre de secondes après l'heure de création de la signature où la signature va expirer. Si il n'est pas présent ou a une valeur de zéro, elle n'expire jamais.

5.2.3.11 Certification exportable

(1 octet d'exportabilité, 1 pour exportable, 0 pour non exportable). Ce sous paquet note si une signature de certification est "exportable", pour être utilisée par d'autres utilisateurs que le producteur de la signature. Le corps du paquet contient un fanion booléen qui indique si la signature est exportable. Si ce sous paquet n'est pas présent, la certification est exportable ; cela est équivalent à un fanion contenant un 1.

Les certifications non exportables, ou "locales", sont des signatures faites par un utilisateur pour marquer une clé comme valide au sein de cette seule mise en œuvre d'utilisateur.

Donc, quand une mise en œuvre prépare une copie d'utilisateur d'une clé pour la transporter à un autre utilisateur (c'est le processus "d'exportation" de la clé) toutes les signatures de certification locales sont supprimées de la clé.

Le receveur d'une clé transportée "l'importe", et de même arrange toutes certifications locales. En fonctionnement normal, il ne va y en avoir aucune, en supposant que l'importation est effectuée sur une clé exportée. Cependant, il y a des instances où cela peut raisonnablement arriver. Par exemple, si une mise en œuvre permet que les clés soient importées d'une base de données de clés en plus d'une clé exportée, cette situation peut alors se produire.

Certaines mises en œuvre ne représentent pas les intérêts d'un seul utilisateur (par exemple, un serveur de clés). De telles mises en œuvre arrangent toujours les certifications locales de toutes les clés qu'elles traitent.

5.2.3.12 Révocable

(1 octet de révocabilité, 1 pour révocable, 0 pour non révocable). L'état de révocabilité de la signature. Le corps de paquet contient un fanion booléen qui indique si la signature est révocable. Les signatures qui ne sont pas révocables ignorent toute révocation de signature ultérieure. Ils représentent un engagement par le signataire de ne pas révoquer cette signature pour la durée de vie de cette clé. Si ce paquet n'est pas présent, la signature est révocable.

5.2.3.13 Signature de confiance

(1 octet "niveau" (profondeur) 1 octet de quantité de confiance). Le signataire affirme que la clé est non seulement valide mais aussi digne de confiance au niveau spécifié. Le niveau 0 a la même signification qu'une signature ordinaire de validité. Le niveau 1 signifie que la clé signée est affirmée être un introducteur de confiance valide, le second octet du corps spécifiant le degré de confiance. Le niveau 2 signifie que la clé signée est affirmée être de confiance pour produire des signatures de confiance de niveau 1, c'est-à-dire, qu'elle est un "méta introducteur". Généralement, une signature de confiance de niveau n affirme qu'une clé est de confiance pour produire des signatures de confiance de niveau n-1. La quantité de confiance est dans une gamme de 0 à 255, interprétée de telle sorte que les valeurs inférieures à 120 indiquent une confiance partielle et les valeurs égales ou supérieures à 120 indiquent une confiance complète. Les mises en œuvre DEVRAIT émettre des values de 60 pour une confiance partielle et de 120 pour une confiance complète.

5.2.3.14 Expression régulière

(Expression régulière terminée par un caractère nul). Utilisé en conjonction avec les paquets Signature de confiance (de niveau > 0) pour limiter la portée de confiance qui est étendue. Seules les signatures par la clé cible sur les identifiants d'utilisateur qui correspondent à l'expression régulière dans le corps de ce paquet ont la confiance étendue par le sous paquet Signature de confiance. L'expression régulière utilise la même syntaxe que le paquetage d'expression régulière [REGEX] de Henry Spencer "presque domaine public". Une description de cette syntaxe se trouve à la Section 8.

5.2.3.15 Clé de révocation

(1 octet de classe, 1 octet d'identifiant d'algorithme de clé publique, 20 octets d'empreinte digitale). Autorise la clé spécifiée à produire des signatures de révocation pour cette clé. L'octet Classe doit avoit le bit 0x80 établi. Si le bit 0x40 est établi, cela signifie alors que les informations de révocation sont sensibles. Les autres bits sont pour une future expansion à d'autres sortes d'autorisations. On le trouve sur une auto signature.

Si le fanion "Sensible" est établi, le détenteur de la clé estime que ce sous paquet contient des informations relevant de la confidentialité qui décrivent une relation sensible dans la réalité. Si ce fanion est établi, les mises en œuvre NE DEVRAIENT PAS exporter cette signature à d'autres utilisateurs sauf dans les cas où les données doivent être disponibles : quand la signature est envoyée au révocateur désigné, ou quand elle est accompagnée d'une signature de révocation provenant de ce révocateur. Noter qu'il peut être approprié d'isoler ce sous paquet dans une signature séparée afin qu'il ne soit pas combiné avec d'autres sous paquets qui ont besoin d'être exportés.

5.2.3.16 Données de notation

(4 octets de fanions, 2 octets de longueur de nom (M) 2 octets de longueur de valeur (N) N octets données de valeur). Ce sous paquet décrit une "notation" sur la signature que le producteur souhaite faire. La notation a un nom et une valeur, chacune étant une chaîne d'octets. Il peut y avoir plus d'une notation dans une signature. Les notations peuvent être utilisées pour toute extension que le producteur de la signature souhaite faire. Le champ "Fanions" contient quatre octets de fanions.

Tous les fanions non définis DOIVENT être à zéro. Les fanions définis sont comme suit :

Premier octet : 0x80 = lisible par l'homme. Cette valeur note le texte.
Autres octets : aucune.

Les noms de notation sont des chaînes arbitraires codées en UTF-8. Ils résident dans deux espaces de noms: L'espace de noms IETF et l'espace de noms d'utilisateur.

L'espace de noms IETF est enregistré par l'IANA. Ces noms NE DOIVENT PAS contenir le caractère "@" (0x40). C'est une étiquette pour l'espace de noms d'utilisateur.

Les noms dans l'espace de noms d'utilisateur consistent en une étiquette de chaîne UTF-8 suivie par "@" suivi par un nom de domaine du DNS. Noter que l'étiquette NE DOIT PAS contenir un caractère "@". Par exemple, l'étiquette "sample" utilisée par Exemple Corporation pourrait être "sample@exemple.com".

Les noms dans un espace d'utilisateur sont possédés et contrôlés par les possesseurs de ce domaine. Évidemment, c'est une mauvaise manière de créer un nouveau nom dans un espace du DNS qu'on ne possède pas.

Comme l'espace de noms d'utilisateur est de la forme d'une adresse de messagerie électronique, les mises en œuvre PEUVENT souhaiter s'arranger pour que cette adresse atteigne une personne qui peut être consultée sur l'utilisation de l'étiquette nommée. Noter que du fait du codage UTF-8, toutes les étiquettes valides de l'espace de noms d'utilisateur ne sont pas des adresses de messagerie électronique valides.

Si c'est une notation critique, la criticité s'applique à cette notation spécifique et non aux notations én général.

5.2.3.17 Préférences de serveur de clés

(N octets de fanions). C'est une liste de fanions d'un bit qui indiquent les préférences qu'a le détenteur de la clé sur la façon dont la clé est traitée au serveur de clés. Tous les fanions non définis DOIVENT être à zéro.

Premier octet : 0x80 = Ne pas modifier. Le détenteur de la clé demande que cette clé ne soit pas modifiée ou mise à jour par le détenteur de la clé ou un administrateur du serveur de clés. Cela se trouve seulement sur une auto signature.

5.2.3.18 Serveur de clés préferé

(Chaîne). C'est un URI d'un serveur de clés que le détenteur de la clé préfère utiliser pour les mises à jour. Noter que les clés avec plusieurs identifiants d'utilisateur peuvent avoir un serveur de clés préféré pour chaque identifiant d'utilisateur. Noter aussi que comme c'est un URI, le serveur de clés peut en fait être une copie de la clé restituée par ftp, http, finger, etc.

5.2.3.19 Identifiant d'utilisateur principal

(1 octet, booléen). C'est un fanion dans une auto signature d'identifiant d'utilisateur qui déclare si cet identifiant d'utiilsateur est l'identifiant d'utilisateur principal pour cette clé. Il est raisonnable pour une mise en œuvre de résoudre les ambiguïtés dans les préférences, etc. en se référant à l'identifiant d'utilisateur principal. Si ce fanion est absent, sa valeur est zéro. Si plus d'un identifiant d'utilisateur dans une clé est marqué comme principal, la mise en œuvre peut résoudre l'ambiguïté de toutes les façons qu'elle estime convenir, mais il est RECOMMANDÉ que la priorité soit donnée à l'identifiant d'utilisateur qui a l'auto signature la plus récente.

Quand il apparaît sur une auto signature d'un paquet Identifiant d'utilisateur, ce sous paquet s'applique seulement aux paquets Identifiant d'utilisateur. Quand il apparaît sur une auto signature sur un paquet Attribut d'utilisateur, ce sous paquet s'applique seulement aux paquets Attribut d'utilisateur. C'est-à-dire, il y a deux "principaux" différents et indépendants -- un pour les identifiants d'utilisateur, et un pour les attributs d'utilisateur.

5.2.3.20 URI de politique

(Chaîne). Ce sous paquet contient un URI d'un document qui décrit la politique sous laquelle la signature a été produite.

5.2.3.21 Fanions de clé

(N octets de fanions). Ce sous paquet contient une liste de fanions binaires qui contiennent des informations sur une clé. C'est une chaîne d'octets, et une mise en œuvre NE DOIT PAS supposer une taille fixe. C'est pour qu'il puisse croître à l'avenir. Si une liste est plus courte que ce qu'attend une mise en œuvre, les fanions non déclarés sont considérés comme étant à zéro. Les fanions définis sont les suivants :

Premier octet :
0x01 - cette clé peut être utilisée pour certifier d'autres clés.
0x02 - cette clé peut être utilisée pour signer des données.
0x04 - cette clé peut être utilisée pour chiffrer des communications.
0x08 - cette clé peut être utilisée pour chiffrer la mémorisation.
0x10 - le composant privé de cette clé peut avoir été partagé par un mécanisme de secret partagé.
0x20 - cette clé peut être utilisée pour l'authentification.
0x80 - le composant privé de cette clé peut être en la possession de plus d'une personne.

Notes d'utilisation :
Les fanions dans ce paquet peuvent apparaître dans des auto signatures ou dans des signatures. Ils signifient des choses différentes selon celui qui fait la déclaration -- par exemple, une signature de certification qui a le fanion "Données de

signature" déclare que la certification est pour cela. Par ailleurs, le fanion "Chiffrement de communications" dans une auto signature déclare une préférence pour qu'une certaine clé soit utilisée pour les communications. Noter cependant, que c'est une question épineuse de déterminer ce qui est "communication" et ce qui est "mémorisation". Cette décision appartient pleinement à la mise en œuvre ; les auteurs de ce document ne prétendent pas avoir de connaissances particulières sur la question et réalisent qu'une opinion acceptée peut changer.

Les fanions "Clé partagée" (0x10) et "Clé de groupe" (0x80) sont placés seulement sur une auto signature ; ils n'ont pas de signification sur une signature de certification. Ils DEVRAIENT n'être placés que sur une signature de clé directe (type 0x1F) ou une signature de sous clé (type 0x18) qui se réfère à la clé à laquelle le fanion s'applique.

5.2.3.22 Identifiant d'utilisateur du signataire

(Chaîne). Ce sous paquet permet au détenteur de la clé de déclarer quel identifiant d'utilisateur est responsable de la signature. De nombreux détenteurs de clé utilisent une seule clé pour différents objets, comme des communications d'affaires aussi bien que des communications personnelles. Ce sous paquet permet à un détenteur de clé de déclarer quel rôle fait une signature.

Ce sous paquet n'est pas approprié pour se référer à un paquet Attribut d'utilisateur.

5.2.3.23 Raison de révocation

(1 octet de code de révocation, N octets de chaîne de raison). Ce sous paquet n'est utilisé que dans la révocation de clé et les signatures de révocation de certification. Il décrit la raison pour laquelle la clé ou le certificat a été révoqué.

Le premier octet contient un code lisible par la machine qui note la raison de la révocation :
0 - pas de raison spécifiée (révocations de clé ou de certificat)
1 - la clé est remplacée (révocations de clé)
2 - le matériel de chiffrement a été compromis (révocations de clé)
3 - la clé est retirée et n'est plus utilisée (révocations de clé)
32 - les informations d'identifiant d'utilisation ne sont plus valides (révocations de certificat)
100-110 - utilisation privée

À la suite du code de révocation est une chaîne d'octets qui donne des informations sur la raison de la révocation en forme lisible par l'homme (UTF-8). La chaîne peut être nulle, c'est-à-dire, de longueur zéro. La longueur du sous paquet est la longueur de la chaîne de raison plus un. Une mise en œuvre DEVRAIT utilier ce sous paquet, l'inclure dans toutes les signatures de révocation, et interpréter les révocations de façon appropriée. Il y a d'importantes différences sémantiques entre les raisons, et il y a donc des raisons importantes pour révoquer des signatures.

Si une clé a été révoquée à cause d'une compromission, toutes les signatures créées avec cette clé sont suspectes. Cependant, si elle a été simplement remplacée ou retirée, les vieilles signatures sont encore valides. Si la signature révoquée est l'auto signature pour certifier un identifiant d'utilisateur, la révocation note que ce nom d'utilisateur n'est plus utilisé. Une telle révocation DEVRAIT inclure un code 0x20.

Noter que toute signature peut être révoquée, incluant une certification sur la clé d'une autre personne. Il y a de nombreuses bonnes raisons pour révoquer une signature de certification, comme le cas où le détenteur de clé quitte l'emploi d'affaires avec une adresse de messagerie électronique. Une certification révoquée ne fait plus partie des calculs de validité.

5.2.3.24 Caractérisitiques

(N octets de fanions). Le sous paquet Caractéristiques note quelles caractéristiques avancées de OpenPGP une mise en œuvre d'utilisateur supporte. C'est pour que lorsque des caractéristiques sont ajoutées à OpenPGP et qu'elles peuvent n'être pas rétro-compatibles, un usager puisse déclarer qu'il peut utiliser cette caractéristique. Les fanions sont d'un seul bit qui indique qu'une certaine caractéristique est supportée.

Ce sous paquet est similaire aux sous paquets de préférences, et n'apparaissent que dans une auto signature.

Une mise en œuvre NE DEVRAIT PAS utiliser une caractéristique mentionnée quand elle envoie à un utilisateur qui ne déclare pas qu'il peut l'utiliser.

Les caractéristiques définies sont les suivantes :

Premier octet :

0x01 - Détection de modification (paquets 18 et 19)
Si une mise en œuvre utilise une des caractéristiques définies, elle DEVRAIT mettre en œuvre aussi le sous paquet Caractéristiques. Une mise en œuvre peut librement déduire des caractéristiques d'autres mécanismes convenables dépendants de la mise en œuvre.

5.2.3.25 Cible de signature

(Algorithme de clé publique d'un octet, algorithme de hachage d'un octet , N octets de hachage). Ce sous paquet identifie une signature spécifique de la cible à laquelle une signature se réfère. Pour les signatures de révocation, ce sous paquet fournit une désignation explicite de la signature révoquée. Pour une signature de tiers ou d'horodatage, cela désigne quelle signature est signée. Le seul argument est un identifiant de cette signature cible.

Les N octets de données de hachage DOIVENT être la taille du hachage de la signature. Par exemple, une signature de cible avec un hachage SHA-1 DOIT avoir 20 octets de données de hachage.

5.2.3.26 Signature incorporée

(Corps de paquet Signature). Ce sous paquet contient un corps complet de paquet Signature comme spécifié au paragraphe 5.2. C'est utile quand une signature a besoin de se référer à, ou être incorporée dans, une autre signature.

5.2.4 Calcul des signatures

Toutes les signatures sont formées en produisant un hachage sur les données de signature, et ensuite en utilisant le hachage résultant dans l'algorithme de signature.

Pour les signatures de document binaire (type 0x00) les données du document sont hachées directement. Pour les signatures de document de texte (type 0x01) le document est canonisé en convertissant les fins de ligne en <CR><LF>, et les données résultantes sont hachées.

Quand une signature est faite sur une clé, les données de hachage commencent par l'octet 0x99, suivi par les deux octets de longueur de la clé, et ensuite le corps de paquet de clé. (Noter que ceci est un en-tête de paquet d'ancien style pour un paquet de clé avec la longueur sur deux octets.) Une signature de lien de sous clé (type 0x18) ou une signature de lien de clé principale (type 0x19) hache ensuite la sous clé en utilisant le même format que la clé principale (en utilisnt aussi 0x99 comme premier octet). Les signatures de révocation de clé (types 0x20) hachent seulement la clé révoquée. La signature de révocation de sous clé (type 0x28) hache d'abord la clé principale et ensuite la sous clé révoquée.

Une signature de certification (types 0x10 à 0x13) hache l'identifiant d'utilisateur à lier à la clé dans le contexte dehachage après les données ci-dessus. Une certification V3 hache le contenu de l'identifiant d'utilisateur ou le paquet Attribut de paquet, sans aucun en-tête. Une certification V4 hache la constante 0xB4 pour les certifications d'identifiant d'utilisateur, ou la constante 0xD1 pour les certifications d'attribut d'utilisateur, suivi par un nombre de quatre octets donnant la longueur des données d'identifiant d'utilisateur ou d'attribut d'utilisateur, et ensuite les données d''identifiant d'utilisateur ou d'attribut d'utilisateur.

Quand une signature est faite sur un paquet Signature (type 0x50) les données de hachage commencent par l'octet 0x88, suivi par la longueur de quatre octets de la signature, suivi par le corps du paquet Signature. (Noter que ceci est un en-tête de paquet d'ancien style pour un paquet Signature avec la longueur de Longueur réglé à zéro). Les données du sous paquet non hachées du paquet Signature à hacher ne sont pas incluses dans le hachage, et la valeur de longueur de données du sous paquet non haché est réglée à zéro.

Une fois le corps des données haché, un en-queue est alors haché. Une signature V3 hache cinq octets du corps du paquet, en commençant au champ Type de signature. Ces données sont le type de signature suivi par les quatre octets de l'heure de signature. Une signature V4 hache le corps du paquet Signature en commençant au premier champ, le numéro de version, jusqu'à la fin des données hachées du sous paquet. Donc, les champs hachés sont la version de signature, le type de signature, l'algorithme de hachage, la longueur du sous paquet haché, et le corps du sous paquet haché.

Les signatures V4 hachent aussi dans en en-queue final de six octets : la version du paquet Signature, c'est-à-dire, 0x04; 0xFF; et un nombre de quatre octets, gros boutien qui est la longueur des données hachées du paquet Signature (noter que ce nombre n'inclut pas ces six octets finaux).

Après que tout cela a été haché dans un seul contexte de hachage, le champ de hachage résultant est utilisé dans l'algorithme de signature et placé à la fin du paquet Signature.

5.2.4.1 Indications de sous paquet

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 plusieurs heures d'expiration. Dans la plupart des cas, une mise en œuvre DEVRAIT utiliser le dernier sous paquet dans la signature, mais PEUT utiliser tout schéma de résolution de conflit qui paraît le mieux adapté. Noter qu'on laisse intentionnellement la résolution de conflit à la mise en œuvre ; la plupart des conflits sont simplement des erreurs de syntaxe, et le langage utilisé ici permet à un receveur d'être généreux dans ce qu'il accepte, tout en mettant la pression sur le créateur pour être strict dans ce qu'il génère.

Certains conflits apparents peuvent réellement avoir un sens -- par exemple, supposons qu'un détenteur de clé ait une clé V3 et une clé V4 qui partagent le même matériel de clé RSA. L'une et l'autre de ces clés peuvent vérifier une signature créée par l'autre, et il peut être raisonnable qu'une signature contienne un sous paquet producteur pour chaque clé, comme moyen de lier explicitement ces clés à la signature.

5.3 Paquets Clé de session chiffrée à clé symétrique (étiquette 3)

Le paquet Clé de session chiffrée à clé symétrique contient la clé de chiffrement symétrique d'une clé de session utilisée pour chiffrer un message. Zéro, un ou plusieurs paquets Clé de session chiffrée à clé publique et/ou paquets Clé de session chiffrée à clé symétrique peuvent précéder un paquet Données chiffrées symétriquement qui contient un message chiffré. Le message est chiffré avec une clé de session, et la clé de session est elle-même chiffrée et mémorisée dans le paquet Clé de session chiffrée à clé publique ou le paquet Clé de session chiffrée à clé symétrique.

Si le paquet de données chiffrées symétriquement est précédé d'un ou plusieurs paquets Clé de session chiffrée à clé symétrique, chacun spécifie un mot de passe qui peut être utilisé pour déchiffrer le message. Cela permet qu'un message soit chiffré avec un certain nombre de clés publiques, et aussi avec un ou plusieurs mots 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 consiste en :
- Un numéro de version de un-octet. La seule version actuellement définie est 4.
- Un numéro d'un octet décrivant l'algorithme symétrique utilisé.
- Un spécificateur de chaîne à clé (S2K, *string-to-key*) dont la longueur a été définie précédemment.
- Facultativement, la clé de session chiffrée elle-même, qui est déchiffrée avec l'objet S2K.

Si la clé de session chiffrée n'est pas présente (ce qui peut être détecté par la longueur du paquet et la taille du spécificateur S2K) alors l'algorithme S2K appliqué au mot de passe produit la clé de session pour déchiffrer le fichier, en utilisant l'algorithme de chiffrement symétrique provenant du paquet Clé de session chiffrée à clé symétrique.

Si la clé de session chiffrée est présente, le résultat de l'application de l'algorithme S2K au mot de passe est utilisé pour déchiffrer juste le champ Clé de session chiffrée, en utilisant le mode CFB avec une IV toute de zéros. Le résultat du déchiffrement consiste en un identifiant d'algorithme de un octet qui spécifie l'algorithme de chiffrement à clé symétrique utilisé pour chiffrer le paquet Données chiffrées symétriquement suivant, suivi par les octets de la clé de session elle-même.

Note : parce qu'une IV toute de zéros est utilisée pour ce déchiffrement, le spécificateur S2K DOIT utiliser une valeur de sel, soit une S2K salée, soit une S2K itérée-salée. La valeur du sel va assurer que la clé de déchiffrement n'est pas répétée même si le mot de passe est réutilisé.

5.4 Paquets Signature à une passe (étiquette 4)

Le paquet Signature à une passe précède les données signées et contient assez d'informations pour permettre au receveur de commencer à calculer tout hachage nécessaire pour vérifier la signature. Il permet que le paquet Signature soit placé à la fin du message, pour que le signataire puisse calculer le message signé entier en un seul passage.

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

Le corps de ce paquet consiste en :
- Un octet de numéro de version. La version actuelle est 3.
- Un octet de type de signature. Les types de signature sont décrits au paragraphe 5.2.1.
- Un octet de numéro décrivant l'algorithme de hachage utilisé.
- Un octet de numéro décrivant l'algorithme de clé publique utilisé.
- Huit octets de numéro contenant l'identifiant de clé de la clé de signature.
- Un octet de numéro contenant un fanion montrant si la signature est incorporée. Une valeur de zéro indique que le paquet suivant est un autre paquet Signature à une passe qui décrit une autre signature à appliquer aux mêmes données de message.

Noter que si un message contient plus d'une signature à une passe, alors les paquets Signature encadrent le message; c'est-à-dire, le premier paquet Signature après le message correspond au dernier paquet à une passe et le paquet Signature final correspond au premier paquet à une passe.

5.5 Paquet Matériel de clé

Un paquet Matériel de clé contient toutes les informations sur une clé publique ou privée. 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 paquet de clé

5.5.1.1 Paquet Clé publique (étiquette 6)

Un paquet Clé publique commence une série de paquets qui forment une clé OpenPGP (parfois appelée un certificat OpenPGP).

5.5.1.2 Paquet Sous clé publique (étiquette 14)

Un paquet Sous clé publique (étiquette 14) a exactement le même format qu'un paquet Clé publique, mais note une sous clé. Une ou plusieurs sous clés peuvent être associées à une clé de niveau supérieur. Par convention, la clé de niveau supérieur fournit des services de signature, et les sous clés fournissent des services de chiffrement.

Note : dans PGP 2.6.x, l'étiquette 14 était destinée à indiquer un paquet de commentaire. La réutilisation de cette étiquette a été choisie parce qu'aucune version précédente de PGP n'a jamais émis de paquet de commentaire mais les ignorait complètement. Les paquets Sous clé publique sont ignorés par PGP 2.6.x et ne causent pas son échec, fournissant un degré limité de rétro compatibilité.

5.5.1.3 Paquet Clé secrète (étiquette 5)

Un paquet Clé secrète contient toutes les informations qui se trouvent dans un paquet Clé publique, incluant le matériel de clé publique, mais inclut aussi le matériel de clé secrète après tous les champs de clé publique.

5.5.1.4 Paquet Sous clé secrète (étiquette 7)

Un paquet Sous clé secrète (étiquette 7) est la sous clé analogue du paquet Clé secrète et a exactement le même format.

5.5.2 Formats de paquet de clé publique

Il y a deux versions de paquet de matériel de clé. Les paquets de version 3 ont été d'abord générés par PGP 2.6. Les clés de version 4 apparaissent d'abord dans PGP 5.0 et sont la version de clé préférée pour OpenPGP.

Les mises en œuvre de OpenPGP DOIVENT créer des clés dansle format de la version 4. Les clé V3 sont déconseillées ; une mise en œuvre NE DOIT PAS générer de clé V3, mais PEUT l'accepter.

Un paquet Clé publique version 3 ou Sous clé publique contient :
- Un octet de numéro de version (3).
- Un nombre de quatre octets notant l'heure de création de la clé.
- Un nombre de deux octets notant le temps en jours pendant lequel cette clé est valide. Si ce nombre est zéro, elle n'expire jamais.
- Un nombre de un octet notant l'algorithme de clé publique de cette clé.
- Une série d'entiers multi précisions comprenant le matériel de la clé :
  - un entier multi précisions (MPI) du RSA public modulo n ;
  - un MPI de l'exposant e du chiffrement RSA public.

Les clés V3 sont déconseillées. Elles contiennent trois faiblesses. D'abord, il est relativement facile de construire une clé V3 qui a le même identifiant de clé qu'une autre clé parce que l'identifiant de clé est simplement les 64 bits de moindre poids du module public. Ensuite, parce que l'empreinte digitale d'une clé V3 hache le matériel de clé, mais pas sa longueur, il y a une opportunité accrue de collisions d'empreintes digitales. Enfin, il y a des faiblesses dans l'algorithme de hachage MD5 qui font que les développeurs préferent d'autres algorithmes. Voir ci-dessous une discussion plus approfondie des identifiants de clé et des empreintes digitales.

Les clés V2 sont identiques aux clés V3 déconseillées sauf pour le numéro de version. Une mise en œuvre NE DOIT PAS les générer et PEUVENT les accepter ou les rejeter comme il lui semble approprié.

Le format de version 4 est similaire au format de version 3 sauf pour l'absence d'une période de validité. Cela a été déplacé au paquet Signature. De plus, les empreintes digitales des clés de version 4 sont calculées différemment des clés de version 3, comme décrit à la Section 12 "Formats de clé améliorés".

Un paquet version 4 contient :
- Un octet de numéro de version (4).
- Un nombre de quatre octets notant l'heure de création de la clé.
- Un nombre d'un octet notant l'algorithme de clé publique de cette clé.
- Une série d'entiers multi précisions composant le matériel de la clé. Cette portion spécifique de l'algorithme est :

Champs spécifiques de l'algorithme pour clés publiques RSA :
- entier multi précisions (MPI) du module RSA public n ;
- MPI de l'exposant e du chiffrement RSA public.

Champs spécifiques de l'algorithme pour clés publiques DSA :
- MPI du nombre premier DSA p ;
- MPI de l'ordre de groupe DSA q (q est un diviseur premier de p-1) ;
- MPI du générateur de groupe DSA g ;
- MPI de la valeur de clé publique DSA y (= g**x mod p où x est secret).

Champs spécifiques de l'algorithme pour clés publiques Elgamal :
- MPI du nombre premier Elgamal p ;
- MPI du générateur de groupe Elgamal g ;
- MPI de la valeur de clé publique Elgamal y (= g**x mod p où x est secret).

5.5.3 Formats de paquet de clé secrète

Les paquets Clé secrète et Sous clé secrète contiennent toutes les données des paquets Clé publique et Sous clé publique, avec l'ajout des données spécifiques de l'algorithme de données de clé secrète, généralement en forme chiffrée.

Le paquet contient :
- Un corps de paquet Clé publique ou Sous clé publique, comme décrit ,ci-dessus.
- Un octet indiquant les conventions d'utilisation de chaîne à clé. Zéro indique que les données de clé secrète ne sont pas chiffrées. 255 ou 254 indique qu'un spécificateur de chaîne à clé est donné. Toute autre valeur est un identifiant d'algorithme de chiffrement à clé symétrique.
- [Facultatif] Si l'octet d'usage de chaîne à clé était 255 ou 254, un octet d'algorithme de chiffrement à clé symétrique.
- [Facultatif] Si l'octet d'usage de chaîne à clé était 255 ou 254, un spécificateur de chaîne à clé. La longueur du spécificateur de chaîne à clé est impliquée par son type, comme décrit ci-dessus.
- [Facultatif] Si les données secrètes sont chiffrées (l'octet d'usage de chaîn à clé n'est pas zéro) une valeur initiale (IV) de la même longueur que la taille de bloc du chiffrement.
- Des entiers multi précisions en clair ou chiffrés comportant les données de clé secrète. Ces champs spécifiques de l'algorithme sont décrits ci-dessous.
- Si l'octet d'usage de chaîne à clé n'est pas 254, alors une somme de contrôle de deux octets du texte en clair de la portion spécifique de l'algorithme (somme de tous les octets, modulo 65536). Si l'octet d'usage de chaîne à clé était 254, alors un hachage de 20 octets SHA-1 du texte en clair de la portion spécifique de l'algorithme. Pour les clés V4 cette somme de contrôle ou hachage est chiffrée ainsi que les champs spécifiques de l'algorithme (si l'octet d'usage de chaîne à clé n'est pas zéro). Noter que pour toutes les autres valeurs, une somme de contrôle de deux octets est exigée.

Champs spécifiques de l'algorithme pour les clés secrètes RSA :
- entier multi précisions (MPI) de l'exposant secret RSA d.
- MPI de la valeur du nombre premier secret RSA p.
- MPI de la valeur du nombre premier secret RSA q (p < q).
- MPI de u, le multiplicateur inverse de p, mod q.

Champs spécifiques de l'algorithme pour les clés secrètes DSA :
- MPI de l'exposant secret DSA x.

Champs spécifiques de l'algorithme pour les clés secrètes Elgamal :
- MPI de l'exposant secret Elgamal x.

Les valeurs secrètes de MPI peuvent être chiffrées en utilisant un mot de passe. Si un spécificateur de chaîne à clé est donné, cela décrit l'algorithme pour convertir le mot de passe en une clé, autrement, un simple hachage MD5 du mot de passe est utilisé. Les mises en œuvre DOIVENT générer un spécificateur de chaîne à clé ; le simple hachage est pour la rétro compatibilité et est déconseillé, bien que les mises en œuvre PEUVENT continuer d'utiliser les clés privées existantes dans l'ancien format. Le chiffrement pour les MPI est spécifié dans le paquet Clé secrète.

Le chiffrement/déchiffrement des données secrètes est fait en mode CFB en utilisant la clé créée à partir du mot de passe et de la valeur initiale provenant du paquet.

Un mode différent d'avec les autres formats de clés est utilisé avec les clés V3 (qui sont seulement RSA). Avec les clés V3, le préfixe de compte de bits de MPI (c'est-à-dire, les deux premiers octets) n'est pas chiffré. Seules les données qui ne sont pas du préfixe de MPI sont chiffrées. De plus, l'état CFB est resynchronisé au début de chaque nouvelle valeur de MPI, de sorte que la limite de bloc CFB est alignée avec le début des données de MPI.

Avec les clés V4, une méthode plus simple est utilisée. Toutes les valeurs secrètes de MPI sont chiffrées en mode CFB, incluant le préfixe de compte de bits de MPI.

La somme de contrôle de deux octets qui suit la portion spécifique de l'algorithme est la somme algébrique, mod 65536, du texte en clair de tous les octets spécifiques de l'algorithme (incluant le préfixe et les données de MPI). Avec les clés V3, la somme de contrôle est mémorisée en clair. Avec les clés V4, la somme de contrôle est chiffrée comme les données spécifiques de l'algorithme. Cette valeur est utilisée pour vérifier que le mot de passe est correct. Cependant, cette somme de contrôle est déconseillée ; une mise en œuvre NE DEVRAIT PAS l'utiliser, mais devrait plutôt utiliser le hachage SHA-1 noté avec un octet d'usage de 254. La raison en est que certaines attaques impliquent une modification indétectable de la clé secrète.

5.6 Paquet Données compressées (étiquette 8)

Le paquet Données compressées contient des données compressées. Normalement, ce paquet se trouve comme contenu d'un paquet chiffré, ou suivant un paquet Signature ou Signature à une passe, et contient un paquet Données littérales.

Le corps de ce paquet consiste en :
- un octet qui donne l'algorithme utilisé pour compresser le paquet,
- des données compressées, qui constituent le reste du paquet.

Un corps de paquet Données compressées contient un bloc qui compresse un certain ensemble de paquets. Voir à la Section 11 "Composition de la séquence de paquets" les détails sur la formation des messages.

Les paquets compressés par ZIP sont compressés avec les blocs DEFLATE bruts de la [RFC1951]. Noter que PGP V2.6 utilise 13 bits de compression. Si une mise en œuvre utilise plus de bits de compression, PGP V2.6 ne peut pas le décompresser.

Les paquets compressés par ZLIB sont compressés avec les blocs de style ZLIB de la [RFC1950].

Les paquets compressés par BZip2 sont compressés en utilisant l'algorithme BZip2 [BZ2].

5.7 Paquet Données à chiffrage symétrique (étiquette 9)

Le paquet Données chiffrées symétriquement contient des données chiffrées avec un algorithme de clé symétrique. Quand il a été déchiffré, il contient d'autres paquets (généralement un paquet de données littérales ou un paquet de données compressées, mais en théorie d'autres paquets Données chiffrées symétriquement ou séquences de paquets qui forment des messages OpenPGP complets).

Le corps de ce paquet consiste en des données chiffrées, le résultat du chiffrement à clé symétrique choisi opérant dans la variante OpenPGP du mode de rebouclage de chiffrement (CFB, *Cipher Feedback*).

Le chiffrement symétrique utilisé peut être spécifié dans un paquet Clé de session chiffrée à clé publique ou à clé symétrique qui précède le paquet Données chiffrées symétriquement. Dans ce cas, l'octet d'algorithme de chiffrement est mis en préfixe de la clé de session avant qu'elle soit chiffrée. Si aucun paquet de ces types ne précède les données chiffrées, l'algorithme IDEA est utilisé avec la clé de session calculée comme le hachage MD5 du mot de passe, bien que cette utilisation soit déconseillée.

Les données sont chiffrées en mode CFB, avec une taille de décalage de CFB égale à la taille de bloc du chiffiement. La valeur initiale (IV) est spécifiée comme toute de zéros. Au lieu d'utiliser une IV, OpenPGP met en préfixe une chaîne de longueur égale à la taille de bloc du chiffrement plus deux aux données avant qu'elles soient chiffrées. Les premiers octets de taille de bloc (par exemple, 8 octets pour une longueur de bloc de 64 bits) sont aléatoires, et les deux octets suivants sont copiés des deux derniers octets de l'IV. Par exemple, dans un bloc de 8 octets, l'octet 9 est une répétition de l'octet 7, et l'octet 10 est une répétition de l'octet 8. Dans un chiffrement de longueur 16, l'octet 17 est une répétition de l'octet 15 et l'octet 18 est une répéttition de l'octet 16. On précise de façon un peu pédante que dans les deux exemples, on considère que le premier octet a le numéro 1.

Après le chiffrement des premiers octets de taille de bloc plus deux, l'état CFB est resynchronisé. Les derniers octets de taille de bloc du texte chiffré sont passés à travers le chiffrement et la limite de bloc est réinitialisée. La répétition de 16 bits dans les données aléatoires en préfixe du message permet au receveur de vérifier immédiatement si la clé de session est incorrecte. Voir à la Section 14 "Considérations sur la sécurité" des conseils sur le bon usage de cette "vérification rapide".

5.8 Paquet Marqueur (rend obsolète le paquet Literal) (étiquette 10)

Une version expérimentale de PGP utilisait ce paquet comme paquet Literal, mais aucune version publiée de PGP n'a généré de paquet Literal avec cette étiquette. Avec PGP 5.x, ce paquet a été réalloué et est réservé pour l'utilisation comme paquet Marqueur.

Le corps de ce paquet consiste en les trois octets 0x50, 0x47, 0x50 (qui épellent "PGP" en UTF-8).

Ce paquet DOIT être ignoré à réception. Il peut être placé au début d'un message qui utilise des caractéristiques non disponibles dans PGP 2.6.x afin de causer le rapport par cette version qu'un logiciel plus récent est nécessaire pour traiter ce message.

5.9 Paquet Données littérales (étiquette 11)

Un paquet Données littérales contient le corps d'un message ; les données ne sont pas à interpréter.

Le corps de ce paquet consiste en :
- Un champ de un octet qui décrit comment les données sont formatées.
  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 de texte, et peut donc devoir convertir les terminaisons de ligne en forme locale, ou autres changement de mode de texte. L'étiquette 'u' (0x75) signifie la même chose que 't', mais indique aussi que la mise en œuvre estime que les données littérales contiennent du texte UTF-8.
  Les anciennes versions de PGP définissaient aussi une valeur de 'l' comme un mode 'local' pour les conversions de machine locale. La [RFC1991] déclarait incorrectement ce fanion de mode local comme '1' (numéral un en ASCII). Ces deux modes locaux sont déconseillés.

- Un nom de fichier comme une chaîne (un octet de longueur, suivi par un nom de fichier). Ce peut être une chaîne de longueur zéro. Couramment, si la source des données chiffrées est un fichier, cela va être le nom du fichier chiffré. Une mise en œuvre PEUT considérer le nom de fichier dans le paquet Littéral comme étant un nom de plus d'autorité que le nom de fichier réel.
  Si le nom spécial "_CONSOLE_" est utilisé, le message est considéré comme étant "à votre seule attention". Cela indique que les données du message sont extrêmement sensibles, et que le programme receveur devrait le traiter avec plus de précautions, peut-être en évitant de mémoriser les données reçues sur le disque, par exemple.

- Un nombre de quatre octets qui indique une date associée aux données littérales. Couramment, la date pourrait être la date de modification d'un fichier, ou l'heure de création du paquet, ou un zéro qui n'indique pas d'heure spécifique.

- Le reste du paquet est les données littérales.

Les données de texte sont mémorisées avec des fins de texte <CR><LF> (c'est-à-dire, les terminaisons normales de ligne du réeau). Cela DEVRAIT être converti en terminaisons de ligne natives par le logiciel receveur.

5.10 Paquet Confiance (étiquette 12)

Le paquet Confiance est seulement utilisé au sein d'anneaux de clés et n'est normalement pas exporté. Les paquets Confiance contiennent des données qui enregistrent les spécifications de l'utilisateur sur quels détenteurs de clés sont des introducteurs dignes de confiance, ainsi que d'autres informations que le logiciel de mise en œuvre utilise pour des informations de confiance. Le format des paquets Confiance est défini pour une certaine mise en œuvre.

Les paquets Confiance NE DEVRAIENT PAS être émis sur des flux en sortie qui sont transférés à d'autres utilisateurs, et ils DEVRAIENT être ignorés sur toute entrée autre que des fichiers d'anneaux de clés locaux.

5.11 Paquet Identifiant d'utilisateur (étiquette 13)

Un corps de paquet Identifiant d'utilisateur consiste en texte UTF-8 qui est destiné à représenter le nom et l'adresse de messagerie du détenteur de la clé. Par convention, il inclut un nom/adresse de messagerie de la [RFC2822], mais il n'y a pas de restriction sur son contenu. La longueur du paquet dans l'en-tête spécifie la longueur de l'identifiant d'utilisateur.

5.12 Paquet Attribut d'utilisateur (étiquette 17)

Le paquet Attribut d'utilisateur est une variante du paquet Identifiant d'utilisateur. Il est capable de mémoriser plus de types de données que le paquet Identifiant d'utilisateur, qui est limité à du texte. Comme le paquet Identifiant d'utilisateur, un paquet Attribut d'utilisateur peut être certifié par le possesseur de la clé ("auto signé") ou tout autre possesseur de clé qui se soucie de la certifier. Sauf comme noté, un paquet Attribut d'utilisateur peut être utilisé partout où un paquet Identifiant d'utilisateur peut être utilisé.

Bien que les paquets Attribut d'utilisateur ne soient pas une partie exigée de la norme OpenPGP, les mises en œuvre DEVRAIENT fournir au moins assez de compatibilité pour traiter correctement une signature de certification sur le paquet Attribut d'utilisateur. Une façon simple de le faire est de traiter le paquet Attribut d'utilisateur comme un paquet Identifiant d'utilisateur avec un contenu opaque, mais une mise en œuvre peut utiliser toute méthode qu'elle désire.

Le paquet Attribut d'utilisateur est constitué d'un ou plusieurs sous paquets d'attributs. Chaque sous paquet consiste en un en-tête de sous paquet et un corps. L'en-tête consiste en :
- la longueur du sous paquet (1, 2, ou 5 octets)
- le type de sous paquet (1 octet)
et est suivi par les données spécifiques du sous paquet.

Le seul type de sous paquet actuellement défini est 1, qui signifie une image. Une mise en œuvre DEVRAIT ignorer tout sous paquet d'un type qu'elle ne reconnaît pas. Les types de sous paquet de 100 à 110 sont réservés pour utilisation privée ou expérimentale.

5.12.1 Sous paquet Attribut d'image

Le sous paquet Attribut d'image est utilisé pour coder une image, en principe (mais ce n'est pas obligé) celle du possesseur de la clé.

Le sous paquet Attribut d'image commence par un en-tête d'image. Les deux premiers octets de l'en-tête d'image contiennent la longueur de l'en-tête d'image. Noter qu'à la différence d'autres valeurs numériques multi octets dans ce document, du fait d'un accident historique, cette valeur est codée comme un nombre petit boutien. La longueur de l'en-tête d'image est suivie pas un seul octet pour la version de l'en-tête d'image. La seule version actuellement définie de l'en-tête d'image est 1, qui est un en-tête d'image de 16 octets. Les trois premiers octets d'un en-tête d'image de version 1 sont donc 0x10, 0x00, 0x01.

Le quatrième octet d'un en-tête d'image de version 1 désigne le format de codage de l'image. Le seul format de codage actuellement défini est la valeur 1 pour indiquer JPEG. Les types de format d'image de 100 à 110 sont réservés pour utilisation privée ou expérimentale. Le reste de l'en-tête d'image de version 1 est constitué de 12 octets réservés, qui DOIVENT tous être réglés à 0.

Le reste du sous paquet Image contient l'image elle-même. Comme le seul type d'image actuellement défini est JPEG, l'image est codée dans le format d'échange de fichier JPEG (JFIF, *JPEG File Interchange Format*) le format standard de fichier pour les images JPEG [JFIF].

Une mise en œuvre PEUT essayer de déterminer le type d'une image en examinant les données de l'image si elle n'est pas capable de traiter une version particulière de l'en-tête d'image ou si une valeur de format de codage spécifiée n'est pas reconnue.

5.13 Paquet Données protégées en intégrité par chiffrement symétrique (étiquette 18)

Le paquet Données protégéres en intégrité par chiffrement symétrique est une variante du paquet Données chiffrées symétriquement. C'est une nouvelle caractéristique créée pour OpenPGP qui traite le problème de la détection d'une modification des données chiffrées. Il est utilisé en combinaison avec un paquet Code de détection de modification.

Il y a une caractéristique correspondante dans le sous paquet Caractéristiques de signature qui note qu'une mise en œuvre peut correctement utiliser ce type de paquet. Une mise en œuvre DOIT prendre en charge le déchiffrement de ces paquets et DEVRAIT préférer les générer plutôt que l'ancien paquet Données chiffrées symétriquement quand c'est possible. Comme ce paquet de données protège contre les attaques de modification, la présente norme encourage sa prolifération. Bien qu'une large adoption de ce paquet de données crée des problèmes d'interopérabilité, une adoption rapide est néanmoins importante. Une mise en œuvre DEVRAIT spécifiquement noter sa prise en charge de ce paquet, mais elle PEUT la déduire d'autres mécanismes.

Par exemple, une mise en œuvre pourrait déduire de l'utilisation d'un chiffrement comme la norme de chiffrement avancé (AES, *Advanced Encryption Standard*) ou Twofish qu'un utilisateur prend cette caractéristique en charge. Elle pourrait placer dans la portion non hachée de la signature de clé d'un autre utilisateur un sous paquet Caractéristiques. Elle pourrait aussi présenter à un utilisateur l'opportunité de régénérer sa propre auto-signature avec un sous paquet Caractéristiques.

Ce paquet contient les données chiffrées avec un algorithme de clé symétrique et protégées contre la modification par l'algorithme de hachage SHA-1. Quand elles ont été déchiffrées, elle vont normalement contenir d'autres paquets (souvent un paquet Données littérales ou un paquet Données compressées). Le dernier paquet déchiffré dans cette charge utile de paquet DOIT être un paquet Code de détection de modification.

Le corps de ce paquet consiste en :
- Un octet de numéro de version. La seule valeur actuellement définie est 1.
- Des données chiffrées, le résultat du chiffrement à clé symétrique choisi en opérant dans le mode CFB avec une quantité de décalage égale à la taille de bloc du chiffrement (CFB-n où n est la taille de bloc).

Le chiffrement symétrique utilisé DOIT être spécifié dans un paquet Clé de session chiffrée à clé publique ou clé symétrique qui précède le paquet Données chiffrées symétriquement. Dans l'un et l'autre cas, l'octet d'algorithme de chiffrement est mis en préfixe à la clé de session avant qu'elle soit chiffrée.

Les données sont chiffrées en mode CFB, avec une taille de décalage CFB égale à la taille de bloc du chiffrement. La valeur initiale (IV, *Initial Vector*) est spécifiée comme toute de zéros. Au lieu d'utiliser une IV, OpenPGP met en préfixe une chaîne d'octets aux données avant qu'elles soient chiffrées. La longueur de la chaîne d'octets est égale à la taille de bloc du chiffrement en octets, plus deux. Les premiers octets dans le groupe, de longueur égale à la taille de bloc du chiffrement, sont aléatoires ; les deux derniers octets sont chacun une copie de leur deuxième octet précédant. Par exemple, avec un chiffrement dont la taille de bloc est 128 bits ou 16 octets, les données de préfixe vont contenir 16 octets aléatoires, puis deux octets de plus, qui sont les copies, respectivement du 15ième et 16ième octet. À la différence du paquet Données chiffrées symétriquement, aucune resynchronisation spéciale de CFB n'est faite après le chiffrement de ces données de préfixe. Voir plus de détails au paragraphe 13.9 "Mode CFB OpenPGP".

La répétition de 16 bits dans les données aléatoires mises en préfixe au message permet au receveur de vérifier immédiatement si la clé de session est incorrecte.

L'enchaînement des données de préfixe décrit ci-dessus, du texte en clair à chiffrer et de deux octets des valeurs 0xD3, 0x14 (elles représentent le codage d'une étiquette de paquet MDC et le champ Longueur de 20 octets) est passé à la fonction de hachage SHA-1.

La valeur de hachage résultante est mémorisée dans un paquet Code de détection de modification (MDC, *Modification Detection Code*) qui DOIT utiliser le codage de deux octets qui vient d'être donné pour représenter son étiquette et son champ Longueur. Le corps du paquet MDC est le résultat de 20 octets du hachage SHA-1.

Le paquet Code de détection de modification est ajouté au texte en clair et chiffré avec le texte clair en utilisant le même contexte de CFB.

Durant le déchiffrement, les données de texte en clair devraient être hachées avec SHA-1, incluant les données de préfixe ainsi que l'étiquette de paquet et le champ Longueur du paquet Code de détection de modification. Le corps du paquet MDC, au déchiffrement, est comparé au résultat du hachage SHA-1.

Tout échec du MDC indique que le message a été modifié et DOIT être traité comme un problème de sécurité. Les échecs incluent une différence dans les valeurs de hachage, mais aussi l'absence d'un paquet MDC, ou un paquet MDC dans toute position autre que la fin du texte en clair. Tout échec DEVRAIT être rapporté à l'utilisateur.

Note : les futures conceptions de nouvelles versions de ce paquet devraient considérer les attaques de dégradation de version car il va être possible à un attaquant de changer la version pour qu'elle revienne à 1.

Explication non normative : Le système MDC, comme sont appelés les paquets 18 et 19, a été créé pour fournir un mécanisme d'intégrité qui est moins fort qu'une signature, mais plus fort que le chiffrement CFB nu. C'est une limitation du chiffrement CFB qu'un dommage au texte chiffré va corrompre les blocs de chiffrement affectés et le bloc suivant. De plus, si des données sont retirées de la fin du bloc de CFB chiffré, cette suppression est indétectable. (Noter aussi que le mode CBC a une limitation similaire, mais les données retirées de la tête du bloc sont indétectables.)

La façon évidente de protéger ou authentifier un bloc chiffré est de le signer numériquement. Cependant, de nombreuses personnes ne souhaitent pas généralement signer les données, pour un grand nombre de raisons qui sortent du domaine d'application du présent document. Il est suffisant de dire que de nombreuses personnes considèrent que des propriétés comme la possibilité de renier sont aussi valables que l'intégrité.

OpenPGP traite ce désir d'avoir plus de sécurité qu'un chiffrement brut et tout en préservant la possibilité de renier avec le système MDC. Un MDC n'est intentionnellement pas un MAC. Son nom n'a pas été choisi par accident. Il est analogue à une somme de contrôle.

En dépit du fait que c'est un système relativement modeste, il a fait ses preuves dans la réalité. C'est une défense efficace contre plusieurs attaques qui sont apparues depuis sa création. Il a admirablement satisfait à ses objectifs modestes.

Par conséquent, comme c'est un système de sécurité modeste, il a des exigences modestes quant aux fonctions de hachage qu'il emploie. Il n'exige pas qu'une fonction de hachage soit sans collision, mais que la fonction de hachage soit unidirectionnelle. Si un faussaire, Frank, souhaite envoyer à Alice un message non signé (numériquement) qui dit "Je t'ai toujours aimé secrètement, signé Bob", il lui est bien plus facile de construire un nouveau message que de modifier quelque chose intercepté de Bob. (Noter aussi que si Bob souhaite communiquer secrètement avec Alice, mais sans authentification ou identification et avec un modèle de menaces qui inclue des fuux, il a un problème qui va bien au delà de la simple cryptographie.)

Noter aussi qu'à la différence de presque tous les autres sous système OpenPGP, il n'y a pas de paramètres dans le système MDC. Il définit obligatoirement SHA-1 comme fonction de hachage. Ce n'est pas par accident. C'est un choix intentionnel pour éviter les attaques en dégradation et en croisement tout en rendant le système simple et rapide. (Une attaque en dégradation serait une attaque qui remplacerait SHA-256 par SHA-1, par exemple. Une attaque de croisement remplacerait SHA-1 par un autre hachage à 160 bits, comme RIPE-MD/160, par exemple.)

Cependant, étant donné l'état présent de la cryptanalyse et de la cryptographie des fonctions de hachage, il peut être souhaitable de mettre à niveau le système MDC avec une nouvelle fonction de hachage. Voir le paragraphe 13.11 dans les "Considérations relatives à l'IANA" pour des lignes directrices.

5.14 Paquet Code de détection de modification (étiquette 19)

Le paquet Code de détection de modification contient un hachage SHA-1 des données de texte en clair, qui est utilisé pour détecter une modification de message. Il est seulement utilisé avec un paquet Données protégées en intégrité par chiffrement symétrique . Le paquet Code de détection de modification DOIT être le dernier paquet dans les données de texte en clair qui sont chiffrées dans le paquet Données protégéres en intégrité par chiffrement symétrique, et NE DOIT PAS apparaître dans un autre endroit.

Un paquet Code de détection de modification DOIT avoir une longueur de corps de 20 octets.

Le corps de ce paquet consiste en un hachage SHA-1 de 20 octets des données de texte en clair précédent du paquet Données protégées en intégrité par chiffrement symétrique, incluant les donn"es de préfixe, l'octet d'étiquette, et l'octet de longueur du paquet Code de détection de modification.

Noter que le paquet Code de détection de modification DOIT toujours utiliser un codage du nouveau format de l'étiquette de paquet, et un codage d'un octet de la longueur de paquet. La raison en est que les règles de hachage pour la détection de modification incluent une étiquette de un octet et une longueur de un octet dans le hachage des données. Bien que ce soit un peu restrictif, cela réduit la complexité.

6. Conversions de Radix-64

Comme indiqué dans l'introduction, la représentation native sous-jacente de OpenPGP pour les objets est un flux d'octets arbitraire, et certains systèmes désirent que ces objets soient protégés contre les dommages causés par la traduction de jeux de caractères, les conversions de données, etc.

En principe, tout schéma de codage imprimable qui satisfait aux exigences du canal non sûr devrait suffire, car il ne va pas changer les flux binaire sous-jacents des structures de données de OpenPGP natif. La norme OpenPGP spécifie un tel schéma de codage imprimable pour assurer l'interopérabilité.

Le codage Radix-64 de OpenPGP est composé de deux parties : un codage base64 des données binaires et une somme de contrôle. Le codage base64 est identique au codage de transfert de contenu base64 de MIME [RFC2045].

La somme de contrôle est un contrôle de redondance cyclique (CRC, *Cyclic Redundancy Check*) de 24 bits converti en quatre caractères de codage radix-64 par la même transformation base64 de MIME, précédée d'un signe égal (=). Le CRC est calculé en utilisant le générateur 0x864CFB et une initialisation de 0xB704CE. L'accumulation est faite sur les données avant qu'elles soient converties en radix-64, plutôt que sur les données converties. Un exemple de mise en œuvre de cet algorithme figure au paragraphe suivant.

La somme de contrôle avec son signe égal en tête PEUT apparaître sur la première ligne après les données codées en base64.

Raison du CRC-24 : la taille de 24 bits tient parfaitement dans le base64 imprimable. L'initialisation non zéro peut détecter plus d'erreurs qu'une initialisation zéro.

6.1 Mise en œuvre de CRC-24 en "C"

```
#define CRC24_INIT 0xB704CEL
#define CRC24_POLY 0x864CFBL

    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 Formation d'une armure ASCII

Quand OpenPGP code des données dans une armure ASCII, il met des en-têtes spécifiques autour des données codées en Radix-64, afin que OpenPGP puisse reconstruire les données ultérieurement. Une mise en œuvre de OpenPGP PEUT utiliser l'armure ASCII pour protéger les données binaires brutes. OpenPGP informe l'utilisateur de la sorte de données qui sont codées dans l'armure ASCII par l'utilisation des en-têtes.

L'enchaînement des données suivantes crée l'armure ASCII :
- Une ligne En-tête d'armure, appropriée au type des données
- Les en-têtes d'armure
- Une ligne blanche (longueur zéro, ou contenant seulement des espaces)
- Les données d'armure ASCII
- Une somme de contrôle d'armure
- La queue de l'armure, qui dépend de la ligne d'en-tête d'armure.

Une ligne d'en-tête d'armure consiste en le texte approprié de ligne d'en-tête entouré de cinq (5) tirets ('-', 0x2D) de chaque côté du texte de ligne d'en-tête. Le texte de la ligne d'en-tête est choisi sur la base du type des données à coder dans l'armure, et de comment il est codé. Les textes de ligne d'en-tête incluent les chaînes suivantes :

BEGIN PGP MESSAGE : utilisé pour des fichiers signés, chiffrés, ou compressés.

BEGIN PGP PUBLIC KEY BLOCK : utilisé pour l'armure de clés publiques.

BEGIN PGP PRIVATE KEY BLOCK : utilisé pour l'armure de clés privées.

BEGIN PGP MESSAGE, PART X/Y : utilisé pour les messages multi-parties, où l'armure est partagée entre Y parties, et c'est la Xième partie de Y.

BEGIN PGP MESSAGE, PART X : utilisé pour les messages multi-parties, où c'est la Xième partie d'un nombre non spécifié de parties. Exige que l'en-tête d'armure MESSAGE-ID soit utilisé.

BEGIN PGP SIGNATURE : utilisé pour les signatures détachées, les signatures OpenPGP/MIME, et les signatures en clair. Noter que PGP 2.x utilise BEGIN PGP MESSAGE pour les signatures détachées.

Noter que toutes ces lignes d'en-tête d'armure constituent une ligne complète. C'est-à-dire qu'il y a toujours une fin de ligne qui précède les cinq tirets de début, et qui suit les cinq tirets de fin. Les lignes d'en-tête DOIVENT donc commencer au début d'une ligne, et NE DOIVENT PAS avoir de texte autre que des espaces qui les suivent sur la même ligne. Ces terminaisons de ligne sont considérées comme une partie de la ligne d'en-tête d'armure pour la détermination du contenu qu'elles délimitent. Ceci est particulièrement important quand on calcule une signature en clair (voir ci-dessous).

Les en-têtes d'armure sont des paires de chaînes qui peuvent donner à l'utilisateur ou à la mise en œuvre OpenPGP receveuse des informations sur la façon de décoder ou utiliser le message. Les en-têtes d'armure sont une partie de l'armure, et non une partie du message, et donc ne sont pas protégés par les signatures appliquées au message.

Le format d'un en-tête d'armure est celui d'une paire clé-valeur. Un caractère deux-points (':' 0x38) et une seule espace (0x20) séparent la clé et la valeur. OpenPGP devrait considérer des en-têtes d'armure improprement formatés comme une corruption de l'armure ASCII. Les clés inconnues devraient être rapportées à l'utilisateur, mais OpenPGP devrait continuer de traiter le message.

Noter que certaines méhodes de transport sont sensibles à la longueur de ligne. Bien qu'il y ait une limite de 76 caractères pour les données de Radix-64 (paragraphe 6.3) il n'y a pas de limite à la longueur des en-têtes d'armure. On devrait veiller à ce que les en-têtes d'armure soient assez courts pour survivre au transport. Une façon de le faire est de répéter plusieurs fois la clé d'en-tête d'armure avec des valeurs différentes à chaque fois de façon à ce qu'aucune ligne ne soit trop longue.

Les clés d'en-tête d'armure actuellement définies sont les suivantes :

- "Version", déclare la mise en œuvre et la version OpenPGP utilisées pour coder le message.

- "Comment", commentaire défini par l'utilisateur. OpenPGP définit tout texte comme UTF-8. Un commentaire peut être toute chaîne UTF-8. Cependant, l'objet de l'armure est de fournir des données en sept bits pur. Par conséquent, si un commentaire a des caractères qui sont hors de la gamme US-ASCII de l'UTF, ils peuvent très bien ne pas survivre au transport.

- "MessageID", une chaîne de 32 caractères imprimables. La chaîne doit être la même pour toutes les parties d'un message multi parties qui utilise l'en-tête d'armure "PART X". Les chaînes MessageID devraient être assez uniques pour que le receveur du message puisse associer toutes les parties d'un message avec chacune des autres. Une bonne somme de contrôle ou fonction de hachage cryptographique est suffisante. Le MessageID NE DEVRAIT PAS apparaître sauf si il est dans un message multi parties. Si il apparaît, il DOIT être calculé à partir du message fini (chiffré, signé, etc.) d'une façon déterministe, plutôt que de contenir une valeur purement aléatoire. C'est pour permettre au receveur légitime de déterminer que le MessageID ne peut pas servir de moyen couvert de faire échapper des informations de clé de chiffrement.

- "Hash", une liste séparée de virgules des algorithmes de hachage utilisés dans ce message. Ce n'est utilisé que dans les messages signés en clair.

- "Charset", une description du jeu de caractères dans lequel est le texte en clair. Noter que OpenPGP définit le texte comme étant en UTF-8. Une mise en œuvre va obtenir les meilleurs résultats en traduisant et en éditant en UTF-8. Cependant, il y a de nombreuses instances où ceci est plus facile à dire qu'à faire. Aussi, il y a des communautés

d'utilisateurs qui n'ont pas besoin de UTF-8 parce qu'ils sont heureux avec un jeu de caractères comme ISO Latin-5 ou un jeu de caractères japonais. Dans de telles instances, une mise en œuvre PEUT outrepasser le UTF-8 par défaut en utilisant cette clé d'en-tête. Une mise en œuvre PEUT utiliser cette clé et toutes les traductions qu'elle veut ; une mise en œuvre PEUT l'ignorer et supposer que tout le texte est en UTF-8.

La ligne de queue d'armure est composée de la même manière que la ligne d'en-tête d'armure, sauf que la chaîne "BEGIN" est remplacée par la chaîne "END".

6.3 Codage binaire en Radix-64

Le processus de codage représente des groupes de 24 bits des bits d'entrée comme des chaînes en sortie de 4 caractères codés. En procédant de gauche à droite, un groupe d'entrée de 24 bits est formé en enchaînant trois groupes d'entrée de 8 bits. Ces 24 bits sont alors traités comme quatre groupes de 6 bits enchaînés, dont chacun est traduit en un seul chiffre dans l'alphabet Radix-64. Lors du codage d'un flux de bits avec le codage Radix-64, le flux de bits doit être supposé être ordonné avec le bit de poids fort en premier. C'est-à-dire, le premier bit dans le flux va être le bit de poids fort du premier octet, et le huitième bit va être le bit de moindre poids dans le premier octet, et ainsi de suite.

```
+-premier octet-+-second octet--+troisième 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 6 bits est utilisé comme un index dans un dispositif de 64 caractères imprimables du tableau ci-dessous. Les caractères référencés par l'index sont placés dans la chaîne de sortie.

Valeur	Codage	Valeur	Codage	Valeur	Codage	Valeur	Codage
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	(bourrage)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Le flux de sortie codé DOIT être représenté en lignes conformément à l'algorithme suivant.
Choisir une longueur de ligne maximale w non supérieure à 76. Insérer après chaque caractère w une coupure de ligne. Si le dernier =3D (le début de la somme de contrôle codée) n'est pas au début d'une ligne, une coupure de ligne PEUT être insérée avant le dernier =3D.

Un traitement spécial est effectué si moins de 24 bits sont disponibles à la fin des données à coder. Il y a trois possibilités :

1. Le dernier groupe de données a 24 bits (3 octets). Aucun traitement spécial n'est nécessaire.

2. Le dernier groupe de données a 16 bits (2 octets). Les deux premiers groupes de 6 bits sont traités comme ci-dessus. Le troisième groupe de données (incomplet) a deux bits de valeur zéro ajoutés, et il est traité comme ci-dessus. Un caractère de bourrage (=) est ajouté au résultat.

3. Le dernier groupe de données a 8 bits (1 octet). Le premier groupe de 6 bits est traité comme ci-dessus. Le second groupe de données (incomplet) a quatre bits de valeur zéro ajoutés, et il est traité comme ci-dessus. Deux caractères de bourrage (=) sont ajoutés au résultat.

6.4 Décodage de Radix-64

Dans les données Radix-64, les caractères autres que ceux du tableau, les coupures de ligne, et autres espaces indiquent probablement une erreur de transmission, au sujet de laquelle un message d'avertisement ou même un rejet de message peut être approprié dans certaines circonstances. Après cette vérification, le logiciel de décodage doit ignorer toutes les espaces.

Parce qu'il est seulement utilisé pour le bourrage à la fin des données, l'occurrence de tout caractère "=" peut être prise comme preuve que la fin des données a été atteinte (sans troncature dans le transit). Une telle assurance n'est pas possible, cependant, quand le nombre des octets transmis était un multiple de trois et qu'aucun caractère "=" n'est présent.

6.5 Exemples de base64

Données d'entrée : 0x14FB9C03D97E

```
Hex:     1   4    F   B    9   C     | 0   3    D   9    7   E
8-bit:   00010100 11111011 10011100  | 00000011 11011001 01111110
6-bit:   000101 001111 101110 011100 | 000000 111101 100101 111110
Decimal: 5      15     46     28       0      61     37     62
Output:  F      P      u      c        A      9      l      +
```

Données d'entrée : 0x14FB9C03D9

```
Hex:     1   4    F   B    9   C     | 0   3    D   9
8-bit:   00010100 11111011 10011100  | 00000011 11011001
                                                bourré avec 00
6-bit:   000101 001111 101110 011100 | 000000 111101 100100
Decimal: 5      15     46     28       0      61     36
                                                 pad with =
Output:  F      P      u      c        A      9      k      =
```

Données d'entrée : 0x14FB9C03

```
Hex:     1   4    F   B    9   C     | 0   3
8-bit:   00010100 11111011 10011100  | 00000011
                                       bourré avec 0000
6-bit:   000101 001111 101110 011100 | 000000 110000
Decimal: 5      15     46     28       0      48
                                         bourré avec =      =
Output:  F      P      u      c        A      w      =      =
```

6.6 Exemple d'un message à armure ASCII

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

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

Noter que cet exemple a des retraits supplémentaires ; un message en armure réel n'aurait pas d'espaces en tête de ligne.

7. Cadre pour la signature en clair

Il est souhaitable d'être capable de signer un flux d'octets textuels sans armure ASCII sur le flux lui-même, afin que le texte signé reste lisible sans logiciel spécial. Afin de lier une signature à un tel texte en clair, on utilise ce cadre. (Noter que ce cadre n'est pas destiné à être réversible. La [RFC3156] définit un autre moyen pour signer les messages en clair pour des environnements qui prennent en charge MIME.)

Le message en clair signé consiste en :
- L'en-tête en clair '-----BEGIN PGP SIGNED MESSAGE-----' sur une seule ligne,
- Une ou plusieurs en-têtes d'armure "Hash",
- Exactement une ligne vide non incluse dans le résumé de message,
- Le texte en clair avec échappement de tirets qui est inclus dans le résumé de message,

- La ou les signatures à armure ASCII incluant les lignes d'en-tête et d'en-queue d'armure '-----BEGIN PGP SIGNATURE-----'.

Si l'en-tête d'armure "Hash" est donné, le ou les algorithmes de résumé de message spécifiés sont utilisés pour la signature. Si il n'y a pas ces en-têtes, MD5 est utilisé. Si MD5 est le seul hachage utilisé, une mise en œuvre PEUT alors omettre cet en-tête pour une compatibilité améliorée avec V2.x. Si plus d'un résumé de message est utilisé dans la signature, l'en-tête d'armure "Hash" contient une liste délimitée par des virgules des résumés de message utilisés.

Les noms actuels de résumé de message sont décrits ci-dessous avec les identifiants d'algorithme.

Une mise en œuvre DEVRAIT ajouter une coupure de ligne après le texte en clair, mais PEUT l'omettre si le texte en clair se temine sur une coupure de ligne. C'est pour la clarté visuelle.

7.1 Texte à échappement de tiret

Le contenu de texte en clair du message doit aussi être échappé par tiret.

Le texte en clair échappé par tiret est le texte en clair ordinaire où chaque ligne commençant par un tiret '-' (0x2D) est préfixée par la séquence tiret '-' (0x2D) et espace ' ' (0x20). Cela empêche l'analyseur de reconnaître des en-têtes d'armure dans le texte en clair lui-même. Une mise en œuvre PEUT échapper par tiret toute ligne, DEVRAIT échapper par tiret les lignes commençant par "From" suivi par une espace, et DOIT échapper par tiret toute ligne commençant par un tiret. Le résumé de message est calculé en utilisant le texte en clair lui-même, pas la forme échappée par tiret.

Comme avec les signatures binaires sur les documents de texte, une signature en clair est calculée sur le texte en utilisant les terminaisons de ligne canoniques <CR><LF>. Les terminaisons de ligne (c'est-à-dire, <CR><LF>) avant la ligne '-----BEGIN PGP SIGNATURE-----' qui termine le texte signé n'est pas considérée comme faisant partie du texte signé.

Quand elle inverse l'échappement par tiret, une mise en œuvre DOIT supprimer la chaîne "- " si elle se produit au début d'une ligne, et DEVRAIT attirer l'attention sur le caractère "-" et tout caractère autre qu'espace au début d'une ligne.

Aussi, toutes les espaces en queue -- espaces (0x20) et tabulations (0x09) -- à la fin de toute ligne sont supprimées quand la signature en clair est générée.

8. Expressions régulières

Une expression régulière est zéro, une ou plusieurs branches, séparées par '|'. Elle correspond à tout ce qui correpond à une des branches.

Une branche est zéro, une ou plusieurs pièces, enchaînées. Elle équivaut à une correspondance pour la première, suivie par une correspondance pour la seconde, etc.

Une pièce est un atome éventuellement suivi par '*', '+', ou '?'. Un atome suivi par '*' correspond à une séquence de 0, une ou plusieurs correspondances de l'atome. Un atome suivi par '+' correspond à une séquence de une ou plusieurs correspondances de l'atome. Un atome suivi par '?' correspond à une correspondance de l'atome, ou à la chaîne nulle.

Un atome est une expression régulière entre parenthèses (correspondant à une correspondance pour l'expression régulière) une gamme (voir ci-dessous) '.' (correspondant à tout caractère seul) '^' (correspondant à la chaîne nulle au début de la chaîne d'entrée) '$' (correspondant à la chaîne nulle à la fin de la chaîne d'entrée) une '\' suivie par un seul caractère (correspondant à ce caractère) ou un seul caractère sans autre signification (correspondant à ce caractère).

Une gamme est une séquence de caractères inclus entre '[]'. Elle correspond normalement à tout caractère seul de la séquence. Si la séquence commence par '^', elle correspond à tout caractère seul qui n'est pas du reste de la séquence. Si deux caractères dans la séquence sont séparés par '-', c'est un abrégé pour la liste complète des caractères ASCII entre eux (par exemple, '[0-9]' correspond à tout chiffre décimal). La séquence de collation est UTF-8. Pour inclure un ']' littéral dans la séquence, on en fait le premier caractère (suivant un possible '^'). Pour inclure un '-' littéral, on en fait le premier ou le dernier caractère.

9. Constantes

Cette Section décrit les constantes utilisées dans OpenPGP.

Noter que ces tableaux ne sont pas des listes exhaustives ; une mise en œuvre PEUT appliquer un algorithme qui n'est pas sur ces listes, pour autant que les numéros d'algorithme soient choisis dans la gamme des algorithmes privés ou expérimentaux.

Voir la Section 13 "Notes sur les algorithmes" pour la discussion des algorithmes.

9.1 Algorithmes de clé publique

ID	Algorithme
1	- RSA (chiffrement ou signature) [HAC]
2	- RSA chiffrement seul [HAC]
3	- RSA signature seule [HAC]
16	- Elgamal (chiffrement seul) [ELGAMAL] [HAC]
17	- DSA *(Digital Signature Algorithm)* [FIPS186] [HAC]
18	- Réservé pour courbe elliptique
19	- Réservé pour ECDSA
20	- Réservé (anciennement Elgamal chiffrement ou signature)
21	- Réservé pour Diffie-Hellman (X9.42, comme défini pour IETF-S/MIME)
100 à 110	- Algorithme privé/expérimental

Les mises en œuvre DOIVENT appliquer DSA pour les signatures, et Elgamal pour le chiffrement. Les mises en œuvre DEVRAIENT appliquer les clés RSA (1). RSA chiffrement seul (2) et RSA signature seule (3) sont déconseillés et NE DEVRAIENT PAS être générés, mais peuvent être interprétés (voir le paragraphe 13.5). Voir au paragraphe 13.8 les notes sur la courbe elliptique (18), ECDSA (19), Elgamal chiffrement ou signature (20), et X9.42 (21). Les mises en œuvre PEUVENT appliquer tout autre algorithme.

9.2 Algorithmes de clés symétriques

ID	Algorithme
0	- Texte en clair ou données non chiffrées
1	- IDEA [IDEA]
2	- TripleDES (DES-EDE, [SCHNEIER] [HAC] - clé de 168 bits déduite de 192)
3	- CAST5 (clé de 128 bits, d'après la [RFC2144])
4	- Blowfish (clé de 128 bits, 16 tours) [BLOWFISH]
5	- Réservé
6	- Réservé
7	- AES avec clé de 128 bits [AES]
8	- AES avec clé de 192 bits
9	- AES avec clé de 256 bits
10	- Twofish avec clé de 256 bits [TWOFISH]
100 à 110	- Algorithme privé/expérimental

Les mises en œuvre DOIVENT utiliser TripleDES. Les mises en œuvre DEVRAIENT utiliser AES-128 et CAST5. Les mises en œuvre qui interopèrent avec PGP 2.6 ou antérieur doivent prendre en charge IDEA, car c'est le seul chiffrement symétrique qu'utilisent ces versions. Les mises en œuvre PEUVENT utiliser tout autre algorithme.

9.3 Algorithmes de compression

ID	Algorithme
0	- Non compressé
1	- ZIP [RFC1951]
2	- ZLIB [RFC1950]
3	- BZip2 [BZ2]
100 à 110	- Algorithme privé/expérimental

Les mises en œuvre DOIVENT traiter les données non compressées. Les mises en œuvre DEVRAIENT appliquer ZIP. Les mises en œuvre PEUVENT utiliser tout autre algorithme.

9.4 Algorithmes de hachage

ID	Algorithme	Nom textuel
1	- MD5 [HAC]	"MD5"
2	- SHA-1 [FIPS180]	"SHA1"
3	- RIPE-MD/160 [HAC]	"RIPEMD160"
4	- Réservé	
5	- Réservé	
6	- Réservé	
7	- Réservé	
8	- SHA256 [FIPS180]	"SHA256"
9	- SHA384 [FIPS180]	"SHA384"
10	- SHA512 [FIPS180]	"SHA512"
11	- SHA224 [FIPS180]	"SHA224"
100 à 110	- Algorithme privé/expérimental	

Les mises en œuvre DOIVENT traiter SHA-1. Les mises en œuvre PEUVENT traiter tout autre algorithme. MD5 est déconseillé.

10. Considérations relatives à l'IANA

OpenPGP est très paramétré, et par conséquent il y a un certain nombre de considérations sur l'allocation de paramètres pour les extensions. Cette Section décrit comment l'IANA devrait traiter les extensions au protocole, comme décrit dans ce document.

10.1 Nouveaux types de spécificateur de chaîne à clé

Les spécificateurs S2K OpenPGP contiennent un mécanisme pour que de nouveaux algorithmes transforment une chaîne en une clé. Cette spécification crée un registre des types de spécificateur S2K. Le registre inclut le type S2K, le nom de la S2K, et une référence à la spécification de définition. Les valeurs initiales de ce registre se trouvent au paragraphe 3.7.1. L'ajout d'un nouveau spécificateur S2K DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2 Nouveaux paquets

Les nouvelles caractéristiques majeures de OpenPGP sont définies par de nouveaux types de paquet. La présente spécification crée un registre des types de paquet. Le registre comporte le type de paquet, le nom du paquet, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 4.3. L'ajout d'un nouveau type de paquet DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.1 Types d'attribut d'utilisateur

Le paquet Attribut d'utilisateur permet un mécanisme extensible à d'autres types d'identification de certificat. La présente spécification crée un registre des types d'attribut d'utilisateur. Le registre inclut le type d'attribut d'utilisateur, le nom de l'attribut d'utilisateur, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.12. Ajouter un nouveau type d'attribut d'utilisateur DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.1.1 Types de sous paquet de format image

Dans les paquets Attribut d'utilisateur, il y a un mécanisme extensible à d'autres types d'attributs d'utilisateur fondés sur l'image. La présente spécification crée un registre des types de sous paquet Attribut d'image. Le registre inclut le type de sous paquet Attribut d'image, le nom du sous paquet Attribut d'image, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.12.1. Ajouter un nouveau type de sous paquet Attribut d'image DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.2 Nouveaux sous paquets Signature

Les signatures OpenPGP contiennent un mécanisme pour que des données signées (ou non signées) leur soient ajoutées pour diverses raisons dans les sous paquets Signature comme expliqué au paragraphe5.2.3.1. La présente spécification crée un registre des types de sous paquet Signature. Le registre comporte le type de sous paquet Signature, le nom du sous

paquet, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.2.3.1. Ajouter un nouveau sous paquet Signature DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.2.1 Sous paquets de données de notation de signature

Les signatures OpenPGP contiennent de plus un mécanisme pour des extensions dans les signatures. Ce sont les sous paquets Données de notation, qui contiennent une paire clé/valeur. Les notations contiennent un espace d'utilisateur qui n'est pas géré du tout et un espace IETF.

La présente spécification crée un registre des types Données de notation de signature. Le registre inclut le type de données de notation de signature, le nom des données de notation de signature, ses valeurs permises, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.2.3.16. Ajouter un nouveau sous paquet Données de notation de signature DOIT être fait par la méthode de revue par expert, comme décrit dans la [RFC2434].

10.2.2.2 Extensions de préférence de serveur de clés

Les signatures OpenPGP contiennent un mécanisme pour que les préférences sur les serveurs de clés soient spécifiées. La présente spécification crée un registre des préférences sur les serveurs de clés. Le registre inclut la clé préférence de serveur de clé, le nom de la préférence, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.2.3.17. Ajouter une nouvelle préférence de serveur de clé DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.2.3 Extensions de fanions de clé

Les signatures OpenPGP contiennent un mécanisme pour que des fanions soient spécifiés sur l'usage des clés. La présente spécification crée un registre des fanions d'usage des clés. Le registre inclut la valeur des fanions de clé, le nom du fanion, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.2.3.21. L'ajout d'un nouveau fanion d'usage de clé DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.2.4 Extensions de raison de révocation

Les signatures OpenPGP contiennent un mécanisme pour que des fanions soient spécifiés sur la raison de la révocation d'une clé. La présente spécification crée un registre des fanions "Raison de révocation". Le registre inclut la valeur des fanions "Raison de révocation", le nom du fanion, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.2.3.23. L'ajout d'un nouveau fanion de caractéristique DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.2.2.5 Caractéristiques de mise en œuvre

Les signatures OpenPGP contiennent un mécanisme pour que des fanions soient spécifiés déclarant quelles caractéristiques facultatives une mise en œuvre prend en charge. La présente spécification crée un registre des fanions de caractéristiques mises en œuvre. Le registre inclut la valeur de caractéristique mise en œuvre, le nom du fanion, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 5.2.3.24. L'ajout d'un nouveau fanion de caractéristiques mises en œuvre DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

Voir aussi au paragraphe 13.12 plus d'informations sur quand des fanions de caractéristiques sont nécessaires.

10.2.3 Nouvelles versions de paquet

Les paquets du cœur de OpenPGP ont tous des numéros de version, et peuvent être révisés par l'introducion d'une nouvelle version d'un paquet existant. La présente spécification crée un registre des types de paquet. Le registre inclut le type de paquet, le numéro de la version, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent à la Section 5. L'ajout d'une nouvelle version de paquet DOIT être faite par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.3 Nouveaux algorithmes

La Section 9 fait la liste des algorithmes principaux que OpenPGP utilise. Ajouter un nouvel algorithme est généralement simple. Par exemple, ajouter un nouveau chiffrement symétrique ne va normalement demander rien de plus que d'allouer une constante pour ce chiffrement. Si ce chiffrement a autre chose qu'une taille de bloc de 64 bits ou 128 bits, il pourrait y avoir besoin d'une documentation supplémentaire décrivant comment le mode CFB OpenPGP va être ajusté. De même, quand DSA a été étendu jusqu'à un maximum de clé publique de 1024 bits à 3072 bits, la révision de FIPS 186 contenait assez d'informations par elle-même pour permettre la mise en œuvre. Les changements au présent document ont été faits principalement pour le souligner.

10.3.1 Algorithmes de clé publique

OpenPGP spécifie un certain nombre d'algorithmes à clé publque. La présente spécification crée un registre des identifiants d'algorithme à clé publique. Le registre comporte le nom de l'algorithme, ses tailles de clés et paramètres, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent à la Section 9. L'ajout d'un nouvel algorithme à clé publique DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.3.2 Algorithmes de clé symétrique

OpenPGP spécifie un certain nombre d'algorithmes à clé symétrique. La présente spécification crée un registre des identifiants d'algorithme à clé symétrique. Le registre comporte le nom de l'algorithme, ses tailles de clés et tailles de bloc, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent à la Section 9. L'ajout d'un nouvel algorithme à clé symétrique DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.3.3 Algorithmes de hachage

OpenPGP spécifie un certain nombre d'algorithmes de hachage. La présente spécification crée un registre des identifiants d'algorithme de hachage. Le registre comporte le nom de l'algorithme, une représentation textuelle de ce nom, sa taille de bloc, un OID de préfixe de hachage, et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent à la Section 9 pour les identifiants d'algorithme et les noms textuels, et au paragraphe 5.2.2 pour les OID et les préfixes étendus de signature. L'ajout d'un nouvel algorithme de hachage DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

10.3.4 Algorithmes de compression

OpenPGP spécifie un certain nombre d'algorithmes de compression. La présente spécification crée un registre des identifiants d'algorithme de compression. Le registre comporte le nom de l'algorithme et une référence à la spécification de définition. Les valeurs initiales pour ce registre se trouvent au paragraphe 9.3. L'ajout d'un nouvel algorithme de compression de clé DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

11. Composition de la séquence de paquets

Les paquets OpenPGP sont assemblés en séquences afin de créer des messages et de transférer les clés. Toutes les séquences de paquet possibles ne sont pas significatives et correctes. Cette section décrit les règles de placement des paquets dans les séquences.

11.1 Clés publiques transférables

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

- un paquet de clé publique
- zéro, une ou plusieurs signatures de révocation
- un ou plusieurs paquets d'identifiant d'utilisateur
- après chaque paquet d'identifiant d'utilisateur, zéro, un ou plusieurs paquets Signature (certifications)
- zéro, un ou plusieurs paquets Attribut d'utilisateur
- après chaque paquet Attribut d'utilisateur, zéro, un ou plusieurs paquets Signature (certifications)
- zéro, un ou plusieurs paquets Sous clé
- après chaque paquet Sous clé, un paquet Signature, plus facultativement une révocation.

Le paquet de clé publique se produit en premier. Chacun des paquets Identifiant d'utiilsateur suivant fournit l'identité du possesseur de cette clé publique. Si il y a plusieurs paquets Identifiant d'utilisateur, cela correspond à plusieurs moyens d'identifier le même unique utilisateur individuel ; par exemple, un utilisateur peut avoir plus d'une adresse de messagerie électronique, et construire un identifiant d'utilisateur pour chacun.

Immédiatement à la suite de chaque paquet Identifiant d'utilisateur, il y a zéro, un ou plusieurs paquets Signature. Chaque paquet Signature est calculé sur le paquet Identifiant d'utilisateur immédiatement précédant et le paquet de clé publique initial. La signature sert à certifier la clé publique et l'identifiant d'utilisateur correspondants. En effet, le signataire atteste de sa certitude que cette clé publique appartient à l'utilisateur identifié par cet identifiant d'utilisateur.

Au sein de la même section que les paquets Identifiant d'utilisateur, il y a zéro, un ou plusieurs paquets Attribut d'utilisateur. Comme les paquets Identifiant d'utilisateur, un paquet Attribute d'utilisateur est suivi par zéro, un ou plusieurs paquets Signature calculés sur le paquet Attribut d'utilisateur immédiatement précédent et le paquet de clé publique initial.

Les paquets Attribut d'utilisateur et les paquets Identifiant d'utilisateur peuvent être librement mélangés dans cette section, tant que les signatures qui les suivent sont conservées sur le paquet Attribut d'utilisateur ou le paquet Identifiant d'utilisateur approprié, et que le premier est un paquet Identifiant d'utilisateur.

Après la séquence avec les paquets Identifiant d'utilisateur et les paquets Attribut d'utilisateur, il peut y avoir zéro, un ou plusieurs paquets Sous clé. En général, les sous clés sont fournies dans des cas où la clé publique de niveau supérieur est une clé seulement de signature. Cependant, toute clé V4 peut avoir des sous clés, et les sous clés peuvent être des clés seulement de chiffrement, des clés seulement de signature, ou des clés d'usage général. Les clés V3 NE DOIVENT PAS avoir de sous clés.

Chaque paquet Sous clé DOIT être suivi d'un paquet Signature, qui devrait être une signature de lien de sous clé produite par la clé de niveau supérieur. Pour les sous clés qui peuvent produire des signatures, la signature de lien de sous clé DOIT contenir un sous paquet Signature incorporé avec une signature de lien de clé principale (0x19) produite par la sous clé sur la clé de niveau supérieur.

Les paquets Sous clé et Clé peuvent chacun être suivis par un paquet Signature de révocation pour indiquer que la clé est révoquée. Les signatures de révocation ne sont acceptées que si elles sont produites par la clé elle-même, ou par une clé qui est autorisée à produire des révocations via un sous paquet Révocation d clé dans une auto signature par la clé de niveau supérieur.

Les séquences de clé publique transférable peuvent être enchaînées pour permettre de transférer plusieurs clés publiques en une opération.

11.2 Clés secrètes transférables

Les utilisateurs OpenPGP peuvent transférer des clés secrètes. Le format d'une clé secrète transférable est le même que celui d'une clé publique transférable sauf que les paquets de clé secrète et de sous clé secrète sont utilisés à la place des paquets de clé publique et sous clé publique. Les mises en œuvre DEVRAIENT inclure des auto signatures sur tout identifiant d'utilisateur et sous clés, car cela permet qu'une clé publique complète soit automatiquement extraite de la clé secrète transférable. Les mises en œuvre PEUVENT choisir d'omettre les auto signatures, en particulier si une clé publique transférable accompagne la clé secrète transférable.

11.3 Messages OpenPGP

Un message OpenPGP est un paquet ou une séquence de paquets qui correspond aux régles grammaticales suivantes (la virgule représente la composition séquentielle, et la barre verticale sépare les solutions de remplacement) :

message OpenPGP :- Message chiffré | Message signé | Message compressé | Message littéral.

Message compressé :- Paquet de données compressées.

Message littéral :- Paquet de données littéales.

ESK :- Paquet Clé de session chiffrée à clé publique | Paquet Clé de session chiffrée à clé symétrique.

Séquence ESK :- ESK | Séquence ESK, ESK.

Données chiffrées :- paquet de données chiffrées symétriquement | paquet de données protégées en intégrité chiffrées symétriquement.

Message chiffré :- Données chiffrées | Séquence ESK, Données chiffrées.

Message signé à une passe :- paquet Signature à une passe, message OpenPGP, paquet Signature correspondant.

Message signé :- paquet Signature, message OpenPGP | Message signé à une passe.

De plus, le déchiffrement d'un paquet Données chiffrées symétriquement ou d'un paquet Données protégées en intégrité à chiffrement symétrique ainsi que la décompression d'un paquet Données compressées doit donner un message OpenPGP valide.

11.4 Signatures détachées

Certaines applications OpenPGP utilisent ce qu'on appelle des "signatures détachées". Par exemple, un bouquet de programmes peut contenir un fichier, et avec lui un second fichier qui est une signature détachée du premier fichier. Ces signatures détachées sont simplement un paquet Signature mémorisé séparément des données pour lesquelles elles sont une signature.

12. Formats de clé améliorés

12.1 Structures de clés

Le format d'une clé OpenPGP V3 est le suivant. Les entrées entre crochets sont facultatives, les barres verticales mentionnent les solutions de remplcement, et les ellipses indiquent la répétition.

```
RSA Public Key
  [Auto signature de révocation]
   Identifiant d'utilisateur [Signature ...]
  [ Identifiant d'utilisateur [Signature ...] ...]
```

Chaque signature certifie la clé publique RSA et l'identifiant d'utilisateur précédent. La clé publique RSA peut avoir de nombreux identifiants d'utilisateur et chaque identifiant d'utilisateur peut avoir plusieurs signatures. Les clés V3 sont déconseillées. Les mises en œuvre NE DOIVENT PAS générer de nouvelles clés V3, mais PEUVENT continuer d'utiliser celles existantes.

Le format d'une clé OpenPGP V4 qui utilise plusieurs clés publiques est similaire excepté que les autres clés sont ajoutées à la fin comme "sous clés" de la clé principale.

```
Clé principale
  [Auto signature de révocation]
  [Signature de clé directe...]
   Identifiant d'utilisateur  [Signature ...]
  [Identifiant d'utilisateur  [Signature ...] ...] |
            [Attribut d'utilisateur [Signature ...] ...]
  [[Sous clé [Signature de révocation de lien]
        Signature de lien de clé principale] ...]
```

Une sous clé a toujours une seule signature après qu'elle est produite en utilisant la clé principale pour lier les deux clés. Cette signature de lien peut être en format V3 ou V4 , mais DEVRAIT être V4. Les sous clés qui peuvent produire des signatures DOIVENT avoir une signature de lien V4 parce que la signature de lien de la clé principale incorporée est EXIGÉE.

Dans le diagramme ci-dessus, si la signature de lien d'une sous clé a été révoquée, la clé révoquée peut être supprimée. Noter que cela comporte le danger d'importer la sous clé à nouveau sans la révocation de signature de lien.

Dans une clé V4, la clé principale DOIT être une clé capable de certification. Les sous clés peuvent être des clés de tout autre type. Ce peut être aussi une autre construction de clés V4. Par exemple, il peut y avoir une seule clé RSA en format V4, une clé principale DSA avec une clé de chiffrement RSA, ou une clé principale RSA avec une sous clé Elgamal, etc.

Il est aussi possible d'avoir une sous clé seulement de signature. Cela permet une clé principale qui collecte les certifications (signatures de clé) mais n'est utilisée que pour certifier les sous clés qui sont utilisées pour le chiffrement et les signatures.

12.2 Identifiants de clés et empreintes digitales

Pour une clé V3, l'identifiant de clé de huit octets consiste en les 64 bits de moindre poids du module public de la clé RSA.

L'empreinte digitale d'une clé V3 est formée par le hachage de l'enchaînement des corps des MPI (MPI sans longueur de deux octets) qui forme le matériel de clé (module public n, suivi par l'exposant e) avec MD5. Noter que les clés V3 et MD5 sont toutes deux déconseillées.

Une empreinte digitale V4 est le hachage SHA-1 de 160 bits de l'octet 0x99, suivi par la longueur de paquet de deux octets, suivie par le paquet de clé publique entier en commençant par le champ Version. L'identifiant de clé est les 64 bits de moindre poids de l'empreinte digitale. Voici les champs du matériel de hachage, avec l'exemple d'une clé DSA :
a.1) 0x99 (1 octet)
a.2) octet de longueur de poids fort de (b)-(e) (1 octet)
a.3) octet de longueur de moindre poids de (b)-(e) (1 octet)
b) numéro de version = 4 (1 octet) ;
c) horodatage de création de clé (4 octets) ;
d) algorithme (1 octet) : 17 = DSA (exemple) ;
e) champs spécifiques de l'algorithme.

Champs spécifiques de l'algorithme pour clés DSA (exemple):
e.1) MPI du nombre premier DSA p ;
e.2) MPI de groupe DSA d'ordre q (q est un nombre premier diviseur de p-1) ;
e.3) MPI de générateur de groupe DSA g ;
e.4) MPI de valeur de clé publique DSA y (= g**x mod p où x est secret).

Noter qu'il est possible qu'il y ait des collisions d'identifiants de clé -- deux clés différentes avec le même identifiant de clé. Noter qu'il y a une probabilité bien plus faible, mais pas nulle, que deux clés différentes aient la même empreinte digitale.

Noter aussi que si les clés de format V3 et V4 partagent le même matériel de clé RSA, elles vont avoir des identifiants de clé différents ainsi que des empreintes digitales différentes.

Finalement, l'identifiant de clé et l'empreinte digitale d'une sous clé sont calculés de la même façon que pour une clé principale, incluant le 0x99 comme premier octet (même si ce n'est pas un identifiant valide de paquet pour une sous clé publique).

13. Notes sur les algorithmes

13.1 Codage PKCS n° 1 dans OpenPGP

La présente norme utilise les fonctions de PKCS n° 1 EME-PKCS1-v1_5 et EMSA-PKCS1-v1_5. Cependant, les conventions d'invocation de ces fonctions ont changé dans le passé. Pour éviter une potentielle confusion et des problèmes d'interopérabilité, on inclut des copies locales dans ce document, adaptées de celles de PKCS n° 1 v2.1 [RFC3447]. La RFC 3447 devrait être traitée comme l'autorité ultime sur PKCS n° 1 pour OpenPGP. Néanmoins, on estime qu'il est valable d'avoir un document autonome qui évite les problèmes à l'avenir avec de nécessaires changements de conventions.

13.1.1 EME-PKCS1-v1_5-ENCODE

Entrée :
k = la longueur en octets du module de clé
M = message à coder, chaîne d'octets de longueur mLen, où mLen ≤ k - 11

Résultat :
EM = message codé, chaîne d'octets de longueur k
Erreur : "message trop long"

1. Vérification de longueur : Si mLen > k - 11, sortir "message trop long" et arrêter.

2. Générer une chaîne d'octets PS de longueur k - mLen - 3 consistant en octets pseudo aléatoires non zéro. La longueur de PS va être d'au moins huit octets.

3. Enchaîner PS, le message M, et autre bourrage pour former un message EM codé de longueur k octets comme
EM = 0x00 || 0x02 || PS || 0x00 || M.

4. Sortir EM.

13.1.2 EME-PKCS1-v1_5-DECODE

Entrée :
EM = message codé, chaîne d'octets

Sortie :
M = message, chaîne d'octets
Erreur : "erreur de déchiffrement"

Pour décoder un message EME-PKCS1_v1_5, on sépare le message EM codé en une chaîne d'octets PS consistant en octets non zéro et un message M comme suit : EM = 0x00 || 0x02 || PS || 0x00 || M.

Si le premier octet de EM n'a pas la valeur hexadécimale 0x00, si le second octet de EM n'a pas la valeur hexadécimale 0x02, si il n'y a pas d'octet de valeur hexadécimale 0x00 pour séparer PS de M, ou si la longueur de PS est moins de 8 octets, sortir "erreur de déchiffrement" et arrêter. Voir aussi la note de sécurité de la Section 14 concernant les différences de rapport entre une erreur de déchiffrement et une erreur de bourrage.

13.1.3 EMSA-PKCS1-v1_5

Cette méthode de codage est déterministe et a seulement une opération de codage.

Option :
Hash - fonction de hachage dans laquelle hLen note la longueur en octets du résultat de la fonction de hachage

Entrée :
M = message à coder

emLen = longueur désirée en octets du message codé, au moins tLen + 11, où tLen est la longueur en octets du codage DER T d'une certaine valeur calculée durant l'opération de codage

Résultat :
EM = message codé, chaîne d'octets de longueur emLen

Erreurs : "message trop long"; "longueur de message codée prévue trop courte"

Étapes :
1. Appliquer la fonction de hachage au message M pour produire une valeur de hachage H : H = Hash(M).
Si la fonction de hachage sort "message trop long," sortir "message trop long" et arrêter.

2. En utilisant la liste du paragraphe 5.2.2, produire une valeur DER ASN.1 pour la fonction de hachage utilisée. Soit T le préfixe de hachage complet du paragraphe 5.2.2, et soit tLen la longueur en octets de T.

3. Si emLen < tLen + 11, sortir "Longueur de message codée prévue trop courte" et arrêter.

4. Générer une chaîne d'octets PS consistant en emLen - tLen - 3 octets avec la valeur hexadécimale 0xFF. La longueur de PS va être d'au moins 8 octets.

5. Enchaîner PS, le préfixe de hachage T, et autre bourrage pour former le message codé EM comme :
EM = 0x00 || 0x01 || PS || 0x00 || T.

6. Sortir EM.

13.2 Préférences d'algorithme symétrique

La préférence d'algorithme symétrique est une liste ordonnée d'algorithmes que le détenteur de clé accepte. Comme il se trouve sur une auto signature, il est possible qu'un détenteur de clé ait plusieurs préférences différentes. Par exemple, Alice peut avoir TripleDES spécifié seulement pour "alice@work.com" mais CAST5, Blowfish, et TripleDES spécifiés pour "alice@home.org". Noter qu'il est aussi possible que les préférences soient dans une signature de lien de sous clé.

Comme TripleDES est l'algorithme de mise en œuvre obligatoire, si il n'est pas explicitement dans la liste, il est ajouté tacitement à sa fin. Cependant, il est de bonne pratique de l'y placer explicitement. Noter aussi que si une mise en œuvre n'applique pas de préférence, il est alors implicite que c'est une mise en œuvre de TripleDES seul.

Une mise en œuvre NE DOIT PAS utiliser un algorithme symétrique qui n'est pas dans la liste des préférences du receveur. Quand elle chiffre pour plus d'un receveur, la mise en œuvre trouve un algorithme convenable en prenant l'intersection des préférences des receveurs. Noter que l'algorithme de mise en œuvre obligatoire, TripleDES, assure que l'intersection n'est pas nulle. La mise en œuvre peut utiliser tout mécanisme pour prendre un algorithme dans l'intersection.

Si une mise en œuvre peut déchiffrer un message qu'un détenteur de clé n'a pas dans ses préférences, la mise en œuvre DEVRAIT déchiffrer quand même le message, mais DOIT avertir le détenteur de clé que le protocole a été violé. Par exemple, supposons qu'Alice, ci-dessus, ait un logiciel qui met en œuvre tous les algorithmes de la présente spécification. Néanmoins, elle préfère des sous ensembles pour le travail ou la maison. Si elle envoie un message chiffré avec IDEA, qui n'est pas dans ses préférences, le logiciel l'avertit que quelqu'un lui a envoyé un message chiffré par IDEA, mais il va en principe le déchiffrer de toutes façons.

13.3 Autres préférences d'algorithme

Les autres préférences d'algorithme fonctionnent de façon similaire à la préférence d'algorithme symétrique, en ce qu'elles spécifient quels algorithmes accepte le détenteur de clé. Il y a deux cas intéressants sur lesquels d'autres commentaires doivent être faits, les préférences de compression et les préférences de hachage.

13.3.1 Préférences de compression

La compression a été une partie intégrante de PGP depuis ses premiers jours. OpenPGP et toutes les versions antérieures de PGP ont offert la compression. Dans la présente spécification, le comportement par défaut est que les messages soient compressés, bien qu'une mise en œuvre ne soit pas obligée de le faire. Par conséquent, la préférence de compression donne le moyen au détenteur de clé de demander que les messages ne soient pas compressés, probablement parce que il utilise une mise en œuvre minimale qui n'inclut pas de compression. De plus, cela donne au détenteur de clé un moyen pour déclarer qu'il peut prendre en charge d'autres algorithmes.

Comme pour les préférences d'algorithme, une mise en œuvre NE DOIT PAS utiliser un algorithme qui n'est pas dans la liste de préférences. Si les préférences ne sont pas présentes, elles sont supposées être [ZIP(1), Non compressé(0)].

De plus, une mise en œuvre DOIT appliquer ces préférences au degré de reconnaissance de quand envoyer un message non compressé. Une mise en œuvre robuste va satisfaire cette exigence en cherchant la préférence du receveur et en agissant en conséquence. Une mise en œuvre minimale peut satisfaire cette exigence en ne générant jamais de message compressé, car toutes les mises en œuvre peuvent traiter les messages qui n'ont pas été compressés.

13.3.2 Préférences d'algorithme de hachage

Normalement, le choix d'un algorithme de hachage est quelque chose que fait le signataire, plutôt que le vérificateur, parce que un signataire sait rarement qui va vérifier la signature. Cette préférence permet cependant qu'un protocole fondé sur les signatures numériques facilite la négociation.

Donc, si Alice s'authentifie à Bob avec une signature, il y a du sens pour elle à utiliser un algorithme de hachage qu'utilise le logiciel de Bob. Cette préférence permet à Bob de déclarer dans sa clé quels algorithmes Alice peut utiliser.

Comme SHA1 est l'algorithme de hachage de mise en œuvre obligatoire, si il n'est pas explicitement dans la liste, il est tacitement ajouté à la fin. Cependant, il est de bonne pratique de l'y placer explicitement.

13.4 Plaintext

L'algorithme 0, "plaintext", ne peut être utilisé que pour noter des clés secrètes qui sont mémorisées en clair. Les mises en œuvre NE DOIVENT PAS utiliser le texte en clair dans les paquets de données à chiffrement symétrique ; elles doivent utiliser les paquets Données littérales pour coder les données non chiffrées ou littérales.

13.5 RSA

Il y a des types d'algorithme pour les clés RSA signature seule, et RSA chiffrement seul. Ces types sont déconseillés. Le sous paquet "Fanions de clés"dans une signature est un bien meilleur moyen d'exprimer la même idée, et la généralise à tous les algorithmes. Une mise en œuvre NE DEVRAIT PAS créer de telle clé, mais PEUT l'interpréter.

Une mise en œuvre NE DEVRAIT PAS utiliser de clés RSA de moins de 1024 bits.

13.6 DSA

Une mise en œuvre NE DEVRAIT PAS utiliser de clés DSA de moins de 1024 bits. Elle NE DOIT PAS utiliser une clé DSA d'une taille q de moins de 160 bits. Les clés DSA DOIVENT aussi être un multiple de 64 bits, et la taille q DOIT être un multiple de 8 bits. La norme de signature numérique (DSS, *Digital Signature Standard*) [FIPS186] spécifie que DSA est utilisé d'une des façons suivantes :

* clé de 1024 bits, q de 160 bits, hachage SHA-1, SHA-224, SHA-256, SHA-384, ou SHA-512
* clé de 2048 bits, q de 224 bits, hachage SHA-224, SHA-256, SHA-384, ou SHA-512
* clé de 2048 bits, q de 256 bits, hachage SHA-256, SHA-384, ou SHA-512
* clé de 3072 bits, q de 256 bits, hachage SHA-256, SHA-384, ou SHA-512

Les paires ci-dessus de taille de clé et de taille de q ont été choisies pour équilibrer au mieux la force de la clé avec la force du hachage. Les mises en œuvre DEVRAIENT utiliser une des paires de taille de clé et de q ci-dessus quand elles génèrent des clés DSA. Si la conformité à DSS est désirée, un des hachages SHA spécifié doit aussi être utilisé. [FIPS186] est l'autorité ultime sur DSS, et devrait être consulté pour toutes les questions de conformité à DSS.

Noter que les versions antérieures de la présente norme ne permettaient qu'un q de 160 bits sans troncature autorisée, de sorte que les mises en œuvre antérieures peuvent n'être pas capables de traiter les signatures avec une taille de q différente ou un hachage tronqué.

13.7 Elgamal

Une mise en œuvre NE DEVRAIT PAS utiliser de clés Elgamal de moins de 1024 bits.

13.8 Numéros d'algorithme réservés

Un certain nombre d'identifiants d'algorithmes ont été réservés pour les algorithmes dont l'utilisation serait utile dans une mise en œuvre de OpenPGP, bien qu'il y ait des problèmes qui empêchent une mise en œuvre d'utiliser en fait l'algorithme. Ils sont marqués au paragraphe 9.1, "Algorithmes de clé publique" comme "réservé pour".

Les algorithmes de clé publique réservés, Courbe elliptique (18), ECDSA (19), et X9.42 (21), n'ont pas défini les paramètres nécessaires, l'ordre des paramètres, ou leur sémantique.

Les versions antérieures de OpenPGP permettaient des signatures Elgamal [ELGAMAL] avec un identifiant de clé publique de 20. Elles ne sont plus permises. Une mise en œuvre NE DOIT PAS générer de telles clés. Une mise en œuvre NE DOIT PAS générer de signatures Elgamal. Voir [BLEICH].

13.9 Mode CFB OpenPGP

OpenPGP fait le chiffrement symétrique en utilisant une variante du mode de rebouclage de chiffrement (mode CFB). Ce paragraphe décrit en détails la procédure utilisée. Ce mode est ce qui est utilisé pour les paquets Données chiffrées symétriquement ; le mécanisme utilisé pour chiffrer le matériel de clé secrète est similaire, et est décrit au paragraphe précédent.

Dans la description ci-dessous, la valeur BS est la taille de bloc en octets du chiffrement. La plupart des chiffrements ont une taille de bloc de 8 octets. AES et Twofish ont une taille de bloc de 16 octets. Noter aussi que la description ci-dessous suppose que l'IV et le dispositif CFB commencent avec un index de 1 (à la différence du langage C, qui suppose que les dispositifs commencent à l'indice zéro).

Le mode CFB de OpenPGP utilise une valeur d'initialisation (IV) toute de zéros, et met en préfixe au texte en clair BS+2 octets de données aléatoires, telles que les octets BS+1 et BS+2 correspondent aux octets BS-1 et BS. Il fait une resynchronisation de CFB après avoir chiffré ces BS+2 octets.

Donc, pour un algorithme qui a une taille de bloc de 8 octets (64 bits) l'IV virtuelle est de 10 octets et les octets 7 et 8 de l'IV virtuelle sont les mêmes que les octets 9 et 10. Pour un algorithme d'une taille de bloc de 16 octets (128 bits) l'IV virtuelle est de 18 octets, et les octets 17 et 18 dupliquent les octets 15 et 16. Ces deux octets supplémentaires sont une vérification facile de la correction de clé.

Voici la procédure, étape par étape :
1. Le registre retour (RF, *feedback register*) est réglé à l'IV, qui est toute de zéros.
2. FR est chiffré pour produire FRE (FR chiffré). C'est le chiffrement d'une valeur toute de zéros.
3. FRE est OUixé avec les BS premiers octets des données aléatoires mises en préfixe au texte clair pour produire C[1] à C[BS], les BS premiers octets du texte chiffré.
4. FR est chargé avec C[1] à C[BS].
5. FR est chiffré pour produire FRE, le chiffrement des BS premiers octets du texte chiffré.
6. Les deux octets de gauche de FRE sont OUixés avec les deux octets suivants des données qui ont été mises en préfixe au texte en clair. Cela produit C[BS+1] et C[BS+2], les deux octets suivants du texte chiffré.
7. (L'étape de resynchronisation) FR est chargé avec C[3] à C[BS+2].
8. FR est chiffré pour produire FRE.
9. FRE est OUixé avec les BS premiers octets du texte en clair donné, maintenant qu'on a fini le chiffrement des BS+2 octets des données de préfixe. Cela donne C[BS+3] à C[BS+(BS+2)], les BS octets suivants de texte chiffré.
10. FR est chargé avec C[BS+3] à C[BS + (BS+2)] (qui est C11-C18 pour un bloc de 8 octets).
11. FR est chiffré pour produire FRE.
12. FRE est OUixé avec les BS octets suivants de texte en clair, pour produire les BS octets suivants de texte chiffré. Ils sont chargés dans FR, et le processus est répété jusqu'à ce que tout le texte en clair soit utilisé.

13.10 Paramètres privés ou expérimentaux

Les spécificateurs S2K, les types de sous paquet Signature, les types d'attribut d'utilisateur, les types de format d'image, et les algorithmes décrits à la Section 9 réservent tous la gamme 100 à 110 pour l'utilisation privée et expérimentale. Les types de paquet réservent la gamme 60 à 63 pour l'utilisation privée et expérimentale. Ils sont intentionnellement gérés avec la méthode UTILISATION PRIVÉE, comme décrit dans la [RFC2434].

Cependant, les mises en œuvre doivent faire attention avec elles et les promouvoir à des paramètres pleinement gérés par l'IANA quand ils sortent au delà de leur système original limité.

13.11 Extension au système MDC

Comme décrit dans les explications non normatives du paragraphe 5.13, le système MDC est uniquement non paramétré dans OpenPGP. C'était une décision intentionnelle pour éviter les attaques de niveau croisé. Si le système MDC est étendu à une fonction de hachage plus forte, il faut faire attention à éviter des attaques en dégradation et de niveau croisé.

Une façon simple de le faire est de créer de nouveaux paquets pour un nouveau MDC. Par exemple, au lieu que le système MDC utilise les paquets 18 et 19, un nouveau MDC pourrait utiliser 20 et 21. Cela a des inconvénient évidents (il utilise deux numéros de paquet pour chaque nouvelle fonction de hachage dans un espace qui est limité à un maximum de 60).

Une autre façon simple d'étendre le système MDC est de créer de nouvelles versions du paquet 18, et de refléter cela dans le paquet 19. Par exemple, supposons que la V2 du paquet 18 utilise implicitement SHA-256. Cela exigerait que le paquet 19 ait une longueur de 32 octets. Le changement de version dans le paquet 18 et la taille du paquet 19 empêchent une attaque en dégradation.

Il y a deux inconvénients à cette dernière approche. Le premier est que l'utilisation du numéro de version d'un paquet pour porter les informations d'algorithme n'est pas correcte du point de vue de la conception de protocole. Il est possible qu'il y ait plusieurs versions du système MDC en usage courant, mais ce désordre reflèterait un désordre dans le consensus cryptographique sur la sécurité de la fonction de hachage. Le second est que différentes versions du paquet 19 vont devoir avoir des tailles uniques. Si il y avait deux versions chacune avec des hachages de 256 bits, elles ne pourraient pas avoir toutes les deux des paquets 19 de 32 octets sans admettre les chances d'une attaques de croisement.

Encore une autre approche complexe pour étendre le système MDC serait un hybride des deux ci-dessus -- créer une nouvelle paire de paquets MDC qui sont pleinement paramétrés, et donc protégés de la dégradation et du croisement.

Tout changement au système MDC DOIT être fait par la méthode du consensus de l'IETF, comme décrit dans la [RFC2434].

13.12 Méta considérations pour l'expansion

Si OpenPGP est étendu d'une façon non rétro compatible, signifiant que les anciennes mises en œuvre ne vont pas traiter en douceur leur absence d'une nouvelle caractéristique, la proposition d'extension peut être déclarée dans l'auto signature du détenteur de la clé comme faisant partie du sous paquet Caractéristiques de signature.

On ne peut pas déclarer de façon définitive quelles extensions ne vont pas être rétro compatibles, mais normalement, les nouveaux algorithmes sont rétro compatibles, tandis que les nouveaux paquets ne le sont pas.

Si une proposition d'extension ne met pas à jour le système de caractéristiques, elle DEVRAIT inclure une explication de pourquoi ce n'est pas nécessaire. Si la proposition ne contient ni une extension du système de caractéristiques ni une explication de pourquoi cette extension est nécessaire, la proposition DEVRAIT être rejetée.

14. Considérations sur la sécurité

* Comme avec toute technologie qui implique la cryptographie, on devrait examiner la littérature actuelle pour déterminer si les algorithmes utilisés ont été trouvés vulnérables à des attaques.

* La présente spécification utilise les technologies de chiffrement à clé publique. On suppose que la portion clé privée d'une paire clé publique - clé privée est contrôlée et sécurisée par la ou les parties appropriées.

* Certaines opérations de la présente spécification impliquent l'utilisation de nombres aléatoires. Une source d'entropie appropriée devrait être utilisée pour générer ces nombres (voir la [RFC4086]).

* Il a été trouvé que l'algorithme de hachage MD5 a des faiblesses, avec des collisions dans certains cas. L'utilisation de MD5 est déconseillée dans OpenPGP. Les mises en œuvre NE DOIVENT PAS générer de nouvelles signatures utilisant MD5 comme fonction de hachage. Elles PEUVENT continuer de considérer les vieilles signatures qui utilisent MD5 comme valides.

* SHA-224 et SHA-384 exigent le même travail que SHA-256 et SHA-512, respectivement. En général, il y a quelques raisons de les utiliser en dehors de la compatibilité avec DSS. Il faut une situation où on a besoin de plus de sécurité qu'avec de plus petits hachages, mais où on ne veut pas avoir toute la longueur de données d'un 256 bits ou 512 bits complet.

* De nombreux concepteurs de protocole de sécurité pensent que c'est une mauvaise idée d'utiliser une seule clé à la fois pour la confidentialité (chiffrement) et l'intégrité (signatures). En fait, c'est une des motivations du format de clé V4 avec des clés séparées de signature et de chiffrement. Si en tant que développeur, on prône l'utilisation de clés à double usage, on devrait au moins connaître cette controverse.

* l'algorithme DSA va fonctionner avec tout hachage, mais est sensible à la qualité de l'algorithme de hachage. Les vérificateurs devraient savoir que même si le signataire a utilisé un hachage fort, un attaquant pourrait avoir modifié la signature pour en utiliser un faible. Seules les signatures qui utilisent des algorithmes de hachage d'une force acceptable devraient être acceptés comme valides.

* Comme OpenPGP combine de nombreux algorithmes asymétriques, symétriques, et de hachage différents, chacun avec des mesures de force différentes, on devrait tenir compte de ce que le plus faible élément d'un message OpenPGP est quand même suffisamment fort pour l'objectif recherché. Bien que le consensus sur la force d'un certain algorithme puisse évoluer, la publication spéciale du NIST 800-57 [SP800-57] recommande la liste suivantes de forces équivalentes :

Taille de clé asymétrique	Taille de hachage	Taille de clé symétrique
1024	160	80
2048	224	112
3072	256	128
7680	384	192
15360	512	256

* Il y a un potentiel de problème de sécurité un peu équivalent avec les signatures. Si un attaquant peut trouver un message qui hache au même hachage avec un algorithme différent, une structure de signature boguée peut être construite qui s'évalue correctement.

Par exemple, supposons que Alice signe avec DSA un message M utilisant l'algorithme de hachage H. Supposons que Mallet trouve un message M' qui a la même valeur de hachage que M avec H'. Mallet peut alors construire un bloc de signature qui se vérifie comme la signature d'Alice de M' avec H'. Cependant, cela constituerait aussi une faiblesse dans H ou H' ou les deux. Si cela devait jamais se produire, une révision du présent document devrait être faite pour revoir les algorithmes de hachage permis.

* Si on construit un système d'authentification, le receveur peut spécifier un algorithme de signature préféré. Cependant, le signataire devrait être assez fou pour utiliser un algorithme faible simplement parce que le receveur le demande.

* Certains des algorithmes de chiffrement mentionnés dans le présent document ont été moins analysés que d'autres. Par exemple, bien que CAST5 soit présentement considéré comme fort, il a moins été analysé que TripleDES. D'autres algorithmes peuvent avoir d'autres controverses à leur sujet.

* À la fin de l'été 2002, Jallad, Katz, et Schneier ont publié une intéressante attaque sur le protocole OpenPGP et certaines de ses mises en œuvre [JKS02]. Dans cette attaque, on modifie un message et on l'envoie à un utilisateur qui retourne alors le message déchiffré de façon erronée à l'attaquant. L'attaquant utilise donc l'utilisateur comme un oracle aléatoire, et peut souvent déchiffrer le message.

Compresser les données peut améliorer cette attaque. Les données incorrectement déchiffrées se décompressent presque toujours d'une façon qui déjoue l'attaque. Cependant, ceci n'est pas un correctif rigoureux, et laisse ouverte certaines petites faiblesses. Par exemple, si une mise en œuvre ne compresse pas un message avant le chiffrement (peut-être parce qu'elle sait qu'il a déjà été compressé) alors ce message est vulnérable. À cause de cet événement fortuit -- cette attaque de modification peut être déjouée par des erreurs de décompression -- une mise en œuvre DEVRAIT traiter une erreur de décompression comme un problème de sécurité, pas simplement comme un problème de données.

Cette attaque peut être déjouée par l'utilisation de la détection de modification, pourvu que la mise en œuvre ne laisse pas l'utilisateur retourner naïvement les données à l'attaquant. Une mise en œuvre DOIT traiter un échec de MDC comme un problème de sécurité, pas simpement comme un problème de données.

Dans l'un et l'autre cas, la mise en œuvre PEUT permettre l'accès de l'utilisateur aux données erronées, mais DOIT avertir l'utilisateur des potentiels problèmes de sécurité si les données sont retournées à l'envoyeur.

Bien que cette attaque soit un peu obscure, exigeant un ensemble de circonstances particulier pour la créer, elle est néanmoins assez sérieuse pour permettre à quelqu'un de conduire un utilisateur à déchiffrer un message. Par conséquent, il est important que :
1. les mises en œuvre traitent les erreur de MDC et les échecs de décompression comme des problèmes de sécurité ;
2. les mises en œuvre utilisent la détection de modification rapidement et encouragent sa diffusion ;
3. les utilisateurs migrent à des mises en œuvre qui prennent en charge la détection de modification dans les meilleurs délais.

* PKCS n° 1 s'est trouvé être vulnérable à des attaques dans lesquelles un système qui rapporte des erreurs de bourrage différemment des erreurs de déchiffrement devient un oracle aléatoire qui peut livrer la clé privée en quelques millions d'interrogations. Les mises en œuvre doivent connaître cette attaque et l'empêcher de se produire. La solution la plus simple est de rapporter un seul code d'erreur pour toutes les variantes d'erreur de déchiffrement afin de ne pas laisser fuir d'informations pour un attaquant.

* Certaines des technologies mentionnées ici peuvent faire l'objet d'un contrôle gouvernemental dans certains pays.

* À l'hiver 2005, Serge Mister et Robert Zuccherato de Entrust on publié un article qui décrit un moyen par lequel la "vérification rapide" dans OpenPGP en mode CFB peut être utilisée avec un oracle aléatoire pour déchiffrer deux octets de tout bloc de chiffrement [MZ05]. Ils recommandent comme précaution de ne pas utiliser du tout la vérification rapide.

De nombreux développeurs ont pris cet avis comme visant toutes les données chiffrées symétriquement et pour lesquelles la clé de session est chiffrée avec une clé publique. Dans ce cas, la vérification rapide n'est pas nécessaire car le chiffrement par clé publique de la clé de session devrait garantir qu'elle est la bonne clé de session. Dans d'autres cas, la mise en œuvre devrait utiliser la vérification rapide avec précaution.

Par ailleurs, il y a un danger à l'utiliser si il y a un oracle aléatoire qui peut laisser échapper des information à un attaquant. Pour parler clair, il y a un danger à utiliser la vérification rapide si des informations de temps sur la vérification peuvent être exposées à un attaquant, en particulier via un service automatique qui permet des interrogations rapidement répétées.

Par ailleurs, il n'est pas pratique pour l'utilisateur d'être informé qu'il a tapé le mauvais mot de passe seulement après qu'un peta octet de données est déchiffré. Il y a de nombreux cas dans l'ingénierie cryptographique où l'utilisateur doit faire preuve de soin et de sagesse, et ceci en est un.

15. Notes de mise en œuvre

Cette section est une collection de commentaires pour aider un utilisateur, en particulier en vue de la rétro compatibilité. Les précédentes mises en œuvre de PGP ne sont pas conformes à OpenPGP. Souvent les différences sont petites, mais les petites différences sont fréquemment plus dérangeantes que de grandes différences. Donc, ceci est une liste non exhaustive de problèmes potentiels et de recettes pour un développeur qui essaye d'être rétro compatible.

* L'algorithme IDEA est breveté, et il est exigé pour l'interopérabilité avec PGP 2.x. Il est aussi l'algorithme préféré de-facto pour une clé V3 avec une auto signature V3 (ou pas d'auto signature).

* Quand on exporte une clé privée, PGP 2.x génère l'en-tête "BEGIN PGP SECRET KEY BLOCK" au lieu de "BEGIN PGP PRIVATE KEY BLOCK". Toutes les versions précédentes ignorent le type de données impliqué, et cherchent directement le type de paquet de données.

* PGP 2.0 à 2.5 généraient des paquets Clé publique V2. Ils sont identiques aux clés V3 déconseillées sauf pour le numéro de version. Une mise en œuvre NE DOIT PAS les générer et peut les accepter ou les rejeter comme bon lui semble. Certaines plus anciennes versions de PGP généraient aussi des paquets V2 PKESK (étiquette 1). Une mise en œuvre peut accepter ou rejeter les paquets V2 PKESK comme bon lui semble, et NE DOIT PAS les générer.

* PGP 2.6.x ne va pas accepter de paquets de matériel de clé avec des versions supérieures à 3.

* Il y a de nombreuses façons possibles pour que deux clés aient le même matériel de clé, mais des empreintes digitales (et donc des identifiants de clé) différentes. Peut-être que le plus intéressant est une clé RSA qui a été "mise à niveau" au format V4, mais comme une empreinte digitale V4 est construite en hachant l'heure de création de la clé ainsi que d'autres choses, deux clés V4 créées à des heures différentes bien qu'avec le même matériel de clé vont avoir des empreintes digitales différentes.

* Si une mise en œuvre utilise zlib pour interopérer avec PGP 2.x, le paramètre "windowBits" devrait être réglé à -13.

* Les rétro signatures 0x19 n'étaient jusqu'à relativement récemment pas exigées pour signer les sous clés. Par conséquent, il peut y avoir dans la nature des clés qui n'ont pas ces rétro signatures. Le logiciel de mise en œuivre peut traiter ces clés comme bon lui semble.

* OpenPGP ne met pas de limite à la taille des clés publiques. Cependant, les plus grandes clés ne sont pas nécessairement les meilleures. Les plus grandes clés prennent plus de temps de calcul, et cela peut rapidement devenir impraticable. Des mises en œuvre OpenPGP différentes peuvent aussi utiliser des limites supérieures différentes pour leurs tailles de clé publique, et donc il faudrait faire attention quand on choisit des tailles à maintenir l'interopérabilité. En 2007, la plupart des mises en œuvre ont une limite supérieure de 4096 bits.

* L'armure ASCII est une caractéristique facultative de OpenPGP. Le groupe de travail OpenPGP s'efforce d'établir un ensemble minimal de caractéristiques de mise en œuvre obligatoire, et comme il pourrait y avoir d'utiles mises en œuvre qui utilisent seulement des formats d'objet binaire, ce n'est pas une exigence "DOIT" pour une mise en œuvre. Par exemple, une mise en œuvre qui utilise OpenPGP comme mécanisme pour les signatures de fichier peut trouver l'armure ASCII inutile. OpenPGP permet qu'une mise en œuvre déclare quelles caractéristiques elle prend ou non en charge, mais l'armure ASCII n'est pas l'une d'elles. Comme la plupart des mises en œuvre permettent que les objets binaires et en armure soient utilisés sans discrimination, une mise en œuvre qui ne met pas en œuvre l'armure ASCII peut se trouver en face de problèmes de compatibilité avec des mises en œuvre d'utilisation générale. De plus, les mises en œuvre de OpenPGP-MIME [RFC3156] ont déjà l'exigence de l'armure ASCII de sorte que ces mises en œuvre vont nécessairement devoir la prendre en charge.

16. Références

16.1 Références normatives

[AES] NIST, FIPS PUB 197, "Advanced Encryption Standard (AES)", novembre 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.{ps, pdf}

[BLOWFISH] Schneier, B. "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)" Fast Software Encryption, Cambridge Security Workshop Proceedings (décembre 1993), Springer-Verlag, 1994, pp191-204 <http://www.counterpane.com/bfsverlag.html >

[BZ2] J. Seward, jseward@acm.org, "The Bzip2 and libbzip2 home page" <http://www.bzip.org/ >

[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.

[FIPS180] "Secure Hash Signature Standard (SHS)", (FIPS PUB 180-2). <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf >

[FIPS186] Digital Signature Standard (DSS) (FIPS PUB 186-2). <http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf > FIPS 186-3 décrit des clés supérieures à 1024 bits. Le dernier projet est à : <http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3%20_March2006.pdf >

[HAC] Alfred Menezes, Paul van Oorschot, et Scott Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996. <http://www.cacr.math.uwaterloo.ca/hac/ >

[IDEA] Lai, X, "On the design et security of block ciphers", ETH Series in Information Processing, J.L. Massey (editor), Vol. 1, Hartung-Gorre Verlag Knostanz, Technische Hochschule (Zurich), 1992

[ISO10646] Norme ISO/CEI 10646, "Technologie de l'information – Jeu de caractères universel codé sur plusieurs octets (UCS) - Partie 1 : Architecture et plan multilingue de base", mai 1993.

[JFIF] Eric Hamilton, "JPEG File Interchange Format (Version 1.02)". C-Cube Microsystems, Milpitas, CA, 1er septembre 1992.

[RFC1950] P. Deutsch et J-L Gailly, "Spécification du format ZLIB de données compressées, version 3.3", mai 1996.

[RFC1951] P. Deutsch, "Spécification du format DEFLATE de données compressées, version 1.3", mai 1996.

[RFC2045] N. Freed et N. Borenstein, "Extensions de messagerie Internet multi-objets (MIME) Partie 1 : Format des corps de message Internet", novembre 1996. *(D. S., MàJ par* 2184, 2231, 5335.)

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2144] C. Adams, "L'algorithme de chiffrement CAST-128", mai 1997. *(Information)*

[RFC2434] T. Narten et H. Alvestrand, "Lignes directrices pour la rédaction d'une section Considérations relatives à l'IANA dans les RFC", BCP 26, octobre 1998. *(Rendue obsolète par la* RFC5226)

[RFC2822] P. Resnick, "Format de message Internet", avril 2001. *(Remplace la* RFC0822, STD 11, *Remplacée par* RFC5322)

[RFC3156] M. Elkins et autres, "Sécurité MIME avec OpenPGP", août 2001.*(P.S.)*

[RFC3447] J. Jonsson et B. Kaliski, "Normes de cryptographie à clés publiques (PKCS) n° 1 : Spécifications de la cryptographie RSA version 2.1", février 2003. *(Obsolète, remplacée par* RFC8017) *(Information)*

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003.

[RFC4086] D. Eastlake 3rd, J. Schiller, S. Crocker, "Exigences d'aléa pour la sécurité", juin 2005. (*Remplace* RFC1750) (BCP0106)

[SCHNEIER] Schneier, B., "Applied Cryptography Second Edition: protocols, algorithms, et source code in C", 1996.

[TWOFISH] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, et N. Ferguson, "The Twofish Encryption Algorithm", John Wiley & Sons, 1999.

16.2 Références pour information

[BLEICH] Daniel Bleichenbacher, "Generating Elgamal signatures without knowing the secret key," Eurocrypt 96. Noter que la version du compte rendu comporte une erreur. Une version révisée est disponible à <ftp://ftp.inf.ethz.ch/pub/publications/papers/ti/isc/ElGamal.ps >

[JKS02] Kahil Jallad, Jonathan Katz, Bruce Schneier "Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG" http://www.counterpane.com/pgp-attack.html

[MAURER] Ueli Maurer, "Modelling a Public-Key Infrastructure", Proc. 1996 European Symposium on Research in Computer Security (ESORICS' 96), Lecture Notes in Computer Science, Springer-Verlag, vol. 1146, pp. 325-350, septembre 1996.

[MZ05] Serge Mister, Robert Zuccherato, "An Attack on CFB Mode Encryption As Used By OpenPGP," IACR ePrint Archive : Report 2005/033, 8 février 2005, à http://eprint.iacr.org/2005/033

[REGEX] Jeffrey Friedl, "Mastering Regular Expressions," O'Reilly, ISBN 0-596-00289-0.

[RFC1423] D. Balenson, D., "Amélioration de la confidentialité pour la messagerie électronique Internet : Partie III -- Algorithmes, modes et identifiants", février 1993. *(Historique)*

[RFC1991] D. Atkins et autres, "Formats d'échange de message PGP", août 1996. (*Obsolète, voir* RFC4880) *(Information)*

[RFC2440] J. Callas, L. Donnerhacke, H. Finney et R. Thayer, "Format de message OpenPGP", novembre 1998. *(Obs. voir* 4880)

[SP800-57] NIST Special Publication 800-57, "Recommendation on Key Management", <http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf > <http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part2.pdf >

Remerciements

Ce mémoire a aussi fait des emprunts à beaucoup de travaux antérieurs d'un certain nombre d'autres auteurs, incluant : Derek Atkins, Charles Breed, Dave Del Torto, Marc Dyksterhouse, Gail Haspert, Gene Hoffman, Paul Hoffman, Ben Laurie, Raph Levien, Colin Plumb, Will Price, David Shaw, William Stallings, Mark Weaver, et Philip R. Zimmermann.

Adresse des auteurs

Le groupe de travail peut être contacté via son actuel président :

Derek Atkins
IHTFP Consulting, Inc.
4 Farragut Ave
Somerville, MA 02144
USA
mél : derek@ihtfp.com

Les auteurs principaux de ce document sont :

Jon Callas
mél : jon@callas.org

Lutz Donnerhacke
IKS GmbH
Wildenbruchstr. 15
07745 Jena,
Germany
mél : lutz@iks-jena.de

Hal Finney
mél : hal@finney.org

David Shaw
mél : dshaw@jabberwocky.com

Rodney Thayer
mél : rodney@canola-jones.com

Déclaration de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par la Internet Society.