Groupe de travail Réseau
Request for Comments : 4648
RFC rendue obsolète : 3548
Catégorie : En cours de normalisation

S. Josefsson
SJD
octobre 2006
Traduction Claude Brière de L'Isle

Codages de données Base16, Base32, et Base64

Statut du présent mémoire
Le présent document spécifie un protocole de normalisation Internet pour la communauté Internet, et appelle à discussion et suggestions en vue de son amélioration. Prière de se rapporter à l'édition en cours des normes officielles du protocole Internet (STD 1) pour connaître l'état de la normalisation et le statut du présent protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Le présent document décrit les schémas de codage base 64, base 32, et base 16 couramment utilisés. Il expose aussi l'utilisation dans les données codées des sauts à la ligne *(line-feed),* du bourrage, des caractères non alphabétiques, d'alphabets de codage différents, et des codages canoniques.

Table des Matières

1. Introduction

Le codage de base des données est utilisé dans de nombreuses situations pour mémoriser ou transférer des données dans des environnements qui, peut-être pour des raisons traditionnelles, sont restreints aux données en US-ASCII [RFC0020]. Le codage de base peut aussi être utilisé dans de nouvelles applications qui ne subissent pas ces restrictions traditionnelles, simplement parce qu'elles rendent possible la manipulation des objets avec les éditeurs de texte.

Dans le passé, différentes applications avaient des exigences différentes et donc mettaient parfois en œuvre les codages de base de façons légèrement différentes. Aujourd'hui, les spécifications de protocole utilisent parfois les codages de base en général, et le "base64" en particulier, sans une description ou référence précise. Les extensions de messagerie Internet multi objet (MIME, *Multipurpose Internet Mail Extensions*) [RFC2425] sont souvent utilisées comme référence pour le base64 sans considération des conséquences pour le saut à la ligne ou les caractères non alphabétiques. L'objet de la présente spécification est d'établir des considérations commune d'alphabet et de codage. On espère que cela réduira l'ambiguïté dans les autres documents, conduisant à une meilleure interopérabilité.

2. Conventions textuelles

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" dans le présent document sont à interpréter comme décrit dans la [RFC2119].

3. Divergences de mise en œuvre

On expose ici les divergences entre les mises en œuvre de codage de base dans le passé et, lorsque approprié, on rend obligatoire à l'avenir un comportement recommandé spécifique.

3.1 Saut à la ligne dans les données codées

MIME [RFC2425] est souvent utilisé comme référence pour le codage base 64. Cependant, MIME ne définit pas en soi le "base 64", mais plutôt un "codage de transfert de contenu en base 64" à utiliser au sein de MIME. À ce titre, MIME met en application une limite de la longueur de ligne des données codées en base 64 de 76 caractères. MIME hérite du codage de la messagerie à confidentialité améliorée (PEM, *Privacy Enhanced Mail*) [RFC1421], qui déclare que c'est "virtuellement identique" ; cependant, PEM utilise une longueur de ligne de 64 caractères. Les limites de MIME et de PEM sont toutes deux dues aux limites dans SMTP.

Les mises en œuvre NE DOIVENT PAS ajouter de retour à la ligne aux données en codage de base sauf si la spécification qui se réfère au présent document demande explicitement que les codeurs de base ajoutent des retours à la ligne après un nombre spécifique de caractères.

3.2 Bourrage de données codées

Dans certaines circonstances, l'utilisation de bourrage ("=") dans les données en codage de base n'est pas exigée ni utilisée. En règle générale, lorsque on ne peut pas faire d'hypothèses sur la taille des données transportées, il est exigé du bourrage qu'il donne les données décodées correctes.

Les mises en œuvre DOIVENT inclure des caractères de bourrage appropriés à la fin des données codées, sauf si la spécification se référant au présent document déclare explicitement le contraire.

Les alphabets base64 et base32 utilisent le bourrage, comme décrit aux sections 4 et 6, mais l'alphabet base16 n'en a pas besoin ; voir la Section 8.

3.3 Interprétation de caractères non alphabétiques dans les données codées

Les codages de base utilisent un alphabet réduit spécifique pour coder les données binaires. Des caractères non alphabétiques pourraient exister au sein des données en codage de base, causées par la corruption des données ou volontairement. Les caractères non alphabétiques peuvent être exploités comme un "canal couvert", où des données qui ne sont pas du protocole peuvent être envoyées pour des projets malveillants. Des caractères non alphabétiques peuvent aussi être envoyés afin d'exploiter des erreurs de mise en œuvre conduisant, par exemple, à des attaques de débordement de mémoire tampon.

Les mises en œuvre DOIVENT rejeter les données codées si elles contiennent des caractères qui sortent de l'alphabet de base lors de l'interprétation de données en codage de base, sauf si la spécification qui se réfère au présent document déclare

explicitement le contraire. De telles spécifications peuvent plutôt déclarer, comme le fait MIME, que les caractères hors de l'alphabet du codage de base devraient simplement être ignorés pour l'interprétation des données ("être libéral dans ce qu'on accepte"). Noter que cela signifie que des caractères retour chariot/saut à la ligne (CRLF) adjacents constituent des caractères "non alphabétiques" et sont ignorés. De plus, de telles spécifications PEUVENT ignorer le caractère de bourrage, "=", et le traiter comme non alphabétique, si il est présent avant la fin des données codées. Si on trouve plus de caractères de bourrage que le nombre admis à la fin de la chaîne (par exemple, une chaîne base 64 terminée par "==="), l'excès de caractères de bourrage PEUT aussi être ignoré.

3.4 Choix de l'alphabet

Des applications différentes ont des exigences différentes quant aux caractères de l'alphabet. Voici quelques exigences qui déterminent quel alphabet devrait être utilisé :

- Traité par l'homme. Les caractères "0" et "O" sont facilement confondus, comme le sont "1", "l", et "I". Dans l'alphabet base32 ci-dessous, où 0 (zéro) et 1 (un) sont absents, un décodeur peut interpréter 0 pour O, et 1 comme I ou L selon la casse. (Cependant, par défaut, il ne devrait pas ; voir le paragraphe précédent.)

- Codé dans des structures qui ont d'autres exigences. Pour le base16 et le base32, ceci détermine l'utilisation d'alphabets majuscules ou minuscules. Pour le base64, les caractères non alphanumériques (en particulier, "/") peuvent être problématiques dans les noms de fichiers et les URL.

- Utilisés comme identifiants. Certains caractères, en particulier "+" et "/" dans l'alphabet base64, sont traités comme des coupures de mot par les outils traditionnels de recherche/indice de texte.

Il n'y a pas d'alphabet universellement accepté qui satisfasse toutes les exigences. Pour un exemple de variante très spécialisée, voir IMAP [RFC3501]. Dans le présent document, on documente et nomme certains des alphabets couramment utilisés.

3.5 Codage canonique

L'étape du bourrage dans les codages en base64 et base32 peut, si elle est mal mise en œuvre, conduire à des altérations qui détruisent la signification des données codées. Par exemple, si l'entrée est seulement d'un octet pour un codage base64, alors tous les six bits du premier symbole sont utilisés, mais seulement les deux premiers bits du symbole suivant sont utilisés. Ces bits de bourrage DOIVENT être réglés à zéro par les codeurs conformes, qui sont décrits dans les descriptions sur le bourrage ci-dessous. Si cette propriété ne tient pas, il n'y a pas de représentation canonique des données en codage de base, et plusieurs chaînes en codage de base peuvent être décodées pour les mêmes données binaires. Si cette propriété (et les autres discutées dans le présent document) est vérifiée, un codage canonique est garanti.

Dans certains environnements, l'altération est critique et donc les décodeurs PEUVENT choisir de rejeter un codage si les bits de bourrage n'ont pas été réglés à zéro. La spécification qui se réfère à cela peut rendre obligatoire un comportement spécifique.

4. Codage en base 64

La description suivante du base 64 est dérivée des [RFC1421], [RFC2425], [RFC2440], et [RFC4033]. Ce codage peut être appelé "base64".

Le codage base64 a été conçu pour représenter des séquences arbitraires d'octets sous une forme qui permette l'utilisation des lettres majuscules et minuscules mais n'a pas besoin d'être lisible par l'homme.

Un sous jeu de 65 caractères de l'US-ASCII est utilisé, permettant à un caractère imprimable d'être représenté par 6 bits. (Le 65e caractère, "=", est utilisé pour signifier une fonction de traitement spécial.)

Le processus de codage représente des groupes de 24 bits de bits d'entrée comme des chaînes résultantes de quatre 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 caractère dans l'alphabet base64.

Chaque groupe de 6 bits est utilisé comme indice dans un dispositif de 64 caractères imprimables. Le caractère référencé par l'indice est placé dans la chaîne de sortie.

Tableau 1 : Alphabet Base 64

Codage de valeur	Codage de valeur	Codage de valeur	Codage de valeur
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 68 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	(bourrage) =
14 O	31 f	48 w	
15 P	32 g	49 x	
16 Q	33 h	50 y	

Un traitement spécial est effectué si moins de 24 bits sont disponibles à la fin des données à coder. Un quantum complet de codage est toujours complété à la fin d'une quantité. Lorsque moins de 24 bits d'entrée sont disponibles dans un groupe d'entrée, des bits de valeur zéro sont ajoutés (à droite) pour former un nombre entier de groupes de 6 bits. Le bourrage à la fin des données est effectué en utilisant le caractère '='. Comme toutes les entrées en base64 sont un nombre entier d'octets, seuls les cas suivants peuvent se présenter :

(1) Le quantum final d'entrées codées est un multiple entier de 24 bits ; ici, l'unité finale de résultat codé sera un multiple entier de 4 caractères sans bourrage de "=".

(2) Le quantum final d'entrées codées est exactement de 8 bits ; ici, l'unité finale de résultat codé sera deux caractères suivis par deux caractères "=" de bourrage.

(3) Le quantum final d'entrées codées est exactement de 16 bits ; ici, l'unité finale de résultat codé sera trois caractères suivis par un caractère "=" de bourrage.

5. Codage Base 64 avec URL et alphabet de nom de fichier sûr

Le codage base64 avec un URL et un alphabet de nom de fichier sûr a été utilisé dans [12].

Un autre alphabet a été suggéré qui utiliserait "~" comme 63e caractère. Comme le caractère "~" a une signification spéciale dans certains environnements de système de fichiers, le codage décrit dans cette section est recommandé à la place. Le caractère d'URI non réservé restant est ".", mais certains environnements de systèmes de fichiers ne permettent pas plusieurs "." dans un nom de fichier, rendant donc le caractère "." sans intérêt non plus.

Le caractère de bourrage "=" est normalement codé en pourcentage lorsque utilisé dans un URI [RFC3986], mais si la longueur des données est connue implicitement, cela peut être évité en sautant le bourrage (voir au paragraphe 3.2).

On peut se référer à ce codage sous le nom de "base64url". Ce codage ne devrait pas être considéré comme le même que le "base64" et ne devrait pas être appelé seulement "base64". Sauf précision contraire, "base64" se réfère au base 64 de la section précédente.

Ce codage est techniquement identique au précédent, sauf pour les caractères 62: et 63: de l'alphabet, comme indiqué au Tableau 2.

Tableau 2 : Alphabet Base 64 "URL et nom de fichier sûr"

Codage de valeur	Codage de valeur	Codage de valeur	Codage de valeur
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 - (moins)
12 M	29 d	46 u	63 _(souligné)
13 N	30 e	47 v	
14 O	31 f	48 w	
15 P	32 g	49 x	
16 Q	33 h	50 y	(bourrage) =

6. Codage Base 32

La description suivante du base 32 est dérivée de la [RFC4752] (avec des corrections). Ce codage peut être appelé "base32".

Le codage base32 est conçu pour représenter des séquences arbitraires d'octets sous une forme qui doit être insensible à la casse mais qui n'a pas besoin d'être lisible par l'homme.

Un sous jeu de 33 caractères de l'US-ASCII est utilisé, permettant de représenter un caractère imprimable sur 5 bits. (Le 33e caractère supplémentaire, "=", est utilisé pour signifier une fonction de traitement spécial.)

Le processus de codage représente des groupes de 40 bits de bits d'entrée comme des chaînes de sortie de 8 caractères codés. Traité de gauche à droite, un groupe de 40 bits d'entrée est formé par l'enchaînement de 5 groupes d'entrée de 8 bits. Ces 40 bits sont alors traités comme 8 groupes de 5 bits enchaînés, dont chacun est traduit dans un seul caractère dans l'alphabet base32. Lorsque un flux de bits est codé via le codage base32, le flux de bits doit être supposé ordonné avec le bit de poids fort en premier. C'est-à-dire, le premier bit dans le flux sera le bit de poids fort dans le premier octet, le huitième bit sera le bit de moindre poids dans le premier octet, et ainsi de suite.

Chaque groupe de 5 bits est utilisé comme un indice dans un dispositif de 32 caractères imprimables. Les caractères référencés par l'indice sont placés dans la chaîne de résultat. Ces caractères, identifiés dans le Tableau 3, ci-dessous, sont choisis dans les chiffres et lettres majuscules de l'US-ASCII.

Tableau 3 : Alphabet Base 32

Codage de valeur	Codage de valeur	Codage de valeur	Codage de valeur
0 A	9 J	18 S	27 3
1 B	10 K	19 T	28 4
2 C	11 L	20 U	29 5
3 D	12 M	21 V	30 6
4 E	13 N	22 W	31 7
5 F	14 O	23 X	
6 G	15 P	24 Y	(bourrage) =
7 H	16 Q	25 Z	
8 I	17 R	26 2	

Un traitement spécial est effectué si moins de 40 bits sont disponibles à la fin des données à coder. Un quantum complet de codage est toujours complété à la fin d'un corps. Lorsque moins de 40 bits d'entrée sont disponibles dans un groupe d'entrés, des bits de valeur zéro sont ajoutés sur la droite pour former un nombre entier de groupes de 5 bits. Le bourrage à la fin des données est effectué en utilisant le caractère "=". Comme toute entrée de base32 est un nombre entier d'octets, seuls les cas suivants vont se produire :

(1) Le quantum final d'entrées codées est un multiple entier de 40 bits ; ici, l'unité finale de résultat codé sera un multiple entier de 8 caractères sans "=" de bourrage.
(2) Le quantum final d'entrées codées est exactement 8 bits ; ici, l'unité finale de résultat codé sera de deux caractères suivis par six caractères "=" de bourrage.
(3) Le quantum final d'entrées codées est exactement de 16 bits ; ici, l'unité finale de résultat codé sera de quatre caractères suivis par quatre caractères "=" de bourrage.
(4) Le quantum final d'entrées codées est exactement de 24 bits ; ici, l'unité finale de résultat codé sera de cinq caractères suivis par trois caractères "=" de bourrage.
(5) Le quantum final d'entrées codées est exactement de 32 bits ; ici, l'unité finale de résultat codé sera de sept caractères suivis d'un caractère "=" de bourrage.

7. Codage Base 32 avec alphabet hexadécimal étendu

La description suivante du base 32 est dérivée de la [RFC2938]. Ce codage peut être appelé "base32hex". Ce codage ne devrait pas être considéré comme le même que le codage "base32" et ne devrait pas être appelé seulement "base32". Ce codage est utilisé, par exemple, par NextSECure3 (NSEC3) [RFC5155].

Une propriété de cet alphabet, qui manque aux alphabets base64 et base32, est que les données codées conservent leur ordre de tri lorsque les données codées sont comparées au bit près.

Ce codage est identique au précédent, sauf pour l'alphabet. On donne le nouvel alphabet au Tableau 4.

Tableau 4 : Alphabet Base 32 "hexadécimal étendu"

Codage de valeur	Codage de valeur	Codage de valeur	Codage de valeur
0 0	9 9	18 I	27 R
1 1	10 A	19 J	28 S
2 2	11 B	20 K	29 T
3 3	12 C	21 L	30 U
4 4	13 D	22 M	31 V
5 5	14 E	23 N	
6 6	15 F	24 O	(bourrage) =
7 7	16 G	25 P	
8 8	17 H	26 Q	

8. Codage Base 16

La description suivante est originale mais analogue aux descriptions précédentes. Essentiellement, le codage Base 16 est le codage hexadécimal standard insensible à la casse et on peut s'y référer sous les noms de "base16" ou "hex".

On utilise un sous jeu de 16 caractères de l'US-ASCII, qui permet de représenter un caractère imprimable sur 4 bits.

Le processus de codage représente des groupes de 8 bits (octets) de bits d'entrée comme des chaînes de résultat de deux caractères codés. En procédant de gauche à droite, une entrée de 8 bits est prise dans les données d'entrée. Ces 8 bits sont alors traités comme deux groupes enchaînés de 4 bits, dont chacun est traduit dans un seul caractère dans l'alphabet base16.

Chaque groupe de 4 bits est utilisé comme indice dans un dispositif de 16 caractères imprimables. Le caractère référencé par l'indice est placé dans la chaîne de résultat.

Tableau 5 : Alphabet Base 16

Codage de valeur	Codage de valeur	Codage de valeur	Codage de valeur
0 0	4 4	8 8	12 C
1 1	5 5	9 9	13 D
2 2	6 6	10 A	14 E
3 3	7 7	11 B	15 F

À la différence du base 32 et du base 64, aucun bourrage particulier n'est nécessaire car un mot de code complet est toujours disponible.

9. Illustrations et exemples

Pour traduire du binaire en codage de base, l'entrée est mémorisée dans une structure, et le résultat est extrait. Le cas du base64 est affiché dans la figure qui suit, empruntée à la [RFC2440].

```
+-premier octet--+-second octet--+- octet trois-+
|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--+
```

Le cas du base32 est montré dans la figure qui suit, empruntée à la [RFC2938]. Chaque caractère successif dans une valeur en base32 représente cinq bits successifs de la séquence d'octets sous-jacente. Donc, chaque groupe de 8 caractères représente une séquence de 5 octets (40 bits).

```
             1          2          3
 01234567 89012345 67890123 45678901 23456789
+--------+--------+--------+--------+--------+
|< 1 >< 2| >< 3 ><|.4 >< 5.|>< 6 ><.|7 >< 8 >|
+--------+--------+--------+--------+--------+
                                        <===> 8e caractère
                                    <====> 7e caractère
                                <===> 6e caractère
                            <====> 5e caractère
                        <====> 4e caractère
                <===> 3e caractère
          <====> 2e caractère
 <===> 1er caractère
```

L'exemple suivant de donnée en base64 est tiré de la [RFC2440], avec des corrections.

```
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
Décimal :  5      15     46     28       0      61     37     62
 Sortie :  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
Décimal :  5      15     46     28       0      61     36
                                            bourré avec =
Sortie  :  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
Décimal :  5      15     46     28       0      48
                                          bourré avec =      =
 Sortie :  F      P      u      c        A      w      =      =
```

10. Vecteurs d'essai

BASE64("") = ""

BASE64("f") = "Zg=="

BASE64("fo") = "Zm8="

BASE64("foo") = "Zm9v"

BASE64("foob") = "Zm9vYg=="

BASE64("fooba") = "Zm9vYmE="

BASE64("foobar") = "Zm9vYmFy"

BASE32("") = ""

BASE32("f") = "MY======"

BASE32("fo") = "MZXQ===="

BASE32("foo") = "MZXW6==="

BASE32("foob") = "MZXW6YQ="

BASE32("fooba") = "MZXW6YTB"

BASE32("foobar") = "MZXW6YTBOI======"

BASE32-HEX("") = ""

BASE32-HEX("f") = "CO======"

BASE32-HEX("fo") = "CPNG===="

BASE32-HEX("foo") = "CPNMU==="

BASE32-HEX("foob") = "CPNMUOG="

BASE32-HEX("fooba") = "CPNMUOJ1"

BASE32-HEX("foobar") = "CPNMUOJ1E8======"

BASE16("") = ""

BASE16("f") = "66"

BASE16("fo") = "666F"

BASE16("foo") = "666F6F"

BASE16("foob") = "666F6F62"

BASE16("fooba") = "666F6F6261"

BASE16("foobar") = "666F6F626172"

11. Mise en œuvre ISO C99 de Base64

Une mise en œuvre ISO C99 de codage et décodage Base64 dont on pense qu'elle suit toutes les recommandations de la présente RFC est disponible à : http://josefsson.org/base-encoding/

Ce code n'est pas normatif.

Le code n'a pas pu être inclus dans la présente RFC pour des raisons de procédure (RFC 3978, paragraphe 5.4).

12. Considérations sur la sécurité

Lors de la mise en œuvre d'un codage et décodage de base, il faut faire attention à ne pas introduire de vulnérabilités aux attaques de débordement de mémoire tampon, ou d'autres attaques sur la mise en œuvre. Un décodeur ne devrait pas couper sur une entrée invalide incluant, par exemple, des caractères NUL (ASCII 0) incorporés.

Si des caractères non alphabétiques sont ignorés, au lieu de causer le rejet du codage entier (comme recommandé) un canal couvert qui peut être utilisé pour des "fuites" d'informations est rendu possible. Les caractères ignorés pourraient aussi être utilisés pour d'autres dessins abominables, comme d'éviter la comparaison d'égalité de chaîne ou de déclencher des erreurs de mise en œuvre. Les implications de l'action d'ignorer des caractères non alphabétiques devraient être bien comprises dans les applications qui ne suivent pas les pratiques recommandées. De même, lorsque les alphabets base 16 et base 32 sont traités comme insensibles à la casse, l'altération de casse peut être utilisée pour faire fuir des informations ou faire échouer des comparaisons d'égalité de chaîne.

Lorsque le bourrage est utilisé, il y a des bits non significatifs qui garantissent des problèmes de sécurité, car ils peuvent être détournés pour faire fuir des informations ou utilisés pour outrepasser des comparaisons d'égalité de chaîne ou pour déclencher des problèmes de mise en œuvre.

Le codage de base cache visuellement des informations autrement aisément reconnaissables, comme les mots de passe, mais n'apporte aucune confidentialité de calcul. Il est connu que cela a causé des incidents de sécurité lorsque, par exemple, un utilisateur rapporte les détails d'un échange de protocole réseau (peut-être pour illustrer un autre problème) et révèle accidentellement le mot de passe parce qu'il ignore que le codage de base ne protège pas le mot de passe.

Le codage de base n'ajoute pas d'entropie au texte source, mais il augmente la quantité de texte source disponible et fournit une signature pour la cryptanalyse sous la forme d'une distribution de probabilité des caractéristiques.

13. Changements depuis la RFC3548

Ajout de "alphabet hexadécimal base32 étendu", nécessaire pour préserver l'ordre de tri des données codées.

Référence à IMAP pour le codage spécial Base64 utilisé ici.

Correction de l'exemple copié de la RFC2440.

Ajout de considérations sur la sécurité concernant la fourniture d'une signature pour la cryptanalyse.

Ajout des vecteurs d'essais.

Corrections typographiques.

14. Remerciements

Plusieurs personnes ont fourni des commentaires et/ou suggestions, dont John E. Hadstate, Tony Hansen, Gordon Mohr, John Myers, Chris Newman, et Andrew Sieber. Le texte utilisé dans le présent document se fonde sur des RFC antérieures qui décrivent des utilisations spécifiques de divers codages de base. L'auteur remercie les Laboratoires RSA de leur soutien aux travaux qui ont conduit au présent document.

Cette version révisée se fonde en partie sur des commentaires et/ou suggestions de Roy Arends, Eric Blake, Brian E Carpenter, Elwyn Davies, Bill Fenner, Sam Hartman, Ted Hardie, Per Hygum, Jelte Jansen, Clement Kent, Tero Kivinen, Paul Kwiatkowski, et Ben Laurie.

15. Conditions de copie

("l'auteur", pour la suite de cette section) l'auteur ne donne aucune garantie et n'est responsable d'aucun dommage résultant des son utilisation. L'auteur accorde à tous une permission irrévocable d'utilisation, de modification, et de distribution de toute façon qui ne diminue pas les droits d'utilisation, modification, et distribution des tiers, pourvu que le travail dérivé redistribué ne contienne aucune information trompeuse sur l'auteur ou la version et ne prétende pas faussement être des documents RFC de l'IETF. Les travaux dérivés n'ont pas besoin d'être licenciés sous des termes similaires.

16. Références

16.1 Références normatives

[RFC0020] V. Cerf, "Format ASCII pour les échanges sur les réseaux", octobre 1969. (STD80)

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

16.2 Références pour information

[RFC1421] J. Linn, "Amélioration de la confidentialité pour la messagerie électronique Internet : Partie I : Chiffrement de message et procédures d'authentification", février 1993. *(Historique)*

[RFC2425] T. Howes, M. Smith, F. Dawson, "Type de contenu MIME pour informations de répertoire", septembre 1998. *(Obsolète, voir* RFC6350*) (P.S.)*

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

[RFC2938] G. Klyne, L. Masinter, "Identification des caractéristiques de support composite", septembre 2000. *(P.S.)*

[RFC3501] M. Crispin, "Protocole d'accès au message Internet - version 4rev1", mars 2003. (*MàJ par* RFC4466, RFC4469, RFC4551, RFC5032, RFC5182) *(P.S.)*

[RFC3986] T. Berners-Lee, R. Fielding et L. Masinter, "Identifiant de ressource uniforme (URI) : Syntaxe générique", STD 66, janvier 2005.

[RFC4033] R. Arends, R. Austein, M. Larson, D. Massey et S. Rose, "Introduction et exigences pour la sécurité du DNS", mars 2005.

[RFC4752] A. Melnikov, éd., "Méthode d'utilisation de l'interface de programme d'application de service générique de sécurité (GSS-API) Kerberos v5 dans le mécanisme d'authentification simple et couche de sécurité (SASL)", novembre 2006.

[RFC5155] B. Laurie et autres, "Déni d'existence authentifié à hachage de la sécurité du DNS (DNSSEC)", mars 2008. *(P.S.)*

[12] Wilcox-O'Hearn, B., "Post to P2P-hackers mailing list", http://zgp.org/pipermail/p2p-hackers/2001-September/000315.html , septembre 2001.

Adresse de l'auteur

Simon Josefsson
SJD
mél : simon@josefsson.org

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par l'activité de soutien administratif (IASA) de l'IETF.