Groupe de travail Réseau
Request for Comments : 3961
Catégorie : En cours de normalisation

K. Raeburn, MIT
février 2005
Traduction Claude Brière de L'Isle

Spécification du chiffrement et de la somme de contrôle pour Kerberos 5

Statut du présent 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.

Résumé
Le présent document décrit un cadre de travail pour la définition des mécanismes de chiffrement et de somme de contrôle à utiliser avec le protocole Kerberos, en définissant une couche d'abstraction entre le protocole Kerberos et les protocoles qui s'y rapportent, et les mécanismes réels eux-mêmes. Le document définit aussi plusieurs mécanismes. Certains sont tirés de la RFC 1510, sous une forme modifiée de façon à rentrer dans ce nouveau cadre, et occasionnellement de contenu modifié lorsque l'ancienne spécification était incorrecte. De nouveaux mécanismes sont également présentés ici. Le présent document N'INDIQUE PAS quels mécanismes peuvent être considérés comme "de mise en œuvre obligatoire".

Table des Matières

1. Introduction....2
2. Concepts....2
3. Profil d'algorithme de chiffrement....3
4. Profil d'algorithme de somme de contrôle....5
5. Profil simplifié pour les chiffrements CBC avec déduction de clé....6
  5.1 Fonction de dérivation de clés....6
  5.2 Paramètres du profil simplifié....7
  5.3 Profil de systèmes de chiffrement fondés sur le profil simplifié....8
  5.4 Profils de somme de contrôle fondés sur le profil simplifié....9
6. Profils pour les algorithmes Kerberos de chiffrement et de somme de contrôle....9
  6.1 Sommes de contrôle sans clé....9
  6.2 Types de chiffrement et de somme de contrôle fondés sur DES....10
  6.3 Types de chiffrement et de somme de contrôle fondés sur triple-DES....15
7. Utilisation du chiffrement Kerberos en dehors de la présente spécification....16
8. Numéros alloués....17
9. Notes de mise en œuvre....18
10. Considérations pour la sécurité....18
11. Considérations relatives à l'IANA....19
12. Remerciements....20
Appendice A. Valeurs d'essai....20
  A.1 n-fold....20
  A.2 mit_des_string_to_key....21
  A.3 DR et DK DES3....24
  A.4 DES3string_to_key....25
  A.5 CRC-32 modifié....25
Appendice B. Changements significatifs par rapport à la RFC 1510....25
Notes....26
Références normatives....27
Références pour information....27
Adresse de l'éditeur....28
Déclaration de droits de reproduction....28

1. Introduction

Les protocoles Kerberos [RFC4120] sont conçus pour chiffrer des messages de taille arbitraire, en utilisant une cryptographie par chiffrement de bloc ou plus rarement, une cryptographie par chiffrement de flux. Le chiffrement est utilisé pour prouver les identités des entités de réseau qui participent aux échanges de messages. Cependant, rien dans le protocole Kerberos n'exige l'utilisation d'un algorithme de chiffrement spécifique, pour autant que l'algorithme comporte certaines opérations.

Les sections qui suivent spécifient les mécanismes de chiffrement et de somme de contrôle actuellement définies pour Kerberos, ainsi qu'un cadre pour la définition de futurs mécanismes. Le codage, chaînage, bourrage, et autres exigences sont décrits pour chacun. L'appendice A donne des vecteurs d'essai pour plusieurs fonctions.

2. Concepts

Les profils des deux mécanismes de chiffrement et de somme de contrôle sont développés dans les paragraphes suivants. Chaque profil spécifie une collection d'opérations et d'attributs qui doivent être définis pour un mécanisme. La spécification d'un mécanisme Kerberos de chiffrement ou de somme de contrôle n'est pas complète si elle ne définit pas toutes ces opérations et attributs.

Un mécanisme de chiffrement doit pourvoir à la confidentialité et l'intégrité du libellé original. (L'incorporation d'une somme de contrôle permet une vérification d'intégrité, si le mode de chiffrement ne pourvoit pas lui-même à la vérification d'intégrité.) Il doit aussi pourvoir à la non malléabilité [Bellare98], [Dolev91]. L'utilisation d'un embrouilleur aléatoire conjoint au libellé est recommandée. Il ne devrait pas être possible de déterminer si deux textes chiffrés correspondent au même libellé sans la clé.

Un mécanisme de somme de contrôle (voir la note [1] à la fin de l'Annexe B) doit fournir la preuve de l'intégrité du message associé et doit en préserver la confidentialité pour le cas où il n'est pas envoyé en clair. Il devrait être impossible en pratique de trouver deux textes avec la même somme de contrôle. Il N'EST PAS exigé qu'un espion soit incapable de déterminer si deux sommes de contrôle sont pour le même message, car les messages eux-mêmes seront vraisemblablement visibles pour un tel espion.

Du fait des avancées de la cryptographie, certains envisagent imprudemment d'utiliser la même clé pour plusieurs objets. Comme les clés sont utilisées pour accomplir un certain nombre de fonctions différentes dans Kerberos, il est souhaitable d'utiliser des clés différentes pour chacun de ces objets, même si on commence avec une seule clé de long terme, appelée clé de session.

Nous allons faire cela en énumérant les différents usages des clés au sein de Kerberos et en faisant du "numéro d'usage" une entrée des mécanismes de chiffrement ou de somme de contrôle ; une telle énumération sort du domaine d'application du présent document. Les paragraphes ultérieurs définissent des gabarits de profils simplifiés pour les mécanismes de chiffrement et de somme de contrôle qui utilisent une fonction de déduction de clé appliquée à un chiffrement en mode CBC (ou similaire) et une somme de contrôle ou algorithme de hachage.

On distingue la "clé de base", spécifiée par d'autres documents, de la "clé spécifique" pour une opération spécifique de chiffrement ou de somme de contrôle. On s'attend, mais ce n'est pas obligatoire, à ce que la clé spécifique soit une ou plusieurs clés séparées déduites de la clé originale de protocole et du numéro d'usage de clé. La clé spécifique ne devrait pas être explicitement mentionnée en dehors du présent document. Le langage normal utilisé dans les autres documents devait être quelque chose comme "chiffrer cette chaîne d'octets en utilisant cette clé et le numéro d'usage" ; la génération de la clé spécifique de l'état de chiffrement (décrit au paragraphe suivant) est implicite. La création d'un nouvel objet d'état de chiffrement, ou la réutilisation d'un objet d'une opération de chiffrement précédente, peut aussi être explicite.

De nouveaux protocoles définis en termes de types de chiffrement et de somme de contrôle Kerberos devraient utiliser leurs propres valeurs d'usage de clés. Les usages de clé sont des entiers non signés de 32 bits ; zéro n'est pas permis.

Toutes les données sont supposées être sous forme de chaînes d'octets. Les environnements qui ont d'autres tailles d'octet devront émuler ce comportement pour obtenir des résultats corrects.

Il est alloué à chaque algorithme un type de chiffrement (ou "etype") ou un numéro de type de somme de contrôle, pour l'identification de l'algorithme au sein du protocole Kerberos. La liste complète des allocations actuelles de numéro de type est données à la Section 8.

3. Profil d'algorithme de chiffrement

Un profil de mécanisme de chiffrement doit définir les attributs et opérations suivants. Les opérations doivent être définies comme des fonctions au sens mathématique. Aucune entrée supplémentaire ou implicite (comme les noms de principaux ou numéros de séquence de message Kerberos) n'est permise.

format de clé de protocole
Cela décrit les valeurs de chaîne d'octet qui représentent des clés valides. Pour les mécanismes de chiffrement qui n'ont pas des espaces de clés parfaitement denses, cela va décrire la représentation utilisée pour les clés de codage. Il n'est pas nécessaire qu'elle décrive des valeurs invalides spécifiques ; tous les programmes de génération de clés devraient éviter de telles valeurs.

structure de clé spécifique
Ce n'est pas du tout un format de protocole mais une description du matériel de chiffrement dérivé de la clé choisie et utilisée pour chiffrer ou déchiffrer les données ou calculer/vérifier une somme de contrôle. Cela peut, par exemple, être une seule clé, un ensemble de clés, ou une combinaison de la clé d'origine et de données supplémentaires. Les auteurs recommandent d'utiliser une ou plusieurs clés dérivées de la clé d'origine via des fonctions de déduction de clé à sens unique.

mécanisme exigé de somme de contrôle
Cela indique un mécanisme de somme de contrôle qui doit être disponible lorsque ce mécanisme de chiffrement est utilisé. Comme Kerberos n'a pas de mécanisme incorporé pour la négociation des mécanismes de somme de contrôle, une fois qu'un mécanisme de chiffrement est décidé, le mécanisme de somme de contrôle correspondant peut être utilisé.

longueur de germe de génération de clé, K
C'est la longueur de la chaîne binaire aléatoire nécessaire pour générer une clé avec la fonction "random-to-key" (décrite ci-dessous de schéma de chiffrement. Ce doit être une valeur fixe de sorte que diverses techniques de production d'une chaîne binaire aléatoire d'une longueur donnée puissent être utilisées avec les fonctions de génération de clé.

fonctions de génération de clé
Les clés doivent être générées dans un certain nombre de cas, à partir de différents types d'entrées. Toutes les spécifications de fonctions doivent indiquer comment générer les clés dans le format de réseau approprié et doivent éviter de générer des clés qui compromettent de façon significative la confidentialité des données chiffrées, sinon le système de chiffrement lui-même. L'entropie provenant de chaque source devrait être préservée autant que possible. Nombre des entrées, bien qu'inconnues, peuvent être au moins en partie prévisibles (par exemple, une chaîne de mot de passe va vraisemblablement être entièrement dans le sous ensemble ASCII et d'une longueur très faible dans de nombreux environnements ; une chaîne semi aléatoire peut inclure des horodatages). Le bénéfice d'une telle prévisibilité pour un attaquant doit être minimisé.

chaîne à clé (chaîne UTF-8, chaîne UTF-8, opaque)->(clé de protocole)
Cette fonction génère une clé à partir de deux chaînes UTF-8 et d'une chaîne d'octets opaque. Une des chaînes est normalement la phrase de passe du principal, mais c'est généralement simplement un chaîne secrète. L'autre chaîne est une chaîne de "sel" destinée à produire différentes clés à partir du même mot de passe pour différents utilisateurs ou domaines. Bien que les chaînes fournies utilisent le codage UTF-8, on ne doit présupposer aucune version spécifique d'Unicode ; toutes les chaînes UTF-8 valides devraient être permises. Les chaînes fournies dans d'autres codages DOIVENT d'abord être converties en UTF-8 avant l'application de cette fonction. Le troisième argument, la chaîne d'octets, peut être utilisé pour faire entrer des paramètres spécifiques du mécanisme dans cette fonction. Comme cette pratique implique la connaissance du système de chiffrement spécifique, la génération de valeurs de paramètre qui ne soient pas les valeurs par défaut devrait être une opération exceptionnelle, et les applications Kerberos normales devraient être capables de traiter ce bloc de paramètres comme un objet opaque fourni par le centre de distribution de clés (KDC, *Key Distribution Center*) ou de revenir par défaut à une valeur constante spécifique du mécanisme. La fonction chaîne à clé *(string-to-key)* devrait être une fonction à sens unique de telle sorte que la compromission de la clé d'un utilisateur dans un domaine ne la compromette pas dans un autre, même si le même mot de passe (mais un sel différent) est utilisé.

aléatoire à clé (chaîne binaire[K])->(clé de protocole)
Cette fonction génère une clé à partir d'une chaîne binaire aléatoire d'une taille spécifique. Tous les bits de la chaîne d'entrée sont supposés être également aléatoires, même si l'entropie présente dans la source d'aléa peut être limitée.

déduction de clé (clé de protocole, entier)->(clé spécifique)
Dans cette fonction, l'entrée d'entier est la valeur d'usage de clé, comme décrit ci-dessus. Un attaquant est supposé connaître les valeurs d'usage. La valeur de sortie de la clé spécifique a été décrite à la Sction 2.

format du paramètre chaîne à clé
Cela décrit le format du bloc de données qui peut être passé à la fonction chaîne à clé ci-dessus pour configurer des paramètres supplémentaires pour cette fonction. Avec le mécanisme des valeurs de paramètres de codage, les limites admises des paramètres devraient aussi être décrites pour éviter de permettre à un KDC factice de compromettre le mot de passe de l'utilisateur. Si c'est pratique, il peut être souhaitable de construire le codage de telle sorte que les valeurs qui affaiblissent de façon inacceptable la clé résultante ne puissent pas être codées. La politique de sécurité locale peut permettre des limites plus strictes pour éviter une consommation excessive de ressources. S'il en est ainsi, la spécification devrait recommander des valeurs par défaut pour ces limites. La description devrait aussi souligner les possibles faiblesses si les vérifications de limites ou autres validations ne sont pas appliquées à la chaîne de paramètres reçue du réseau. Comme mentionné ci-dessus, ceci devrait être considéré comme opaque pour la plupart des applications normales.

paramètres chaîne à clé par défaut (chaîne d'octets)
Cette valeur par défaut pour l'argument "params" de la fonction chaîne à clé devrait être utilisée lorsque le protocole d'application (Kerberos ou autre) ne règle pas de façon explicite la valeur du paramètre. Comme indiqué ci-dessus, dans la plupart des cas ce bloc de paramètres devrait être traité comme un objet opaque.

état de chiffrement
Cela décrit toute information qui peut être portée d'une opération de chiffrement ou déchiffrement à l'autre, pour être utilisée avec une clé spécifique donnée. Par exemple, un bloc de chiffrement utilisé en mode CBC peut mettre une valeur initiale (IV, *initial vector*) d'un bloc dans l'état de chiffrement. D'autres modes de chiffrement peuvent garder la trace de noms occasionnels ou d'autres données. Cet état doit être non vide et doit influencer le chiffrement de telle sorte que les messages soient déchiffrés dans le même ordre que leur chiffrement, si l'état de chiffrement est transporté d'un chiffrement à l'autre. La distinction des messages décalés ou manquants des messages corrompus n'est pas exigée. Si on le désire, cela peut être fait à un niveau supérieur en incluant des numéros de séquence et en "n'enchaînant pas" l'état de chiffrement entre les opérations de chiffrement. L'état de chiffrement peut n'être pas réutilisé dans plusieurs opérations de chiffrement et déchiffrement. Ces opérations génèrent toutes un nouvel état de chiffrement qui peut être utilisé pour les opérations suivantes en utilisant la même clé et la même opération. Le contenu de l'état de chiffrement doit être traité comme opaque en dehors des spécifications de système de chiffrement.

état de chiffrement initial (clé spécifique, direction)->(état)
Cela décrit la génération de la valeur initiale de l'état de chiffrement si il n'est pas transporté d'une opération précédente de chiffrement ou déchiffrement. Cela décrit tout établissement d'état initial nécessaire avant le chiffrement de quantités arbitraires de données avec une clé spécifique donnée. La clé spécifique et la direction des opérations à effectuer (chiffrement contre déchiffrement) doit être la seule entrée nécessaire pour cette initialisation. Cet état devrait être traité comme opaque dans toute utilisation en dehors d'une définition d'algorithme de chiffrement.
Note de mise en œuvre : La [RFC1510] était vague sur la question de savoir si, et dans quelle mesure, un protocole d'application pouvait exercer un contrôle sur le vecteur initial utilisé dans les opérations CBC de DES. Certaines mises en œuvre existantes permettent le réglage de la valeur initiale. Le présent cadre de travail ne prévoit pas de contrôle d'application pour l'état de chiffrement (au-delà de "initialise" et "report du chiffrement précédent"), car la forme et le contenu de l'état de chiffrement initial peuvent varier d'un système de chiffrement à l'autre et peut n'être pas toujours un seul bloc de données aléatoires. Les nouveaux protocoles d'application Kerberos ne devraient pas supposer un contrôle sur la valeur initiale, ni même que celle-ci existe. Cependant, une mise en œuvre tout venant peut souhaiter fournir cette capacité, au cas où elle rencontrerait des applications l'établissant explicitement.

chiffre (clé spécifique, état, chaîne d'octets)->(état, chaîne d'octets)
Cette fonction prend la clé spécifique, l'état de chiffrement, et une chaîne de texte en clair non vide comme entrée et génère du texte chiffré et un nouvel état de chiffrement en sortie. Si l'algorithme de chiffrement de base ne fournit pas lui-même la protection de l'intégrité (par exemple, DES en mode CBC), une forme quelconque de MAC vérifiable ou de somme de contrôle doit être incluse. Un facteur aléatoire du genre confondeur devrait être inclus de telle sorte qu'un observateur ne puisse pas savoir si deux messages contiennent le même texte en clair, même si l'état de chiffrement et les clés spécifiques sont les mêmes. La longueur exacte du texte en clair n'est pas nécessairement codée, mais si elle ne l'est pas et si un bourrage est nécessaire, le bourrage doit être ajouté à la fin de la chaîne de sorte que la version déchiffrée puisse être analysée à partir du début. La spécification de la fonction de chiffrement doit indiquer non seulement le contenu précis de la chaîne d'octets de sortie, mais aussi l'état de chiffrement de sortie. Le protocole d'application peut porter l'état de chiffrement de sortie directement d'un chiffrement avec une clé spécifique à une autre ; l'effet de ce "chaînage" doit être défini (voir la note [2] à l'Annexe B). En supposant que les valeurs de la clé spécifique et de l'état de chiffrement sont produits correctement, aucune chaîne d'octets d'entrée ne peut résulter en une indication d'erreur.

déchiffre (clé spécifique, état, chaîne d'octets)->(état, chaîne d'octets)
Cette fonction prend la clé spécifique, l'état de chiffrement, et le texte chiffré comme entrées et vérifie l'intégrité du texte chiffré fourni. Si l'intégrité du texte chiffré est intacte, cette fonction produit le texte en clair et un nouvel état de chiffrement en sortie ; autrement, une indication d'erreur doit être retournée, et les données doivent être éliminées. Le résultat du déchiffrement peut être plus long que le texte en clair d'origine, comme, par exemple, lorsque le mode de codage

ajoute un bourrage pour atteindre un multiple d'une taille de bloc. Si c'est le cas, tout octet supplémentaire doit venir après le texte en clair décodé. Un protocole d'application qui a besoin de savoir la longueur exacte du message doit coder une longueur ou un marqueur de "fin de message" reconnaissable au sein du texte en clair (voir la note [3] à la fin de l'Annexe B). Comme avec la fonction de chiffrement, une spécification correcte de cette fonction doit indiquer non seulement le contenu de la chaîne d'octets de sortie, mais aussi l'état de chiffrement résultant.

pseudo aléatoire (clé de protocole, chaîne d'octets)->(chaîne d'octets)
Cette fonction pseudo aléatoire (PRF, *pseudo random function*) devrait générer une chaîne d'octets d'une certaine taille indépendante de la chaîne d'octets d'entrée. La chaîne de sortie PRF devrait convenir pour la génération de clés, même si la chaîne d'octets d'entrée est publique. Elle ne devrait pas révéler la clé d'entrée, même si le résultat est rendu public.

Ces opérations et attributs sont tout ce qui est exigé pour la prise en charge de Kerberos et de divers schémas de pré authentification proposés.

Pour l'agrément de certains protocoles d'application qui peuvent souhaiter utiliser le profil de chiffrement, on ajoute la contrainte que, pour toute taille d'entrée de texte en clair donnée, une taille de message doit exister entre cette taille donnée et cette taille plus 65 535 telle que la longueur de la version déchiffrée du texte chiffré n'ait jamais d'octets supplémentaires à la fin.

Exprimée mathématiquement, pour toute longueur de message L1, il existe une taille de message L2 telle que

$L2 \geq L1$
$L2 < L1 + 65\,536$
pour tout message M avec |M| = L2, decrypt(encrypt(M)) = M

Un document définissant un nouveau type de chiffrement devrait aussi décrire les faiblesses ou attaques connues, de sorte que sa sécurité puisse être attestée de bonne foi, et devrait inclure des valeurs d'essai ou autres procédures de validation pour les opérations définies. Des références spécifiques aux informations déjà disponibles ailleurs sont suffisantes.

4. Profil d'algorithme de somme de contrôle

Un profil de mécanisme de somme de contrôle doit définir les attributs et opérations suivants :

Algorithmes de chiffrement associés
Cela indique les types de clés de chiffrement avec lesquelles ce mécanisme de somme de contrôle peut être utilisé.

Un mécanisme de somme de contrôle à clés peut avoir plus d'un algorithme de chiffrement associé si ils partagent le même format de clé de réseau *(wire-key)*, la même fonction de chaîne à clé, les mêmes paramètres par défaut de chaîne à clé, et la même fonction de déduction de clé. (Cette combinaison signifie que, par exemple, un type de somme de contrôle, une valeur d'usage de clé, et un mot de passe sont adéquats pour obtenir la clé spécifique utilisée pour calculer une somme de contrôle.)

Un mécanisme de somme de contrôle sans clé peut être utilisé avec tout type de chiffrement, car la clé est ignorée, mais son utilisation doit être limitée aux cas où la somme de contrôle elle-même est protégée, pour éviter des attaques triviales.

Fonction get_mic
Cette fonction génère un jeton MIC pour une clé spécifique donnée (voir la Section 3) et un message (représenté comme une chaîne d'octets) qui peut être utilisé pour vérifier l'intégrité du message associé. Cette fonction n'est pas obligée de retourner le même résultat déterministe à chaque utilisation ; il a seulement besoin de générer un jeton que le programme verify_mic puisse vérifier. Le résultat de cette fonction va aussi dicter la taille de la somme de contrôle. Elle ne doit pas être supérieure à 65 535 octets.

Fonction verify_mic
Étant donnés une clé spécifique, un message, et un jeton MIC, cette fonction constate si l'intégrité du message a été compromise. Pour un programme get_mic déterministe, le verify_mic correspondant peut simplement générer une autre somme de contrôle et comparer les deux.

Les opérations get_mic et verify_mic doivent permettre des entrées de longueur arbitraire ; si du bourrage est nécessaire, le schéma de bourrage doit être spécifié au titre de ces fonctions.

Ces opérations et attributs sont tout ce qui devrait être exigé pour la prise en charge de Kerberos et des divers schémas de

pré authentification proposés.

Comme avec les documents de définition de mécanisme de chiffrement, les documents qui définissent de nouveaux mécanismes de somme de contrôle devraient indiquer les processus de validation et les faiblesses connues.

5. Profil simplifié pour les chiffrements CBC avec déduction de clé

Le profil dessiné aux sections 3 et 4 décrit un grand nombre d'opérations qui doivent être définies pour les algorithmes de chiffrement et de somme de contrôle à utiliser avec Kerberos. Nous décrivons ici un profil plus simple qui peut générer à la fois des définitions de chiffrement de mécanisme de somme de contrôle, remplissant les usages de déduction de clés aux endroits appropriés, fournissant la protection de l'intégrité, et définissant les multiples opérations du profil de système de chiffrement sur la base d'un plus petit ensemble d'opérations. Tous les systèmes de chiffrement existants pour Kerberos s'adaptent à ce profil simplifié, mais nous recommandons que les systèmes de chiffrement futurs l'utilisent, ou bien quelque chose fondé sur lui (voir la note [4] à la fin de l'Annexe B).

Toutes les opérations des profils complets ne sont pas définies par ce mécanisme ; plusieurs doivent toujours être définies pour chaque nouvelle paire d'algorithmes.

5.1 Fonction de dérivation de clés

Plutôt que de définir un schéma par lequel une "clé de protocole" est composée d'un grand nombre de clés de chiffrement, nous utilisons des clés dérivées d'une clé de base pour effectuer les opérations cryptographiques. La clé de base doit être utilisée seulement pour générer les clés dérivées, et cette dérivation doit être non réversible et préserver l'entropie. Avec ces restrictions, la compromission d'une clé dérivée ne compromet pas les autres. Les attaques contre la clé de base sont limitées, car elle n'est utilisée que pour la dérivation et n'est pas exposée à des données d'utilisateur.

Pour générer une clé dérivée à partir d'une clé de base, on génère une chaîne d’octets pseudo aléatoire en utilisant un algorithme DR, décrit plus loin, et on génère une clé à partir de cette chaîne d’octets en utilisant une fonction qui dépend de l'algorithme de chiffrement. La longueur d'entrée nécessaire pour cette fonction, qui dépend elle aussi de l'algorithme de chiffrement, dicte la longueur de la chaîne à générer avec l'algorithme DR (la valeur "k" ci-dessous. Ces procédures sont fondées sur la dérivation de clé dans [Blument].

Clé dérivée = DK(Clé de base, Constante bien connue)

DK(Clé, Constante) = aléatoire à clé(DR(Clé, Constante))

DR(Clé, Constante) = k-truncate(E(Clé, Constante, état-de-chiffrement-initial))

Ici, DR est la fonction de génération d'octet aléatoire décrite ci-dessous, et DK est la fonction de dérivation de clé produite pour cela. Dans cette construction, E(Clé, Libellé, ÉtatDeChiffrement) est un chiffre, Constante est une constante bien connue déterminée par l'utilisation spécifique de cette fonction, et k-truncate tronque son argument en retirant les k premiers bits. Ici, k est la longueur de germe de génération de clés nécessaire pour le système de chiffrement.

Le résultat de la fonction DR est une chaîne de bits ; la clé réelle est produite en appliquant l'opération aléatoire à clé du système de chiffrement à cette chaîne binaire.

Si la Constante est plus petite que la taille du bloc de chiffrement de E, elle doit alors être développée avec n-fold() de façon qu'elle puisse être chiffrée. Si le résultat de E est plus court que k bits, il doit être repassé dans le processus de chiffrement autant de fois que nécessaire. La construction est la suivante (où | indique l'enchaînement) :

K1 = E(Key, n-fold(Constante), état-de-chiffrement-initial)
K2 = E(Key, K1, état-de-chiffrement-initial)
K3 = E(Key, K2, état-de-chiffrement-initial)
K4 = ... DR(Key, Constante) = k-truncate(K1 | K2 | K3 | K4 ...)

n-fold est un algorithme qui prend m bits d'entrée et les "étire" pour former n bits en sortie avec une égale contribution de la part de chaque bit d'entrée au résultat, comme décrit dans [Blument]:

On définit d'abord une primitive appelée n-folding *(replie n fois)*, qui prend un bloc d'entrée de longueur variable et produit une séquence de sortie de longueur fixe. L'intention est de donner à chaque bit d'entrée un poids approximativement égal

dans la détermination de la valeur de chaque bit de sortie. Noter que chaque fois que nous avons besoin de traiter une chaîne d'octets comme un nombre, la représentation est supposée être gros boutienne – l'octet de poids fort en premier.

Pour appliquer n-fold à un nombre X, dupliquer la valeur d'entrée sur une longueur qui est le plus petit commun multiple de n et de la longueur de X. Avant chaque répétition, l'entrée subit une rotation sur la droite de 13 positions binaires. Les tronçons successifs de n-bits sont ajoutés les uns aux autres en utilisant l'addition de complément à un (c'est-à-dire, avec report circulaire) pour donner un résultat de n bits....

Les valeurs d'essai pour l'opération n-fold sont donnés à l'appendice A (Note [5]).

Dans cette section, n-fold est toujours utilisé pour produire c bits de résultat, et c est la taille du bloc de chiffrement de E.

La taille de la constante ne doit pas être supérieure à celle de c, parce que la réduction de la longueur de la constante de n fois peut causer des collisions.

Si la taille de la constante est inférieure à celle de c, la constante doit alors être répliquée n fois jusqu'à la longueur de c. Cette chaîne est utilisée comme entrée pour E. Si la taille de bloc de E est inférieure à la taille de l'entrée de aléatoire à clé, le résultat de E est alors pris comme entrée pour une seconde invocation de E. Ce processus est répété jusqu'à ce que le nombre de bits accumulés soit supérieur ou égal à la taille d'entrée de aléatoire à clé. Lorsque assez de bits ont été calculés, les k premiers sont pris comme données aléatoires pour créer la clé avec la fonction aléatoire à clé qui dépend de l'algorithme.

Comme la clé dérivée est le résultat d'un ou plusieurs chiffrements dans la clé de base, déduire la clé de base de la clé dérivée est équivalent à déterminer la clé à partir d'un très petit nombre de paires libellé/texte chiffré. Et donc, cette construction est aussi forte que le système de chiffrement lui-même.

5.2 Paramètres du profil simplifié

Voici les opérations et attributs qui doivent être définis :
- format de clé du protocole
- fonction string-to-key *(chaîne à clé)*
- paramètres string-to-key par défaut
- longueur du germe de génération de clé, k
- fonction random-to-key *(aléatoire à clé)*

Comme ci-dessus pour le profil normal de mécanisme de chiffrement.

algorithme de hachage sans clé, H
Ce devrait être un algorithme de hachage résistant aux collisions avec un résultat de taille fixe, convenant à l'utilisation dans un HMAC [RFC2104]. Il doit accepter des entrées de longueur arbitraire. Sa sortie doit être au moins de la taille du bloc de message (voir ci-dessous).

taille de sortie HMAC, h
Cela indique la taille de la sous chaîne produite en tête par la fonction HMAC qui devrait être utilisée dans les messages transmis. Elle devrait être au moins de la moitié de la taille du résultat de la fonction de hachage H, et d'au moins 80 bits ; il n'est pas nécessaire qu'elle corresponde à la taille du résultat.

taille de bloc de message, m
C'est la taille de la plus petite unité que le chiffrement peut traiter dans le mode dans lequel il est utilisé. Les messages seront bourrés à un multiple de cette taille. Si un chiffrement de bloc est utilisé dans un mode qui peut traiter les messages qui ne sont pas des multiples de la taille du bloc de chiffrement, comme le mode CBC avec dissimulation du texte chiffré (CTS, voir la [RFC2040]) cette valeur sera d'un octet. Pour le mode CBC traditionnel avec bourrage, ce serait la taille du bloc du chiffrement sous-jacent. Cette valeur doit être un multiple de huit bits (un octet).

fonctions de chiffrement/déchiffrement, E et D
Ce sont les fonctions de base de chiffrement et déchiffrement pour les messages de tailles qui sont des multiples de la taille de bloc du message. Aucune vérification d'intégrité ni de confondeur ne devrait être incluse ici. Pour les entrées de ces fonctions, prendre l'IV ou des données similaires, une clé du format de protocole, et une chaîne d'octets, retournant une nouvelle IV et chaîne d'octets.

Il n'est pas exigé de la fonction de chiffrement qu'elle utilise le mode CBC mais on suppose qu'elle utilise quelque chose qui a des propriétés similaires. En particulier, d'ajouter devant un confondeur de taille de bloc de chiffrement au texte en

clair qui devrait altérer le texte chiffré entier (comparable à choisir et inclure une valeur initiale aléatoire pour le mode CBC).
Le résultat du chiffrement d'un bloc de chiffrement (de taille c, ci-dessus) doit être déterministe pour que la fonction de génération d'octet aléatoire DR dans le paragraphe précédent fonctionne. Pour une meilleure sécurité, il devrait aussi être inférieur ou égal à c.

taille de bloc de chiffrement, c
C'est la taille de bloc du chiffrement de bloc sous-jacent à la fonction de chiffrement et déchiffrement indiquée ci-dessus, utilisée pour la déduction de clé et pour la taille du confondeur de message et de la valeur initiale. (Si un chiffrement de bloc n'est pas utilisé, un paramètre comparable devrait être déterminé.) Il doit être au moins de 5 octets.

Ce n'est en fait pas un paramètre indépendant ; c'est plutôt une propriété des fonctions E et D. Il est mentionné ici pour préciser la distinction entre lui et la taille de bloc du message, m.

Bien qu'il y ait encore un certain nombre de propriétés à spécifier, elles sont moins nombreuses et plus simples que dans le profil complet.

5.3 Profil de systèmes de chiffrement fondés sur le profil simplifié

La fonction de déduction de clé ci-dessus est utilisée pour produire trois clés intermédiaires. Une est utilisée pour calculer les sommes de contrôle des données non chiffrées. Les deux autres sont utilisées pour chiffrer et déchiffrer le texte source à envoyer non chiffré.

Le résultat du texte chiffré est l'enchaînement du résultat de la fonction de chiffrement de base E et d'un HMAC (éventuellement tronqué) utilisant la fonction de hachage spécifiée H, toutes deux appliquées au texte source avec un préfixe de confondeur aléatoire et un bourrage suffisant pour l'amener à un multiple de la taille de bloc de message. Lorsque le HMAC est calculé, la clé est utilisée sous la forme d'une clé de protocole.

Le déchiffrement est effectué en retirant le HMAC (partiel), en déchiffrant le reste, et en vérifiant le HMAC. L'état de chiffrement est une valeur initiale, initialisée à zéro.

La notation de sous chaîne "[1..h]" dans le tableau qui suit devrait être lue en utilisant un indice de base 1 ; les chaînes de tête sont utilisées.

Systèmes de chiffrement du profil simplifié

format de clé de protocole	Tel quel.
structure de clé spécifique	Trois clés au format du protocole : { Kc, Ke, Ki }.
longueur de germe de génération de clé	Tel quel.
mécanisme exigé de somme de contrôle	Comme défini au paragraphe 5.4.
état de chiffrement	Valeur initiale (usuellement de longueur c)
état de chiffrement initial	Tous les bits à zéro
fonction de chiffrement	conf = chaîne aléatoire de longueur c bourrage = plus courte chaîne qui amène le confondeur et le texte source à une longueur multiple de m. (C1, newIV) = E(Ke, conf | plaintext | bourrage, oldstate.ivec) H1 = HMAC(Ki, conf | plaintext | bourrage) texte chiffré = C1 | H1[1..h] newstate.ivec = newIV
fonction de déchiffrement	(C1,H1) = texte chiffré (P1, newIV) = D(Ke, C1, oldstate.ivec) si (H1 != HMAC(Ki, P1)[1..h]) rapporter l'erreur newstate.ivec = newIV
paramètres par défaut string-to-key	tel quel.
fonction pseudo aléatoire	tmp1 = H(chaîne d'octets) tmp2 = tronque tmp1 à un multiple de m PRF = E(DK(clé de protocole, prfconstant), tmp2, état-de-chiffrement-initial)

La "prfconstant" utilisée dans le fonctionnement du PRF est la chaîne de trois octets "prf".

Systèmes de chiffrement du profil simplifié	
fonctions de génération de clé :	
fonction chaîne à clé	Telle quelle.
fonction aléatoire à clé	Telle quelle.
fonction dérivation de clé	La "constante bien connue" utilisée pour la fonction DK est le numéro d'usage de clé, exprimé par quatre octets en ordre gros boutien, suivis par un octet indiqué ci-dessous : Kc = DK(base-key, usage | 0x99); Ke = DK(base-key, usage | 0xAA); Ki = DK(base-key, usage | 0x55);

5.4 Profils de somme de contrôle fondés sur le profil simplifié

Lorsque un système de chiffrement est défini avec le profil simplifié donné au paragraphe 5.2, un algorithme de somme de contrôle peut être défini comme suit pour lui :

Mécanisme de somme de contrôle provenant du profil simplifié	
cryptosystème associé	comme défini ci-dessus.
get_mic	HMAC(Kc, message)[1..h]
verify_mic	get_mic et comparer

La fonction HMAC et la clé Kc sont décrites au paragraphe 5.3.

6. Profils pour les algorithmes Kerberos de chiffrement et de somme de contrôle

Ces profils décrivent les systèmes de chiffrement et de somme de contrôle définis pour Kerberos. Le lecteur astucieux remarquera que certains d'entre eux ne satisfont pas aux exigences mentionnées dans les sections précédentes. Ces systèmes sont définis pour la rétro compatibilité ; les nouvelles mises en œuvre devraient (chaque fois que possible) essayer d'utiliser des systèmes de chiffrement qui satisfont toutes les exigences du profil.

La liste complète des allocations actuelles de numéro de type de chiffrement et de somme de contrôle, y compris les valeurs actuellement réservées mais non définies dans le présent document, est donnée à la Section 8.

6.1 Sommes de contrôle sans clé

Ces types de somme de contrôle n'utilisent pas de clé de chiffrement et peuvent donc être utilisés en combinaison avec tout type de chiffrement, mais ils ne doivent être utilisés qu'avec des précautions, dans des circonstances limitées où l'absence de clé n'ouvre pas une fenêtre à des attaques, de préférence au sein d'un message chiffré (Note [6]). Les algorithmes à somme de contrôle à clés sont recommandés.

6.1.1 Somme de contrôle en RSA MD5

La somme de contrôle RSA-MD5 calcule une somme de contrôle en utilisant l'algorithme RSA MD5 [RFC1321]. L'algorithme prend en entrée un message d'entrée de longueur arbitraire et produit en résultat une somme de contrôle de 128 bits (seize octets).

	rsa-md5
cryptosystème associé	tout
get_mic	rsa-md5(msg)
verify_mic	get_mic et comparer

L'algorithme de somme de contrôle rsa-md5 a reçu le numéro de type de somme de contrôle de sept (7).

6.1.2 Somme de contrôle en RSA MD4

La somme de contrôle RSA-MD4 calcule une somme de contrôle en utilisant l'algorithme RSA MD4 [RFC1320]. L'algorithme prend en entrée un message d'entrée de longueur arbitraire et produit comme résultat une somme de contrôle de 128 bits (seize octets).

	rsa-md4
cryptosystème associé	tout
get_mic	md4(msg)
verify_mic	get_mic et comparer

L'algorithme de somme de contrôle rsa-md4 a reçu le numéro de type de somme de contrôle de deux (2).

6.1.3 Somme de contrôle en CRC-32

Cette somme de contrôle CRC-32 calcule une somme de contrôle fondée sur un contrôle de redondance cyclique comme décrit dans la norme ISO 3309 [CRC] mais modifiée comme décrit ci-dessous. La somme de contrôle résultante est longue de quatre (4) octets. Le CRC-32 n'a ni clé ni protection contre la collision ; donc, l'utilisation de cette somme de contrôle n'est pas recommandée. Un attaquant qui utilise une attaque probabiliste de texte source choisi comme décrit dans [SG92] peut être capable de générer un message de remplacement qui satisfasse la somme de contrôle.

La somme de contrôle utilisée dans le mode de chiffrement des-cbc-crc est identique à la séquence de vérification de trame (FCS, *Frame Check Sequence*) de 32 bits décrite dans ISO 3309 avec deux exceptions : la somme avec le polynôme tout à un fois x**k est omise, et le reste final n'est pas complété par des uns. ISO 3309 décrit la FCS en termes de bits, tandis que le présent document décrit le protocole Kerberos en termes d'octets. Pour préciser la définition de ISO 3309 pour les besoins du calcul du CRC-32 dans le mode de chiffrement des-cbc-crc, l'ordre des bits dans chaque octet devra être supposé avec le bit de moindre poids en premier. Étant donné cet ordre supposé des bits au sein d'un octet, la transposition des bits en coefficients polynomiaux devra être identique à celle spécifiée dans ISO 3309.

Les valeurs d'essai pour cette fonction modifiée de CRC sont incluses dans l'Appendice A.5.

	crc32
cryptosystème associé	tout
get_mic	crc32(msg)
verify_mic	get_mic et comparer

L'algorithme de somme de contrôle crc32 a reçu le numéro de type de somme de contrôle de un (1).

6.2 Types de chiffrement et de somme de contrôle fondés sur DES

Ces systèmes de chiffrement chiffrent l'information avec la norme de chiffrement de données (DES, *Data Encryption Standard*) [DES77] en utilisant le mode de chaînage de bloc de chiffrement (CBC, *cipher block chaining*) [DESM80]. Une somme de contrôle est calculée comme décrit ci-dessous et placée dans le champ Somme de contrôle. Les blocs DES font huit octets. Par suite, les données à chiffrer (l'enchaînement du confondeur, de la somme de contrôle, et du message) doivent être bourrées jusqu'à une limite de huit octets avant le chiffrement. Les valeurs des octets de bourrage ne sont pas spécifiées.

Le texte source et le texte chiffré en DES sont codés comme des blocs de huit octets, qui sont enchaînés pour constituer les entrées de 64 bits des algorithmes DES. Le premier octet fournit les huit bits de plus fort poids (MSB, *most significant bit*) (avec le MSB de l'octet utilisé comme MSB du bloc d'entrée DES, etc.) le second octet des huit bits suivants, et ainsi de suite. Le huitième octet fournit les 8 bits de moindre poids (LSB, *least significant bit*).

Le chiffrement sous DES en utilisant le chaînage de bloc de chiffrement exige une entrée supplémentaire sous la forme d'une valeur d'initialisation ; cette valeur est spécifiée ci-dessous pour chaque système de chiffrement.

Les spécifications de DES [DESI81] identifient quatre clés 'faibles' et douze clés 'semi faibles' ; ces clés NE DEVRONT PAS être utilisées pour chiffrer les messages à utiliser dans Kerberos. Les "clés variantes" générées pour les types de somme de contrôle RSA-MD5-DES, RSA-MD4-DES, et DES-MAC par une opération 'OU exclusif' d'une clé DES avec une constante ne sont pas essayées pour cette propriété.

Une clé DES est faite de huit octets de données. Cela consiste en 56 bits de données de clé réelles, et 8 bits de parité, un par octet. La clé est codée comme une série de 8 octets écrits avec le MSB en premier. Les bits au sein de la clé sont aussi codés dans le même ordre. Par exemple, si la clé de chiffrement est (B1,B2,...,B7,P1,B8,...,B14,P2,B15,...,B49,P7, B50, ... , B56,P8), où B1,B2,...,B56 sont les bits de clé dans l'ordre MSB, et P1,P2,...,P8 sont les bits de parité, le premier octet de la clé serait B1,B2,...,B7,P1 (avec B1 comme bit de poids fort). Voir les références dans l'introduction à [DESM80].

Format de chiffrement des données
Le format pour les données à chiffrer comporte un confondeur de un bloc, une somme de contrôle, le texte source codé, et tout le bourrage nécessaire, comme décrit dans le diagramme qui suit. Le champ msg-seq contient la partie du message de protocole à chiffrer.

confondeur somme de contrôle

On génère un confondeur aléatoire de un bloc, en le plaçant dans 'confondeur' ; ont complète de zéros le champ 'somme de contrôle' (de la longueur appropriée pour tenir exactement la somme de contrôle à calculer) ; on ajoute le bourrage nécessaire ; on calcule la somme de contrôle appropriée sur la totalité de la séquence, et on place le résultat dans 'somme de contrôle' ; et on chiffre ensuite en utilisant le type de chiffrement spécifié et la clé appropriée.

Transformation de chaîne ou données aléatoires en clé
Pour générer une clé DES à partir de deux chaînes de texte UTF-8 (le mot de passe et le sel) les deux chaînes sont enchaînées, le mot de passe d'abord, et le résultat est ensuite bourré d'octets à zéro jusqu'à un multiple de huit octets.

Le bit de poids fort de chaque octet (toujours un zéro si le mot de passe est en ASCII, comme on l'a supposé lors de la rédaction de la spécification d'origine) est éliminé, et les sept bits restants de chaque octet forment une chaîne binaire. Ceci est alors replié en éventail et traité avec l'opérateur OU exclusif sur lui-même pour produire une chaîne de 56 bits. Une clé de huit octets est formée à partir de cette chaîne, chaque octet utilisant sept bits de la chaîne binaire, en laissant le bit de moindre poids non affecté. La clé est alors "corrigée" en corrigeant la parité de la clé, et si la clé correspond à une clé 'faible' ou 'semi faible' comme décrit dans la spécification DES, elle est traitée avec l'opérateur OU exclusif avec la constante 0x00000000000000F0. Cette clé est alors utilisée pour générer une somme de contrôle CBC DES sur la chaîne initiale avec le sel rajouté. Le résultat de la somme de contrôle CBC est alors "corrigé" comme décrit ci-dessus pour former le résultat, qui est retourné comme la clé.

Pour les besoins de la fonction de chaîne à clé, la somme de contrôle CBC DES est calculée par CBC qui chiffre une chaîne en utilisant la clé comme valeur d'initialisation (IV, *initialization vector*) et le bloc final de huit octets comme somme de contrôle.

Voici le pseudocode :

```
removeMSBits(8byteblock) {
/* Traite un bloc de 64 bits comme 8 octets et retire le MSB de chaque octet (en mode gros boutien) et enchaîne le
résultat. /* Par exemple, la chaîne d'octets d'entrée :
/*  01110000 01100001 11110011  01110011 11110111 01101111 11110010 01100100
/* résulte en la chaîne binaire de sortie :
/*  1110000 1100001 1110011  1110011 1110111 1101111 1110010 1100100  */
        }

reverse(56bitblock) {
/* Traite un bloc de 56 bits comme une chaîne binaire et l'inverse.
/* Par exemple, la chaîne d'entrée :
/*  1000001 1010100 1001000  1000101 1001110 1000001 0101110 1001101
/*  résulte en la chaîne de sortie :
/*  1011001 0111010 1000001  0111001 1010001 0001001 0010101 1000001  */
        }

add_parity_bits(56bitblock) {
/* Copie un bloc de 56 bits en bloc de 64 bits, décale le contenu à gauche dans chaque octet, et ajoute le bit de parité DES.
/* Par exemple, la chaîne d'entrée :
/* 1100000 0001111 0011100  0110100 1000101 1100100 0110110 0010111
/* résulte en la chaîne de sortie :
/* 11000001 00011111 00111000  01101000 10001010 11001000 01101101 00101111  */
        }

key_correction(key) {
        fixparity(key);
        if (is_weak_key(key))
        key = key XOR 0xF0;
        return(key);
        }
```

```
mit_des_string_to_key(string,salt) {
        odd = 1;
        s = string | salt;
        tempstring = 0; /* 56-bit string */
        pad(s); /* avec des zéro jusqu'à la limite de 8 octets */
        for (8byteblock in s) {
        56bitstring = removeMSBits(8byteblock);
        if (odd == 0) reverse(56bitstring);
        odd = ! odd;
        tempstring = tempstring XOR 56bitstring;
        }
        tempkey = key_correction(add_parity_bits(tempstring));
        key = key_correction(DES-CBC-check(s,tempkey));
        return(key);
        }

des_string_to_key(string,salt,params) {
        if (length(params) == 0)
        type = 0;
        else if (length(params) == 1)
        type = params[0];
        else
        error("invalid params");
        if (type == 0)
        mit_des_string_to_key(string,salt);
        else
        error("invalid params");
    }
```

Une extension courante est de prendre en charge l'algorithme "AFS string-to-key" qui n'est pas défini ici, si la valeur de type ci-dessus est un (1).

Pour générer une clé à partir d'une chaîne binaire aléatoire, on commence par une chaîne de 56 bits et, comme avec l'opération de chaîne à clé ci-dessus, on insère les bits de parité. Si le résultat est une clé faible ou semi faible, on la modifie avec l'opération OU exclusif avec la constante 0x00000000000000F0 :

```
des_random_to_key(bitstring) {
        return key_correction(add_parity_bits(bitstring));
        }
```

6.2.1 DES avec MD5

Le mode de chiffrement des-cbc-md5 chiffre les informations sous DES en mode CBC avec une valeur initiale toute à zéro et avec une somme de contrôle MD5 (décrite dans la [RFC1321]) calculée et placée dans le champ Somme de contrôle.

Les paramètres du système de chiffrement pour des-cbc-md5 sont les suivants :

	des-cbc-md5
format de clé du protocole	8 octets, parité dans le bit de moindre poids de chacun
structure de clé spécifique	copie de la clé d'origine
mécanisme exigé de somme de contrôle	rsa-md5-des
longueur de germe de génération de clé	8 octets
état de chiffrement	8 octets (valeur initiale CBC)
état de chiffrement initial	tout à zéro
fonction de chiffrement	des-cbc(confondeur | somme de contrôle | msg | bourrage, ivec=oldstate) où somme de contrôle = md5(confondeur | 0000... | msg | bourrage) newstate = dernier bloc du résultat de des-cbc
fonction de déchiffrement	déchiffre le texte chiffré et vérifie la somme de contrôle newstate = dernier bloc de texte chiffré

	des-cbc-md5
paramètres de chaîne à clé par défaut	chaîne vide
fonction pseudo-aléatoire	des-cbc(md5(input-string), ivec=0)
fonctions de génération de clé:	
string-to-key	des_string_to_key
random-to-key	des_random_to_key
key-derivation	identité

La valeur de etype trois (3) est allouée au type de chiffrement des-cbc-md5.

6.2.2 DES avec MD4

Le mode de chiffrement des-cbc-md4 chiffre aussi les informations sous DES en mode CBC, avec une valeur initiale toute à zéro. Une somme de contrôle MD4 (décrite dans la [RFC1320]) est calculée et placée dans le champ Somme de contrôle.

	des-cbc-md4
format de clé du protocole	8 octets, parité dans le bit de moindre poids de chacun.
structure de clé spécifique	copie de la clé d'origine
mécanisme exigé de somme de contrôle	rsa-md4-des
longueur de germe de génération de clé	8 octets
état de chiffrement	8 octets (valeur initiale CBC)
état de chiffrement initial	tout à zéro
fonction de chiffrement	des-cbc(confondeur | somme de contrôle | msg | bourrage, ivec=oldstate) où somme de contrôle = md4(confondeur | 0000... | msg | bourrage) newstate = dernier bloc du résultat de des-cbc

	des-cbc-md4
fonction de déchiffrement	déchiffre le texte chiffré et vérifie la somme de contrôle newstate = dernier bloc du texte chiffré
paramètres de chaîne à clé par défaut	chaîne vide
fonction pseudo aléatoire	des-cbc(md5(input-string), ivec=0)
fonctions de génération de clé :	
string-to-key	des_string_to_key
random-to-key	copie l'entrée, puis corrige les bits de parité
key-derivation	identité

Noter que des-cbc-md4 utilise md5, et non md4, dans la définition de PRF.

La valeur de etype de deux (2) est allouée à l'algorithme de chiffrement des-cbc-md4.

6.2.3 DES avec CRC

Le type de chiffrement des-cbc-crc utilise DES en mode CBC avec la clé utilisée comme valeur d'initialisation, avec une somme de contrôle de quatre octets fondée sur le CRC calculée comme décrit au paragraphe 6.1.3. Noter que ce n'est pas une somme de contrôle standard CRC-32, mais qu'elle est légèrement modifiée.

	des-cbc-crc
format de clé du protocole	8 octets, parité dans le bit de moindre poids de chacun.
structure de clé spécifique	copie de la clé d'origine
mécanisme exigé de somme de contrôle	rsa-md5-des
longueur de germe de génération de clé	8 octets
état de chiffrement	8 octets (valeur initiale CBC)

	des-cbc-crc
état de chiffrement initial	copie de la clé d'origine
fonction de chiffrement	des-cbc(confondeur | somme de contrôle | msg | bourrage, ivec=oldstate) où, somme de contrôle = crc(confondeur | 00000000 | msg | bourrage)

	newstate = dernier bloc du résultat de des-cbc
fonction de déchiffrement	déchiffre le texte chiffré et vérifie la somme de contrôle newstate = dernier bloc du texte chiffré
paramètres de chaîne à clé par défaut	chaîne vide
fonction pseudo aléatoire	des-cbc(md5(chaîne d'entrée), ivec=0)
fonctions de génération de clé :	
string-to-key	des_string_to_key
random-to-key	copie l'entrée,puis corrige les bits de parité
key-derivation	identité

La valeur de etype de un (1) est allouée à l'algorithme de chiffrement des-cbc-crc.

6.2.4 Somme de contrôle cryptographique MD5 RSA utilisant DES

La somme de contrôle RSA-MD5-DES calcule une somme de contrôle chiffrée à l'épreuve des collisions en ajoutant un confondeur de huit octets avant le texte, en appliquant l'algorithme de somme de contrôle RSA MD5, et en chiffrant le confondeur et la somme de contrôle en utilisant DES en mode de chaînage de bloc de chiffrement (CBC, *cipher-block-chaining*) avec une variante de la clé, où la variante est calculée en appliquant l'opération OU exclusif à la clé avec la constante hexadécimale 0xF0F0F0F0F0F0F0F0. La valeur d'initialisation devrait être zéro. La somme de contrôle résultante fait 24 octets.

	rsa-md5-des
système de chiffrement associé	des-cbc-md5, des-cbc-md4, des-cbc-crc
get_mic	des-cbc(key XOR 0xF0F0F0F0F0F0F0F0, conf | rsa-md5(conf | msg))
verify_mic	déchiffre et vérifie la somme de contrôle rsa-md5

L'algorithme de somme de contrôle rsa-md5-des a reçu le numéro de type de somme de contrôle de huit (8).

6.2.5 Somme de contrôle cryptographique MD4 RSA utilisant DES

La somme de contrôle RSA-MD4-DES calcule une somme de contrôle chiffrée à l'épreuve des collisions en ajoutant un confondeur de huit octets avant le texte, en appliquant l'algorithme de somme de contrôle RSA MD4 [RFC1320], et en chiffrant le confondeur et la somme de contrôle en utilisant DES en mode de chaînage de bloc de chiffrement (CBC) avec une variante de la clé, où la variante est calculée en appliquant l'opération OU exclusif à la clé avec la constante hexadécimale 0xF0F0F0F0F0F0F0F0 (Note [7]). La valeur d'initialisation devrait être zéro. La somme de contrôle résultante fait 24 octets.

	rsa-md4-des
système de chiffrement associé	des-cbc-md5, des-cbc-md4, des-cbc-crc
get_mic	des-cbc(key XOR 0xF0F0F0F0F0F0F0F0, conf | rsa-md4(conf | msg), ivec=0)
verify_mic	déchiffre et vérifie la somme de contrôle rsa-md4

L'algorithme de somme de contrôle rsa-md4-des a reçu le numéro de type de somme de contrôle de trois (3).

6.2.6 Somme de contrôle cryptographique RSA MD4 avec une alternative à DES

La somme de contrôle RSA-MD4-DES-K calcule une somme de contrôle chiffrée à l'épreuve des collisions en appliquant l'algorithme de somme de contrôle RSA MD4 et en chiffrant le résultat avec DES en mode de chaînage de bloc de chiffrement (CBC) avec une clé DES à la fois comme clé et comme valeur d'initialisation. La somme de contrôle résultante fait 16 octets. Cette somme de contrôle est à l'épreuve de l'altération et est estimée à l'épreuve des collisions. Noter que ce type de somme de contrôle est la vieille méthode pour coder la somme de contrôle RSA-MD4-DES ; elle n'est plus recommandée.

	rsa-md4-des-k
système de chiffrement associé	des-cbc-md5, des-cbc-md4, des-cbc-crc
get_mic	des-cbc(clé, md4(msg), ivec=clé)
verify_mic	déchiffre, calcule la somme de contrôle et compare

L'algorithme de somme de contrôle rsa-md4-des-k a reçu le numéro de type de somme de contrôle de six (6).

6.2.7 Somme de contrôle DES CBC

La somme de contrôle DES-MAC est calculée en ajoutant un confondeur de huit octets devant le texte source, en bourrant avec des octets à zéro si nécessaire pour amener la longueur à un multiple de huit octets, en effectuant un chiffrement DES en mode CBC sur le résultat avec la clé et la valeur d'initialisation de zéro, en prenant le dernier bloc du texte chiffré, en ajoutant devant le même confondeur, et en chiffrant la paire au moyen de DES en mode de chaînage de bloc de chiffrement (CBC) avec une variante de la clé, où la variante est calculée par l'opération OU exclusif de la clé avec la constante hexadécimale 0xF0F0F0F0F0F0F0F0. La valeur d'initialisation devrait être zéro. La somme de contrôle résultante fait 128 bits (seize octets) dont 64 bits sont redondants. Cette somme de contrôle est à l'épreuve de l'altération et de la collision.

	des-mac
système de chiffrement associé	des-cbc-md5, des-cbc-md4, des-cbc-crc
get_mic	des-cbc(key XOR 0xF0F0F0F0F0F0F0F0, conf | des-mac(key, conf | msg | bourrage, ivec=0), ivec=0)
verify_mic	déchiffre, calcule le MAC DES en utilisant un confondeur, compare

L'algorithme de somme de contrôle des-mac a reçu un numéro de type de somme de contrôle de quatre (4).

6.2.8 Autre somme de contrôle DES CBC

La somme de contrôle DES-MAC-K est calculée en effectuant un chiffrement DES en mode CBC du texte source, avec un bourrage d'octets à zéro si nécessaire pour amener la longueur à un multiple de huit octets, et en utilisant le dernier bloc du texte chiffré comme valeur de somme de contrôle. Elle est chiffrée avec une clé de chiffrement qui est aussi utilisée comme valeur d'initialisation. La somme de contrôle résultante fait 64 bits (huit octets). Cette somme de contrôle est à l'épreuve de l'altération et des collisions. Noter que ce type de somme de contrôle est la vieille méthode de codage de la somme de contrôle DESMAC ; elle n'est plus recommandée.

	des-mac-k
système de chiffrement associé	des-cbc-md5, des-cbc-md4, des-cbc-crc
get_mic	des-mac(key, msg | bourrage, ivec=key)
verify_mic	calcule le MAC et compare

L'algorithme de somme de contrôle des-mac-k a reçu un numéro de type de somme de contrôle de cinq (5).

6.3 Types de chiffrement et de somme de contrôle fondés sur triple-DES

Cette paire de chiffrement et de type de somme de contrôle se fonde sur le système de chiffrement Triple DES en mode CBC externe *(Outer-CBC)* et sur l'algorithme d'authentification de message HMAC-SHA1.

Une clé Triple DES est l'enchaînement de trois clés DES comme décrit ci-dessus pour des-cbc-md5. Une clé Triple DES est générée à partir de données aléatoires en créant trois clés DES) partir de séquences séparées de données aléatoires.

Les données chiffrées en utilisant ce type doivent être générées comme décrit au paragraphe 5.3. Si la longueur des données d'entrée n'est pas un multiple de la taille de bloc, des octets de valeur zéro doivent être utilisés pour bourrer le texte source jusqu'à la prochaine limite de huit octets. Le confondeur doit être huit octets aléatoires (un bloc).

Le profil simplifié pour Triple DES, avec la déduction de clé définie à la section 5, est le suivant :

	des3-cbc-hmac-sha1-kd, hmac-sha1-des3-kd
format de clé du protocole	24 octets, parité dans le bit de moindre poids de chacun
longueur de germe de génération de clé	21 octets

	des3-cbc-hmac-sha1-kd, hmac-sha1-des3-kd
fonction de hachage	SHA-1
taille du résultat HMAC	160 bits
taille du bloc de message	8 octets
paramètres par défaut de chaîne à clé	chaîne vide
fonctions de chiffrement et déchiffrement	chiffre et déchiffre triple-DES, en mode CBC externe (taille de bloc de chiffrement 8 octets)
fonctions de génération de clé :	
random-to-key	DES3random-to-key (voir ci-dessous)
string-to-key	DES3string-to-key (voir ci-dessous)

Le type de chiffrement des3-cbc-hmac-sha1-kd a reçu la valeur de seize (16). L'algorithme de somme de contrôle hmac-sha1-des3-kd a reçu le numéro de type de somme de contrôle de douze (12).

6.3.1 Production de clé triple DES (aléatoire à clé , chaîne à clé)

Les 168 bits de données de clé aléatoires sont convertis en une valeur de clé de protocole comme suit. D'abord, les 168 bits sont divisés en trois groupes de 56 bits, qui sont développés individuellement en 64 bits comme suit :

DES3 aléatoire à clé :
1 2 3 4 5 6 7 p
9 10 11 12 13 14 15 p
17 18 19 20 21 22 23 p
25 26 27 28 29 30 31 p
33 34 35 36 37 38 39 p
41 42 43 44 45 46 47 p
49 50 51 52 53 54 55 p
56 48 40 32 24 16 8 p

Les bits "p" sont les bits de parité calculés sur les bits de données. Le résultat des trois expansions, chacune corrigée pour éviter les clés "faibles" et "semi faibles" comme au paragraphe 6.2, est enchaîné pour former la valeur de clé de protocole.

La fonction chaîne à clé est utilisée pour transformer les mots de passe UTF-8 en clés DES3. La fonction DES3 chaîne à clé s'appuie sur l'algorithme "N-fold" et la fonction DK, décrits à la Section 5.

l'algorithme n-fold est appliqué à la chaîne de mot de passe enchaînée avec une valeur de sel. Pour le triple DES à trois clés, l'opération va impliquer un 168-fold de la chaîne de mot de passe d'entrée, pour générer une clé intermédiaire, à partir de laquelle la clé à long terme de l'utilisateur sera déduite avec la fonction DK. La fonction DES3 chaîne à clé est montrée ci-dessous en pseudocode :

```
DES3string-to-key(passwordString, salt, params)
        if (params != emptyString)
        error("invalid params");
        s = passwordString + salt
        tmpKey = random-to-key(168-fold(s))
        key = DK (tmpKey, KerberosConstant)
```

La vérification de clés faibles est effectuée dans les opérations aléatoire à clé et DK. La valeur KerberosConstant est la chaîne d'octets {0x6b 0x65 0x72 0x62 0x65 0x72 0x6f 0x73}. Ces valeurs correspondent au codage ASCII pour la chaîne "kerberos".

7. Utilisation du chiffrement Kerberos en dehors de la présente spécification

Plusieurs protocoles d'application et systèmes de pré authentification fondés sur Kerberos ont été conçus et déployés qui effectuent le chiffrement et la vérification d'intégrité de message de différentes façons. Bien que dans certains cas, il puisse y avoir de bonnes raisons pour spécifier ces protocole en termes d'algorithmes spécifiques de chiffrement ou de somme de contrôle, on prévoit que dans de nombreux cas, cela ne sera pas vrai, et que des approches plus génériques indépendantes des algorithmes particuliers seront souhaitables. Plutôt que de voir chaque concepteur de protocole réinventer des schémas pour protéger les données, utiliser plusieurs clés, etc., on a tenté de présenter dans cette section un cadre général qui devrait être suffisant non seulement pour le protocole Kerberos lui-même mais aussi pour de nombreux systèmes de pré authentification et protocoles d'application, tout en essayant d'éviter certaines des hypothèses qui pourraient se faire jour dans de telles conceptions de protocole.

Certaines hypothèses problématiques qu'on a vu (et parfois faites) incluent ce qui suit : une chaîne binaire aléatoire est toujours valide comme clé (mais ce n'est pas vrai pour les clés DES avec parité) ; le mode de chaînage de chiffrement de bloc de base ne fournit pas de vérification d'intégrité, ou peut facilement être séparé d'une telle vérification (ce n'est pas vrai pour de nombreux modes en développement qui font les deux simultanément) ; une somme de contrôle pour un message résulte toujours en la même valeur (ce n'est pas vrai si un confondeur est incorporé) ; une valeur initiale est utilisée (peut n'être pas vrai si un chiffrement de bloc en mode CBC n'est pas utilisé). Bien que de telles hypothèses puissent tenir pour un certain ensemble d'algorithmes de chiffrement et de somme de contrôle, elles peuvent n'être pas vraies des prochains algorithmes à définir, laissant le protocole d'application incapable d'utiliser ces algorithmes sans mise à jour de

leur spécification.

Le protocole Kerberos utilise seulement les attributs et opérations décrits aux sections 3 et 4. Les systèmes de pré authentification et protocoles d'application qui utilisent Kerberos sont encouragés à les utiliser aussi. Les paramètres spécifiques de clé et de chaîne à clé devraient généralement être traités comme opaques. Bien que les paramètres de chaîne à clé soient manipulés comme une chaîne d'octets, la représentation de la structure de clé spécifique est définie par la mise en œuvre ; elle peut même être un seul objet.

On ne recommande pas de faire comme cela, mais certains protocoles d'application vont sans doute continuer d'utiliser directement les données de clé, même si c'est seulement dans certaines spécifications de protocole actuellement existantes. Une mise en œuvre destinée à prendre en charge des applications Kerberos générales peut donc avoir besoin de rendre les données de clé disponibles, ainsi que les attributs et opérations décrites aux sections 3 et 4 (Note [8]).

8. Numéros alloués

Les numéros de type de chiffrement suivants sont déjà alloués ou réservés pour l'usage de Kerberos et des protocoles qui s'y rapportent.

Type de chiffrement	**paragraphe ou commentaire**
des-cbc-crc	6.2.3
des-cbc-md4	6.2.2
des-cbc-md5	6.2.1
[réservé]	
des3-cbc-md5	
[réservé]	
des3-cbc-sha1	
dsaWithSHA1-CmsOID	(pkinit)
md5WithRSAEncryption-CmsOID	(pkinit)
sha1WithRSAEncryption-CmsOID	(pkinit)
rc2CBC-EnvOID	(pkinit)
rsaEncryption-EnvOID	(pkinit d'après PKCS#1 v1.5)
rsaES-OAEP-ENV-OID	(pkinit d'après PKCS#1 v2.0)
des-ede3-cbc-Env-OID	(pkinit)
des3-cbc-sha1-kd	6.3
aes128-cts-hmac-sha1-96	[RFC3962]
aes256-cts-hmac-sha1-96	[RFC3962]
rc4-hmac	(Microsoft)
rc4-hmac-exp	(Microsoft)
subkey-keymaterial	(opaque; PacketCable)

(L'allocation "des3-cbc-sha1" est une version déconseillée qui n'utilise pas de déduction de clé. Elle ne devrait pas être confondue avec des3-cbc-sha1-kd.) Plusieurs numéros ont été réservés pour l'usage de systèmes de chiffrement non définis ici. Les numéros de type de chiffrement ont malheureusement été surchargés à l'occasion dans les protocoles en rapport avec Kerberos, de sorte que certains des numéros réservés ne correspondent pas à des systèmes de chiffrement cohérents avec le profil présenté ici.

Les numéros de type de somme de contrôle ont été alloués ou réservés. Comme avec les numéros de type de chiffrement, il s'est produit une certaine surcharge des numéros de somme de contrôle.

Type de s. de contrôle	**Valeur du type de s.**	**Taille de s. de contrôle**	**Paragraphe ou référence**
CRC32	1	4	6.1.3
rsa-md4	2	16	6.1.2
rsa-md4-des	3	24	6.2.5
des-mac	4	16	6.2.7
des-mac-k	5	8	6.2.8
rsa-md4-des-k	6	16	6.2.6
rsa-md5	7	16	6.1.1
rsa-md5-des	8	24	6.2.4
rsa-md5-des3	9	24	??
sha1 (unkeyed)	10	20	??

hmac-sha1-des3-kd	12	20	6.3
hmac-sha1-des3	13	20	??
sha1 (unkeyed)	14	20	??
hmac-sha1-96-aes128	15	20	[RFC3962]
hmac-sha1-96-aes256	16	20	[RFC3962]
[réservé]	0x8003	?	[RFC1964]

Les numéros de chiffrement et de type de somme de contrôle sont des valeurs signées de 32 bits. Le zéro est invalide, et le numéros négatifs sont réservés à l'utilisation locale. Toutes les valeurs normalisées doivent être positives.

9. Notes de mise en œuvre

L'interface décrite ici constitue les informations minimales qui doivent être définies pour rendre un système de chiffrement utile d'une façon interopérable dans Kerberos. L'utilisation de la notation fonctionnelle à certains endroits n'est pas une tentative pour définir une API pour la fonction cryptographique au sein de Kerberos. Les mises en œuvre réelles qui fournissent de vraies API vont probablement rendre disponibles des informations supplémentaires, qui pourraient être déduites d'une spécification écrite sur le cadre donné ici. Par exemple, un concepteur d'application peut souhaiter déterminer le plus grand nombre d'octets qui peuvent être chiffrés sans déborder une mémoire tampon d'une certaine taille ou, à l'inverse, le nombre maximum d'octets qui pourrait être obtenu en déchiffrant un message de texte chiffré d'une certaine taille. (En fait, une mise en œuvre du mécanisme GSS-API Kerberos [RFC1964] va en exiger un certain nombre.)

La présence d'un mécanisme dans ce document ne devrait pas être prise comme l'indication qu'il doit être mis en œuvre pour être conforme à une spécification ; les mécanismes exigés seront spécifiés ailleurs. Bien sûr, certains des mécanismes décrits ici pour la rétro compatibilité sont maintenant considérés comme plutôt faibles pour protéger des données critiques.

10. Considérations pour la sécurité

Les années récentes ont vu de telles améliorations des capacités d'attaque à grande échelle contre le DES qu'il n'est plus considéré comme un mécanisme de chiffrement fort. Triple-DES est généralement préféré, en dépit de ses moindres performances. Voir dans la [RFC2405] un résumé des attaques potentielles et dans [EFF-DES] une discussion détaillée de la mise en œuvre d'attaques particulières. Cependant, la plupart des mises en œuvre de Kerberos ont toujours le DES comme principal type de chiffrement interopérable.

DES a quatre clés 'faibles' et douze clés 'semi faibles', et l'utilisation ici du DES seul les évite. Cependant, DES a aussi 48 clés 'possiblement faibles' [Schneier96] (noter que les tableaux dans de nombreuses éditions de la référence contiennent des erreurs) qui ne sont pas évitées.

Les clés faibles DES ont la propriété que E1(E1(P)) = P (où E1 note le chiffrement d'un seul bloc avec la clé 1). Les clés DES semi faibles, ou clés "duelles", sont des paires de clés avec la propriété que E1(P) = D2(P), et donc E2(E1(P)) = P. À cause de l'utilisation du mode CBC et du confondeur aléatoire en tête, cependant, ces propriétés ont peu de chances de poser un problème de sécurité.

Beaucoup des choix concernant le moment où effectuer les corrections de clés faibles se rapportent plus à la compatibilité avec les mises en œuvre existantes qu'à aucune analyse de risque.

Bien que des vérifications soient aussi faites sur les clés qui composent DES dans une clé triple-DES, la nature des clés faibles rend très peu probable qu'elles affaiblissent le chiffrement triple-DES. Il est seulement légèrement plus probable d'avoir la clé du milieu des trois sous clés qui correspond à une des deux autres, qui convertit effectivement le chiffrement en un DES seul – cas qu'on a fait aucun effort pour éviter.

La vraie somme de contrôle CRC-32 n'est pas à l'épreuve des collisions ; un attaquant pourrait utiliser une attaque probabiliste de texte source choisi pour générer un message valide même si un confondeur est utilisé [SG92]. L'utilisation de sommes de contrôle à l'épreuve des collisions est bien sûr recommandée pour des environnements où de telles attaques représentent une menace significative. Les "simplifications" (lire des erreurs) introduites lorsque le CRC-32 a été mis en œuvre pour Kerberos sont cause que les zéros en tête sont effectivement à ignorer, de sorte que des messages qui diffèrent seulement par des bits à zéro en tête auront la même somme de contrôle.

La [RFC2104] et la [RFC2404] discutent des faiblesses de l'algorithme HMAC. À la différence de la [RFC2404], la spécification du triple-DES n'utilise ici pas la troncature suggérée pour le résultat HMAC. Comme indiqué dans la

[RFC2404], SHA-1 n'a pas été développé pour être utilisé comme fonction de hachage chiffré, qui est un critère de HMAC. La [RFC2202] contient des valeurs d'essai pour HMAC-SHA-1.

La fonction mit_des_string_to_key a été originellement construite avec l'hypothèse que toutes les entrées seraient en ASCII ; elle ignore le bit de poids fort de chaque octet d'entrée. Le repli avec OUX n'est pas non plus un très bon mécanisme pour préserver l'aléa.

La fonction n-fold utilisée dans l'opération chaîne à clé pour des3-cbc-hmac-sha1-kd a été conçue pour faire que chaque bit d'entrée contribue également au résultat. Elle n'a pas été conçue pour maximiser ou distribuer également l'aléa dans le résultat, et on peut concevoir que l'aléa puisse être perdu dans des cas d'entrées partiellement structurées. Ceci ne devrait cependant poser problème que pour des mots de passe très structurés.

La [RFC1851] discute de la force relative du chiffrement triple-DES. La vitesse relativement faible du chiffrement triple-DES peut aussi poser problème pour certaines applications.

[Bellovin91] suggère que les analyses des schémas de chiffrement incluent un modèle d'un attaquant capable de soumettre des textes source connus à chiffrer avec une clé inconnue, ainsi que capable d'effectuer de nombreux types d'opérations sur des messages de protocole connus. Les expériences récentes d'attaques de texte source connu sur Kerberos version 4 mettent à mal la valeur de cette suggestion.

L'utilisation de sommes de contrôle chiffrées sans clé, comme celles utilisées dans les systèmes de chiffrement DES seul spécifié dans la [RFC1510], permet des attaques de copié collé, en particulier si on utilise pas de confondeur. De plus, les sommes de contrôle chiffrées sans clé sont vulnérables aux attaques de texte source choisi : un attaquant qui a accès à un oracle de chiffrement peut facilement chiffrer la somme de contrôle sans clé requise avec un texte source choisi [Bellovin99]. Ces faiblesses, combinées à un choix courant de conception de mise en œuvre décrit ci-dessus, permet une attaque de protocole croisée de la version 4 à la version 5.

L'utilisation d'un confondeur aléatoire est un moyen important pour empêcher un attaquant de faire un usage efficace des échanges de protocole comme un oracle de chiffrement. Dans Kerberos version 4, le chiffrement de texte source constant en texte chiffré constant constitue un oracle de chiffrement efficace pour un attaquant. L'utilisation de confondeurs aléatoires dans la [RFC1510] contre cette sorte d'attaque de texte source choisi.

Utiliser la même clé à plusieurs fins peut permettre ou augmenter la portée des attaques de texte source choisi. Certains logiciels qui mettent en œuvre les deux versions 4 et 5 du protocole Kerberos utilisent les mêmes clés pour les deux versions. Cela permet d'utiliser l'oracle de chiffrement de la version 4 pour attaquer la version 5. Les vulnérabilités aux attaques telles que cette attaque inter protocoles rendent peu sage d'utiliser une clé à plusieurs fins.

Le présent document, comme le protocole Kerberos, ne traite pas de la limitation de la quantité de données sur laquelle une clé peut être utilisée à une quantité fondée sur la robustesse de l'algorithme ou la taille de la clé. On suppose que tout algorithme et taille de clé définis seront assez forts pour prendre en charge de très grandes quantités de données, ou ils seront déconseillés dès que des attaques significatives seront connues.

Le présent document ne place aucune limite à la quantité de données qui peut être traitée dans les diverses opérations. Pour éviter des attaques de déni de service, les mises en œuvre vont probablement chercher à restreindre les tailles de message à un niveau supérieur.

11. Considérations relatives à l'IANA

Deux registres ont été créés pour les valeurs numériques : Numéros de type de chiffrement Kerberos et Numéros de type de somme de contrôle Kerberos. Ce sont des valeurs signées dans la gamme de -2 147 483 648 à 2 147 483 647. Les valeurs positives ne devraient être allouées que pour les algorithmes spécifiés en accord avec la présente spécification pour être utilisées avec Kerberos ou les protocoles qui s'y rapportent. Les valeurs négatives sont pour usage privé ; les algorithmes locaux et expérimentaux devraient utiliser ces valeurs. Zéro est réservé et ne peut pas être alloué.

Les numéros positifs de chiffrement et de type de somme de contrôle peuvent être alloués suivant l'une ou l'autre des politiques décrites dans la [RFC2434].

Les spécifications sur la voie de la normalisation peuvent recevoir des valeurs sous la politique d'action de normalisation.

Les spécifications dans des RFC qui ne sont pas sur la voie de la normalisation peuvent recevoir des valeurs après revue par un expert. Une spécification qui n'est pas de l'IETF peut recevoir des valeurs en publiant une RFC pour information ou une

RFC sur la voie de la normalisation qui fait référence à la spécification externe ; cette spécification doit être publique et être publiée sous une forme permanente, assez proche d'une RFC de l'IETF. Il est très souhaitable, bien que non exigé, que la spécification complète soit publiée comme RFC de l'IETF.

Les plus petites valeurs de type de chiffrement devraient être utilisées pour les mécanismes sur la voie de la normalisation de l'IETF, et des valeurs beaucoup plus élevées (16 777 216 et au-dessus) pour les autres mécanismes. (La raison en est que dans le codage Kerberos ASN.1, les plus petits numéros codent les plus petites séquences d'octets, de sorte que ceci favorise les mécanismes en cours de normalisation avec des messages légèrement plus petits.) À part cette directive, l'IANA peut choisir les numéros comme bon lui semble.

Les spécifications de projet Internet ne devraient pas inclure de valeurs de numéros de chiffrement et de type de somme de contrôle. À la place, elles devraient indiquer que les valeurs seront allouées par l'IANA lorsque le document sera approuvé comme RFC. Pour les essais de développement et d'interopérabilité, les valeurs dans la gamme d'utilisation privée (valeurs négatives) peuvent être utilisées mais ne devraient pas être incluse dans le projet de spécification.

Chaque valeur enregistrée devrait avoir un nom de référence unique associé. La liste données à la Section 8 a été utilisée pour créer le registre initial ; elle inclut des réservations pour des spécifications en cours en parallèle avec le présent document, et certaines autres valeurs dont on pense qu'elles sont déjà utilisées.

12. Remerciements

Le présent document est une extension de la spécification de chiffrement incluse dans la [RFC1510] par B. Clifford Neuman et John Kohl, et beaucoup du texte de la présentation des concepts et des spécifications de DES, est tiré directement de ce document. Le cadre abstrait présenté dans ce document a été construit par Jeff Altman, Sam Hartman, Jeff Hutzelman, Cliff Neuman, Ken Raeburn, et Tom Yu, et les détails ont été reprécisés plusieurs fois sur la base des commentaires de John Brezak et d'autres. Marc Horowitz a écrit la spécification d'origine de triple-DES et les déductions de clés dans deux projets Internet (sous les noms de draft-horowitz-key-derivation et draft-horowitz-kerb-key-derivation) qui ont été ensuite fondus dans un projet de révision de la [RFC1510] à partir de laquelle le présent document a été ensuite extrait. Tom Yu a fourni le texte qui décrit les modifications de l'algorithme de CRC standard comme les mises en œuvre de Kerberos l'utilisent actuellement, et une partie du texte de la section sur les considérations pour la sécurité. Miroslav Jurisic a fourni les informations pour un des cas d'essai de UTF-8 pour les fonctions de chaîne à clé. Marcus Watts a fait remarquer des erreurs des versions antérieures et que le profil simplifié pouvait être facilement modifié pour prendre en charge des modes de vol du texte chiffré. Simon Josefsson a contribué à des précisions de la "somme de contrôle CBC" DES et des descriptions de chaîne à clé et de clé faible, et de certaines valeurs d'essai. Simon Josefsson, Louis LeVay, et d'autres ont aussi relevé des erreurs dans les versions antérieures.

Appendice A. Valeurs d'essai

Cette section donne des valeurs d'essai pour diverses fonctions définies ou décrites dans le présent document. Par convention, la plupart des entrées sont des chaînes ASCII, bien que quelques échantillons d'UTF-8 soient fournis pour les fonctions de chaîne à clé. Les clés et autres données binaires sont spécifiées comme des chaînes hexadécimales.

A.1 n-fold

La fonction n-fold est définie au paragraphe 5.1. Comme il y est noté, la valeur d'échantillon donnée dans le texte d'origine qui définit l'algorithme est incorrect. Voici des cas d'essai fournis par Marc Horowitz et Simon Josefsson :

64-fold("012345") =
64-fold(303132333435) = be072631276b1955

56-fold("password") =
56-fold(70617373776f7264) = 78a07b6caf85fa

64-fold("Rough Consensus, and Running Code") =
64-fold(526f75676820436f6e73656e7375732c20616e642052756e6e696e6720436f6465) = bb6ed30870b7f0e0

168-fold("password") =
168-fold(70617373776f7264) = 59e4a8ca7c0385c3c37b3f6d2000247cb6e6bd5b3e

192-fold("MASSACHVSETTS INSTITVTE OF TECHNOLOGY")
192-fold(4d41535341434856534554545320494e5354495456544520
4f4620544543484e4f4c4f4759) = db3b0d8f0b061e603282b308a50841229ad798fab9540c1b

168-fold("Q") =
168-fold(51) = 518a54a2 15a8452a 518a54a2 15a8452a518a54a2 15

168-fold("ba") =
168-fold(6261) = fb25d531 ae897449 9f52fd92 ea9857c4ba24cf29 7e

Voici des valeurs supplémentaires qui correspondent aux valeurs repliées de la chaîne "kerberos" ; la forme en 64 bits est utilisée dans la chaîne à clé des3 (paragraphe 6.3.1).

64-fold("kerberos") = 6b657262 65726f73
128-fold("kerberos") = 6b657262 65726f73 7b9b5b2b 93132b93
168-fold("kerberos") = 8372c236 344e5f15 50cd0747 e15d62ca 7a5a3bce a4
256-fold("kerberos") = 6b657262 65726f73 7b9b5b2b 93132b935c9bdcda d95c9899 c4cae4de e6d6cae4

Noter que les octets initiaux correspondent exactement à la chaîne d'entrée lorsque la longueur de sortie est un multiple de la longueur d'entrée.

A.2 mit_des_string_to_key

La fonction mit_des_string_to_key est définie au paragraphe 6.2. On présente ici plusieurs valeurs d'essai, avec certains des résultats intermédiaires. Le quatrième essai montre l'utilisation de UTF-8 avec trois caractères. Les deux derniers essais sont spécifiquement construits afin de déclancher les arrangements de clé faible pour les clés intermédiaires produites par repliage en éventail *(fan-folding)* ; on n'a pas de cas d'essai qui cause une telle correction pour la clé finale.

Codages UTF-8 utilisés dans la valeur d'essai :

	C5 A1
U+1011E	F0 9D 84 9E

Valeur d'essai :

sel : "ATHENA.MIT.EDUraeburn" 415448454e412e4d49542e4544557261656275726e
mot de passe : "password" 70617373776f7264
fan-fold result: c01e38688ac86c2e
clé intermédiaire : c11f38688ac86d2f
clé DES : cbc22fae235298e3

sel : "WHITEHOUSE.GOVdanny" 5748495445484f5553452e474f5664616e6e79
mot de passe : "potatoe" 706f7461746f65
fan-fold result: a028944ee63c0416
clé intermédiaire : a129944fe63d0416
clé DES : df3d32a74fd92a01

sel : "EXAMPLE.COMpianist" 4558414D504C452E434F4D7069616E697374
mot de passe : g-clef (U+1011E) f09d849e
fan-fold result: 3c4a262c18fab090
clé intermédiaire : 3d4a262c19fbb091
clé DES : 4ffb26bab0cd9413

sel : "ATHENA.MIT.EDUJuri" + s-caron(U+0161) + "i" + c-acute(U+0107)
415448454e412e4d49542e4544554a757269c5a169c487
mot de passe : eszett(U+00DF) c39f
fan-fold result: b8f6c40e305afc9e
clé intermédiaire : b9f7c40e315bfd9e
clé DES : 62c81a5232b5e69d

sel : "AAAAAAAA" 4141414141414141
mot de passe : "11119999" 3131313139393939
fan-fold result: e0e0e0e0f0f0f0f0

```
clé intermédiaire :          e0e0e0e0f1f1f101
clé DES :           984054d0f1a73e31

sel :                    "FFFFAAAA"   4646464641414141
mot de passe :           "NNNN6666"   4e4e4e4e36363636
fan-fold result:         1e1e1e1e0e0e0e0e
clé intermédiaire :      1f1f1f1f0e0e0efe
clé DES :                c4bf6b25adf7a4f8
```

Ce schéma fourni par Simon Josefsson montre les étapes de traitement intermédiaire d'une des entrées de l'essai :

```
string_to_key (des-cbc-md5, chaîne, sel)
        ;; chaîne:
        ;; `mot de passe' (longueur 8 octets)
        ;; 70 61 73 73 77 6f 72 64
        ;; sel :
        ;; `ATHENA.MIT.EDUraeburn' (longueur 21 octets)
        ;; 41 54 48 45 4e 41 2e 4d  49 54 2e 45 44 55 72 61
        ;; 65 62 75 72 6e
des_string_to_key (chaîne, sel)
        ; chaîne :
        ; `mot de passe' (longueur 8 octets)
        ; 70 61 73 73 77 6f 72 64
        ; Sel :
        ; `ATHENA.MIT.EDUraeburn' (longueur 21 octets)
        ; 41 54 48 45 4e 41 2e 4d  49 54 2e 45 44 55 72 61
        ; 65 62 75 72 6e
odd = 1;
s = chaîne | sel ;
tempstring = 0; /* chaîne de 56 bits */
pad(s); /* avec des nuls jusqu'à une frontière de 8 octet */
        ;; s = pad(chaîne|sel):
        ;; `mot-de-passeATHENA.MIT.EDUraeburn\x00\x00\x00'
        ;; (longueur 32 octets)
        ;; 70 61 73 73 77 6f 72 64  41 54 48 45 4e 41 2e 4d
        ;; 49 54 2e 45 44 55 72 61  65 62 75 72 6e 00 00 00
for (8byteblock in s) {
        ;; loop iteration 0
        ;; 8byteblock:
        ;; `password' (longueur 8 octets)
        ;; 70 61 73 73 77 6f 72 64
        ;; 01110000 01100001 01110011  01110011 01110111 01101111
        ;; 01110010 01100100
56bitstring = removeMSBits(8byteblock);
        ;; 56bitstring:
        ;; 1110000 1100001 1110011  1110011 1110111 1101111
        ;; 1110010 1100100
if (odd == 0) reverse(56bitstring);    ;; odd=1
odd = ! odd
tempstring = tempstring XOR 56bitstring;
        ;; tempstring
        ;; 1110000 1100001 1110011  1110011 1110111 1101111
        ;; 1110010 1100100

    for (8byteblock in s) {
        ;; loop iteration 1
        ;; 8byteblock:
        ;; `ATHENA.M' (longueur 8 octets)
        ;; 41 54 48 45 4e 41 2e 4d
        ;; 01000001 01010100 01001000  01000101 01001110 01000001
        ;; 00101110 01001101
    56bitstring = removeMSBits(8byteblock);
        ;; 56bitstring:
```

```
        ;; 1000001 1010100 1001000  1000101 1001110 1000001
        ;; 0101110 1001101
    if (odd == 0) reverse(56bitstring);    ;; odd=0
    reverse(56bitstring)
        ;; 56bitstring après inversion
        ;; 1011001 0111010 1000001  0111001 1010001 0001001
        ;; 0010101 1000001
    odd = ! odd
    tempstring = tempstring XOR 56bitstring;
        ;; tempstring
        ;; 0101001 1011011 0110010  1001010 0100110 1100110
        ;; 1100111 0100101

    for (8byteblock in s) {
        ;; loop iteration 2
        ;; 8byteblock:
        ;; `IT.EDUra' (longueur 8 octets)
        ;; 49 54 2e 45 44 55 72 61
        ;; 01001001 01010100 00101110  01000101 01000100 01010101
        ;; 01110010 01100001
    56bitstring = removeMSBits(8byteblock);
        ;; 56bitstring:
        ;; 1001001 1010100 0101110  1000101 1000100 1010101
        ;; 1110010 1100001
    if (odd == 0) reverse(56bitstring);    ;; odd=1
    odd = ! odd
    tempstring = tempstring XOR 56bitstring;
        ;; tempstring
        ;; 1100000 0001111 0011100  0001111 1100010 0110011
        ;; 0010101 1000100

    for (8byteblock in s) {
        ;; loop iteration 3
        ;; 8byteblock:
        ;; `eburn\x00\x00\x00' (longueur 8 octets)
        ;; 65 62 75 72 6e 00 00 00
        ;; 01100101 01100010 01110101  01110010 01101110 00000000
        ;; 00000000 00000000
    56bitstring = removeMSBits(8byteblock);
        ;; 56bitstring:
        ;; 1100101 1100010 1110101  1110010 1101110 0000000
        ;; 0000000 0000000
    if (odd == 0) reverse(56bitstring);    ;; odd=0
    reverse(56bitstring)
        ;; 56bitstring after reverse
        ;; 0000000 0000000 0000000  0111011 0100111 1010111
        ;; 0100011 1010011
    odd = ! odd
    tempstring = tempstring XOR 56bitstring;
        ;; tempstring
        ;; 1100000 0001111 0011100  0110100 1000101 1100100
        ;; 0110110 0010111

    for (8byteblock in s) {
    }
        ;; pour la boucle terminée

    tempkey = key_correction(add_parity_bits(tempstring));
        ;; tempkey
        ;; `\xc1\x1f8h\x8a\xc8m\x2f' (longueur 8 octets)
        ;; c1 1f 38 68 8a c8 6d 2f
        ;; 11000001 00011111 00111000  01101000 10001010 11001000
        ;; 01101101 00101111
```

```
key = key_correction(DES-CBC-check(s,tempkey));
      ;; key
      ;; `\xcb\xc2\x2f\xae\x23R\x98\xe3' (longueur 8 octets)
      ;; cb c2 2f ae 23 52 98 e3
      ;; 11001011 11000010 00101111  10101110 00100011 01010010
      ;; 10011000 11100011

      ;; clé string_to_key :
      ;; `\xcb\xc2\x2f\xae\x23R\x98\xe3' (longueur 8 octets)
      ;; cb c2 2f ae 23 52 98 e3
```

A.3 DR et DK DES3

Ces essais montrent les valeurs dérivées aléatoires et dérivées de clé pour le schéma de chiffrement des3-hmac-sha1-kd, en utilisant les fonctions DR et DK définies au paragraphe 6.3.1. Les clés d'entrée ont été générées au hasard ; les valeurs d'usage sont tirées de la présente spécification.

clé : dce06b1f64c857a11c3db57c51899b2cc1791008ce973b92
usage : 0000000155
DR : 935079d14490a75c3093c4a6e8c3b049c71e6ee705
DK : 925179d04591a79b5d3192c4a7e9c289b049c71f6ee604cd

clé : 5e13d31c70ef765746578531cb51c15bf11ca82c97cee9f2
usage : 00000001aa
DR : 9f58e5a047d894101c469845d67ae3c5249ed812f2
DK : 9e58e5a146d9942a101c469845d67a20e3c4259ed913f207

clé : 98e6fd8a04a4b6859b75a176540b9752bad3ecd610a252bc
usage : 0000000155
DR : 12fff90c773f956d13fc2ca0d0840349dbd39908eb
DK : 13fef80d763e94ec6d13fd2ca1d085070249dad39808eabf

clé : 622aec25a2fe2cad7094680b7c64940280084c1a7cec92b5
usage : 00000001aa
DR : f8debf05b097e7dc0603686aca35d91fd9a5516a70
DK : f8dfbf04b097e6d9dc0702686bcb3489d91fd9a4516b703e

clé : d3f8298ccb166438dcb9b93ee5a7629286a491f838f802fb
usage : 6b65726265726f73 ("kerberos")
DR : 2270db565d2a3d64cfbfdc5305d4f778a6de42d9da
DK : 2370da575d2a3da864cebfdc5204d56df779a7df43d9da43

clé : c1081649ada74362e6a1459d01dfd30d67c2234c940704da
usage : 0000000155
DR : 348056ec98fcc517171d2b4d7a9493af482d999175
DK : 348057ec98fdc48016161c2a4c7a943e92ae492c989175f7

clé : 5d154af238f46713155719d55e2f1f790dd661f279a7917c
usage : 00000001aa
DR : a8818bc367dadacbe9a6c84627fb60c294b01215e5
DK : a8808ac267dada3dcbe9a7c84626fbc761c294b01315e5c1

clé : 798562e049852f57dc8c343ba17f2ca1d97394efc8adc443
usage : 0000000155
DR : c813f88b3be2b2f75424ce9175fbc8483b88c8713a
DK : c813f88a3be3b334f75425ce9175fbe3c8493b89c8703b49

clé : 26dce334b545292f2feab9a8701a89a4b99eb9942cecd016
usage : 00000001aa
DR : f58efc6f83f93e55e695fd252cf8fe59f7d5ba37ec
DK : f48ffd6e83f83e7354e694fd252cf83bfe58f7d5ba37ec5d

A.4 DES3string_to_key

Voici les clés générées pour certaines des chaînes d'entrée ci-dessus pour triple-DES avec la déduction de clé comme définie au paragraphe 6.3.1.

sel : "ATHENA.MIT.EDUraeburn"
mot de passe : "password"
clé : 850bb51358548cd05e86768c313e3bfef7511937dcf72c3e

sel : "WHITEHOUSE.GOVdanny"
mot de passe : "potatoe"
clé : dfcd233dd0a43204ea6dc437fb15e061b02979c1f74f377a

sel : "EXAMPLE.COMbuckaroo"
mot de passe : "penny"
clé : 6d2fcdf2d6fbbc3ddcadb5da5710a23489b0d3b69d5d9d4a

sel : "ATHENA.MIT.EDUJuri" + s-caron(U+0161) + "i" + c-acute(U+0107)
mot de passe : eszett(U+00DF)
clé : 16d5a40e1ce3bacb61b9dce00470324c831973a7b952feb0

sel : "EXAMPLE.COMpianist"
mot de passe : g-clef(U+1011E)
clé : 85763726585dbc1cce6ec43e1f751f07f1c4cbb098f40b19

A.5 CRC-32 modifié

Ci-dessous figurent les valeurs modifiées de CRC32 pour diverses chaîne ASCII et d'octets. Seuls les caractères ASCII imprimables font l'objet d'une somme de contrôle, sans un octet en queue de style C- de valeur zéro. Le CRC de 32 bits modifié et la séquence d'octets de résultat sont utilisés dans Kerberos comme montré. (Les valeurs d'octet sont séparées ici pour souligner que ce sont des valeurs d'octet et non des nombres de 32 bits, ce qui serait la forme la plus convenable de manipulation dans certaines mises en œuvre. L'ordre des bits et des octets utilisé en interne pour un tel nombre n'est pas pertinent ; la séquence d'octets générée est seule importante.)

mod-crc-32("foo") =	33 bc 32 73
mod-crc-32("test0123456789") =	d6 88 3e b8
mod-crc-32("MASSACHVSETTS INSTITVTE OF TECHNOLOGY") =	f7 80 41 e3
mod-crc-32(8000) =	4b 98 83 3b
mod-crc-32(0008) =	32 88 db 0e
mod-crc-32(0080) =	20 83 b8 ed
mod-crc-32(80) =	20 83 b8 ed
mod-crc-32(80000000) =	3b b6 59 ed
mod-crc-32(00000001) =	96 30 07 77

Appendice B. Changements significatifs par rapport à la RFC 1510

Les profils de chiffrement et de mécanisme de somme de contrôle sont nouveaux. L'ancienne spécification définissait quelques opérations pour divers mécanismes mais ne précisait pas quelles propriétés abstraites devraient être exigées des nouveaux mécanismes, ni comment assurer qu'une spécification de mécanisme soit assez complète pour l'interopérabilité entre les mises en œuvre. Les nouveaux profils diffèrent dans une certaine mesure de l'ancienne spécification :

Certaines définitions de message de la [RFC1510] pourraient être lus comme permettant que la valeur initiale soit spécifiée par l'application ; le texte était trop vague. Cela est explicitement interdit dans la présente spécification. Certains algorithmes de chiffrement peuvent ne pas utiliser de valeurs d'initialisation, de sorte que s'appuyer sur des valeurs d'initialisation secrètes choisies pour la sécurité n'est pas raisonnable. Aussi, le confondeur ajouté dans les algorithmes existants est en gros équivalent à une valeur d'initialisation par message qui est révélée sous forme chiffrée. Cependant, porter l'état d'un chiffrement à un autre est explicitement permis à travers l'objet opaque "état de chiffrement".

L'utilisation de la déduction de clés est nouvelle.

Plusieurs nouvelles méthodes sont introduites, y compris la génération d'une clé dans un format de protocole filaire à partir de données d'entrée aléatoires.

Les moyens d'influencer l'algorithme string-to-key sont exposés plus clairement.

La prise en charge de Triple-DES est introduite.

La fonction pseudo aléatoire est introduite.

Les descriptions de des-cbc-crc, DES string-to-key et de CRC ont été mises à jour pour les aligner sur les mises en œuvre existantes.

La [RFC1510] n'indiquait pas quel jeu de caractères ou quel codage pouvaient être utilisés pour les phrases de passe et les sels.

Dans la [RFC1510], les types de clé, les algorithmes de chiffrement, et les algorithmes de somme de contrôle étaient seulement associés de façon lâche, et l'association n'était pas bien décrite. Dans la présente spécification, les types de clés et les algorithmes de chiffrement ont une correspondance biunivoque, et les associations entre algorithmes de chiffrement et de somme de contrôle sont décrites de telle sorte que les sommes de contrôle puissent être calculées en fonction des clés négociées, sans exiger d'autre négociation pour les types de sommes de contrôle.

Notes

[1] Bien que Code d'authentification de message (MAC) ou vérification d'intégrité du message (MIC) soient des termes beaucoup plus appropriés pour la plupart des utilisations du présent document, nous continuerons d'utiliser le terme de somme de contrôle pour des raisons historiques.

[2] L'extension du mode CBC à travers les messages serait un exemple évident de ce chaînage. Un autre en serait l'utilisation du mode compteur, avec un compteur initialisé au hasard et attaché au texte chiffré ; un second message pourrait continuer d'incrémenter le compteur lors du chaînage de l'état de chiffrement, évitant ainsi d'avoir à transmettre une autre valeur de compteur. Cependant, ce chaînage n'est utile que pour les séquence de messages ininterrompues et ordonnées.

[3] Dans le cas de Kerberos, les objets chiffrés seront généralement de l'ASN.1 codé en DER, qui contiennent l'indication de leur longueur dans les premiers octets.

[4] Au moment de la rédaction, de nouveaux modes de fonctionnement ont été proposés, dont certains permettent simultanément le chiffrement et la protection d'intégrité. Lorsque ces propositions auront été soumises à une analyse adéquate, il serait souhaitable de formuler un nouveau profil simplifié sur la base de l'une d'entre elles.

[5] Il devrait être noté que le vecteur échantillon de l'appendice B.2 de l'article d'origine s'est révélé incorrect. Deux mises en œuvre indépendantes de la spécification (une en C par Marc Horowitz, et l'autre en Scheme par Bill Sommerfeld) s'accordent sur une valeur différente de celle de [Blument].

[6] Par exemple, dans la mise en œuvre du MIT de la [RFC1510], la somme de contrôle sans clés rsa-md5 des données d'application peut être incluse dans un authentifiant chiffré dans une clé de service.

[7] Utiliser une variante de la clé limite l'utilisation d'une clé à une fonction particulière, séparant les fonctions de génération d'une somme de contrôle des autres chiffrements effectués en utilisant la clé de session. La constante 0xF0F0F0F0F0F0F0F0 a été choisie parce qu'elle maintient la parité de clé. Les propriétés de DES empêchaient l'utilisation du complément. La même constante est utilisée pour des besoins similaires dans la vérification d'intégrité de message dans la norme de messagerie à confidentialité améliorée.

[8] Peut-être qu'une des raisons les plus courantes pour effectuer directement le chiffrement est le contrôle direct sur la négociation et le choix d'un algorithme de chiffrement "suffisamment fort" (quoi que cela veuille dire dans le contexte d'une application données). Bien que Kerberos ne fournisse directement aucune facilité directe pour négocier les types de chiffrement entre le client et le serveur de l'application, il y a d'autres moyens pour atteindre des objectifs similaires (par exemple, de demander seulement des types de clé de session "forts" au KDC, et en supposant que le type réellement retourné par le KDC sera compris et pris en charge par le serveur d'application).

Références normatives

[Bellare98] Bellare, M., Desai, A., Pointcheval, D., and P. Rogaway, "Relations Among Notions of Security for Public-Key Encryption Schemes". Résumé long publié dans le compte-rendu de Advances in Cryptology-Crypto 98, Notes de lecture dans Computer Science Vol. 1462, H. Krawcyzk éditeur, Springer-Verlag, 1998.

[Blument] Blumenthal, U. and S. Bellovin, "A Better Key Schedule for DES-Like Ciphers", Compte-rendu de PRAGOCRYPT '96, 1996.

[CRC] Organisation internationale de normalisation, "Systèmes de traitement de l'information – Communications de données – Procédure de contrôle de liaison de données à haut niveau – Structure de trame," IS 3309, 3è édition, octobre 1984.

[DES77] National Bureau of Standards, U.S. Department of Commerce, "Data Encryption Standard," Federal Information Processing Standards Publication 46, Washington, DC, 1977.

[DESI81] National Bureau of Standards, U.S. Department of Commerce, "Guidelines for implementing and using NBS Data Encryption Standard," Federal Information Processing Standards Publication 74, Washington, DC, 1981.

[DESM80] National Bureau of Standards, U.S. Department of Commerce, "DES Modes of Operation," Federal Information Processing Standards Publication 81, Springfield, VA, décembre 1980.

[Dolev91] Dolev, D., Dwork, C., and M. Naor, "Non-malleable cryptography", Compte-rendu du 23ème Symposium annuel sur la Théorie du calcul, ACM, 1991.

[RFC1320] R. Rivest, "Algorithme de résumé de message MD4", avril 1992. *(Historique, Information)*

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997.

[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. *(Obsolète, voir la* RFC5226)

[RFC3962] K. Raeburn, "Chiffrement de la norme de chiffrement évolué (AES) pour Kerberos 5", février 2005.

[SG92] Stubblebine, S. and V. D. Gligor, "On Message Integrity in Cryptographic Protocols," dans Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, California, mai 1992.

Références pour information

[Bellovin91] Bellovin, S. M. and M. Merrit, "Limitations of the Kerberos Authentication System", dans Proceedings of the Winter 1991 Usenix Security Conference, janvier 1991.

[Bellovin99] Bellovin, S. M. and D. Atkins, communications privées, 1999.

[EFF-DES] Electronic Frontier Foundation, "Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design", O'Reilly & Associates, Inc., mai 1998.

[RFC1510] J. Kohl et C. Neuman, "Service Kerberos d'authentification de réseau (v5)", septembre 1993. *(Obsolète, voir* RFC6649)

[RFC1851] P. Karn, P. Metzger et W. Simpson, "Transformation ESP de triple DES", septembre 1995.

[RFC1964] J. Linn, "Mécanisme GSS-API de Kerberos version 5", juin 1996. (*MàJ par* RFC4121 et RFC6649)

[RFC2040] R. Baldwin et R. Rivest, "Algorithmes RC5, RC5-CBC, RC5-CBC-Pad, et RC5-CTS", octobre 1996. *(Information)*

[RFC2202] P. Cheng et R. Glenn, "Cas d'essai pour HMAC-MD5 et HMAC-SHA-1", septembre 1997. *(Information)*

[RFC2404] C. Madson, R. Glenn, "Utilisation de HMAC-SHA-1-96 au sein d'ESP et d'AH", novembre 1998. *(P.S.)*

[RFC2405] C. Madson et N. Doraswamy, "Algorithme de chiffrement ESP DES-CBC avec IV explicite", novembre 1998. *(P.S.)*

[RFC4120] C. Neuman et autres, "Service Kerberos d'authentification de réseau (V5)", juillet 2005. (*MàJ par* RFC4537, RFC5021, RFC6649)

[Schneier96] B. Schneier, "Applied Cryptography Second Edition", John Wiley & Sons, New York, NY, 1996. ISBN 0-471-12845-7.

Adresse de l'éditeur

Kenneth Raeburn
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
mél : raeburn@mit.edu

Déclaration de droits de reproduction

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