Internet Engineering Task Force (IETF)
Request for Comments : 8489
RFC rendue obsolète : 5389
Catégorie : Sur la voie de la normalisation
ISSN : 2070-1721

Traduction Claude Brière de L'Isle

M. Petit-Huguenin, Impedance Mismatch
G. Salgueiro, Cisco
J. Rosenberg, Five9
D. Wing, Citrix
R. Mahy
P. Matthews, Nokia
février 2020

Utilitaires de traversée de session pour les NAT (STUN)

Résumé
Utilitaires de traversée de session pour les traducteurs d'adresse réseau (STUN, *Session Traversal Utilities for NAT*) est un protocole qui sert d'outil pour d'autres protocoles qui traitent de la traversée de NAT. Il peut être utilisé par un point d'extrémité pour déterminer l'adresse et l'accès IP qui lui sont alloués par un NAT. Il peu aussi être utilisé pour vérifier la connexité entre deux points d'extrémité, et de protocole de maintien en vie des liens de NAT. STUN fonctionne avec de nombreux NAT existants et n'exige aucun comportement particulier de leur part.

STUN n'est pas par lui-même une solution de traversée de NAT. C'est plutôt un outil à utiliser dans le contexte d'une solution de traversée de NAT.

Ce document rend obsolète la RFC 5389.
(Cette traduction incorpore les errata 6268 et 6290;)

Statut de ce mémoire
Ceci est un document de l'Internet sur la voie de la normalisation.

Le présent document a été produit par l'équipe d'ingénierie de l'Internet (IETF). Il représente le consensus de la communauté de l'IETF. Il a subi une révision publique et sa publication a été approuvée par le groupe de pilotage de l'ingénierie de l'Internet (IESG). Tous les documents approuvés par l'IESG ne sont pas candidats à devenir une norme de l'Internet ; voir la Section 2 de la RFC5741.

Les informations sur le statut actuel du présent document, tout errata, et comment fournir des réactions sur lui peuvent être obtenues à http://www.rfc-editor.org/info/rfc8489

Table des matières

1. Introduction

Le protocole défini dans la présente spécification, Utilitaires de traversée de session pour les NAT (STUN, *Session Traversal Utilities for NAT*) fournit un outil pour traiter les traducteurs d'adresse de réseau (NAT, *Network Address Translator*). Il donne un moyen pour qu'un point d'extrémité détermine l'adresse et l'accès IP alloués par un NAT qui correspondent à son adresse et accès IP privés. Il donne aussi un moyen pour qu'un point d'extrémité garde en vie un lien de NAT. Avec certaines extensions, le protocole peut être utilisé pour faire des vérifications de connexité entre deux points d'extrémité [RFC8445] ou pour relayer les paquets entre deux points d'extrémité [RFC5766].

En gardant cette nature d'outil, la présente spécification définit un format de paquet extensible, elle définit le fonctionnement sur plusieurs protocoles de transport, et donne deux formes d'authentification.

STUN est destiné à être utilisé dans le contexte d'une ou plusieurs solutions de traversée de NAT. Ces solutions sont appelées des "usages STUN". Chaque usage décrit comment STUN est utilisé pour réaliser la solution de traversée de NAT. Normalement, un usage indique quand les messages STUN sont envoyés, quels attributs facultatifs inclure, quel serveur est utilisé, et quel mécanisme d'authentification est à utiliser. L'établissement de connexité interactive (ICE, *Interactive Connectivity Establishment*) [RFC8445] est un usage de STUN. SIP sortant [RFC5626] est un autre usage de STUN. Dans certains cas, un usage va exiger des extensions à STUN. Une extension de STUN peut être sous forme de nouvelles méthodes, attributs, ou codes de réponse d'erreur. Plus d'informations sur les usages de STUN se trouvent à la Section 13.

2. Vue d'ensemble du fonctionnement

Cette section est seulement descriptive.

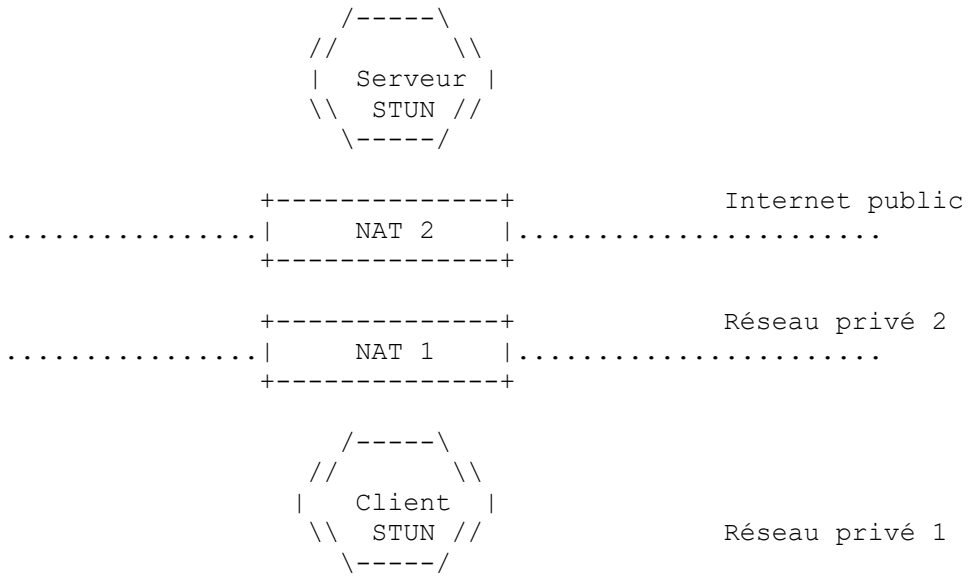

Figure 1 : Configuration STUN possible

Une configuration possible de STUN est montrée à la Figure 1. Dans cette configuration, il y a deux entités (appelées des agents STUN) qui mettent en œuvre le protocole STUN. L'agent inférieur dans la figure est le client, qui est connecté au réseau privé 1. Ce réseau connecte au réseau privé 2 à travers le NAT 1. Le réseau privé 2 connecte à l'Internet public à travers le NAT 2. L'agent supérieur dans la figure est le serveur, qui réside sur l'Internet public.

STUN est un protocole de client-serveur. Il prend en charge deux types de transactions. Une est une transaction de demande/réponse dans laquelle un client envoie une demande à un serveur, et le serveur retourne une réponse. Le second type est une transaction d'indication dans laquelle un des agents -- client ou serveur – envoie une indication qui ne génère pas de réponse. Les deux types de transactions incluent un identifiant de transaction, qui est un nombre choisi au hasard de 96 bits. Pour les transactions de demande/réponse, cet identifiant de transaction permet au client d'associer la réponse à la demande qui l'a générée ; pour les indications, l'identifiant de transaction sert à aider au débogage.

Tous les messages STUN commencent par un en-tête fixe qui inclut une méthode, une classe, et l'identifiant de transaction. La méthode indique de quelle demande ou indication il s'agit ; la présente spécification définit juste une méthode, Binding *(lien)* mais il est prévu que d'autres méthodes seront définies dans d'autres documents. La classe indique si c'est une demande, une réponse de succès, une réponse d'erreur, ou une indication. À la suite de l'en-tête fixe viennent zéro, un ou plusieurs attributs, qui sont des extensions de Type-Longueur-Valeur portant des informations supplémentaires pour le message spécifique.

Le présent document définit une seule méthode appelée "Binding". La méthode Binding peut être utilisée dans des transactions de demande/réponse ou dans des transactions d'indication. Quand elle est utilisée dans des transactions demande/réponse, la méthode Binding peut être utilisée pour déterminer le lien particulier qu'un NAT a alloué à un client STUN. Quand elle est utilisée dans des transactions de demande/réponse ou d'indication, la méthode Binding peut aussi être utilisée pour garder ces liens en vie.

Dans la transaction de demande/réponse Binding, une demande Binding est envoyée d'un client STUN à un serveur STUN. Quand la demande Binding arrive au serveur STUN, elle peut être passée à travers un ou plusieurs NAT entre le client STUN et le serveur STUN (dans la Figure 1, il y a deux de ces NAT). Lorsque le message de demande Binding passe à travers un NAT, le NAT modifie l'adresse de transport de source (c'est-à-dire, l'adresse IP de source et l'accès de source) du paquet. Par suite, l'adresse de transport de source de la demande reçue par le serveur va être l'adresse et accès IP publics créés par le NAT le plus proche du serveur. C'est appelé une "adresse de transport réflexive". Le serveur STUN copie cette adresse de transport de source dans un attribut XOR-MAPPED-ADDRESS dans la réponse STUN Binding et renvoie la réponse Binding au client STUN. Lorsque ce paquet repasse à travers un NAT, le NAT modifie l'adresse de transport de destination dans l'en-tête IP, mais l'adresse de transport dans l'attribut XOR-MAPPED-ADDRESS au sein du corps de la réponse STUN va rester inchangée. De cette façon, le client peut connaître l'adresse de transport réflexive allouée par le NAT le plus externe par rapport au serveur STUN.

Dans certains usages, STUN doit être multiplexé avec d'autres protocoles (par exemple, de la [RFC8445] et de la [RFC5626]). Dans ces usages, il doit y avoir un moyen d'inspecter un paquet et de déterminer si c'est un paquet STUN ou non. STUN fournit trois champs dans l'en-tête STUN avec des valeurs fixes qui peuvent être utilisées à cette fin. Si ce n'est pas suffisant, alors les paquets STUN peuvent aussi contenir une valeur FINGERPRINT *(empreinte digitale)* qui peut de plus être utilisée pour distinguer les paquets.

STUN définit un ensemble de procédures facultatives qu'un usage peut décider d'utiliser, appelés des "mécanismes". Ces mécanismes incluent la découverte DNS, une technique de redirection sur un serveur de remplacement, un attribut d'empreinte digitale pour d"multiplexer, et deux échanges d'authentification et d'intégrité de message. Les mécanismes d'authentification tournent autour de l'utilisation d'un nom d'utilisateur, d'un mot de passe, et d'une valeur d'intégrité de message. Deux mécanismes d'authentification, le mécanisme d'accréditif à long terme et le mécanismes d'accréditif à court terme, sont définis dans la présente spécification. Chaque usage spécifie les mécanismes permis avec lui.

Dans le mécanisme d'accréditif à long terme, le client et le serveur partagent un nom d'utilisateur et un mot de passe pré-provisionnés et effectuent un échange de défi/réponse résumé inspiré de celui défini pour HTTP [RFC7616] mais différent dans les détails. Dans le mécanisme d'accréditif à court terme, le client et le serveur échangent un nom d'utilisateur et un mot de passe par une méthode hors bande avant l'échange STUN. Par exemple, dans l'usage ICE [RFC8445], les deux points d'extrémité utilisent la signalisation hors bande pour échanger un nom d'utilisateur et un mot de passe. Ils sont utilisés pour protéger l'intégrité et authentifier la demande et la réponse. Aucun défi ou nom occasionnel n'est utilisé.

3. Terminologie

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119], [RFC8174] quand, et seulement quand ils apparaissent tout en majuscules, comme montré ci-dessus.

4. Définitions

agent STUN : un agent STUN est une entité qui met en œuvre le protocole STUN. L'entité peut être un client STUN ou un serveur STUN.

client STUN : un client STUN est une entité qui envoie des demandes STUN et reçoit des réponses STUN et des indications STUN. Un client STUN peut aussi envoyer des indications. Dans la présente spécification, les termes de "client STUN" et de "client" sont synonymes.

serveur STUN : un serveur STUN est une entité qui reçoit des demandes STUN et des indications STUN et qui envoie des réponses STUN. Un serveur STUN peut aussi envoyer des indications. Dans cette spécification, les termes "serveur STUN" et "serveur" sont synonymes.

adresse de transport : combinaison d'une adresse et d'un numéro d accès IP (comme un numéro d'accès UDP ou TCP).

adresse de transport réflexive : adresse de transport apprise par un client qui identifie ce client comme il est vu par un autre hôte ou réseau IP, normalement un serveur STUN. Quand il y a un NAT interposé entre le client et l'autre hôte, l'adresse de transport réflexive représente l'adresse transposée allouée au client sur le côté public du NAT. Les adresses

de transport réflexives sont apprises de l'attribut de l'adresse transposée (MAPPED-ADDRESS ou XOR-MAPPED-ADDRESS) dans les réponses STUN.

adresse transposée : même signification que adresse réflexive. Ce terme est conservé seulement pour des raisons historiques et du fait de la dénomination des attributs MAPPED-ADDRESS et XOR-MAPPED-ADDRESS.

accréditif à long terme : un nom d'utilisateur et le mot de passe associé qui représente un secret partagé entre client et serveur. Les accréditifs à long terme sont généralement accordés au client quand un abonné s'inscrit à un service et persiste jusqu'à ce que l'abonné quitte le service ou change explicitement l'accréditif.

mot de passe à long terme : mot de passe provenant d'un accréditif à long terme.

accréditif à court terme : nom d'utilisateur temporaire et mot de passe associé qui représente un secret partagé entre client et serveur. Les accréditifs à court terme sont obtenus par un mécanisme de protocole entre le client et le serveur, précédant l'échange STUN. Un accréditif à court terme a une portée temporelle explicite, qui peut être fondée sur une durée spécifique (comme 5 minutes) ou sur un événement (comme la terminaison d'un dialogue du protocole d'initialisation de session (SIP, *Session Initiation Protocol*) [RFC3261]). La portée spécifique d'un accréditif à court terme est définie par l'usage d'application.

mot de passe à court terme : composant mot de passe d'un accréditif à court terme.

indication STUN : message STUN qui ne reçoit pas de réponse.

attribut : terme STUN pour un objet Type-Longueur-Valeur (TLV) qui peut être ajouté à un message STUN. Les attributs sont divisés en deux types : compréhension exigée et compréhension facultative. Les agents STUN peuvent en toute sécurité ignorer les attributs de compréhension facultative qu'ils ne comprennent pas mais ne peuvent pas réussir à traiter un message si il contient des attributs de compréhension exigée qui ne sont pas compris.

RTO *(Retransmission TimeOut)* : fin de temporisation de retransmission, qui définit la période initiale entre la transmission d'une demande et la première retransmission de cette demande.

5. Structure du message STUN

Les messages STUN sont codés en binaire en utilisant le format du réseau (octet de plus fort poids en premier, aussi couramment appelé gros boutien). L'ordre de transmission est décrit en détails à l'Appendice B de la [RFC0791]. Sauf mention contraire, les constantes numériques sont en décimal (base 10).

Tous les messages STUN comportent un en-tête de 20 octets suivi par zéro, un ou plusieurs attributs. L'en-tête STUN contient un type de message STUN, une longueur de message, un mouchard magique, et un identifiant de transaction.

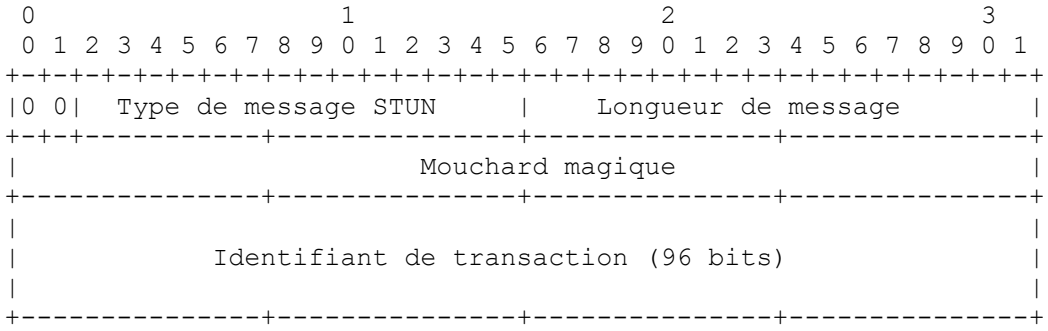

Figure 2 : Format d'en-tête de message STUN

Les deux bits de poids fort de chaque message STUN DOIVENT être à zéro. Ils peuvent être utilisés pour différencier les paquets STUN provenant d'autres protocoles quand STUN est multiplexé avec d'autres protocoles sur le même accès.

Le type de message définit la classe de message (demande, réponse de succès, réponse d'erreur, ou indication) et la méthode de message (la fonction principale) du message STUN. Bien qu'il y ait quatre classes de messages, il y a

seulement deux types de transactions dans STUN : les transactions de demande/réponse (qui consistent en un message de demande et un message de réponse) et les transactions d'indication (qui consistent en un seul message d'indication). Les classes de réponses sont partagées en réponses d'erreur et de succès pour aider au traitement rapide du message STUN.

Le champ Type de message STUN est décomposé plus en détails dans la structure suivante :

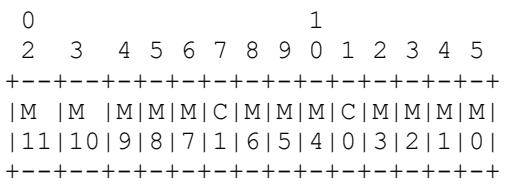

Figure 3 : Format du champ Type de message STUN

Les bits dans le champ Type de message STUN sont montrés du bit de plus fort poids (M11) au bit de moindre poids (M0). M11 à M0 représente un codage de 12 bits de la méthode. C1 et C0 représentent les 2 bits de codage de la classe. une classe de 0b00 est une demande, une classe de 0b01 est une indication, une classe de 0b10 est une réponse de succès, et une classe de 0b11 est une réponse d'erreur. La présente spécification définit une seule méthode, Binding. La méthode et la classe sont orthogonales, de sorte que chaque méthode, demande, réponse de succès, réponse d'erreur, et indication est possible pour cette méthode. Les extensions définissant de nouvelles méthodes DEVRONT indiquer quelles classes sont permises pour cette méthode.

Par exemple, une demande Binding a la classe=0b00 (demande) et la méthode=0b000000000001 (Binding) et est codée dans les 16 premiers bits comme 0x0001. Une réponse Binding a la classe=0b10 (réponse de succès) et la méthode 0b000000000001 et est codée sur les 16 premiers bits par 0x0101.

Note : ce codage malheureux est dû à l'allocation des valeurs dans la [RFC3489] qui ne prend pas en compte le codage des messages d'indication, des réponses de succès, et des réponses en utilisant des champs binaires.

Le champ Mouchard magique DOIT contenir la valeur fixe de 0x2112A442 dans l'ordre des octets du réseau. Dans la [RFC3489], les 32 bits composant le champ Mouchard magique faisaient partie de l'identifiant de transaction ; placer le mouchard magique à cet endroit permet à un serveur de détecter si le client va comprendre certains attributs qui ont été ajoutés à STUN par la [RFC5389]. De plus, cela aide à distinguer les paquets STUN des paquets d'autres protocoles quand STUN est multiplexé avec d'autres protocoles sur le même accès.

L'identifiant de transaction est un identifiant de 96 bits, utilisé pour identifier de façon univoque les transactions STUN. Pour les transactions de demande/réponse, l'identifiant de transaction est choisi par le client STUN pour la demande et le serveur y fait écho dans la réponse. Pour les indications, il est choisi par l'agent qui envoie l'indication. Il sert principalement à corréler les demandes aux réponses, bien qu'il joue aussi un petit rôle pour aider à prévenir certains types d'attaques. Le serveur utilise aussi l'identifiant de transaction comme clé pour identifier chaque transaction parmi tous les clients. À ce titre, l'identifiant de transaction DOIT être choisi uniformément et aléatoirement sur l'intervalle 0 à 2**96-1 et DOIT être cryptographiquement aléatoire. Les nouveaux envois de la même demande réutilisent le même identifiant de transaction, mais le client DOIT choisir un nouvel identifiant de transaction pour les nouvelles transactions sauf si la nouvelle demande est identique au bit près à la demande précédente et envoyée de la même adresse de transport à la même adresse IP. Les réponses de succès et d'erreur DOIVENT porter le même identifiant de transaction que la demande correspondante. Quand un agent agit comme serveur STUN et client STUN sur le même accès, l'identifiant de transaction dans la demande envoyée par l'agent n'a pas de relation avec l'identifiant de transaction de la demande reçue par l'agent.

La longueur de message DOIT contenir la taille du message en octets, sans inclure les 20 octets de l'en-tête STUN. Comme tous les attributs STUN sont bourrés à un multiple de 4 octets, les 2 derniers bits de ce champ sont toujours à zéro. Cela donne un autre moyen de distinguer les paquets STUN des paquets des autres protocoles.

À la suite de la portion fixe de l'en-tête STUN sont zéro, un ou plusieurs attributs. Chaque attribut est codé sous forme de TLV (Type-Longueur-Valeur). Les détails du codage et les attributs eux-mêmes sont donnés à la Section 14.

6. Procédures de base du protocole

Cette Section définit les procédures de base du protocole STUN. Elle décrit comment les messages sont formés, comment

ils sont envoyés, et comment ils sont traités quand ils sont reçus. Elle définit aussi le traitement détaillé de la méthode Binding. Les autres sections du document décrivent les procédures facultatives qu'un usage peut choisir d'utiliser dans certaines situations. D'autres documents pourront définir d'autres extensions à STUN, en ajoutant de nouvelles méthodes, de nouveaux attributs, ou de nouveaux codes de réponse d'erreur.

6.1 Formation d'une demande ou d'une indication

Quand il formule un message de demande ou d'indication, l'agent DOIT suivre les règles de la Section 5 quand il crée l'en-tête. De plus, la classe de message DOIT être "Demande" ou "Indication" (comme approprié) et la méthode doit être Binding ou une autre méthode définie dans un autre document.

L'agent ajoute alors tous les attributs spécifiés par la méthode ou l'usage. Par exemple, certains usages peuvent spécifier que l'agent utilise une méthode d'authentification (Section 9) ou l'attribut FINGERPRINT (Section 7).

Si l'agent envoie une demande, il DEVRAIT ajouter un attribut SOFTWARE à la demande. Les agents PEUVENT inclure un attribut SOFTWARE dans les indications, selon la méthode. Les extensions à STUN devraient discuter si SOFTWARE est utile dans de nouvelles indications. Noter que l'inclusion d'un attribut SOFTWARE peut avoir des implications sur la sécurité ; voir les détails au paragraphe 16.1.2.

Pour la méthode Binding sans authentification, aucun attribut n'est exigé sauf si l'usage le spécifie autrement.

Tous les messages STUN envoyés sur UDP ou DTLS sur UDP [RFC6347] DEVRAIENT faire moins que la MTU du chemin, si elle est connue.

Si la MTU du chemin est inconnue pour UDP, les messages DEVRAIT être le plus petit de 576 octets et de la MTU du premier bond pour IPv4 [RFC1122] et 1280 octets pour IPv6 [RFC8200]. Cette valeur correspond à la taille globale du paquet IP. Par conséquent, pour IPv4, le message STUN réel va devoir faire moins de 548 octets (576 moins 20 octets d'en-tête IP, moins 8 octets d'en-tête UDP, en supposant qu'aucune option IP n'est utilisée).

Si la MTU du chemin est inconnue pour DTLS sur UDP, les règles décrites au paragraphe précédent doivent être ajustées pour prendre en compte la taille de l'en-tête d'enregistrement DTLS (13 octets) la taille du code d'authentification de message (MAC, *Message Authentication Code*) et la taille du bourrage.

STUN ne donne pas de capacité de traiter le cas où la demande est plus petite que la MTU mais la réponse y est supérieure. Il n'est pas envisagé que cette limitation soit un problème pour STUN. La limitation de la MTU est un DEVRAIT, non un DOIT, pour tenir compte des cas où STUN lui-même est utilisé pour sonder les caractéristiques de MTU [RFC5780]. Voir aussi [PMTUD] pour un cadre qui utilise STUN pour ajouter la découverte de la MTU de chemin aux protocoles qui n'ont pas ce mécanisme. En dehors de cela ou d'applications similaires, la contrainte de MTU DOIT être suivie.

6.2 Envoi de la demande ou indication

L'agent envoie ensuite la demande ou indication. Le présent document spécifie comment envoyer les messages STUN sur UDP, TCP, TLS sur TCP, ou DTLS sur UDP ; d'autres protocoles de transport pourront être ajoutés à l'avenir. L'usage STUN doit spécifier quel protocole de transport est utilisé et comment l'agent détermine l'adresse et accès IP du receveur. La Section 8 décrit une méthode fondée sur le DNS pour déterminer l'adresse et l'accès IP d'un serveur qu'un usage peut choisir d'utiliser.

À tout moment, un client PEUT avoir plusieurs demandes STUN en instance avec le même serveur STUN (c'est-à-dire, plusieurs transactions en cours, avec des identtifiants de transaction différents). En l'absence d'autres limites sur le taux de nouvelles transactions (comme celles spécifiées par ICE pour les vérifications de connexité ou quand STUN fonctionne sur TCP) un client DEVRAIT se limiter lui-même à dix transactions en instance sur le même serveur.

6.2.1 Envoi sur UDP ou DTLS sur UDP

Avec STUN sur UDP ou STUN sur DTLS sur UDP [RFC7350], il est possible que le message STUN soit éliminé par le réseau. La fiabilité des transactions de demande/réponse STUN est réalisée par des retransmissions du message de demande par l'application du client lui-même. Les indications STUN ne sont pas retransmises ; donc, les transactions d'indication sur UDP ou de DTLS sur UDP ne sont pas fiables.

Un client DEVRAIT retransmettre un message de demande STUN commençant par un intervalle de temporisation de retransmission (RTO, *Retransmission TimeOut*) doublant après chaque retransmission. Le RTO est une estimation du délai d'aller retour (RTT, *Round-Trip Time*) et est calculé comme décrit dans la [RFC6298], avec deux exceptions. D'abord, la valeur initiale du RTO DEVRAIT être supérieure ou égale à 500 ms. Les cas d'exception pour cela "DEVRAIENT" être quand d'autres mécanismes sont utilisés pour déduire les seuils d'encombrement (comme ceux définis dans ICE pour les flux à taux fixe) ou quand STUN est utilisé dans des environnements non Internet avec des capacités de réseau connues. Dans des liaisons d'accès à ligne fixe, une valeur de 500 ms est RECOMMANDÉE. Ensuite, la valeur de RTO NE DEVRAIT PAS être arrondie à la seconde la plus proche. Une précision d'une milliseconde DEVRAIT être conservée. Comme avec TCP, l'usage de l'algorithme de Karn est RECOMMANDÉ [KARN87]. Quand il est applqué à STUN, il signifie que les estimations de RTT NE DEVRAIENT PAS être calculées à partir des transactions STUN qui ont résulté en la retransmission d'une demande.

La valeur du RTO DEVRAIT être mise en antémémoire par un client après l'achèvement de la transaction et utilisée comme valeur de démarrage du RTO pour la prochaine transaction sur le même serveur (fondée sur l'égalité d'adresse IP). La valeur DEVRAIT être considérée comme périmée et éliminée si aucune transaction ne s'est produite sur le même serveur dans les 10 dernières minutes.

Les retransmissions continuent jusqu'à ce qu'une réponse soit reçue ou jusqu'à ce qu'un total de Rc demandes aient été envoyées. Rc DEVRAIT être configurable et DEVRAIT être 7 par défaut. Si, après la dernière demande, une durée égale à Rm fois le RTO est passée sans réponse (ce qui donne largement le temps d'obtenir une réponse si seulement cette demande finale a réussi) le client DEVRAIT considérer que la transaction a échoué. Rm DEVRAIT être configurable et DEVRAIT avoir une valeur par défaut de 16. Une transaction STUN sur UDP ou DTLS sur UDP est aussi considérée avoir échoué si il y a eu une erreur ICMP [RFC1122]. Par exemple, si on suppose un RTO de 500 ms, les demandes vont être envoyées aux temps 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms, et 31500 ms. Si le client n'a pas reçu de réponse après 39500 ms, il va considérer que la transaction a expiré.

6.2.2 Envoi sur TCP ou TLS sur TCP

Pour TCP et TLS sur TCP [RFC8446], le client ouvre une connexion TCP avec le serveur.

Dans certains usages de STUN, STUN est le seul protocole sur la connexion TCP. Dans ce cas, il peut être envoyé sans l'aide d'aucun tramage ou démultiplexage supplémentaire. Dans d'autres usages, ou avec d'autres extensions, il peut être multiplexé avec d'autres données sur une connexion TCP. Dans ce cas, STUN DOIT être mis par dessus une sorte de protocole de tramage, spécifié par l'usage ou l'extension, qui va permettre à l'agent d'extraire les messages STUN complets et les messages de couche d'application complets. Le service STUN fonctionnant sur le ou les accès bien connus découverts par les procédures du DNS à la Section 8 est pour STUN seul, et non pour STUN multiplexé avec d'autres données. Par conséquent, aucun protocole de tramage n'est utilisé dans les connexions à ces serveurs. Quand un tramage supplémentaire est utilisé, l'usage va spécifier comment le client apprend comment l'appliquer et à quel accès se connecter. Par exemple, dans le cas des vérifications de connexité ICE, ces informations sont apprises par une négociation hors bande entre client et serveur.

La fiabilité de STUN sur TCP et TLS sur TCP est traitée par TCP lui-même, et il n'y a pas de retransmissions au niveau du protocole STUN. Cependant, pour une transaction de demande/réponse, si le client n'a pas reçu de réponse dans les Ti secondes après l'envoi du message de demande, il considère que la transaction a expiré. Ti DEVRAIT être configurable et DEVRAIT avoir une valeur par défaut de 39,5 s. Cette valeur a été choisie pour égaliser les temporisations de TCP et d'UDP pour le RTO initial par défaut.

De plus, si le client est dans l'incapacité d'établir la connexion TCP, ou si la connexion TCP est réinitialisée ou échoue avant la réception d'une réponse, toute transaction de demande/réponse en cours est considérée avoir échoué.

Le client PEUT envoyer plusieurs transactions sur une seule connexion TCP (ou TLS sur TCP) et il PEUT envoyer une autre demande avant de recevoir une réponse à la demande précédente. Le client DEVRAIT garder la connexion ouverte jusqu'à ce que :

- il n'ait plus d'autres demandes ou indications STUN à envoyer sur cette connexion,
- il ne prévoit pas d'utiliser de ressources (comme une adresse transposée (MAPPED-ADDRESS ou XOR-MAPPED-ADDRESS) ou adresse relayée [RFC5766]) apprises par des demandes STUN envoyées sur cette connexion,
- si il multiplexe d'autres protocoles d'application sur cet accès, il a fini d'utiliser ces autres protocoles,
- si il utilise cet accès appris avec un homologue distant, il a établi des communications avec cet homologue distant, comme exigé par les techniques de traversée de NAT TCP (par exemple, [RFC6544]).

Les détails d'un éventuel mécanisme de maintien en vie sont laissés à chaque usage STUN. En tout cas, si une transaction échoue à cause d'une connexion TCP inactive qui ne fonctionne plus, le client DEVRAIT envoyer un RST et essayer d'ouvrir une nouvelle connexion TCP.

De son côté, le serveur DEVRAIT garder la connexion ouverte et laisser le client la clore, sauf si le serveur a déterminé que la connection a expiré (par exemple, du fait que le client s'est déconnecté du réseau). Les liens appris par le client ne vont rester valides dans les NAT intermédiaires que tant que la connexion reste ouverte. Seul le client sait combien de temps il a besoin du lien. Le serveur NE DEVRAIT PAS clore une connexion si une demande a été reçue sur cette connexion pour laquelle une réponse n'a pas été envoyée. Un serveur NE DOIT PAS rouvrir une connexion avec le client afin d'envoyer une réponse. Les serveurs DEVRAIENT suivre les bonnes pratiques concernant la gestion de connexion dans les cas de surcharge.

6.2.3 Envoi sur TLS sur TCP ou DTLS sur UDP

Quand STUN fonctionne lui-même sur TLS sur TCP ou DTLS sur UDP, les suites de chiffrement TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 et TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 DOIVENT être mises en œuvre (pour la compatibilité avec les anciennes versions de ce protocole) sauf si elles sont déconseillées par les règles d'un usage STUN spécifique. D'autres suites de chiffrement PEUVENT être mises en œuvre. Noter que les clients et serveurs STUN qui mettent en œuvre TLS version 1.3 [RFC8446] ou des versions suivantes sont aussi requis de mettre en œuvre les suites de chiffrement obligatoires de ces spécifications et DEVRAIENT désactiver l'usage des suites de chiffrement déconseillées quand ils détectent la prise en charge de ces spécifications. Les suites de chiffrement de secret parfait vers l'avant (PFS, *Perfect Forward Secrecy*) DOIVENT être préférées à des suites de chiffrement non PFS. Les suites de chiffrement qui ont des faiblesses connues, comme celles fondées sur un (seul) DES et RC4, NE DOIVENT PAS être utilisées. Les mises en œuvre DOIVENT désactiver la compression de niveau TLS.

Ces recommandations sont juste une partie des recommandations de la [RFC7525] que les mises en œuvre et déploiements d'un usage STUN avec TLS ou DTLS DOIVENT suivre.

Quand il reçoit le message Certificat TLS, le client DOIT vérifier le certificat et inspecter le site identifié par le certificat. Si le certificat est invalide ou révoqué, ou si il n'identifie pas la partie appropriée, le client NE DOIT PAS envoyer le message STUN ou le traiter autrement avec la transaction STUN. Le client DOIT vérifier l'identité du serveur. Pour ce faire, il suit les procédures d'identification définies dans la [RFC6125], avec un certificat contenant un identifiant de type DNS-ID ou CN-ID, facultativement avec un caractère générique comme étiquette de gauche, mais pas de type SRV-ID ou URI-ID.

Quand STUN fonctionne multiplexé avec d'autres protocoles sur une connexion TLS sur TCP ou une association DTLS sur UDP, les suites de chiffrement obligatoires et les procédures de traitement TLS opèrent comme défini par ces protocoles.

6.3 Réception d'un message STUN

Cette section spécifie le traitement d'un message STUN. Le traitement spécifié ici est pour les messages STUN tels que définis dans la présente spécification ; des règles supplémentaires pour la rétro compatibilité sont définies à la Section 11. Ces procédures supplémentaires sont facultatives, et les usages peuvent choisir de les utiliser. D'abord, un ensemble d'opérations est appliqué indépendamment de la classe. Cela est suivi par un traitement spécifique de la classe, décrit dans les sous paragraphes qui suivent.

Quand un agent STUN reçoit un message STUN, il vérifie d'abord que le message respecte les règles de la Section 5. Il vérifie que les deux premiers bits sont 0, que le champ Mouchard magique a la valeur correcte, que la longueur du message est raisonnable, et que la valeur de la méthode est prise en charge. Il vérifie que la classe de message est permise pour la méthode particulière. Si la classe de message est "réponse de succès" ou "réponse d'erreur", l'agent vérifie que l'identifiant de transaction correspond à une transaction qui est encore en cours. Si l'extension FINGERPRINT est utilisée, l'agent vérifie que l'attribut FINGERPRINT est présent et contient la valeur correcte. Si des erreurs sont détectées, le message est éliminé en silence. Quand STUN est multiplexé avec un autre protocole, une erreur peut indiquer que ce n'est pas réellement un message STUN ; dans ce cas, l'agent devrait essayer d'analyser le message comme étant d'un protocole différent.

L'agent STUN fait alors toutes les vérification exigées par un mécanisme d'authentification que l'usage a spécifié (voir la Section 9).

Une fois les vérifications d'authentification effectuées, l'agent STUN cherche des attributs inconnus et des attributs connus mais inattendus dans le message. Les attributs inconnus de compréhension facultative DOIVENT être ignorés par l'agent. Les attributs connus mais inattendus DEVRAIENT être ignorés par l'agent. Les attributs inconnus de compréhension obligatoire causent un traitement qui dépend de la classe de message, et qui est décrit ci-dessous.

À ce point, la suite du traitement dépend de la classe de message de la demande.

6.3.1 Traitement d'une demande

Si la demande contient un ou plusieurs attributs inconnus de compréhension exigée, le serveur réplique avec une réponse d'erreur et un code d'erreur de 420 (Attribut inconnu) et inclut un attribut UNKNOWN-ATTRIBUTES dans la réponse qui fait la liste des attributs inconnus de compréhension exigée.

Autrement, le serveur fait alors toutes les vérifications supplémentaires qu'exige la méthode ou l'usage spécifique. Si toutes les vérifications réussissent, le serveur formule une réponse de succès comme décrit ci-dessous.

Quand on fonctionne sur UDP ou DTLS sur UDP, une demande reçue par le serveur pourrait être la première demande d'une transaction ou pourrait être une retransmission. Le serveur DOIT répondre aux retransmissions de façon à ce que la propriété suivante soit préservée : si le client reçoit la réponse à la retransmission et pas la réponse qui était envoyée à la demande originale, l'état global sur le client et le serveur est identique au cas où seule la réponse à la retransmission originale est reçue ou où les deux réponses sont reçues (auquel cas le client va utiliser la première). La façon la plus simple de satisfaire cette exigence est que le serveur se souvienne de tous les identifiants de transaction reçus sur UDP ou DTLS sur UDP et de leurs réponses correspondantes dans les 40 dernières secondes. Cependant, cela exige que le serveur conserve l'état et c'est inapproprié pour toutes les demandes qui ne sont pas authentifiées. Une autre façon est de retraiter la demande et de recalculer la réponse. Cette dernière technique DOIT seulement être appliquée aux demandes qui sont idempotentes (une demande est considérée idempotente quand la même demande peut être répétée en toute sécurité sans impacter l'état global du système) et résulte en la même réponse de succès pour la même demande. La méthode Binding est considérée être idempotente. Noter qu'il y a certains rares événements de réseau qui pourraient causer le changement de la valeur d'adresse réflexive de transport, résultant en une adresse transposée différente dans les différentes réponses de succès. Des extensions à STUN DOIVENT discuter les implications des retransmissions de demandes sur les serveurs qui ne mémorisent pas l'état de transaction.

6.3.1.1 Formation d'une réponse de succès ou d'erreur

Quand il forme la réponse (de succès ou d'erreur) le serveur suit les règles de la Section 6. La méthode de la réponse est la même que dans la demande, et la classe du message est soit "Réponse de succès", soit "Réponse d'erreur".

Pour une réponse d'erreur, le serveur DOIT ajouter un attribut ERROR-CODE contenant le code d'erreur spécifié dans le traitement ci-dessus. La phrase de raison n'est pas fixée mais DEVRAIT être quelque chose de convenable pour le code d'erreur. Pour certaines erreurs, des attributs supplémentaires sont ajoutés au message. Ces attributs sont invoqués dans la description où le code d'erreur est spécifié. Par exemple, pour un code d'erreur de 420 (Attribut inconnu) le serveur DOIT inclure un attribut UNKNOWN-ATTRIBUTES. Certaines erreurs d'authentification causent aussi l'ajout d'attributs (voir la Section 9). Des extensions peuvent définir d'autres erreurs et/ou attributs supplémentaires ajoutés aux cas d'erreur.

Si le serveur a authentifié la demande en utilisant un mécanisme d'authentification, le serveur DEVRAIT alors ajouter les attributs d'authentification appropriés à la réponse (voir la Section 9).

Le serveur ajoute aussi tous les attributs requis par la méthode ou usage spécifique. De plus, le serveur DEVRAIT ajouter un attribut SOFTWARE au message.

Pour la méthode Binding, aucune vérification supplémentaire n'est requise sauf si l'usage le spécifie autrement. Quand il forme la réponse de succès, le serveur ajoute un attribut XOR-MAPPED-ADDRESS à la réponse ; cet attribut contient l'adresse de transport de source du message de demande. Pour UDP ou DTLS sur UDP, c'est l'adresse IP de source et l'accès UDP de source du message de demande. Pour TCP et TLS sur TCP, c'est l'adresse IP de source et l'accès TCP de source de la connexion TCP telle que vue par le serveur.

6.3.1.2 Envoi d'une réponse de succès ou d'erreur

La réponse (succès ou erreur) est envoyée sur le même transport que celui sur lequel la demande a été reçue. Si la demande a été reçue sur UDP ou DTLS sur UDP, l'adresse de destination et l'accès IP de la réponse sont l'adresse et accès IP de

source du message de demande reçu, et l'adresse et accès IP de source de la réponse sont égaux à l'adresse et accès IP de destination du message de demande reçu. Si la demande était reçue sur TCP ou TLS sur TCP, la réponse est renvoyée sur la même connexion TCP que celle sur laquelle la demande a été reçue.

Il est permis au serveur d'envoyer des réponses dans un ordre différent de celui des demandes reçues.

6.3.2 Traitement d'une indication

Si l'indication contient des attributs inconnus de compréhension exigée, l'indication est éliminée et le traitement cesse.

Autrement, l'agent fait alors toutes les vérifications supplémentaires que la méthode ou l'usage spécifique exige. Si toutes les vérifications ont réussi, l'agent traite alors l'indication. Aucune réponse n'est générée pour une indication.

Pour la méthode Binding, aucune vérification ni traitement supplémentaire n'est requise, sauf si l'usage le spécifie autrement. La simple réception du message par l'agent a rafraîchi les liens chez les NAT intermédiaires.

Comme les indications ne sont pas retransmises sur UDP ou DTLS sur UDP (à la différence des demandes) il n'est pas besoin de traiter les retransmissions des indications chez l'agent envoyeur.

6.3.3 Traitement d'une réponse de succès

Si la réponse de succès contient des attributs inconnus de compréhension exigée, la réponse est éliminée et la transaction est considérée comme ayant échoué.

Autrement, le client fait alors toutes les vérifications supplémentaires exigées par la méthode ou usage spécifique. Si toutes les vérifications réussissent, le client traite alors la réponse de succès.

Pour la méthode Binding, le client vérifie que l'attribut XOR-MAPPED-ADDRESS est présent dans la réponse. Le client vérifie la famille d'adresses spécifiée. Si c'est une famille d'adresses non prise en charge, l'attribut DEVRAIT être ignoré. Si c'est une famille d'adresses non attendue mais prise en charge (par exemple, la transaction Binding était envoyée sur IPv4, mais la famille d'adresses spécifiée est IPv6) alors le client PEUT accepter et utiliser la valeur.

6.3.4 Traitement d'une réponse d'erreur

Si la réponse d'erreur contient des attributs inconnus de compréhension exigée, ou si la réponse d'erreur ne contient pas d'attribut ERROR-CODE, alors la transaction est simplement considérée avoir échoué.

Autrement, le client fait alors tout le traitement spécifié par le mécanisme d'authentification (voir la Section 9). Il peut en résulter une nouvelle tentative de transaction.

Le traitement à ce point dépend du code d'erreur, de la méthode, et de l'usage ; les règles par défaut sont les suivantes :

- Si le code d'erreur est de 300 à 399, le client DEVRAIT considérer que la transaction a échoué sauf si l'extension ALTERNATE-SERVER (Section 10) est utilisée.
- Si le code d'erreur est de 400 à 499, le client déclare que la transaction a échoué ; dans le cas de 420 (Attribut inconnu) la réponse devrait contenir un attribut UNKNOWN-ATTRIBUTES qui donne des informations supplémentaires.
- Si le code d'erreur est de 500 à 599, le client PEUT renvoyer la demande ; les clients qui le font DOIVENT limiter le nombre de fois qu'ils le font. Sauf si un code d'erreur spécifie une valeur différente, le nombre de retransmissions DEVRAIT être limité à 4.

Tout autre code d'erreur fait que le client considère que la transaction a échoué.

7. Mécanisme FINGERPRINT

Cette section décrit un mécanisme facultatif pour STUN qui aide à distinguer les messages STUN des paquets d'autres protocoles quand les deux sont multiplexés sur la même adresse de transport. Ce mécanisme est facultatif, et un usage STUN doit décrire si et quand il est utilisé. Le mécanisme FINGERPRINT n'est pas rétro compatible avec la RFC 3489 et ne peut pas être utilisé dans des environnements où une telle compatibilité est requise.

Dans certains usages, les messages STUN sont multiplexés sur la même adresse de transport que d'autres protocoles, comme le protocole de transport en temps réel (RTP, *Real-Time Transport Protocol*). Afin d'appliquer le traitement décrit à la Section 6, les messages STUN doivent d'abord être séparés des paquets de l'application.

La Section 5 décrit trois champs fixes dans l'en-tête STUN qui peuvent être utilisés à cette fin. Cependant, dans certains cas, ces trois champs fixes peuvent n"être pas suffisants.

Quand l'extension FINGERPRINT est utilisée, un agent inclut l'attribut FINGERPRINT dans les messages qu'il envoie à un autre agent. Le paragraphe 14.7 décrit le placement et la valeur de cet attribut.

Quand l'agent reçoit ce qu'il croit être un message STUN, alors, en plus des autres vérifications de base, il vérifie aussi que le message contient un attribut FINGERPRINT et que l'attribut contient la valeur correcte. Le paragraphe 6.3 décrit quand dans le traitement global d'un message STUN est effectuée la vérification de FINGERPRINT. Cette vérification supplémentaire aide l'agent à détecter les messages d'autres protocoles qui pourraient autrement sembler être des messages STUN.

8. Découverte DNS d'un serveur

Cette section décrit une procédure facultative pour STUN qui permet à un client d'utiliser le DNS pour déterminer l'adresse et l'accès IP d'un serveur. Un usage STUN doit décrire si et quand cette extension est utilisée. Pour utiliser cette procédure, le client doit connaître un URI STUN [RFC7064] ; l'usage doit aussi décrire comment le client obtient cet URI. L'incorporation d'un URI STUN dans le logiciel N'EST PAS RECOMMANDÉE au cas où le nom de domaine serait perdu ou devrait changer pour des raisons légales ou autres.

Quand un client souhaite localiser un serveur STUN sur l'Internet public qui accepte les transactions de demande/réponse Binding, le schéma d'URI STUN est "stun". Quand il souhaite localiser un serveur STUN qui accepte les transactions de demande/réponse Binding sur une session TLS ou DTLS, le schéma d'URI est "stuns".

La syntaxe des URI "stun" et "stuns" est définie au paragraphe 3.1 de la [RFC7064]. Les usages STUN PEUVENT définir des schémas d'URI supplémentaires.

8.1 Sémantique du schéma d'URI STUN

Si la partie <hôte> d'un URI "stun" contient une adresse IP, alors cette adresse IP est utilisée directement pour contacter le serveur. Un URI "stuns" qui contient une adresse IP DOIT être rejeté. Une future extension ou usage de STUN peut assouplir cette exigence, pourvu qu'elle montre comment authentifier le serveur STUN et empêcher des attaques par interposition.

Si l'URI ne contient pas d'adresse IP, le nom de domaine contenu dans la partie <hôte> est résolue en adresse de transport en utilisant les procédures SRV spécifiées dans la [RFC2782]. Le nom de service SRV du DNS est le contenu de la partie <scheme>. Le protocole dans la recherche de SRV est le protocole de transport sur lequel le client va faire fonctionner STUN : "udp" pour UDP et "tcp" pour TCP.

Les procédures de la RFC 2782 sont suivies pour déterminer le serveur à contacter. La RFC 2782 décrit les détails de la façon dont un ensemble d'enregistrements SRV est trié puis essayé. Cependant, la RFC 2782 déclare seulement que le client devrait "essayer de se connecter au (protocole, adresse, service)" sans donner de détails sur ce qui arrive en cas d'échec. Quand on suit ces procédures, si la transaction STUN arrive à expiration sans recevoir de réponse, le client DEVRAIT ressayer la demande au prochain serveur dans l'ordre défini par la RFC 2782. Un tel nouvel essai n'est possible que pour des transmissions de demande/réponse, car les transactions d'indication ne génèrent ni réponse ni fin de temporisation.

De plus, au lieu d'interroger les enregistrements de ressource A ou AAAA sur un nom de domaine, un client de double pile IPv4/IPv6 DOIT interroger et essayer les demandes sur toutes les adresses IP reçues, comme spécifié dans la [RFC8305].

L'accès par défaut pour les demandes STUN est 3478, pour TCP et UDP. L'accès par défaut pour les demandes STUN sur TLS et STUN sur DTLS est 5349. Les serveurs peuvent faire fonctionner STUN sur DTLS sur le même accès que STUN sur UDP si le logiciel de serveur prend en charge de déterminer si le message initial est un message DTLS ou STUN. Les

serveurs peuvent faire fonctionner STUN sur TLS sur le même accès que STUN sur TCP si le logiciel de serveur prend en charge de déterminer si le message initial est un message TLS ou STUN.

Les administrateurs de serveurs STUN DEVRAIENT utiliser ces accès dans leurs enregistrements SRV pour UDP et TCP. Dans tous les cas, l'accès dans le DNS DOIT refléter celui sur lequel le serveur écoute.

Si aucun enregistrement SRV n'est trouvé, le client effectue une recherche d'enregistrement A et AAAA sur le nom de domaine, comme décrit dans la [RFC8305]. Le résultat va être une liste d'adresses IP, dont chacune peut être contactée simultanément à l'accès par défaut en utilisant UDP ou TCP, indépendamment de l'usage STUN. Pour les usages qui exigent TLS, le client se connecte aux adresses IP en utilisant l'accès par défaut de STUN sur TLS. Pour les usages qui exigent DTLS, le client se connecte aux adresses IP en utilisant l'accès par défaut de STUN sur DTLS.

9. Mécanismes d'authentification et d'intégrité de message

Cette Section définit deux mécanismes pour STUN qu'un client et un serveur peuvent utiliser pour assurer l'authentification et l'intégrité de message ; ces deux mécanismes sont appelés le mécanisme d'accréditif à court terme et le mécanisme d'accréditif à long terme. Ces deux mécanismes sont facultatifs, et chaque usage doit spécifier si et quand ils sont utilisés. Par conséquent, les clients et les serveurs vont savoir quel mécanisme (si il en est) suivre sur la base de leur connaissance de l'usage qui s'applique. Par exemple, un serveur STUN sur l'Internet public qui prend en charge ICE ne va pas avoir d'authentification, tandis que la fonction de serveur STUN dans un agent qui prend en charge les vérifications de connexité va utiliser des accréditifs à court terme. Une vue d'ensemble de ces deux mécanismes est donnée à la Section 2.

Chaque mécanisme spécifie le traitement supplémentaire exigé pour utiliser ce mécanisme, étendant le traitement spécifié à la Section 6. Le traitement supplémentaire se produit en trois endroits différents : à la formation d'un message, à la réception d'un message immédiatement après avoir effectué les vérifications de base, et lors du traitement détaillé des réponses d'erreur.

Noter que les agents DOIVENT ignorer tous les attributs qui suivent MESSAGE-INTEGRITY, à l'exception des attributs MESSAGE-INTEGRITY-SHA256 et FINGERPRINT. De même, les agents DOIVENT ignorer tous les attributs qui suivent l'attribut MESSAGE-INTEGRITY-SHA256 si l'attribut MESSAGE-INTEGRITY n'est pas présent, à l'exception de l'attribut FINGERPRINT.

9.1 Mécanisme d'intégrité à court terme

Le mécanisme d'accréditif à court terme suppose que, avant la transaction STUN, le client et le serveur ont utilisé un autre protocole pour échanger un accréditif sous la forme d'un nom d'utilisateur et mot de passe. Cet accréditif est limité dans le temps. La limite de temps est définie par l'usage. Par exemple, dans l'usage ICE [RFC8445], les deux points d'extrémité utilisent la signalisation hors bande pour s'accorder sur un nom d'utilisateur et un mot de passe, et ce nom d'utilisateur et mot de passe sont applicables pour la durée de la session de supports.

Cet accréditif est utilisé pour former une vérification d'intégrité de message dans chaque demande et dans de nombreuses réponses. Il n'y a pas de défi et réponse comme dans le mécanisme de long terme ; par conséquent, la répétition est limitée par la vertu de la nature limitée dans le temps de l'accréditif.

9.1.1 Clé HMAC

Pour les accrédifs à court terme , la clé de code d'authentification de message fondée sur le hachage (HMAC, *Hash-Based Message Authentication Code*) est définie comme suit :

clé = OpaqueString(mot de passe)

où le profil OpaqueString est défini dans la [RFC8265]. Le codage utilisé est UTF-8 [RFC3629].

9.1.2 Formation d'une demande ou indication

Pour un message de demande ou d'indication, l'agent DOIT inclure les attributs USERNAME, MESSAGE-INTEGRITY-SHA256, et MESSAGE-INTEGRITY dans le message sauf si l'agent sait d'un mécanisme externe que l'algorithme

d'intégrité de message est pris en charge par les deux agents. Dans ce cas, MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 DOIT être inclus en plus de USERNAME. Le HMAC pour l'attribut MESSAGE-INTEGRITY est calculé comme décrit au paragraphe 14.5, et le HMAC pour les attributs MESSAGE-INTEGRITY-SHA256 est calculé comme décrit au paragraphe 14.6. Noter que le mot de passe n'est jamais inclus dans la demande ou indication.

9.1.3 Réception d'une demande ou indication

Après que l'agent a fait le traitement de base d'un message, il effectue les vérifications énumérées ci-dessous dans l'ordre spécifié :

- Si le message ne contient pas 1) un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 et 2) un attribut USERNAME :
  - Si le message est une demande, le serveur DOIT rejeter la demande avec une réponse d'erreur. Cette réponse DOIT utiliser un code d'erreur de 400 (Mauvaise demande).
  - Si le message est une indication, l'agent DOIT éliminer en silence l'indication.
- Si le USERNAME ne contient pas une valeur de nom d'utilisateur actuellement valide pour le serveur :
  - Si le message est une demande, le serveur DOIT rejeter la demande avec une réponse d'erreur. Cette réponse DOIT utiliser un code d'erreur de 401 (Non authentifiée).
  - Si le message est une indication, l'agent DOIT éliminer en silence l'indication.
- Si l'attribut MESSAGE-INTEGRITY-SHA256 est présent, calculer la valeur pour l'intégrité de message comme décrit au paragraphe 14.6, en utilisant le mot de passe associé au nom d'utilisateur. Si l'attribut MESSAGE-INTEGRITY-SHA256 n'est pas présent, utiliser alors le même mot de passe pour calculer la valeur pour l'intégrité de message comme décrit au paragraphe 14.5. Si la valeur résultante ne correspond pas au contenu de l'attribut correspondant (MESSAGE-INTEGRITY-SHA256 ou MESSAGE-INTEGRITY) :
  - Si le message est une demande, le serveur DOIT rejeter la demande avec une réponse d'erreur. Cette réponse DOIT utiliser un code d'erreur de 401 (Non authentifiée).
  - Si le message est une indication, l'agent DOIT éliminer en silence l'indication.

Si ces vérifications réussissent, l'agent continue le traitement de la demande ou indication. Toute réponse générée par un serveur à une demande qui contient un attribut MESSAGE-INTEGRITY-SHA256 DOIT inclure l'attribut MESSAGE-INTEGRITY-SHA256, calculé en utilisant le mot de passe utilisé pour authentifier la demande. Toute réponse générée par un serveur sur une demande qui contient seulement un attribut MESSAGE-INTEGRITY DOIT inclure l'attribut MESSAGE-INTEGRITY, calculé en utilisant le mot de passe utilisé pour authentifier la demande. Cela signifie que seulement un de ces attributs peut apparaître dans une réponse. La réponse NE DOIT PAS contenir l'attribut USERNAME.

Si une des vérifications échoue, un serveur NE DOIT PAS inclure d'attribut MESSAGE-INTEGRITY-SHA256, MESSAGE-INTEGRITY, ou USERNAME dans la réponse d'erreur. C'est parce que, dans ces cas d'échec, le serveur ne peut pas déterminer le secret partagé nécessaire pour calculer les attributs MESSAGE-INTEGRITY-SHA256 ou MESSAGE-INTEGRITY.

9.1.4 Réception d'une réponse

Le client cherche l'attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 dans la réponse. Si il est présent et si le client a seulement envoyé un des attributs MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 dans la demande (à cause de l'indication externe du paragraphe 9.1.2 ou parce que c'est une demande suivante comme défini au paragraphe 9.1.5) l'algorithme dans la réponse doit correspondre ; autrement, la réponse DOIT être éliminée.

Le client calcule alors l'intégrité du message sur la réponse comme défini au paragraphe 14.5 pour l'attribut MESSAGE-INTEGRITY ou au paragraphe 14.6 pour l'attribut MESSAGE-INTEGRITY-SHA256, en utilisant le même mot de passe qu'utilisé pour la demande. Si la valeur résultante correspond au contenu de l'attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, respectivement, la réponse est considérée authentifiée. Si la valeur ne correspond pas, ou si MESSAGE-INTEGRITY et MESSAGE-INTEGRITY-SHA256 sont tous deux absents, le traitement dépend de si la demande était envoyée sur un transport fiable ou non fiable.

Si la demande était envoyé sur un transport non fiable, la réponse DOIT être éliminée, comme si elle n'avait jamais été reçue. Cela signifie que les retransmissions, si applicables, vont continuer. Si toutes les réponses reçues sont éliminées, alors au lieu de signaler une fin de temporisation après la fin de la transaction, la couche DOIT signaler que la protection d'intégrité était violée.

Si la demande était envoyée sur un transport fiable, la réponse DOIT être éliminée, et la couche DOIT immédiatement terminer la transaction et signaler que la protection d'intégrité était violée.

9.1.5 Envoi des demandes suivantes

Un client qui envoie des demandes suivantes au même serveur DOIT envoyer seulement l'attribut MESSAGE-INTEGRITY-SHA256 ou MESSAGE-INTEGRITY qui correspond à l'attribut qui a été reçu dans la réponse à la demande initiale. Ici, "même serveur" signifie la même adresse et numéro d'accès IP, pas juste le même URI ou résultat de recherche de SRV.

9.2 Mécanisme d'accréditifs à long terme

Le mécanisme d'accréditifs à long terme s'appuie sur un accréditif à long terme, sous la forme d'un nom d'utilisateur et d'un mot de passe partagés entre client et serveur. L'accréditif est considéré comme à long terme car il est supposé qu'il est provisionné pour un utilisateur et reste en vigueur jusqu'à ce que l'utilisateur ne soit plus un abonné du système ou jusqu'à ce qu'il soit changé. C'est fondamentalement un traditionnel nom d'utilisateur et mot de passe de connexion donné aux utilisateurs.

Parce que ces noms d'utilisateur et mots de passe sont supposés être valides pour des durées étendues, la prévention de répétition est fournie sous la forme d'un défi de résumé. Dans ce mécanisme, le client envoie initialement une demande, sans offrir d'accréditifs ou de vérification d'intégrité. Le serveur rejette cette demande, en fournissant à l'utilisateur un domaine (utilisé pour guider l'utilisateur ou agent dans le choix d'un nom d'utilisateur et mot de passe) et un nom occasionnel. Le nom occasionnel donne une protection limitée contre la répétition. C'est un mouchard, choisi par le serveur et codé de façon à indiquer une durée de validité ou une identité de client à partir de laquelle il est valide. Seul le serveur a besoin de connaître la structure interne du mouchard. Le client réessaye la demande, cette fois en incluant son nom d'utilisateur et le domaine et en faisant écho au nom occasionnel fourni par le serveur. Le client inclut aussi un des attributs d'intégrité de message définis dans ce document, qui fournit un HMAC sur la demande entière, incluant le nom occasionnel. Le serveur valide le nom occasionnel et vérifie l'intégrité du message. Si il correspond, la demande est authentifiée. Si le nom occasionnel n'est plus valide, il est considéré "périmé", et le serveur rejette la demande, fournissant un nouveau nom occasionnel.

Dans les demandes suivantes au même serveur, le client réutilise le nom occasionnel, le nom d'utilisateur, le domaine, et le mot de passe utilisés précédemment. De cette façon, les demandes suivantes ne sont pas rejetées tant que le nom occasionnel n'est pas invalidé par le serveur, dans ce cas le rejet fournit un nouveau nom occasionnel au client.

Noter que le mécanisme d'accréditifs à long terme ne peut pas être utilisé pour protéger les indications, car les indications ne peuvent pas être mises au défi. Les usages qui utilisent des indications doivent soit utiliser un accréditif à court terme, soit omettre l'authentification et l'intégrité de message pour elles.

Pour indiquer qu'il prend en charge la présente spécification, un serveur DOIT ajouter devant la valeur d'attribut NONCE avec les chaînes de caractères composées de "obMatJos2" enchaînées avec le codage (4 caractères) en base64 [RFC4648] des 24 bits de caractéristiques de sécurité STUN comme défini au paragraphe 18.1. L'ensemble de 24 bits de caractéristiques de sécurité est codé comme trois octets, avec un bit 0 comme bit de poids fort du premier octet et le bit 23 comme bit de moindre poids du troisième octet. Si aucune caractéristique de sécurité n'est utilisée, alors un dispositif d'octets avec tous les 24 bits réglés à zéro DOIT être codé à la place. Pour le reste de ce document, le terme de "mouchard de nom occasionnel" se réfère à la chaîne complète de 13 caractères ajoutée devant la valeur de l'attribut NONCE.

Comme le mécanisme d'accréditifs à long terme est susceptible d'attaques de dictionnaire hors ligne, les déploiements DEVRAIENT utiliser des mots de passe difficiles à deviner. Dans le cas où les accréditifs ne sont pas entrés par l'utilisateur, mais sont plutôt placés sur un appareil client durant le provisionnement de l'appareil, le mot de passe DEVRAIT avoir au moins 128 bits d'aléa. Dans le cas où les accréditifs sont entrés par l'utilisateur, ils devraient suivre les meilleures pratiques courantes en matière de structure de mot de passe.

9.2.1 Prévention d'attaque en dégradation

Le présent document introduit deux nouvelles caractéristiques de sécurité qui assurent la capacité de choisir l'algorithme utilisé pour la protection du mot de passe ainsi que la capacité d'utiliser un nom d'utilisateur anonyme. Ces ceux capacités sont facultatives afin de rester rétro compatibles avec les précédentes versions du protocole STUN.

Ces nouvelles capacités sont sujettes à des attaques en dégradation par lesquelles un attaquant sur le chemin du message peut supprimer ces capacités et forcer à utiliser des propriétés de sécurité plus faibles. Pour empêcher ces sortes d'attaques de rester non détectées, le nom occasionnel est amélioré avec des informations supplémentaires.

La valeur du "mouchard de nom occasionnel" va varier sur la base des bits spécifiques de caractéristique de sécurité STUN choisis. Quand le présent document fait référence au "mouchard de nom occasionnel" dans un paragraphe discutant d'une caractéristique de sécurité STUN spécifique, il est compris que le bit Caractéristique de sécurité STUN correspondant dans le "mouchard de nom occasionnel" est réglé à 1.

Par exemple, quand la caractéristique de sécurité PASSWORD-ALGORITHMS (définie au paragraphe 9.2.4) est utilisée, le bit correspondant "Algorithmes de mot de passe" (défini au paragraphe 18.1) est réglé à 1 dans le "mouchard de nom occasionnel".

9.2.2 Clé HMAC

Pour les accréditifs à long terme qui n'utilisent pas un algorithme différent, comme spécifié par l'attribut PASSWORD-ALGORITHM, la clé est de 16 octets :

clé = MD5(nom d'utilisateur ":" OpaqueString(realm) ":" OpaqueString(mot de passe))

Où MD5 est défini dans les [RFC1321] et [RFC6151], et le profil OpaqueString est défini dans la [RFC8265]. Le codage utilisé est UTF-8 [RFC3629].

La clé de 16 octets est formée en prenant le hachage MD5 du résultat de l'enchaînement des cinq champs suivants : (1) le nom d'utilisateur, en supprimant tous les guillemets et nuls en queue, tel que pris dans l'attribut USERNAME (et dans ce cas OpaqueString a déjà été appliqué) ; (2) un seul caractère deux-points ; (3) le domaine, en supprimant tous les guillemets et nuls en queue et après traitement en utilisant OpaqueString ; (4) un seul caractère deux-points ; et (5) le mot de passe, en supprimant tous les nuls en queue et après traitement en utilisant OpaqueString. Par exemple, si le nom d'utilisateur est "user", le domaine est "realm", et le mot de passe est "pass", alors la clé HMAC de 16 octets serait le résultat de l'exécution d'un hachage MD5 sur la chaîne "user:realm:pass", le hachage résultant étant 0x8493fbc53ba582fb4c044c456bdc40eb.

La structure de la clé quand elle est utilisée avec des accréditifs à long terme facilite le déploiement dans des systèmes qui utilisent aussi SIP [RFC3261]. Normalement, les systèmes SIP qui utilisent le mécanisme d'authentification par résumé de SIP ne mémorisent en fait pas le mot de passe dans la base de données. Ils mémorisent plutôt une valeur appelée "H(A1)", qui est égale à la clé définie ci-dessus. Par exemple, ce mécanisme peut être utilisé avec les extensions d'authentification définies dans la [RFC5090]. Quand un PASSWORD-ALGORITHM est utilisé, la longueur de clé et l'algorithme à utiliser sont décrits au paragraphe 18.5.1.

9.2.3 Formation d'une demande

La première demande du client au serveur (comme identifiée par le nom d'hôte si les procédures du DNS de la Section 8 sont utilisées, et par l'adresse IP sinon) est traitée en accord avec les règles du paragraphe 9.2.3.1. Quand le client initie une demande suivante une fois qu'une précédente transaction de demande/réponse s'est achevée avec succès, il suit les règles du paragraphe 9.2.3.2. Former une demande par suite d'une réponse d'erreur 401 (Non authentifié) ou 438 (Nom occasionnel périmé) est traité au paragraphe 9.2.5 et n'est pas considéré être une "demande suivante" et donc n'utilise pas les règles décrites au paragraphe 9.2.3.2. Chacun de ces types de demandes a des attributs obligatoires différents.

9.2.3.1 Première demande

Si le client n'a pas achevé une transaction de demande/réponse réussie avec le serveur, il DOIT omettre les attributs USERNAME, USERHASH, MESSAGE-INTEGRITY, MESSAGE-INTEGRITY-SHA256, REALM, NONCE, PASSWORD-ALGORITHMS, et PASSWORD-ALGORITHM. En d'autres termes, la première demande est envoyée comme si il n'y avait pas d'authentification ou d'intégrité de message appliquée.

9.2.3.2 Demandes suivantes

Une fois qu'une transaction de demande/réponse est achevée, le serveur aura présenté un domaine et un nom occasionnel au client et celui-ci aura choisi un nom d'utilisateur et le mot de passe avec lesquels s'authentifier. Le client DEVRAIT mettre en antémémoire le nom d'utilisateur, le mot de passe, le domaine, et le nom occasionnel pour la suite de la communication avec le serveur. Quand le client envoie une demande suivante, il DOIT inclure soit l'attribut USERNAME

soit les attributs USERHASH, REALM, NONCE, et PASSWORD-ALGORITHM avec ces valeurs mises en antémémoire. Il DOIT inclure un attribut MESSAGE-INTEGRITY ou un attribut MESSAGE-INTEGRITY-SHA256, calculé comme décrit aux paragraphes 14.5 et 14.6 en utilisant le mot de passe mis en antémémoire. Le choix entre les deux attributs dépend de l'attribut reçu dans la réponse à la première demande.

9.2.4 Réception d'une demande

Après que le serveur a fait le traitement de base d'une demande, il effectue les vérifications mentionnées ci-dessous dans l'ordre spécifié. Noter qu'il est RECOMMANDÉ que la valeur de REALM soit le nom de domaine du fournisseur du serveur STUN :

- Si le message ne contient pas un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, le serveur DOIT générer une réponse d'erreur avec un code d'erreur de 401 (Non authentifié). Cette réponse DOIT inclure une valeur de REALM. La réponse DOIT inclure un NONCE, choisi par le serveur. Le serveur NE DOIT PAS choisir le même NONCE pour deux demandes sauf si elles ont la même adresse et accès IP de source. Le serveur PEUT prendre en charge des algorithmes de mot de passe de remplacement, et dans ce cas, il peut en faire la liste en ordre de préférence dans un attribut PASSWORD-ALGORITHMS. Si le serveur ajoute un attribut PASSWORD-ALGORITHMS, il DOIT régler le bit de caractéristique de sécurité STUN "Algorithmes de mot de passe" à 1. Le serveur PEUT prendre en charge un nom d'utilisateur anonyme, et dans ce cas, il DOIT régler le bit de caractéristique de sécurité STUN "Anonymat du nom d'utilisateur" à 1. La réponse NE DEVRAIT PAS contenir d'attribut USERNAME, USERHASH, MESSAGE-INTEGRITY, ou MESSAGE-INTEGRITY-SHA256.

Note : réutiliser un NONCE pour des adresses ou accès de source IP différents n'était pas explicitement interdit dans la [RFC5389].

- Si le message contient un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, mais que manque l'attribut USERNAME ou USERHASH, REALM, ou NONCE, le serveur DOIT générer une réponse d'erreur avec un code d'erreur de 400 (Mauvaise demande). Cette réponse NE DEVRAIT PAS inclure d'attribut USERNAME, USERHASH, NONCE, ou REALM. La réponse ne peut pas contenir un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, car les attributs requis pour les générer sont manquants.

- Si l'attribut NONCE commence par le "mouchard de nom occasionnel" avec le bit de caractéristiques de sécurité STUN "Algorithmes de mot de passe" réglé à 1, le serveur effectue ces vérifications dans l'ordre spécifié :
  * Si la demande ne contient ni l'algorithme PASSWORD-ALGORITHMS ni l'algorithme PASSWORD-ALGORITHM, alors la demande est traitée comme si PASSWORD-ALGORITHM était MD5 ;
  * autrement, sauf si (1) PASSWORD-ALGORITHM et PASSWORD-ALGORITHMS sont tous deux présents, (2) PASSWORD-ALGORITHMS correspond à la valeur envoyée dans la réponse qui a envoyé ce NONCE, et (3) PASSWORD-ALGORITHM correspond à une des entrées dans PASSWORD-ALGORITHMS, le serveur DOIT générer une réponse d'erreur avec un code d'erreur de 400 (Mauvaise demande).

- Si la valeur de l'attribut USERNAME ou USERHASH n'est pas valide, le serveur DOIT générer une réponse d'erreur avec un code d'erreur de 401 (Non authentifié). Cette réponse DOIT inclure une valeur de REALM. La réponse DOIT inclure un NONCE, choisi par le serveur. La réponse DOIT inclure un attribut PASSWORD-ALGORITHMS. La réponse NE DEVRAIT PAS contenir d'attribut USERNAME ou USERHASH. La réponse PEUT inclure un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, en utilisant la clé précédente pour le calculer.

- Si l'attribut MESSAGE-INTEGRITY-SHA256 est présent, calculer la valeur pour l'intégrité de message comme décrit au paragraphe 14.6, en utilisant le mot de passe associé au nom d'utilisateur. Autrement, en utilisant le même mot de passe, calculer la valeur pour l'attribut MESSAGE-INTEGRITY comme décrit au paragraphe 14.5. Si la valeur résultante ne correspond pas au contenu de l'attribut MESSAGE-INTEGRITY ou de l'attribut MESSAGE-INTEGRITY-SHA256, le serveur DOIT rejeter la demande avec une réponse d'erreur. Cette réponse DOIT utiliser un code d'erreur de 401 (Non authentifié). Il DOIT inclure les attributs REALM et NONCE et NE DEVRAIT PAS inclure l'attribut USERNAME, USERHASH, MESSAGE-INTEGRITY, ou MESSAGE-INTEGRITY-SHA256.

- Si le NONCE n'est plus valide, le serveur DOIT générer une réponse d'erreur avec un code d'erreur de 438 (Nom occasionnel périmé). Cette réponse DOIT inclure les attributs NONCE, REALM, et PASSWORD-ALGORITHMS et NE DEVRAIT PAS inclure d'attributs USERNAME et USERHASH. La valeur de l'attribut NONCE DOIT être valide. La réponse PEUT inclure un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, en utilisant le nom occasionnel précédent pour le calculer. Les serveurs peuvent révoquer les noms occasionnels afin de fournir une sécurité supplémentaire. Voir les lignes directrices au paragraphe 5.4 de la [RFC7616].

Si ces vérifications réussissent, le serveur continue de traiter la demande. Toute réponse générée par le serveur DOIT inclure l'attribut MESSAGE-INTEGRITY-SHA256, calculé en utilisant le nom d'utilisateur et le mot de passe utilisés pour authentifier la demande, sauf si la demande était traitée comme si PASSWORD-ALGORITHM était MD5 (parce que la demande ne contenait ni PASSWORD-ALGORITHMS ni PASSWORD-ALGORITHM). Dans ce cas, l'attribut MESSAGE-INTEGRITY DOIT être utilisé à la place de l'attribut MESSAGE-INTEGRITY-SHA256, et les attributs REALM, NONCE, USERNAME, et USERHASH NE DEVRAIENT PAS être inclus.

9.2.5 Réception d'une réponse

Si la réponse est une réponse d'erreur avec un code d'erreur de 401 (Non authentifié) ou 438 (Nom occasionnel périmé) le client DOIT vérifier si la valeur de l'attribut NONCE commence par le "mouchard de nom occasionnel". Si il en est ainsi et si le "mouchard de nom occasionnel" a le bit de caractéristique de sécurité STUN "Algorithmes de mot de passe" réglé à 1 mais si aucun attribut PASSWORD-ALGORITHMS n'est présent, alors le client NE DOIT PAS réessayer la demande avec une nouvelle transaction.

Si la réponse est une réponse d'erreur avec un code d'erreur de 401 (Non authentifié) le client DEVRAIT réessayer la demande avec une nouvelle transaction. Cette demande DOIT contenir un USERNAME ou un USERHASH, déterminé par le client comme le nom d'utilisateur approprié pour le REALM provenant de la réponse d'erreur. Si le "mouchard de nom occasionnel" est présent et a le bit de caractéristique de sécurité STUN "Nom d'utilisateur anonyme" réglé à 1, alors l'attribut USERHASH DOIT être utilisé ; autrement, l'attribut USERNAME DOIT être utilisé. La demande DOIT contenir le REALM, copié de la réponse d'erreur. La demande DOIT contenir le NONCE, copié de la réponse d'erreur. Si la réponse contient un attribut PASSWORD-ALGORITHMS, la demande DOIT contenir l'attribut PASSWORD-ALGORITHMS avec le même contenu. Si la réponse contient un attribut PASSWORD-ALGORITHMS, et si cet attribut contient au moins un algorithme pris en charge par le client, alors la demande DOIT contenir un attribut PASSWORD-ALGORITHM avec le premier algorithme pris en charge de la liste. Si la réponse contient un attribut PASSWORD-ALGORITHMS, et si cet attribut ne contient aucun algorithme pris en charge par le client, le client NE DOIT PAS réessayer la demande avec une nouvelle transaction. Le client NE DOIT PAS effectuer ce nouvel essai si il n'a pas changé le USERNAME, USERHASH, REALM, ou son mot de passe associé de la tentative précédente.

Si la réponse est une réponse d'erreur avec un code d'erreur de 438 (Nom occasionnel périmé) le client DOIT réessayer la demande, en utilisant le nouvel attribut NONCE fourni dans la réponse 438 (Nom occasionnel périmé). Ce nouvel essai DOIT aussi inclure le USERNAME ou USERHASH, le REALM, et soit l'attribut MESSAGE-INTEGRITY, soit l'attribut MESSAGE-INTEGRITY-SHA256.

Pour toutes les autres réponses, si l'attribut NONCE commence par le "mouchard de nom occasionnel" avec le bit de caractéristique de sécurité STUN "Algorithmes de mot de passe" réglé à 1 mais si PASSWORD-ALGORITHMS n'est pas présent, la réponse DOIT être ignorée.

Si la réponse est une réponse d'erreur avec un code d'erreur de 400 (Mauvaise demande) et ne contient d'attribut ni MESSAGE-INTEGRITY ni MESSAGE-INTEGRITY-SHA256, alors la réponse DOIT être éliminée, comme si elle n'avait jamais été reçue. Cela signifie que les retransmissions, si applicables, vont continuer.

Note : dans ce cas, la réponse 400 ne va jamais atteindre l'application, résultant en un fin de temporisation.

Le client cherche l'attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 dans la réponse (de succès ou d'échec). Si il est présent, le client calcule l'intégrité du message sur la réponse comme défini aux paragraphes 14.5 ou 14.6, en utilisant le même mot de passe qu'utilisé pour la demande. Si la valeur résultante correspond au contenu de l'attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, la réponse est considérée authentifiée. Si la valeur ne correspond pas ou si MESSAGE-INTEGRITY et MESSAGE-INTEGRITY-SHA256 sont tous deux absents, le traitement dépend de si la demande a été envoyée sur un transporta fiable ou non fiable.

Si la demande était envoyée sur un transport non fiable, la réponse DOIT être éliminée, comme si elle n'avait jamais été reçue. Cela signifie que les retransmissions, si applicable, vont continuer. Si toutes les réponses reçues sont éliminées, alors au lieu de signaler une fin de temporisation après la fin de la transaction, la couche DOIT signaler que la protection d'intégrité était violée.

Si la demande était envoyée sur un transport fiable, la réponse DOIT être éliminée, et la couche DOIT immédiatement terminer la transaction et signaler que la protection d'intégrité était violée.

Si la réponse contient un attribut PASSWORD-ALGORITHMS, toutes les demandes suivantes DOIVENT être authentifiées en utilisant MESSAGE-INTEGRITY-SHA256 seulement.

10. Mécanisme ALTERNATE-SERVER

Cette Section décrit un mécanisme de STUN qui permet à un serveur de rediriger un client sur un autre serveur. Cette extension est facultative, et un usage doit définir si et quand cette extension est utilisée. L'attribut ALTERNATE-SERVER porte une adresse IP.

Un serveur qui utilise cette extension redirige un client sur un autre serveur en répondant à un message de demande par un message de réponse d'erreur avec un code d'erreur de 300 (Essayer un autre). Le serveur DOIT inclure au moins un attribut ALTERNATE-SERVER dans la réponse d'erreur, qui DOIT contenir une adresse IP de la même famille d'adresses que l'adresse IP de source du message de demande. Le serveur DEVRAIT inclure un attribut ALTERNATE-SERVER supplémentaire, après celui obligatoire, contenant une adresse IP de la famille d'adresses autre que l'adresse IP de source du message de demande. Le message de réponse d'erreur PEUT être authentifié ; cependant, il y a des cas d'utilisation de ALTERNATE-SERVER où l'authentification de la réponse n'est pas possible ou praticable. Si la transaction utilise TLS ou DTLS, si la transaction est authentifiée par un attribut MESSAGE-INTEGRITY-SHA256, et si le serveur veut rediriger sur un serveur qui utilise un certificat différent, il DOIT alors inclure un attribut ALTERNATE-DOMAIN contenant le nom à l'intérieur du subjectAltName de ce certificat. Cette série de conditions sur l'attribut MESSAGE-INTEGRITY-SHA256 indique que la transaction est authentifiée et que le client met en œuvre la présente spécification et donc peut traiter l'attribut ALTERNATE-DOMAIN.

Un client qui utilise cette extension traite un code d'erreur 300 (Essayer un autre) comme suit. Le client cherche un attribut ALTERNATE-SERVER dans la réponse d'erreur. Si il en trouve une, il considère que la transaction en cours a échoué et retente la demande avec le serveur spécifié dans l'attribut, en utilisant le même protocole de transport qu'utilisé pour la demande précédente. Cette demande, si elle est authentifiée, DOIT utiliser les mêmes accréditifs que le client aurait utilisé dans la demande au serveur qui a effectué la redirection. Si le protocole de transport utilise TLS ou DTLS, alors le client cherche un attribut ALTERNATE-DOMAIN. Si l'attribut est trouvé, le domaine DOIT être utilisé pour valider le certificat en utilisant les recommandations de la [RFC6125]. Le certificat DOIT contenir un identifiant de type DNS-ID ou CN-ID (éventuellement avec des caractères génériques) mais pas de type SRV-ID ou URI-ID. Si l'attribut n'est pas trouvé, le même domaine qu'utilisé pour la demande originale DOIT être utilisé pour valider le certificat. Si le client a été redirigé sur un serveur auquel il a déjà envoyé cette demande dans les cinq dernières minutes, il DOIT ignorer la redirection et considérer que la transaction a échoué. Cela empêche un ping-pong infini entre serveurs en cas de boucles de redirection.

11. Rétro compatibilité avec la RFC 3489

En plus de la rétro compatibilité déjà décrite à la Section 12 de la [RFC5389], DTLS NE DOIT PAS être utilisé avec la [RFC3489] (appelée le "STUN classique"). Toute demande ou indication STUN sans le mouchard magique (voir la Section 6 de la [RFC5389]) sur DTLS DOIT être considérée comme invalide : toutes les demandes DOIVENT générer une réponse d'erreur 500 (Erreur du serveur) et les indications DOIVENT être ignorées.

12. Comportement de base du serveur

Cette Section définit le comportement d'un serveur STUN de base, autonome.

Historiquement, le "STUN classique" de la [RFC3489] définissait seulement le comportement d'un serveur qui fournissait aux clients des adresses de transport de serveur réflexives en recevant et répondant aux demandes STUN Binding. La [RFC5389] a redéfini le protocole comme un cadre extensible, et la fonction de serveur est devenue le seul usage STUN défini dans ce document. Cet usage STUN est aussi appelé "serveur STUN de base".

Le serveur STUN DOIT prendre en charge la méthode Binding. Il NE DEVRAIT PAS utiliser le mécanisme d'accréditifs à court terme ou long terme. C'est parce que le travail impliqué dans l'authentification de la demande est supérieur au travail de simplement le traiter. Il NE DEVRAIT PAS utiliser le mécanisme ALTERNATE-SERVER pour la même raison. Il DOIT prendre en charge UDP et TCP. Il PEUT prendre en charge STUN sur TCP/TLS ou STUN sur UDP/DTLS ; cependant, DTLS et TLS fournissent des avantages de sécurité minimaux dans ce mode de fonctionnement de base. Il

n'exige pas de mécanisme de maintien en vie parce que une connexion TCP ou TLS sur TCP est close après la fin de la transaction Binding. Il PEUT utiliser le mécanisme FINGERPRINT mais NE DOIT PAS l'exiger. Comme le serveur autonome fonctionne seulement sur STUN, FINGERPRINT ne procure aucun avantage. L'exiger romprait la compatibilité avec la RFC 3489, et une telle compatibilité est désirable dans un serveur autonome. Les serveurs STUN autonomes DEVRAIENT prendre en charge la rétro compatibilité avec les clients qui utilisent la [RFC3489], comme décrit à la Section 11.

Il est RECOMMANDÉ que les administrateurs de serveurs STUN fournissent des entrées du DNS pour ces serveurs comme décrit à la Section 8. Si des enregistrements de ressource A et AAAA sont retournés, alors le client peut simultanément envoyer des demandes STUN Binding aux adresses IPv4 et IPv6 (comme spécifié dans la [RFC8305]) puisque la demande Binding est idempotente. Noter que les attributs MAPPED-ADDRESS ou XOR-MAPPED-ADDRESS qui sont retournés ne vont pas nécessairement correspondre à la famille d'adresses de l'adresse de serveur utilisée.

Un serveur STUN de base n'est pas par lui-même une solution pour la traversée de NA. Cependant, il peut être utilisé comme partie d'une solution à travers des usages STUN. Ceci est discuté plus en détails à la Section 13.

13. Usages de STUN

STUN par lui-même n'est pas une solution au problème de la traversée de NAT. STUN définit plutôt un outil qui peut être utilisé dans une solution plus large. Le terme de "usage STUN" est utilisé pour toute solution qui utilise STUN comme composant.

Un usage STUN définit comment STUN est en fait utilisé -- quand envoyer les demandes, que faire avec les réponses, et quelles procédures facultatives définies ici (ou dans une extension à STUN) sont à utiliser. Un usage définit aussi :

- Quelles méthodes STUN sont utilisées.
- Quels transports sont utilisés. Si DTLS sur UDP est utilisé, la mise en œuvre des contre-mesures de déni de service décrite au paragraphe 4.2.1 de la [RFC6347] est obligatoire.
- Quels mécanismes d'authentification et d'intégrité de message sont utilisés.
- Les considérations sur la déduction manuelle ou automatique de clé pour le mécanisme d'intégrité, comme discutées dans la [RFC4107].
- Quels mécanismes sont utilisés pour distinguer les messages STUN des autres messages. Quand STUN fonctionne sur TCP ou TLS sur TCP, un mécanisme de tramage peut être requis.
- Comment un client STUN détermine l'adresse et l'accès IP du serveur STUN.
- Comment une utilisation simultanée d'adresses IPv4 et IPv6 (Happy Eyeballs [RFC8305]) fonctionne avec des transactions non idempotentes quand les deux familles d'adresses sont trouvées pour le serveur STUN.
- Si la rétro compatibilité avec la RFC 3489 est requise.
- Quels attributs facultatifs définis ici (comme FINGERPRINT et ALTERNATE-SERVER) ou dans d'autres extensions sont requis.
- Si la troncature de MESSAGE-INTEGRITY-SHA256 est permise, et les limites permises pour la troncature.
- Le mécanisme de maintien en vie si STUN fonctionne sur TCP ou TLS sur TCP.
- Si des adresses d'envoi à la cantonade peuvent être utilisées pour le serveur en cas d'utilisation 1) de TCP ou TLS sur TCP, ou 2) d'authentification.

De plus, tout usage STUN doit considérer les implications de sécurité de l'utilisation de STUN dans cet usage. Un certain nombre d'attaques contre STUN sont connues (voir la Section des considérations sur la sécurité du présent document) et tout usage doit considérer comment ces attaques peuvent être déjouées ou atténuées.

Finalement, un usage doit considérer si son usage de STUN est un exemple d'approche de correction d'auto adresse unilatérale de la traversée de NAT et, si il l'est, il règle les questions soulevées dans la [RFC3424].

14. Attributs STUN

Après l'en-tête STUN sont zéro, un ou plusieurs attributs. Chaque attribut DOIT être codé comme TLV, avec un type de 16 bits, une longueur de 16 bits, et une valeur. Chaque attribut STUN DOIT finir sur une limite de 32 bits. Comme mentionné ci-dessus, tous les champs dans un attribut sont transmis avec le bit de poids fort en premier.

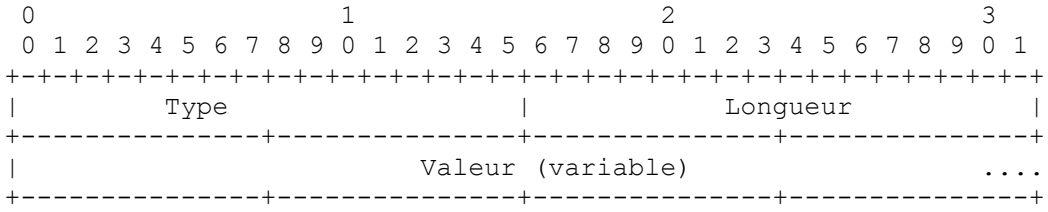

Figure 4 : Format des attributs STUN

La valeur dans le champ Longueur DOIT contenir la longueur de la partie Valeur de l'attribut, avant le bourrage, mesurée en octets. Comme STUN aligne les attributs sur des limites de 32 bits, les attributs dont le contenu n'est pas un multiple de 4 octets sont bourrés avec 1, 2, ou 3 octets de bourrage afin que sa valeur contienne un multiple de 4 octets. Les bits de bourrage DOIVENT être réglés à zéro à l'envoi et DOIVENT être ignorés par le receveur.

Tout type d'attribut PEUT apparaître plus d'une fois dans un message STUN. Sauf mention contraire, l'ordre d'apparition est significatif ; seule la première occurrence a besoin d'être traitée par un receveur, et tous les dupliqués PEUVENT être ignorés par un receveur.

Pour permettre que de futures révisions de cette spécification ajoutent de nouveaux attributs si nécessaire, l'espace d'attributs est divisé en deux gammes. Les attributs avec des valeurs de type entre 0x0000 et 0x7FFF sont des attributs de compréhension exigée, ce qui signifie que l'agent STUN ne peut pas réussir à traiter le message à moins de comprendre l'attribut. Les attributs avec des valeurs de type entre 0x8000 et 0xFFFF sont des attributs de compréhension facultative, ce qui signifie que ces attributs peuvent être ignorés par l'agent STUN si il ne les comprend pas.

L'ensemble des types d'attribut STUN est maintenu par l'IANA. L'ensemble initial défini par la présente spécification est décrit au paragraphe 18.3.

Le reste de cette section décrit le format des divers attributs définis dans cette spécification.

14.1 MAPPED-ADDRESS

L'attribut MAPPED-ADDRESS indique une adresse de transport réflexive du client. Il consiste en une famille d'adresses de 8 bits et un accès de 16 bits, suivis par une valeur de longueur fixe qui représente l'adresse IP. Si la famille d'adresses est IPv4, l'adresse DOIT faire 32 bits. Si la famille d'adresses est IPv6, l'adresse DOIT faire 128 bits. Tous les champs doivent être dans l'ordre des octets du réseau.

Le format de l'attribut MAPPED-ADDRESS est :

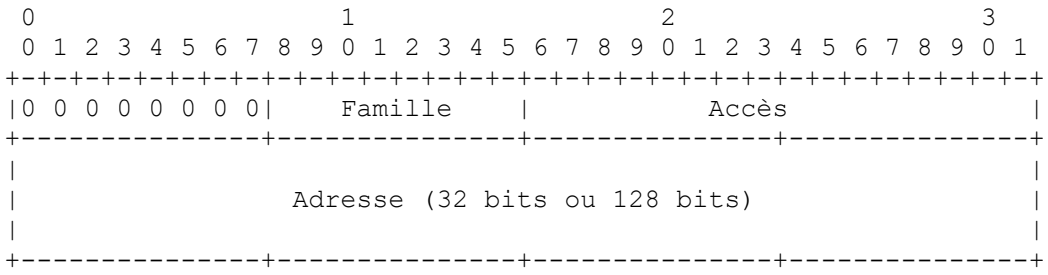

Figure 5 : Format de l'attribut MAPPED-ADDRESS

La famille d'adresses peut prendre les valeurs suivantes :
0x01 : IPv4
0x02 : IPv6

Les 8 premiers bits de MAPPED-ADDRESS DOIVENT être réglés à 0 et DOIVENT être ignorés des receveurs. Ces bits sont présents pour aligner les paramètres sur les frontières naturelles de 32 bits.

Cet attribut est utilisé seulement par les serveurs pour réaliser la rétro compatibilité avec les clients de la [RFC3489].

14.2 XOR-MAPPED-ADDRESS

L'attribut XOR-MAPPED-ADDRESS est identique à l'attribut MAPPED-ADDRESS, sauf que l'adresse de transport réflexive est masquée par la fonction OUX.

Le format de XOR-MAPPED-ADDRESS est :

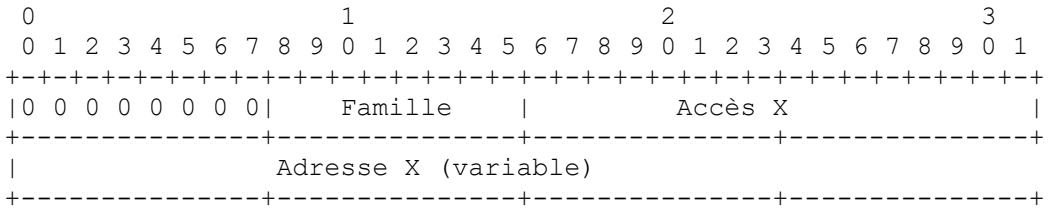

Figure 6 : Format de l'attribute XOR-MAPPED-ADDRESS

Le champ Famille représente la famille d'adresses IP et est codé de façon identique au champ Famille de MAPPED-ADDRESS.

Accès X est calculé en OUXant l'accès transposé avec les 16 bits de poids fort du mouchard magique. Si la famille d'adresses IP est IPv4, Adresse X est calculée en OUXant l'adresse IP transposée avec le mouchard magique. Si la famille d'adresses IP est IPv6, Adresse X est calculée en OUXant l'adresse IP transposée avec l'enchaînement du mouchard magique et des 96 bits de l'identifiant de transaction. Dans tous les cas, l'opération OUX fonctionne sur son entrée dans l'ordre des octets du réseau (c'est-à-dire, l'ordre dans lequel il va être codé dans le message).

Les règles de codage et de traitement des 8 premiers bits de la valeur de l'attribut, les règles du traitement de plusieurs occurrences de l'attribut, et les règles et traitement des familles d'adresses sont les mêmes que pour MAPPED-ADDRESS.

Note : XOR-MAPPED-ADDRESS et MAPPED-ADDRESS diffèrent seulement par leur codage de l'adresse de transport. Le premier code l'adresse de transport en l'OUXant avec le mouchard magique. Le dernier la code directement en binaire. La RFC 3489 spécifiait à l'origine seulement MAPPED-ADDRESS. Cependant, l'expérience du déploiement a trouvé que certains NAT réécrivent les charges utiles binaires de 32 bits qui contiennent l'adresse IP publique du NAT, comme l'attribut MAPPED-ADDRESS de STUN, dans une tentative bien intentionnée mais malheureuse de fournir une fonction générique de passerelle de couche application (ALG, *Application Layer Gateway*). Ce comportement interfère avec le fonctionnement de STUN et cause aussi l'échec de la vérification d'intégrité de message de STUN.

14.3 USERNAME

L'attribut USERNAME est utilisé pour l'intégrité de message. Il identifie la combinaison de nom d'utilisateur et mot de passe utilisée dans la vérification d'intégrité de message.

La valeur de USERNAME est une valeur de longueur variable qui contient le nom d'utilisateur de l'authentification. Il DOIT contenir une séquence codée en UTF-8 [RFC3629] de moins de 509 octets et DOIT avoir été traité en utilisant le profil OpaqueString de la [RFC8265]. Une mise en œuvre conforme DOIT être capable d'analyse une séquence codée en UTF-8 de 763 octets ou moins pour être compatible avec la [RFC5389].

Note : la [RFC5389] faisant par erreur référence à la définition de UTF-8 dans la [RFC2279]. La [RFC2279] supposait jusqu'à 6 octets par caractère codé. La [RFC2279] a été remplacée par la [RFC3629], qui permet seulement 4 octets par caractère codé, en cohérence avec les changements faits dans Unicode 2.0 et la norme ISO/CEI 10646.

Note : la présente spécification utilise le profil OpaqueString au lieu du profil UsernameCasePreserved pour le traitement de la chaîne de nom d'utilisateur afin d'améliorer la compatibilité avec les magasins déployés de mots de passe. De nombreuses bases de données de mots de passe utilisées pour l'authentification HTTP et par résumé SIP mémorisent le hachage MD5 du nom d'utilisateur:domaine:mot de passe au lieu de mémoriser un mot de passe en clair. Dans la [RFC3489], l'authentification STUN était destinée à être compatible avec ces bases de données existantes dans la mesure du possible, qui, comme SIP et HTTP n'effectuaient pas de pré-traitement des noms d'utilisateur et de mots de passe autre que de prohiber les caractères de contrôle non espace ASCII. La révision suivante de la spécification STUN, la [RFC5389], utilisait le profil SASLprep [RFC4013] stringprep [RFC3454] pour pré-traiter les noms d'utilisateur et mots de passe. SASLprep utilise la forme de normalisation Unicode KC (Décomposition de

compatibilité, suivie par composition canonique) [UAX15] et interdit les divers codets de contrôle, espace, et non texte, déconseillés, ou inappropriés. Le cadre PRECIS [RFC8264] rend obsolète stringprep. Le traitement PRECIS des noms d'utilisateur et mots de passe [RFC8265] utilise la forme de normalisation Unicode C (décomposition canonique, suivie par la composition canonique). Bien qu'il y ait des cas spécifiques où des chaînes différentes de nom d'utilisateur sous HTTP Digest pourraient être transposées en un seul nom d'utilisateur STUN traité avec OpaqueString, ces cas sont extrêmement improbables et faciles à détecter et corriger. Avec un profil UsernameCasePreserved, il serait plus probable que des noms d'utilisateur valides sous HTTP Digest ne correspondent pas à leur forme traitée (spécifiquement les noms d'utilisateur contenant du texte bidirectionnel et des formes de compatibilité). Les opérateurs sont libres de restreindre encore les codets permis dans les noms d'utilisateur pour éviter des caractères problématiques.

14.4 USERHASH

L'attribut USERHASH est utilisé comme remplacement de l'attribut USERNAME quand l'anonymat du nom d'utilisateur est pris en charge.

La valeur de USERHASH a une longueur fixe de 32 octets. Le nom d'utilisateur DOIT avoir été traité en utilisant le profil OpaqueString [RFC8265], et le domaine DOIT avoir été traité en utilisant le profil OpaqueString [RFC8265] avant le hachage.

Voici l'opération que le client va effectuer pour hacher le nom d'utilisateur :

userhash = SHA-256(OpaqueString(nom d'utilisateur) ":" OpaqueString(realm))

14.5 MESSAGE-INTEGRITY

L'attribut MESSAGE-INTEGRITY contient un HMAC-SHA1 [RFC2104] du message STUN. L'attribut MESSAGE-INTEGRITY peut être présent dans tout type de message STUN. Comme il utilise le hachage SHA-1, le HMAC va être de 20 octets.

La clé pour le HMAC dépend du mécanisme d'accréditifs utilisé. Le paragraphe 9.1.1 définit la clé pour le mécanisme d'accréditif à court terme, et le paragraphe 9.2.2 définit la clé pour le mécanisme d'accréditif à long terme. Les autres mécanismes d'accréditifs DOIVENT définir la clé qui est utilisée pour le HMAC.

Le texte utilisé comme entrée au HMAC est le message STUN, jusqu'à et inclu l'attribut précédant l'attribut MESSAGE-INTEGRITY. Le champ Longueur de l'en-tête de message STUN est ajusté pour pointer sur la fin de l'attribut MESSAGE-INTEGRITY. La valeur de l'attribut MESSAGE-INTEGRITY est réglée à une valeur factice.

Une fois le calcul effectué, la valeur de l'attribut MESSAGE-INTEGRITY est remplie, et la valeur de la longueur dans l'en-tête STUN est réglée à sa valeur correcte -- la longueur du message entier. De même, quand on valide le MESSAGE-INTEGRITY, le champ Longueur dans l'en-tête STUN doit être ajusté à pointer sur la fin de l'attribut MESSAGE-INTEGRITY avant le calcul du HMAC sur le message STUN, jusqu'à et incluant l'attribut précédant l'attribut MESSAGE-INTEGRITY. Cet ajustement est nécessaire quand des attributs, comme FINGERPRINT et MESSAGE-INTEGRITY-SHA256, apparaissent après MESSAGE-INTEGRITY. Voir aussi dans la [RFC5769] des exemples de tels calculs.

14.6 MESSAGE-INTEGRITY-SHA256

L'attribut MESSAGE-INTEGRITY-SHA256 contient un HMAC-SHA256 [RFC2104] du message STUN. L'attribut MESSAGE-INTEGRITY-SHA256 peut être présent dans tout type de message STUN. L'attribut MESSAGE-INTEGRITY-SHA256 contient une portion initiale du HMAC-SHA-256 [RFC2104] du message STUN. La valeur va être d'au plus 32 octets, mais elle DOIT être d'au moins 16 octets et DOIT être un multiple de 4 octets. La valeur doit être sur les 32 octets complets sauf si l'usage STUN spécifié explicitement que la troncature est permise. Les usages STUN peuvent spécifier une longueur minimum de plus de 16 octets.

La clé pour le HMAC dépend du mécanisme d'accréditifs utilisé. Le paragraphe 9.1.1 définit la clé pour le mécanisme d'accréditif à court terme, et le paragraphe 9.2.2 définit la clé pour le mécanisme d'accréditif à long terme. Les autres mécanismes d'accréditifs DOIVENT définir la clé qui est utilisée pour le HMAC.

Le texte utilisé comme entrée au HMAC est le message STUN, jusqu'à et inclu l'attribut précédant l'attribut MESSAGE-

INTEGRITY-SHA256. Le champ Longueur de l'en-tête du message STUN est ajusté pour pointer sur la fin de l'attribut MESSAGE-INTEGRITY-SHA256. La valeur de l'attribut MESSAGE-INTEGRITY-SHA256 est réglée à une valeur factice.

Une fois le calcul effectué, la valeur de l'attribut MESSAGE-INTEGRITY-SHA256 est remplie, et la valeur de la longueur dans l'en-tête STUN est réglée à sa valeur correcte -- la longueur du message entier. De même, quand on valide le MESSAGE-INTEGRITY-SHA256, le champ Longueur dans l'en-tête STUN doit être ajusté à pointer sur la fin de l'attribut MESSAGE-INTEGRITY-SHA256 avant le calcul du HMAC sur le message STUN, jusqu'à et inclu l'attribut précédant l'attribut MESSAGE-INTEGRITY-SHA256. Cet ajustement est nécessaire quand des attributs, comme FINGERPRINT, apparaissent après MESSAGE-INTEGRITY-SHA256. Voir aussi à l'Appendice B.1 des exemples de tels calculs.

14.7 FINGERPRINT

L'attribut FINGERPRINT PEUT être présent dans tous les messages STUN.

La valeur de l'attribut est calculée comme le CRC-32 du message STUN jusqu'à (mais exclu) l'attribut FINGERPRINT lui-même, OUXé avec la valeur de 32 bits 0x5354554e. (l'opération OUX assure que l'essai de FINGERPRINT ne va pas rapporter un faux positif sur un paquet contenant un CRC-32 généré par un protocole d'application.) Le CRC de 32 bits est celui défini dans la Recommandation UIT-T [V.42], qui a un générateur polynomial de x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1. Voir l'exemple de code pour le CRC-32 à la Section 8 de la [RFC1952].

Quand il est présent, l'attribut FINGERPRINT DOIT être le dernier attribut dans le message et donc va apparaître après MESSAGE-INTEGRITY et MESSAGE-INTEGRITY-SHA256.

L'attribut FINGERPRINT peut aider à distinguer les paquets STUN des paquets d'autres protocoles. Voir la Section 7.

Comme avec MESSAGE-INTEGRITY et MESSAGE-INTEGRITY-SHA256, le CRC utilisé dans l'attribut FINGERPRINT couvre le champ Longueur de l'en-tête du message STUN. Donc, avant le calcul du CRC, cette valeur doit être correcte et inclure l'attribut CRC au titre de la longueur du message. Quand on utilise l'attribut FINGERPRINT dans un message, l'attribut est d'abord placé dans le message avec une valeur factice ; puis, le CRC est calculé, et la valeur de l'attribut est mise à jour. Si l'attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 est aussi présent, il doit alors être présent avec la valeur correcte d'intégrité de message avant que le CRC soit calculé, car le CRC est fait aussi sur la valeur des attributs MESSAGE-INTEGRITY et MESSAGE-INTEGRITY-SHA256.

14.8 ERROR-CODE

L'attribut ERROR-CODE est utilisé dans les messages de réponse d'erreur. Il contient une valeur numérique de code d'erreur dans la gamme de 300 à 699 plus une phrase textuelle de raison codée en UTF-8 [RFC3629] ; il est aussi cohérent dans ses allocations et sa sémantique de code avec SIP [RFC3261] et HTTP [RFC7231]. La phrase de raison est destinée aux diagnostics et peut être n'importe quoi d'approprié pour le code d'erreur. Les phrases de raison recommandées pour les codes d'erreur définis sont incluses dans le registre IANA des codes d'erreur. La phrase de raison DOIT être une séquence codée en UTF-8 [RFC3629] de moins de 128 caractères (qui peuvent être jusqu'à 509 octets quand on les code ou 763 octets quand on les décode).

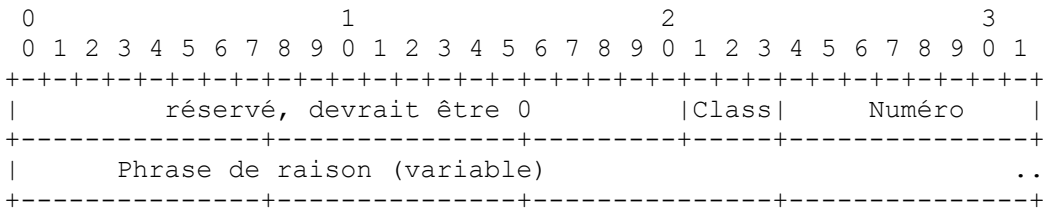

Figure 7 : Format de l'attribut ERROR-CODE

Pour faciliter le traitement, la classe du code d'erreur (le chiffre des centaines) est codée séparément du reste du code, comme le montre la Figure 7.

Les bits "réservé" DEVRAIENT être à 0 et sont pour l'alignement sur les frontières de 32 bits. Les receveurs DOIVENT

ignorer ces bits. La Classe représente le chiffre des centaines du numéro de code d'erreur. La valeur DOIT être entre 3 et 6. Le Numéro représente le codage binaire du code d'erreur modulo 100, et sa valeur DOIT être entre 0 et 99.

Les codes d'erreur suivants, avec leur phrase de raison recommandée, sont définis :

300 Essayer un autre : le client devrait contacter un autre serveur pour cette demande. Cette réponse d'erreur DOIT seulement être envoyée si la demande incluait un attribut USERNAME ou USERHASH et un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 valide ; autrement, il NE DOIT PAS être envoyé et le code d'erreur 400 (Mauvaise demande) est suggéré. Cette réponse d'erreur DOIT être protégée avec l'attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256, et les receveurs DOIVENT valider le MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 de cette réponse avant de se rediriger sur un autre serveur.

Note : l'échec à générer et valider l'intégrité de message pour une réponse 300 permet à un attaquant sur le chemin de falsifier une réponse 300 causant ainsi l'envoi des messages STUN suivants à une victime.

400 Mauvaise demande : la demande était mal formée. Le client NE DEVRAIT PAS réessayer la demande sans modification de la tentative précédente. Le serveur peut n'être pas capable de générer un MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 valide pour cette erreur, de sorte que le client NE DOIT PAS attendre un attribut MESSAGE-INTEGRITY ou MESSAGE-INTEGRITY-SHA256 valide sur cette réponse.

401 Non authentifié : la demande ne contenait pas les accréditifs corrects pour poursuivre. Le client devrait réessayer la demande avec des accréditifs appropriés.

420 Attribut inconnu : le serveur a reçu un paquet STUN contenant un attribut de compréhension exigée qu'il ne comprend pas. Le serveur DOIT mettre cet attribut inconnu dans l'attribut UNKNOWN-ATTRIBUTE de sa réponse d'erreur.

438 Nom occasionnel périmé : le NONCE utilisé par le client n'était plus valide. Le client devrait réessayer, en utilisant le NONCE fourni dans la réponse.

500 Erreur du serveur : le serveur a subi une erreur temporaire. Le client devrait réessayer.

14.9 REALM

L'attribut REALM peut être présent dans des demandes et des réponses. Il contient un texte qui satisfait à la grammaire pour "realm-value" comme décrit dans la [RFC3261] mais sans les guillemets et leur espace environnante. C'est-à-dire, c'est un realm-value sans guillemets (et est donc une séquence de qdtext ou quoted-pair). Ce DOIT être une séquence codée en UTF-8 [RFC3629] de moins de 128 caractères (qui peut faire jusqu'à 509 octets au codage et jusqu'à 763 octets au décodage) et DOIT avoir été traitée en utilisant le profil OpaqueString [RFC8265].

La présence de l'attribut REALM dans une demande indique que des accréditifs à long terme sont utilisés pour l'authentification. Sa présence dans certaines réponses d'erreur indique que le serveur souhaite que le client utilise un accréditif à long terme dans ce domaine pour l'authentification.

14.10 NONCE

L'attribut NONCE peut être présent dans les demandes et les réponses. Il contient une séquence de qdtext ou quoted-pair, qui sont définis dans la [RFC3261]. Noter que cela signifie que l'attribut NONCE ne va pas contenir les caractères guillemets autour. L'attribut NONCE DOIT faire moins de 128 caractères (qui peuvent être jusqu'à 509 octets quand on les code ou 763 octets quand on les décode). Voir au paragraphe 5.4 de la [RFC7616] des directives sur le choix des valeurs de nom occasionnel dans un serveur.

14.11 PASSWORD-ALGORITHMS

L'attribut PASSWORD-ALGORITHMS peut être présent dans les demandes et les réponses. Il contient la liste des algorithmes que le serveur peut utiliser pour déduire le mot de passe à long terme.

L'ensemble des algorithmes connus est tenu par l'IANA. L'ensemble initial défini par la présente spécification se trouve au paragraphe 18.5.

L'attribut contient une liste de numéros d'algorithmes et des paramètres de longueur variable. Le numéro d'algorithme est une valeur de 16 bits comme défini au paragraphe 18.5. Les paramètres commencent par la longueur (avant le bourrage) des paramètres comme une valeur de 16 bits, suivie par les paramètres qui sont spécifiques de chaque algorithme. Les paramètres sont bourrés à une frontière de 32 bits, de la même manière qu'un attribut.

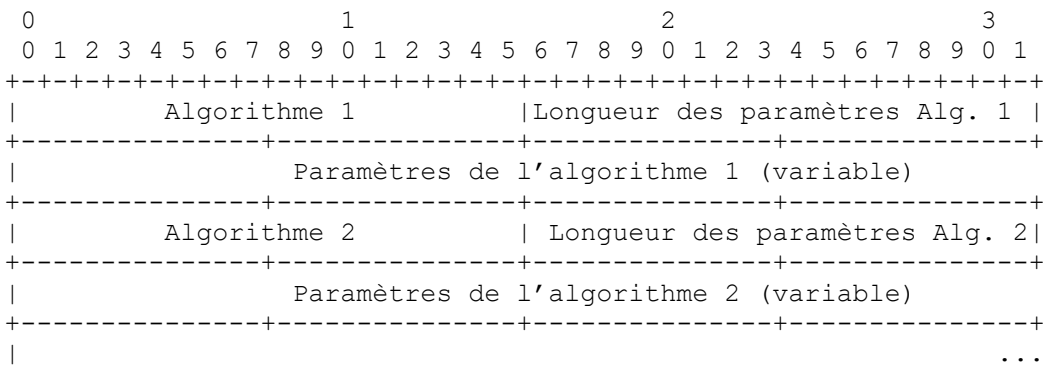

Figure 8 : Format de l'attribut PASSWORD-ALGORITHMS

14.12 PASSWORD-ALGORITHM

L'attribut PASSWORD-ALGORITHM est présent seulement dans les demandes. Il contient l'algorithme que le serveur doit utiliser pour déduire une clé du mot de passe à long-terme.
L'ensemble des algorithmes connus est tenu par l'IANA. L'ensemble initial défini par la présente spécification se trouve au paragraphe 18.5.

L'attribut contient un numéro d'algorithme et des paramètres de lungueur variable. Le numéro d'algorithme est une valeur de 16 bits comme défini au paragraphe 18.5. Les paramètres commencent par la longueur (avant le bourrage) des paramètres comme une valeur de 16 bits, suivie par les paramètres qui sont spécifiques de l'algorithme. Les paramètres sont bourrés à une limite de 32 bits, de la même manière qu'un attribut. De même, les bits de bourrage DOIVENT être mis à zéro à l'envoi et DOIVENT être ignorés par le receveur.

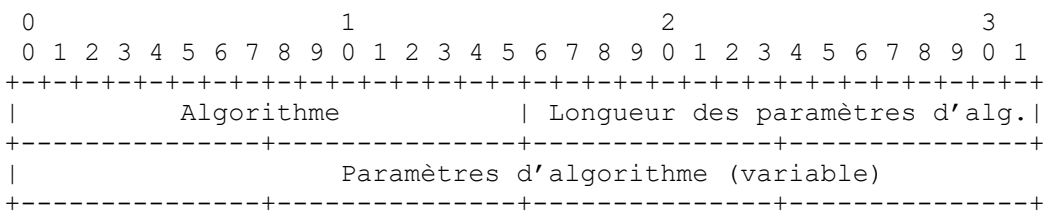

Figure 9 : Format de l'attribut PASSWORD-ALGORITHM

14.13 UNKNOWN-ATTRIBUTES

L'attribut UNKNOWN-ATTRIBUTES est présent seulement dans une réponse d'erreur quand le code de réponse dans l'attribut ERROR-CODE est 420 (Attribut inconnu).
L'attribut contient une liste de valeurs de 16 bits, dont chacune représente un type d'attribut qui était non compris par le serveur.

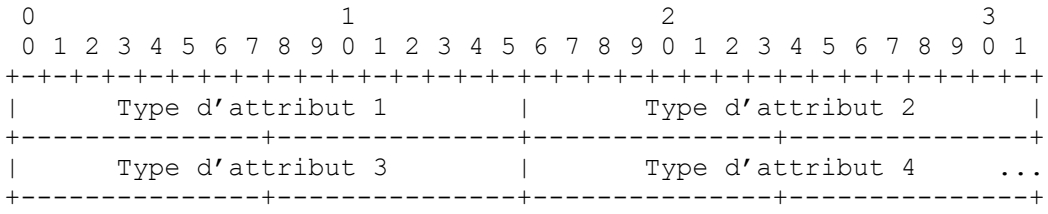

Figure 10 : Format de l'attribut UNKNOWN-ATTRIBUTES

Note : dans la [RFC3489], ce champ était bourré à 32 en dupliquant le dernier attribut. Dans cette version de la spécification, les règles normales de bourrage des attributs sont utilisées.

14.14 SOFTWARE

L'attribut SOFTWARE contient une description textuelle du logiciel utilisé par l'agent qui envoie le message. Il est utilisé par les clients et les serveurs. Sa valeur DEVRAIT inclure le nom du fabriant et le numéro de version. L'attribut n'a pas d'impact sur le fonctionnement du protocole et sert seulement d'outil de diagnostic et de débogage. La valeur de SOFTWARE est de longueur variable. Il DOIT être une séquence codée en UTF-8 [RFC3629] de moins de 128 caractères (qui peuvent être jusqu'à 509 octets quand on les code ou 763 octets quand on les décode).

14.15 ALTERNATE-SERVER

Le serveur de remplacement représente une autre adresse de transport qui identifie un serveur STUN différent que le client STUN devrait essayer.

Il est codé de la même façon que MAPPED-ADDRESS et donc se réfère à un seul serveur par adresse IP.

14.16 ALTERNATE-DOMAIN

Le domaine de remplacement représente le nom de domaine qui est utilisé pour vérifier l'adresse IP dans l'attribut ALTERNATE-SERVER quand le protocole de transport utilise TLS ou DTLS.

La valeur de ALTERNATE-DOMAIN est de longueur variable. Elle DOIT être un nom DNS valide [RFC1123] (incluant les étiquettes A [RFC5890]) de 255 caractères ASCII ou moins.

15. Considérations de fonctionnement

STUN PEUT être utilisé avec des adresses d'envoi à la cantonade, mais seulement avec UDP et dans des usages STUN où l'authentification n'est pas utilisée.

16. Considérations de sécurité

Les mises en œuvre et déploiements d'un usage STUN avec TLS ou DTLS DOIVENT suivre les recommandations de la [RFC7525].

Les mises en œuvre et déploiements d'un usage STUN qui utilisent le mécanisme d'accréditif à long terme (paragraphe 9.2) DOIVENT suivre les recommandations de la Section 5 de la [RFC7616].

16.1 Attaques contre le protocole

16.1.1 Attaques de l'extérieur

Un attaquant peut essayer de modifier les messages STUN en transit, afin de causer une défaillance du fonctionnement de STUN. Ces attaques sont détectées pour les demandes et les réponses par le mécanisme d'intégrité de message, en utilisant un accréditif à court terme ou à long terme. Bien sûr, une fois détecté, les paquets manipulés vont être éliminés, causant effectivement l'échec de la transaction STUN. Cette attaque est possible seulement par un attaquant sur le chemin.

Un attaquant qui peut observer, mais non modifier, les messages STUN en transit (par exemple, un attaquant présent sur un support en accès partagé, comme un Wi-Fi) peut voir une demande STUN et envoyer alors immédiatement une réponse STUN, typiquement une réponse d'erreur, afin de perturber le traitement STUN. Cette attaque est aussi empêchée pour les messages qui utilisent MESSAGE-INTEGRITY. Cependant, certaines réponses d'erreur, celles relatives à l'authentification en particulier, ne peuvent pas être protégées par MESSAGE-INTEGRITY. Quand STUN lui-même fonctionne sur un protocole de transport sûr (par exemple, TLS) ces attaques sont complètement déjouées.

Selon l'usage STUN, ces attaques peuvent être de faibles conséquences et donc n'exigent pas d'être atténuées par l'intégrité de message. par exemple, quand STUN est utilisé sur un serveur STUN de base pour découvrir un candidat serveur réflexif pour l'usage avec ICE, l'authentification et l'intégrité de message ne sont pas requises car ces attaques sont détectées durant la phase de vérification de connexité. Les vérifications de connexité elles-mêmes, exigent cependant la protection du fonctionnement approprié globale de ICE. Comme décrit à la Section 13, les usages STUN décrivent quand l'authentification et l'intégrité de message sont nécessaires.

Comme STUN utilise le HMAC d'un secret partagé pour l'authentification et la protection de l'intégrité, il est soumis à des attaques de dictionnaire hors ligne. Quand l'authentification est utilisée, elle DEVRAIT l'être avec un mot de passe fort qui ne soit pas directement l'objet d'attaques de dictionnaire hors ligne. La protection du canal lui-même, en utilisant TLS ou DTLS, atténue ces attaques.

STUN prend en charge MESSAGE-INTEGRITY et MESSAGE-INTEGRITY-SHA256, ce qui fait qu'il est sujet à des attaques en dégradation par un attaquant sur le chemin. Un attaquant pourrait supprimer l'attribut MESSAGE-INTEGRITY-SHA256, laissant seulement l'attribut MESSAGE-INTEGRITY et exploitant donc une potentielle vulnérabilité. La protection du canal lui-même, en utilisant TLS ou DTLS, atténue ces attaques. La suppression le moment venu de la prise en charge de MESSAGE-INTEGRITY dans une future version de STUN est nécessaire.

Note : l'utilisation de SHA-256 pour le hachage du mot de passe ne satisfait pas aux normes modernes, qui visent à ralentir les recherches exhaustives de mot de passe en fournissant un temps minimum relativement faible pour calculer le hachage. Bien que de meilleurs algorithmes comme Argon2 [RFC9106] soient disponibles, SHA-256 a été choisi pour la cohérence avec la [RFC7616].

16.1.2 Attaques de l'intérieur

Un client félon peut essayer de lancer une attaque de DoS contre un serveur en lui envoyant un grand nombre de demandes STUN. Heureusement, les demandes STUN peuvent être traitées sans état par un serveur, rendant de telles attaques difficiles à lancer effectivement.

Un client félon peut utiliser un serveur STUN comme réflecteur, lui envoyant des demandes avec une adresse et accès IP de source falsifiés. Dans ce cas, la réponse va être livrée à cette source et accès IP. Il n'y a pas d'amplification du nombre de paquets avec cette attaque (le serveur STUN envoie un paquet pour chaque paquet envoyé par le client) bien qu'il y ait une petite augmentation de la quantité de données, car les réponses STUN sont normalement plus grandes que les demandes. Cette attaque est atténuée par le filtrage d'adresse de source.

Révéler la version de logiciel spécifique de l'agent par l'attribut SOFTWARE pourrait permettre de devenir plus vulnérable aux attaques contre le logiciel connu pour contenir des trous de sécurité. Les mises en œuvre DEVRAIENT faire usage de l'attribut SOFTWARE une option configurable.

16.1.3 Attaques en dégradation

Le présent document ajoute la possibilité de choisir différents algorithmes pour protéger la confidentialité des mots de passe mémorisés du côté du serveur quand on utilise le mécanisme d'accréditif à long terme tout en s'assurant de la compatibilité avec MD5, qui était l'algorithme utilisé dans la [RFC5389]. Ce choix fonctionne en ayant le serveur qui envoie au client la liste des algorithmes pris en charge dans un attribut PASSWORD-ALGORITHMS et le client qui renvoie un attribut PASSWORD-ALGORITHM contenant l'algorithme choisi.

Parce que l'attribut PASSWORD-ALGORITHMS doit être envoyé dans une réponse non authentifiée, un attaquant sur le chemin qui veut exploiter une éventuelle vulnérabilité de MD5 peut juste supprimer l'attribut PASSWORD-ALGORITHMS de la réponse non protégée, faisant donc agir ensuite le serveur comme si le client mettait en œuvre la version de ce protocole définie dans la [RFC5389].

Pour protéger contre cette attaque et autres attaques en dégradation similaires, le nom occasionnel est enrichi d'un ensemble de bits de sécurité qui indiquent quelles caractéristiques de sécurité sont utilisées. Dans le cas du choix de l'algorithme de mot de passe, le bit correspondant est établi dans le nom occasionnel retourné par le serveur dans la même réponse qui contient l'attribut PASSWORD-ALGORITHMS. Parce que le nom occasionnel utilisé dans les transactions authentifiés suivantes est vérifié par le serveur comme étant identique à celui originellement envoyé, il ne peut pas être modifié par un attaquant sur le chemin. De plus, le client est obligé de copier l'attribut PASSWORD-ALGORITHMS reçu

dans la prochaine transaction authentifiée à ce serveur.

Un attaquant sur le chemin qui retire le PASSWORD-ALGORITHMS va être détecté parce que le client ne va pas être capable de le renvoyer au serveur dans la prochaine transaction authentifiée. Le client va détecter cette attaque parce que le bit de sécurité est établi mais l'attribut correspondant manque ; cela va terminer la session. Un client qui utilise une plus vieille version de ce protocole ne va pas renvoyer le PASSWORD-ALGORITHMS mais peut de toutes façons seulement utiliser MD5, de sorte que l'attaque est sans conséquence.

L'attaquant sur le chemin peut aussi essayer de supprimer le bit de sécurité ainsi que l'attribut PASSWORD-ALGORITHMS, mais le serveur va découvrir cela quand la prochaine transaction authentifiée contient un nom occasionnel invalide.

Un attaquant sur le chemin qui retire certains algorithmes de l'attribut PASSWORD-ALGORITHMS va également être mis en échec parce que cet attribut va être différent de l'original quand le serveur va le vérifier dans la transaction authentifiée suivante.

Noter que le mécanisme de protection contre la dégradation introduit dans ce document est par nature limité par le fait qu'il n'est pas possible de détecter une attaque tant que le serveur ne reçoit pas la seconde demande après la réponse 401 (Non authentifié).

SHA-256 a été choisi comme nouveau hachage de mot de passe par défaut pour sa compatibilité avec la [RFC7616], mais parce que SHA-256 (comme MD5) est un algorithme relativement rapide, il fait peu pour empêcher les attaques en force brute. Précisément, cela signifie que si l'utilisateur a un mot de passe faible, un attaquant qui capture un seul échange peut utiliser une attaque en force brute pour apprendre le mot de passe de l'utilisateur et ensuite potentiellement se faire passer pour l'utilisateur auprès du serveur et d'autres serveurs où le même mot de passe était utilisé. Noter qu'un tel attaquant peut se faire passer pour l'utilisateur auprès du serveur lui-même sans aucune attaque en force brute.

Un algorithme plus fort (‘est-à-dire, plus lent) comme Argon2 [RFC9106], va aider dans ces deux cas ; cependant, dans le premier cas, il va seulement aider après que l'entrée de base de données pour cet usager est mise à jour pour utiliser exclusivement ce mécanisme plus fort.

Les défenses contre la dégradation dans ce protocole empêchent un attaquant de forcer le client et le serveur d'achever une prise de contact en utilisant des algorithmes plus faibles que ceux qu'ils prennent conjointement en charge, mais seulement si l'algorithme plus faible commun est assez fort pour qu'il ne puisse pas être compromis par une attaque en force brute. Cependant, cela ne défend pas contre de nombreuses attaques sur ces algorithmes ; précisément, un attaquant sur le chemin pourrait effectuer une attaque en dégradation sur un client qui prend en charge à la fois Argon2 [RFC9106] et SHA-256 pour le hachage de mot de passe et utiliser cela pour collecter une valeur de MESSAGE-INTEGRITY-SHA256 qu'il peut alors utiliser pour une attaque en force brute hors ligne. Cela va être détecté quand le serveur va recevoir la seconde demande, mais cela n'empêche pas l'attaquant d'obtenir la valeur de MESSAGE-INTEGRITY-SHA256.

De même, une attaque contre le mécanisme USERHASH ne va pas réussir à établir une session parce que le serveur va détecter que la caractéristique a été éliminée sur le chemin, mais le client aurait quand même été convaincu d'envoyer son nom d'utilisateur en clair dans l'attribut USERNAME, le divulguant donc à l'attaquant.

Finalement, quand le mécanisme de protection contre la dégradation est employé pour une future mise à niveau de l'algorithme HMAC utilisé pour protéger les messages, il va offrir seulement une protection limitée si l'algorithme HMAC actuel est déjà compromis.

16.2 Attaques affectant l'usage

Ce paragraphe fait la liste des attaques qui pourraient être lancées contre un usage de STUN. Chaque usage STUN doit examiner si ces attaques lui sont applicables, et si elles le sont, discuter des contre-mesures.

La plupart des attaques de ce paragraphe tournent autour d'un attaquant qui modifie l'adresse réflexive apprise par un client STUN par une transaction de demande/réponse Binding. Comme l'usage de l'adresse réflexive est une fonction de l'usage, l'applicabilité et les remèdes à ces attaques sont spécifiques de l'usage. Dans les situations courantes, la modification de l'adresse réflexive par un attaquant sur le chemin est facile à faire. Considérons, par exemple, la situation courante où STUN fonctionne directement sur UDP. Dans ce cas, un attaquant sur le chemin peut modifier l'adresse IP de source de la demande Binding avant qu'elle arrive au serveur STUN. Le serveur STUN va alors retourner cette adresse IP dans l'attribut XOR-MAPPED-ADDRESS au client et renvoyer la réponse à cette adresse et accès IP (falsifiés). Si l'attaquant peut aussi

intercepter cette réponse, il peut la rediriger sur le client. Protéger contre cette attaque en utilisant une vérification d'intégrité de message est impossible, car une valeur d'intégrité de message ne peut pas couvrir l'adresse IP de source et le NAT intermédiaire doit être capable de modifier cette valeur. Une solution pour empêcher les attaques mentionnées ci-dessous est plutôt que le client vérifie l'adresse réflexive apprise, comme c'est fait dans ICE [RFC8445].

D'autres usages peuvent utiliser d'autres moyens pour empêcher ces attaques.

16.2.1 Attaque I : DoS réparti contre une cible

Dans cette attaque, l'attaquant fournit à un ou plusieurs clients la même adresse réflexive falsifiée qui pointe sur la cible destinée. Cela va amener les clients STUN à penser à tort que leurs adresses réflexives sont égales à celle de la cible. Si les clients prennent cette adresse réflexive pour recevoir le trafic (par exemple, dans des messages SIP) le trafic va plutôt être envoyé à la cible. Cette attaque peut fournir une amplification substantielle, en particulier quand utilisée avec des clients qui utilisent STUN pour permettre des applications multimédia. Cependant, elles peuvent seulement être lancées contre des cibles pour lesquelles les paquets du serveur STUN à la cible passent à travers l'attaquant, limitant les cas où elle est possible.

16.2.2 Attaque II : réduire un client au silence

Dans cette attaque, l'attaquant fournit à un client STUN une fausse adresse réflexive. L'adresse réflexive qu'il fournit est une adresse de transport qui achemine sur nulle part. Par suite, le client ne va recevoir aucun des paquets qu'il s'attend à recevoir quand il distribue l'adresse réflexive. Cette exploitation n'est pas très intéressante pour l'attaquant. Elle impacte un seul client, qui n'est fréquemment pas la cible désirée. De plus, tout attaquant qui peut monter l'attaque pourrait aussi dénier le service au client par d'autres moyens, comme d'empêcher le client de recevoir de réponse du serveur STUN, ou même d'un serveur DHCP. Comme avec l'attaque décrite au paragraphe 16.2.1, cette attaque est seulement possible quand l'attaquant est sur le chemin des paquets envoyés du serveur STUN vers cette adresse IP inutilisée.

16.2.3 Attaque III : prendre l'identité d'un client

Cette attaque est similaire à l'attaque II. Cependant, la fausse adresse réflexive pointe sur l'attaquant lui-même. Cela permet à l'attaquant de recevoir le trafic qui était destiné au client.

16.2.4 Attaque IV : espionnage

Dans cette attaque, l'attaquant force le client à utiliser une adresse réflexive qui achemine sur lui-même. Il transmet alors tous les paquets qu'il reçoit au client. Cette attaque permet à l'attaquant d'observer tous les paquets envoyés au client. Cependant, afin de lancer l'attaque, l'attaquant doit avoir déjà été capable d'observer les paquets du client au serveur STUN. Dans la plupart des cas (comme quand l'attaque est lancée depuis un réseau d'accès) cela signifie que l'attaquant pourrait déjà observer les paquets envoyés au client. Cette attaque est, par suite, seulement utile pour observer le trafic par des attaquants sur le chemin du client au serveur STUN, mais pas généralement sur le chemin des paquets acheminés vers le client.

Noter que cette attaque peut être lancée de façon triviale par le serveur STUN lui-même, de sorte que les utilisateurs de serveurs STUN devraient avoir le même niveau de confiance dans les utilisateurs de serveurs STUN que tout autre nœud qui peut s'insérer lui-même dans le flux de communication.

16.3 Plan d'agilité de hachage

La présente spécification utilise HMAC-SHA256 pour le calcul de l'intégrité de message, parfois en combinaison avec HMAC-SHA1. Si, plus tard, HMAC-SHA256 se trouve être compromis, le remède suivant devrait être appliqué :

- Un nouvel attribut d'intégrité de message et un nouveau bit de caractéristique de sécurité STUN vont être alloués dans un document sur la voie de la normalisation. Le nouvel attribut d'intégrité de message va avoir sa valeur calculée en utilisant un nouveau hachage. Le bit de caractéristique de sécurité STUN va être utilisé pour simultanément 1) signaler au client STUN qui utilise le mécanisme d'accréditif à long terme que ce serveur prend en charge ce nouvel algorithme de hachage et 2) empêcher les attaques en dégradation sur le nouvel attribut d'intégrité de message.

- Les clients et serveurs STUN qui utilisent le mécanisme d'accréditif à court terme vont devoir mettre à jour jour le

mécanisme externe qu'ils utilisent pour signaler quels attributs d'intégrité de message sont utilisés.

Le mécanisme de protection contre la dégradation décrit dans ce document est nouveau et donc ne peut pas actuellement protéger contre une attaque en dégradation qui diminue la force de l'algorithme de hachage à HMAC-SHA1. C'est pourquoi, après une période de transition, un nouveau document qui mettra celui-ci à jour allouera un nouveau bit de caractéristique de sécurité STUN pour déconseiller HMAC-SHA1. Quand il sera utilisé, ce bit signalera que HMAC-SHA1 est déconseillé et ne devrait plus être utilisé.

De même, si HMAC-SHA256 se trouve être compromis, un nouvel attribut userhash et un nouveau bit de caractéristique de sécurité STUN seront alloués dans un document sur la voie de la normalisation. Le nouvel attribut userhash aura sa valeur calculée en utilisant un nouveau hachage. Le bit de caractéristique de sécurité STUN sera utilisé pour simultanément 1) signaler à un client STUN qui utilise le mécanisme d'accréditif à long terme que ce serveur prend en charge ce nouvel algorithme de hachage pour l'attribut userhash et 2) empêcher les attaques en dégradation sur le nouvel attribut userhash.

17. Considérations de l'IAB

L'IAB a étudié le problème de l'auto correction d'adressage unilatérale (UNSAF, *Unilateral Self-Address Fixing*) qui est le processus général par lequel un client tente de déterminer son adresse dans un autre domaine de l'autre côté d'un NAT par un mécanisme de réflexion d'un protocole collaboratif [RFC3424]. STUN peut être utilisé pour effectuer cette fonction en utilisant une transaction de demande/réponse Binding si un agent est derrière un NAT et l'autre est du côté public du NAT.

L'IAB a suggéré que les protocoles développés à cette fin documentent un ensemble spécifique de considérations. Parce que certains usages STUN fournissent des fonctions UNSAF (comme ICE [RFC8445]) et d'autres ne le font pas (comme SIP sortant [RFC5626]) les réponses à ces considérations doivent être traitées par les usages eux-mêmes.

18. Considérations relatives à l'IANA

18.1 Registre des caractéristiques de sécurité de STUN

Un ensemble de caractéristique de sécurité STUN définit 24 bits comme des fanions.

L'IANA a créé un nouveau registre contenant les caractéristiques de sécurité STUN qui sont protégées par le mécanisme de prévention d'attaque en dégradation décrite au paragraphe 9.2.1.

Les caractéristique de sécurité STUN initiales sont :
Bit 0 : algorithmes de mot de passe
Bit 1 : anonymat du nom d'utilisateur
Bits 2 à 23 : non alloués
Les bits sont alloués en commençant par le côté de moindre poids de bit établi, de sorte que le bit 0 est le bit de droite, et le bit 23 est le bit de gauche

Les nouvelles caractéristique de sécurité sont allouées par action de normalisation [RFC8126].

18.2 Registre des méthodes de STUN

Une méthode STUN est un nombre hexadécimal dans la gamme de 0x000 à 0x0FF. Le codage d'une méthode STUN dans un message STUN est décrit à la Section 5.

Les méthodes STUN dans la gamme de 0x000 à 0x07F sont allouées par revue de l'IETF [RFC8126]. Les méthodes STUN dans la gamme de 0x080 à 0x0FF sont allouées par revue d'expert [RFC8126]. La responsabilité de l'expert est de vérifier que les codets choisis ne sont pas utilisés et que la demande n'est pas pour un nombre anormalement grand de codets. La revue technique de l'extension elle-même sort du domaine de la responsabilité de l'expert désigné.

L'IANA a mis à jour le nom de la méthode 0x002 comme décrit ci-dessous ainsi que la référence de la RFC 5389 à la RFC 8489 pour les méthodes STUN suivantes :

0x000 : réservé

0x001 : Binding
0x002 : réservé; était SharedSecret avant la [RFC5389]

18.3 Registre des attributs de STUN

Un type d'attribut STUN est un nombre hexadécimal dans la gamme de 0x0000 à 0xFFFF. Les types d'attribut STUN dans la gamme de 0x0000 à 0x7FFF sont considérés comme de compréhension exigée ; les types d'attribut STUN dans la gamme de 0x8000 à 0xFFFF sont considérés comme de compréhension facultative. Un agent STUN traite différemment les attributs inconnus de compréhension exigée et de compréhension facultative.

Les types d'attribut STUN dans la première moitié de la gamme de compréhension exigée (0x0000 à 0x3FFF) et dans la première moitié de la gamme de compréhension facultative (0x8000 à 0xBFFF) sont alloués par revue de l'IETF [RFC8126]. Les types d'attribut STUN dans la seconde moitié de la gamme de compréhension exigée (0x4000 à 0x7FFF) et dans la seconde moitié de la gamme de compréhension facultative (0xC000 à 0xFFFF) sont alloués par revue d'expert [RFC8126]. La responsabilité de l'expert est de vérifier que les codets choisis ne sont pas utilisés et que la demande n'est pas pour un nombre anormalement grand de codets. La revue technique de l'extension elle-même sort du domaine de compétence de l'expert désigné.

18.3.1 Attributs mis à jour

L'IANA a mis à jour les noms des attributs 0x0002, 0x0004, 0x0005, 0x0007, et 0x000B ainsi qu'a mis à jour la référence de la RFC 5389 à la RFC 8489 pour chacune des méthodes STUN suivantes. De plus, la [RFC5389] introduisait une faute dans le nom de l'attribut 0x0003 ; la [RFC5389] l'appelait CHANGE-ADDRESS quand il était en fait précédemment appelé CHANGE-REQUEST. Donc, l'IANA a mis à jour la description de 0x0003 comme "réservé ; était CHANGE-REQUEST avant la [RFC5389]".

Gamme de compréhension exigée (0x0000 à 0x7FFF) :
0x0000 : réservé
0x0001 : MAPPED-ADDRESS
0x0002 : réservé; était RESPONSE-ADDRESS avant la [RFC5389]
0x0003 : réservé; était CHANGE-REQUEST avant la [RFC5389]
0x0004 : réservé; était SOURCE-ADDRESS avant la [RFC5389]
0x0005 : réservé; était CHANGED-ADDRESS avant la [RFC5389]
0x0006 : USERNAME
0x0007 : réservé; était PASSWORD avant la [RFC5389]
0x0008 : MESSAGE-INTEGRITY
0x0009 : ERROR-CODE
0x000A : UNKNOWN-ATTRIBUTES
0x000B : réservé ; était REFLECTED-FROM avant la [RFC5389]
0x0014 : REALM
0x0015 : NONCE
0x0020 : XOR-MAPPED-ADDRESS

Gamme de compréhension facultative (0x8000 à 0xFFFF) :
0x8022 : SOFTWARE
0x8023 : ALTERNATE-SERVER
0x8028 : FINGERPRINT

18.3.2 Nouveaux attributs

L'IANA a ajouté l'attribut suivant au registre "Attributs STUN" :

Gamme de compréhension exigée (0x0000 à 0x7FFF) :
0x001C : MESSAGE-INTEGRITY-SHA256
0x001D : PASSWORD-ALGORITHM
0x001E : USERHASH

Gamme de compréhension facultative (0x8000 à 0xFFFF) :
0x8002 : PASSWORD-ALGORITHMS

0x8003 : ALTERNATE-DOMAIN

18.4 Registre des codes d'erreur STUN

Un code d'erreur STUN est un nombre dans la gamme de 0 à 699. Les codes d'erreur STUN sont accompagnés d'une phrase de raison textuelle en UTF-8 [RFC3629] qui est destinée seulement à la consommation humaine et peut être n'importe quoi d'approprié ; le présent document propose seulement des valeurs suggérées.

Les codes d'erreur STUN sont cohérents dans les allocations de codets et la sémantique avec SIP [RFC3261] et HTTP [RFC7231].

De nouveaux codes d'erreur STUN sont alloués sur la base d'une revue de l'IETF [RFC8126]. La spécification doit considérer avec attention comment les clients qui ne comprennent pas ce code d'erreur vont le traiter avant d'accorder la demande. Voir les règles du paragraphe 6.3.4.

L'IANA a mis à jour la référence de la RFC 5389 à la RFC 8489 pour les codes d'erreur définis au paragraphe 14.8.

L'IANA a changé le nom du code d'erreur 401 de "Non autorisé" en "Non authentifié".

18.5 Registre des algorithmes de mot de passe de STUN

L'IANA a créé un nouveau registre intitulé "Algorithmes de mot de passe STUN".

Un algorithme de mot de passe est un nombre hexadécimal dans la gamme de 0x0000 à 0xFFFF.

Le contenu initial du registre des "Algorithmes de mot de passe STUN" est le suivant :

0x0000 : réservé
0x0001 : MD5
0x0002 : SHA-256
0x0003-0xFFFF : non alloué

Les algorithmes de mot de passe dans la première moitié de la gamme (0x0000-0x7FFF) sont alloués par revue de l'IETF [RFC8126]. Les algorithmes de mot de passe dans la seconde moitié de la gamme (0x8000-0xFFFF) sont alloués sur revue d'expert [RFC8126].

18.5.1 Algorithmes de mot de passe

18.5.1.1 MD5

Cet algorithme de mot de passe est tiré de la [RFC1321].

La longueur de clé est 16 octets, et la valeur de paramètres est vide.

Note : cet algorithme DOIT seulement être utilisé pour la compatibilité avec les systèmes traditionnels.

clé = MD5(nom d'utilisateur ":" OpaqueString(domaine) ":" OpaqueString(mot de passe))

18.5.1.2 SHA-256

Cet algorithme de mot de passe est tiré de la [RFC7616].

La longueur de clé est 32 octets, et la valeur de paramètres est vide.

clé = SHA-256(nom d'utilisateur ":" OpaqueString(domaine) ":" OpaqueString(mot de passe))

18.6 Numéros d'accès UDP et TCP de STUN

L'IANA a mis à jour la référence de la RFC 5389 à la RFC 8489 pour les accès suivants dans le "Registre des noms de

service et des numéros d'accès de protocole de transport".

stun	3478/tcp	accès d'utilitaires de traversée de session pour les NAT (STUN)
stun	3478/udp	accès d'utilitaires de traversée de session pour les NAT (STUN)
stuns	5349/tcp	accès d'utilitaires de traversée de session pour les NAT (STUN)

19. Changements par rapport à la RFC 5389

La présente spécification rend obsolète la [RFC5389]. La présente spécification diffère de la RFC 5389 par ce qui suit :
- o Ajout de la prise en charge de DTLS sur UDP [RFC6347].
- o Précision que le RTO est considéré comme périmé si il n'y a pas de transaction avec le serveur.
- o Alignement du calcul du RTO sur la [RFC6298].
- o Mise à jour des suites de chiffrement pour TLS.
- o Ajout de la prise en charge de l'URI STUN [RFC7064].
- o Ajout de la prise en charge de l'intégrité de message SHA256.
- o Mise à jour de la prise en charge de la préparation, application, et comparaison des chaînes internationalisées (PRECIS) de la [RFC8265].
- o Ajout du protocole et du registre pour choisir l'algorithme de chiffrement de mot de passe.
- o Ajout de la prise en charge du nom d'utilisateur anonyme.
- o Ajout du protocole et registre pour la prévention des attaques en dégradation.
- o Spécifie que le partage d'un nom occasionnel n'est plus permis.
- o Ajout de la possibilité d'utiliser un nom de domaine dans le mécanisme de serveur de remplacement.
- o Ajout d'appliquettes en langage C.
- o Ajout de vecteur d'essai.

20. Références

20.1 Références normatives

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

[KARN87] Karn, P. et C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM '87, Proceedings of the ACM workshop on Frontiers in computer communications technology, Pages 2-7, DOI 10.1145/55483.55484, août 1987.

[RFC0791] J. Postel, éd., "Protocole Internet - Spécification du protocole du programme Internet", STD 5, DOI 10.17487/RFC0791, septembre 1981.

[RFC1122] R. Braden, "Exigences pour les hôtes Internet – couches de communication", STD 3, DOI 10.17487/RFC1122, octobre 1989. *(MàJ par* RFC6633, 8029, 9293)

[RFC1123] R. Braden, éditeur, "Exigences pour les hôtes Internet – Application et prise en charge", STD 3, DOI 10.17487/RFC1123, octobre 1989. *(MàJ par* RFC7766, RFC9210)

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992, DOI 10.17487/RFC1321, *(Information)*

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997, DOI 10.17487/RFC2104.

[RFC2782] A. Gulbrandsen, P. Vixie et L. Esibov, "Enregistrement de ressource DNS pour la spécification de la localisation des services (DNS SRV)", février 2000, DOI 10.17487/RFC2782.

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003, DOI 10.17487/RFC3629.

[RFC4648] S. Josefsson, "Codages de données Base16, Base32 et Base64", octobre 2006, DOI 10.17487/RFC4648,

(*Remplace* RFC3548) *(P.S.).*

[RFC5890] J. Klensin, "Noms de domaine internationalisés pour les applications (IDNA) : Définitions et cadre documentaire", août 2010, DOI 10.17487/RFC5890. *(Remplace RFC*3490) *(P.S.)*

[RFC6125] P. Saint-André, J. Hodges, "Représentation et vérification d'identité de service d'application fondé sur le domaine au sein de l'infrastructure Internet de clé publique utilisant les certificats X.509 (PKIX) dans le contexte de la sécurité de la couche Transport (TLS)", mars 2011, DOI 10.17487/RFC6125. *(P.S.)*

[RFC6151] S. Turner, L. Chen, "Mise à jour des considérations de sécurité pour les algorithmes de résumé de message MD5 et le HMAC-MD5", mars 2011, DOI 10.17487/RFC6151, (*MàJ* RFC1321, RFC2104) *(Information)*

[RFC6298] V. Paxson, M. Allman, J. Chu, M. Sargent, "Calcul du temporisateur de retransmission de TCP", DOI 10.17487/RFC6298, juin 2011. *(Remplace la* RFC2988) *(MàJ la* RFC1122) *(P.S.)*

[RFC6347] E. Rescorla, N. Modadugu, "Sécurité de la couche transport de datagrammes, version 1.2", DOI 10.17487/RFC6347, janvier 2012. *(Remplace la* RFC4347) *(P.S. ; MàJ par* RFC7905, 9146 ; *remplacée par* RFC9147)

[RFC7064] S. Nandakumar et autres, "Schéma d'URI pour protocole d'utilitaires de traversée de session pour les NAT (STUN)", novembre 2013, DOI 10.17487/RFC7064. *(P.S.)*

[RFC7350] M. Petit-Huguenin, G. Salgueiro, "Sécurité de la couche transport de datagrammes (DTLS) comme transport pour les utilitaires de traversée de session pour les NAT (STUN)", août 2014, DOI 10.17487/RFC7350. *(P.S.) (MàJ* 5389*,* 5928*)*

[RFC7616] R. Shekh-Yusef, et autres, "Authentification d'accès HTTP par résumé", septembre 2015, DOI 10.17487/RFC7616. *(P.S. ; Remplace* RFC2617*)*

[RFC8174] B. Leiba, "Ambiguïté des mots clés en majuscules ou minuscules dans la RFC2119", DOI 10.17487/RFC8174, mai 2017. BCP14. *(MàJ RFC*2119)

[RFC8200] S. Deering, R. Hinden, "Spécification du protocole Internet version 6 (IPv6)", juillet 2017, DOI 10.17487/RFC8200. STD 86. *(Remplace* 2460)

[RFC8265] P. Saint-André, A. Melnikov, "Préparation, application et comparaison de chaînes internationalisées représentant des noms d'utilisateur et des mots de passe", octobre 2017, DOI 10.17487/RFC8265. *(P.S. ; remplace* RFC7613*)*

[RFC8305] D. Schinazi, "Algorithme Happy Eyeballs version 2 : meilleure connexité en utilisant la concurrence", décembre 2017, DOI 10.17487/RFC8305. *(P.S. ; remplace la* RFC6555*)*

[V.42] Recommandation UIT-T V.42, "Procédures de correction d'erreur pour les équipements terminaux de circuit de données qui utilisent la conversion asynchrone à synchrone", mars 2002.

20.2 Références pour information

[RFC1952] P. Deutsch, "Spécification du format de fichier GZIP version 4.3", DOI 10.17487/RFC1952, mai 1996. *(Info)*

[RFC2279] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", janvier 1998, DOI 10.17487/RFC2279, (*Obsolète, voir* RFC3629) *(D.S.)*

[RFC3261] J. Rosenberg et autres, "SIP : Protocole d'initialisation de session", juin 2002, DOI 10.17487/RFC3261. (*Mise à jour par* 3265, 3853, 4320, 4916, 5393, 6665, 8217, 8760)

[RFC3424] L. Daigle, éd., IAB, "Considérations de l'IAB sur l'auto correction d'adressage unilatérale (UNSAF) à travers la traduction d'adresse réseau", novembre 2002, DOI 10.17487/RFC3424. *(Information)*

[RFC3454] P. Hoffman et M. Blanchet, "Préparation de chaînes internationalisées ("stringprep")", décembre 2002, DOI

10.17487/RFC3454. *(P.S.)*

[RFC3489] J. Rosenberg et autres, "STUN - Simple traversée par le protocole de datagramme d'utilisateur (UDP) des traducteurs d'adresse réseau (NAT)", mars 2003, DOI 10.17487/RFC3489. (*Obsolète, voir* RFC5389) *(P.S.)*

[RFC4013] K. Zeilenga, "SASLprep : Profil Stringprep pour les noms d'utilisateur et mots de passe", février 2005, DOI 10.17487/RFC4013.

[RFC4107] S. Bellovin, R. Housley, "Lignes directrices pour la gestion des clés de chiffrement", juin 2005, DOI 10.17487/RFC4107, (BCP0107)

[RFC5090] B. Sterman et autres, "Extension à RADIUS pour l'authentification par résumé", février 2008, DOI 10.17487/RFC5090, (*Remplace* RFC4590) *(P.S.)*

[RFC5389] J. Rosenberg et autres, "Utilitaires de traversée de session pour les NAT (STUN)", octobre 2008, DOI 10.17487/RFC5389. *(P.S. ; Remplace* RFC3489 *; remplacée par RFC*8489)

[RFC5626] C. Jennings, R. Mahy, F. Audet, éd., "Gestion des connexions initiées par le client dans le protocole d'initialisation de session (SIP)", octobre 2009, DOI 10.17487/RFC5626. *(MàJ* RFC3261, RFC3327) *(P. S.)*

[RFC5766] R. Mahy, P. Matthews, J. Rosenberg, "Traversée de NAT au moyen d'un relais (TURN) : Extensions de relais aux utilitaires de traversée de session pour les NAT (STUN)", avril 2010, DOI 10.17487/RFC5766. *(P. S. ; MàJ par* RFC8155 *; Remplacée par* RFC8656)

[RFC5769] R. Denis-Courmont, "Vecteurs d'essais pour les utilitaires de traversée de session pour les NAT (STUN)", avril 2010, DOI 10.17487/RFC5769. *(Information)*

[RFC5780] D. MacDonald, B. Lowekamp, "Découverte du comportement de NAT avec les utilitaires de traversée de session pour les NAT (STUN)", mai 2010, DOI 10.17487/RFC5780. *(Expérimentale)*

[RFC6544] J. Rosenberg et autres, "Candidats TCP avec établissement de connexité interactive (ICE)", mars 2012, DOI 10.17487/RFC6544. *(P.S.)*

[RFC7231] R. Fielding, et J. Reschke, "Protocole de transfert Hypertexte (HTTP/1.1) : sémantique et contenu", juin 2014, DOI 10.17487/RFC7231. *(P.S. ; remplace* RFC2616, *MàJ* RFC2617 *; MàJ par* RFC9110*)*

[RFC7525] Y. Scheffer, R. Holz, P. Saint-André, "Recommandations pour l'utilisation sûre de TLS et DTLS", DOI 10.17487/RFC7525, mai 2015. BCP195.

[RFC8126] M. Cotton, B. Leiba, T. Narten, "Lignes directrices pour la rédaction d'une section de considérations relatives à l'IANA dans les RFC", juin 2017. BCP 26, DOI 10.17487/RFC8126, *(Remplace* RFC5226)

[RFC8264] P. Saint-André, M. Blanchet, "Cadre PRECIS : Préparation, application et comparaison de chaînes internationalisées dans les protocoles d'application", octobre 2017, DOI 10.17487/RFC8264. *(P.S. ; remplace* RFC7564*)*

[RFC8445] A. Keranen, et autres, "Établissement de connexité interactive (ICE) : un protocole pour la traversée de NAT", DOI 10.17487/RFC8445, juillet 2018. *(P.S. ; remplace la* RFC 5245 *; MàJ par* RFC8863)

[RFC8446] E. Rescorla, "Version 1.3 du protocole de sécurité de la couche transport (TLS)", DOI 10.17487/RFC8446, août 2018. *(P.S. ; MàJ* RFC5705, 6066 *; rend obsolètes les* RFC 5077, 5246, 6961)

[RFC9106] A. Biryukov, D. Dinu, D. Khovratovich, S. Josefsson, "Fonction Argon2 de mémoire dure pour hachage de mot de passe et applications de preuve de travail", septembre 2021. (DOI : 10.17487/RFC9106) *(Info.*)

[PMTUD] M. Petit-Huguenin, G. Salgueiro, et F. Garrido, "Packetization Layer Path MTU Discovery (PLMTUD) For UDP Transports Using Session Traversal Utilities for NAT (STUN)", Travail en cours, draft-ietf-tram-stun-pmtud-15, décembre 2019.

[UAX15] Unicode Standard Annex #15, "Unicode Normalization Forms", édité par Mark Davis et Ken Whistler. Partie intégrante de la norme Unicode, <http://unicode.org/reports/tr15/ >.

Appendice A. Appliquette en langage C pour déterminer le type de message STUN

Étant donnée une valeur de type de message STUN de 16 bits dans l'ordre des octets de l'hôte dans le paramètre msg_type, les macros en langage C ci-dessous déterminent les types de message STUN :

```
<CODE BEGINS>
#define IS_REQUEST(msg_type)       (((msg_type) & 0x0110) == 0x0000)
#define IS_INDICATION(msg_type)    (((msg_type) & 0x0110) == 0x0010)
#define IS_SUCCESS_RESP(msg_type)  (((msg_type) & 0x0110) == 0x0100)
#define IS_ERR_RESP(msg_type)      (((msg_type) & 0x0110) == 0x0110)
<CODE ENDS>
```

Une fonction pour convertir méthode et classe en un type de message :

```
<CODE BEGINS>
int type(int method, int cls) {
  return (method & 0x1F80) << 2 | (method & 0x0070) << 1
    | (method & 0x000F) | (cls & 0x0002) << 7
    | (cls & 0x0001) << 4;
  }
<CODE ENDS>
```

Une fonction pour extraire la méthode du type de message :

```
<CODE BEGINS>
int method(int type) {
  return (type & 0x3E00) >> 2 | (type & 0x00E0) >> 1
    | (type & 0x000F);
  }
<CODE ENDS>
```

Une fonction pour extraire la classe du type de message :

```
<CODE BEGINS>
int cls(int type) {
  return (type & 0x0100) >> 7 | (type & 0x0010) >> 4;
  }
<CODE ENDS>
```

Appendice B. Vecteurs d'essais

Cette section augmente la liste des vecteurs d'essai définie dans la [RFC5769] avec MESSAGE-INTEGRITY-SHA256. Tous les formats et définitions mentionnés à la Section 2 de la [RFC5769] s'appliquent ici.

B.1 Exemple de demande avec authentification à long terme avec MESSAGE-INTEGRITY-SHA256 et USERHASH

Cette demande utilise les paramètres suivants :

Noom d'utilisateur : "<U+30DE><U+30C8><U+30EA><U+30C3><U+30AF><U+30B9>" (sans les guillemets) non affecté par le traitement OpaqueString [RFC8265]

Mot de passe : "Le <U+00AD>M<U+00AA>tr<U+2168>" et "TheMatrIX" (sans les guillemets) respectivement avant et après le traitement OpaqueString [RFC8265]

Nom occasionnel : "obMatJos2AAACf//499k954d6OL34oL9FSTvy64sA" (sans les guillemets)

Domaine : "example.org" (sans les guillemets)

Algorithme de mot de passe : SHA-256 (0x0002), et longueur de paramètres (0)

00 01 00 90	Type de demande et longueur de message
21 12 a4 42	Mouchard magique
78 ad 34 33 }	
c6 ad 72 c0 }	Identifiant de transaction
29 da 41 2e }	
00 1e 00 20	En-tête d'attribut USERHASH
4a 3c f3 8f }	
ef 69 92 bd }	
a9 52 c6 78 }	
04 17 da 0f }	Valeur de Userhash (32 octets)
24 81 94 15 }	
56 9e 60 b2 }	
05 c4 6e 41 }	
40 7f 17 04 }	
00 15 00 29	En-tête d'attribut NONCE
6f 62 4d 61 }	
74 4a 6f 73 }	
32 41 41 41 }	
43 66 2f 2f }	
34 39 39 6b }	Valeur de nom occasionnel et bourrage (3 octets)
39 35 34 64 }	
36 4f 4c 33 }	
34 6f 4c 39 }	
46 53 54 76 }	
79 36 34 73 }	
41 00 00 00 }	
00 14 00 0b	En-tête d'attribut REALM
65 78 61 6d }	
70 6c 65 2e }	Valeur de Realm (11 octets) et bourrage (1 octet)
6f 72 67 00 }	
00 1d 00 04	En-tête d'attribut PASSWORD-ALGORITHM
00 02 00 00	Valeur de PASSWORD-ALGORITHM (4 octets)
00 1c 00 20	En-tête d'attribut MESSAGE-INTEGRITY-SHA256
b5 c7 bf 00 }	
5b 6c 52 a2 }	
1c 51 c5 e8 }	
92 f8 19 24 }	Valeur de HMAC-SHA256
13 62 96 cb }	
92 7c 43 14 }	
93 09 27 8c }	
c6 51 8e 65 }	

Remerciements

Merci à Michael Tuexen, Tirumaleswar Reddy, Oleg Moskalenko, Simon Perreault, Benjamin Schwartz, Rifaat Shekh-Yusef, Alan Johnston, Jonathan Lennox, Brandon Williams, Olle Johansson, Martin Thomson, Mihaly Meszaros, Tolga Asveren, Noriyuki Torii, Spencer Dawkins, Dale Worley, Matthew Miller, Peter Saint-Andre, Julien Elie, Mirja Kuehlewind, Eric Rescorla, Ben Campbell, Adam Roach, Alexey Melnikov, et Benjamin Kaduk pour les commentaires, suggestions, et questions qui ont aidé à améliorer ce document.

La section des remerciements de la RFC 5389 apparaît comme suit :

Les auteurs tiennent à remercier Cedric Aoun, Pete Cordell, Cullen Jennings, Bob Penfield, Xavier Marjou, Magnus Westerlund, Miguel Garcia, Bruce Lowekamp, et Chris Sullivan de leues commentaires, et Baruch Sterman et Alan Hawrylyshen de leurs mises en œuvre initiales. Merci à Leslie Daigle, Allison Mankin, Eric Rescorla, et Henning Schulzrinne pour les apports de l'IESG et de l'IAB sur ce travail.

Contributeurs : Christian Huitema et Joel Weinberger sont les co-auteurs originaux de la RFC 3489.

Adresse des auteurs

Marc Petit-Huguenin
Impedance Mismatch
mél : marc@petit-huguenin.org

Gonzalo Salgueiro
Cisco
mél : gsalguei@cisco.com

Jonathan Rosenberg
Five9
mél : jdrosen@jdrosen.net

Dan Wing
Citrix Systems, Inc.
mél : dwing-ietf@fuggles.com

Rohan Mahy
mél : rohan.ietf@gmail.com

Philip Matthews
Nokia
mél : philip_matthews@magma.ca