rfc5906

NTPv4 - Spécification Autokey

Un service réseau distribué nécessite un provisionnement sûr, non-ambigus et survivable pour empêcher des accidents ou des attaques malicieuses sur les serveurs et clients dans le réseau ou sur les valeurs échangées. La fiabilité nécessite que les clients puissent déterminer que les paquets reçus sont authentiques ; c'est à dire, envoyés pas le serveur attendu qu'un client peut valider l'authenticité en utilisant des informations publiques uniquement.

Ce mémo décris une méthodologie à utiliser dans NTPv4. Les divers schémas d'agréments de clé proposés nécessitent des variables d'état par association, qui contredisent les principes du paradigme RPC dans lequel les serveurs ne conservent aucun état pour une population possiblement très grande. Une évaluation du modèle PKI et des algorithmes, par ex, tel qu'implémenté dans tout paquet NTP pour gérer une signature numérique PKI résulte dans des performances de temps inacceptablement pauvre.

Le protocole Autokey est basé sur une combinaison de PKI et une séquence pseudo-aléatoire générée par des hashs répétés d'une valeur cryptographique impliquant des composants publiques et privées. Ce schéma a été implémenté, testé, et déployé dans l'Internet d'aujourd'hui. Une description détaillée du modèle de sécurité, principes de design, et d'implémentation sont présentés dans ce mémo.

Modèle de sécurité NTP

Les exigences de sécurité NTP sont plus stricts que la plupart des services distribués. D'abord, l'opération du mécanisme d'authentification et le mécanisme de synchronisation de temps sont inextricablement liés. La synchronisation de temps fiable exige des clés cryptographiques qui sont valides seulement quand les serveurs participants et les clients sont synchronisés en UTC.

Un client peut se prétendre authentique aux applications dépendantes seulement si tous les serveurs dans le chemin depuis les serveurs primaires sont authentiques. Afin de souligner cette exigence, dans ce mémo, la notion d'authentique est remplacée par proventique, un adjectif dérivé de provenance. Dans NTP, chaque serveur authentifie la strate inférieur et proventique les serveurs de strate inférieur.

Il est important de noter que la notion de proventique n'implique pas nécessairement que le temps est correct. Un client NTP mobilise des associations concurrentes avec différents serveurs et utilise un algorithme d'accord pour sélectionner les truechimers. Une association particulière est proventique si le certificat serveur est l'identité ont été vérifiés. Cependant, la déclaration "le client est synchronisé à des sources proventiques" signifie que l'horloge système est définie en utilisant des valeurs de temps d'une ou plusieurs associations proventiques et en accord avec les algorithmes de mitigation NTP.

Pendant des années IETF a définis et amélioré l'infrastructure IPsec pour la protection des données et l'authentification de la source dans l'Internet. L'infrastructure inclus ESP (rfc4303) et AH (rfc4302) pour IPv4 et IPv6. Les algorithmes cryptographiques qui utilisent ces en-têtes incluent ceux développés pour la PKI, incluant des algorithmes de hash, signature et d'agréments. Ce mémo ne prend pas position sur les algorithmes utilisés. C'est établis par un profile pour chaque communauté d'utilisateurs.

Approche

Le protocole Autokey est conçus pour répondre aux objectifs suivants :

1. Il doit interropérer avec le modèle d'architecture NTP existant et le design du protocole. En particulier, il doit supporter le schéma de clé symétrique décris dans la rfc1305.
2. Il doit fournir une collection de valeurs cryptographiques et de valeurs de temps indépendants.

3. Il ne doit pas significativement dégrader la précision potentielle des algorithmes de synchronisation NTP.
4. Il doit être résistant aux attaques cryptographiques
5. Il doit permettre une large palette d'algorithmes cryptographiques.
6. Il doit fonctionner dans tous les modes supportés par NTP.

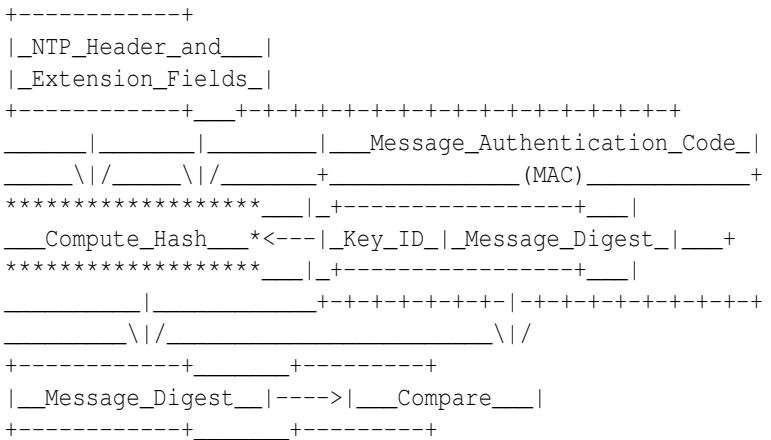
Cryptographie Autokey

La cryptographie Autokey est basée sur les algorithems PKI communément utilisés dans les applications SSH et SSL. Comme dans ces applications, Autokey utilise les message digest pour détecter les modifications de paquet, les signatures numérique pour vérifier les accréditifs, et les certificats publiques pour fournir une autorité traçable. e qui rend Autokey cryptographiquement unique est la manière dans laquelle les algorithmes sont utilisés pour dévier les attaquants tout en maintenant l'intégrité et la précision des fonction de synchronisation de temps.

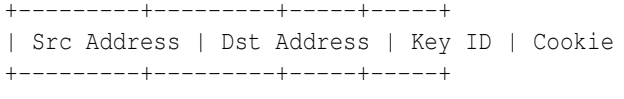
Autokey, comme de nombreuses autres protocoles RPC dépendent des messages digest pour l'authentification de base ; cependant, il est important de comprendre que les messages digest sont également utilisés par NTP quand Autokey n'est pas disponible ou non configuré. La sélection de l'algorithme de hashage est une fonction de NTP est est transparent à Autokey.

Le design de protocole et l'implémentation de référence supporte les algorithmes de hashage 128-bit et 160-bits, chacun avec un ID de clé 32-bit. Pour rester compatible avec NTPv3, l'ID de clé NTPv4 est scindé en 2 parties.

La cryptographie à clé symétrique et à clé publique authentifient comme décris dans le schéma ci-dessous. Le serveur recherche la clé associée avec l'ID de clé et calcule le hash. L'ID de clé est le hash forment le MAC inclus dans le message. Le client fait le même calcule en utilisant sa copie local de la clé et compare le résultat avec le hash dans le MAC. Si les valeurs concordent, le message est authentique



Autokey utilise des clés de session artificielles, appelés autokeys, et une séquence pseudo-aléatoire précalculée d'autokeys qui sont sauvés dans la liste autokey. Le protocole Autokey opère séparément pour chaque association, donc il peut y avoir de nombreuses séquences autokey opérant indépendemment à la fois.



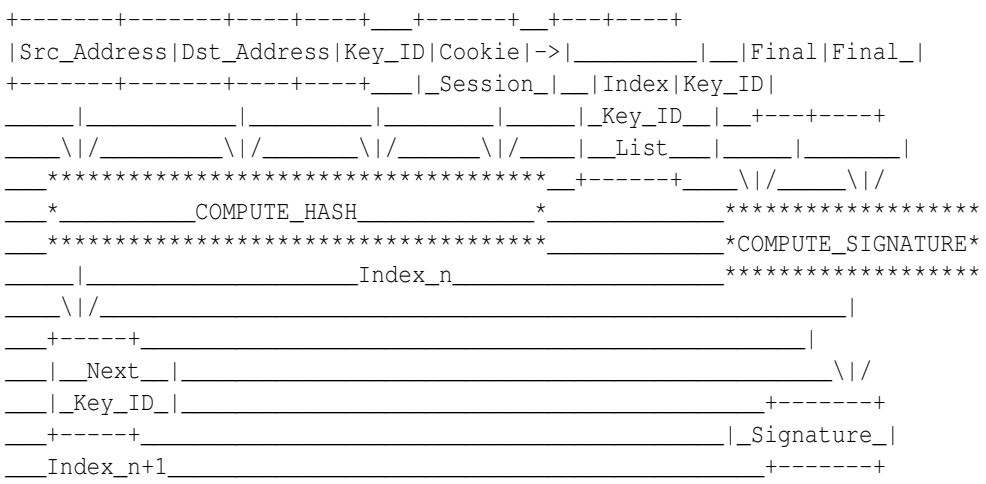
Une autokey est calculée depuis 4 champs dans l'ordre des octets réseaux. Les 4 valeurs sont hashés en utilisant l'algorithme MD5 pour produire la valeur autokey 128-bits, qui est stocké avec l'ID de clé dans un cache utilisé pour les clés symétriques comme les autokeys. Les clés sont récupérées depuis le cache par key ID en utilisant les tables de hash et un algorithme de recherche rapide.

Avec IPv4, les champs Src Address et Dst Address contiennent 32bits ; avec IPv6 ces champs contiennent 128-bits. Dans les 2 cas, les champs Key ID et Cookie contiennent 32 bits. Donc, une autokey IPv4 à 4 mots de 32 bits et IPv6 à 10 mots de 32 bits. Seul le champs

cookie n'est pas visible dans le paquet.

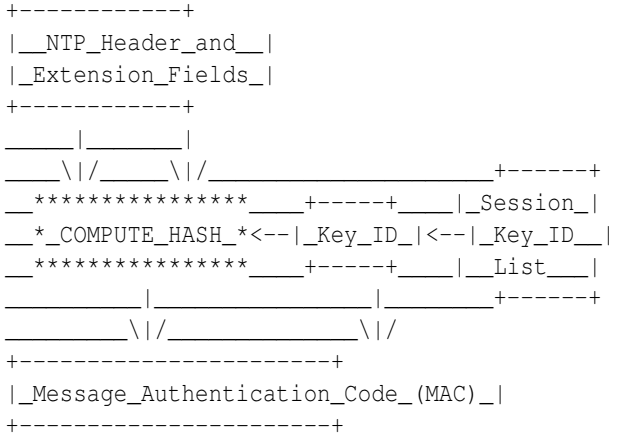
Le format de paquet NTP a été augmenté pour inclure un ou plusieurs champs d'extensions entre l'en-tête NTP et le MAC. Pour les paquets sans champs d'extension, le cookie est une valeur privée partagée. Pour les paquets avec des champs d'extension, le cookie a une valeur publique par défaut de 0, vu que ces paquets sont validés indépendamment en utilisant les signatures numériques.

Il y a certains scénarios où l'utilisation d'adresse IP terminal peut être difficile ou impossible. Cela inculs les configuration où un NAT est utilisé ou quand les adresse sont changées durant la durée de vie d'une association. Pour Autokey, la seule restriction est que les champs d'adresse qui sont visibles dans le paquet transmis doivent être les même que celles utilisées pour construire la liste autokey et que ces champs soient les même que ceux visible dans le paquet reçus.



Cette illustration montre comment la liste autokey et les valeurs autokey sont calculés. Les Key ID utilisé dans la liste autokey consiste d'une séquence avec un 32bits aléatoire supérieur ou égal au pivot du premier key ID. Le premier autokey est calculé comme ci-dessus en utilisant le cookie donné et autokey a l'index assigné 0. Les premiers 32 bits du résultat dans l'ordre d'octet réseau deviennent le prochain Key ID. Le hash MD5 d'autokey est la valeur de clé sauvée dans le cache de clé avec le Key ID. Les premiers 32 bits de la clé deviennent le Key ID pour la prochaine autokey assignée avec l'index 1.

Les opérations continuent pour générer la liste entière. Il peut arriver qu'un nouveau Key ID généré soit inférieur au pivot ou entre en collision avec un autre déjà généré. Quand cela se produit, la liste de clé est terminée à ce point. La durée de vie de chaque clé est définie pour expirer à un interval d'interrogation après sont utilisation. Dans l'implémentation de référence, la liste est terminée quand la durée de vie maximum de clé est d'environ une heure, donc pour les intervals d'interrogation supérieur à une heure, une nouvelle liste de clé contenant seulement une seule entrée est regénérée à chaque poll.



L'index de la dernière autokey dans la liste est sauvée avec l'id de clé pour cette entrée, collectivement appelée les valeurs autokey. Les valeurs autokey sont ensuite signées pour une utilisation ultérieure. La liste est utilisée dans l'ordre inverse pour que la première autokey

utilisé soit la dernière générée.

Le protocole Autokey inclus un message pour récupérer les valeurs autokey et vérifier la signature, donc les paquets suivants peuvent être validés en utilisant un ou plusieurs hash qui éventuellement matchent le dernier Key ID (valide) ou excède l'index (invalide). C'est appelé le test autokey dans la suite et est effectué pour chaque paquet, incluant ceux avec et sans champs d'extension. Dans l'implémentation de référence le Key ID le plus récent est sauvé pour comparaison avec les premiers 32 bits dans l'ordre d'octets réseau de la valeur de clé suivante. Cela minimise le nombre d'opérations de hash dans le cas où un paquet est perdu.

Le protocole Autokey inclus des échanges requête/réponse qui doivent être complétés dans l'ordre. Dans chaque échange, un client envoie une requête et attend un message de réponse d'une serveur. Les requêtes et les réponses sont contenus dans les champs d'extension. Un paquet NTP peut contenir un message requête et un ou plusieurs messages réponse. La suite liste ces messages.

Échange de paramètres La requête inclus le nom d'hôte du client et le status ; la réponse inclus le nom d'hôte du serveur et le status. Le status spécifie le schéma digest/signature à utiliser et le schéma d'intentité supportés.

Échange d'identité Le chemin du certificat n'est généralement pas considéré comme une protection suffisante contre les attaquets MITM sauf si une protection additionnelle tels qu'un schéma de preuve de possession (rfc2875) est disponible, mais c'est coûteux est exige que les serveurs retiennent l'état.

Échange de cookie La requête inclus la clé publique du serveur. La réponse inclus le cookie du serveur chiffré avec cette clé. Le client utilise cette valeur en construisant la liste de clé. La fin de cet échange active le bit COOK.

Échange Autokey La requête inclus soit aucune donnée ou les valeurs autokey en modes symétriques. La réponse inclus les valeurs autokey du serveur. Ces valeurs sont utilisées pour vérifier la séquence autokey. La fin de cet échange active le bit AUT

échange de signature Cet échange est exécuté seulement quand le client est synchronisé à une source proventique. La requête inclus le certificat client auto-signé. Le serveur agit comme une CA interprète le certificat comme requête de certificat X.509v3. Il extrait le sujet, émetteur, et les champs d'extension, construit un nouveau certificat avec un numéro de série et dates de validité, puis le signe avec sa clé privée et l'inclus dans la réponse. Le client utilise le certificat signé dans son propre rôle comme serveur pour des clients dépendants. La fin de cet échange active le bit SIGN.

Échange leapseconds Cet échange est exécuté seulement quand le client a été synchronisé à une source proventique. Cet échange se produit quand le serveur a les valeurs leapseconds, comme indiqué dans le status de l'hôte. Si c'est le cas, le client demande les valeurs et les compare avec ses propres valeurs, si disponible. Si les valeurs serveur sont plus récents que les valeurs client, le client remplace ses valeurs avec celles du serveur. Le client, agissant comme serveur, peut maintenant fournir les valeurs les plus récentes à ses clients. La fin de cet échange active le bit LPT.

Une fois le certificat et l'identitié validés, les paquets suivants sont validés par des signature numérique et la séquence autokey. L'association est maintenant proventique, mais n'est pas encore sélectionnable pour discipliner l'horloge système. Les associations accumulent les valeurs de temps, et les algorithmes de mitigation continuent de manière normale. Quand ces algorithmes ont supprimés les falsetickers et que cluster a au moins 3 survivants, l'horloge système est synchronisée à une source proventique.

Les valeurs de temps pour les sources truechimers forment un ordre partiel proventique relatif aux horodatages de signatures applicable. Cela soulèves le problème intéressant de comme différentier entre les horodatages des différentes associations. Il peut arriver, par exemple, que le timestamp d'un message Autokey soit est en devant l'horloge système. Pour cette raison, les comparaisons d'horodatage entre les différentes associations et entre les associations et les horloges système sont évités, excepté dans l'intersection NTP et les algorithmes cluster et en déterminant si un certificat a expiré.

Groupe sécurisé NTP

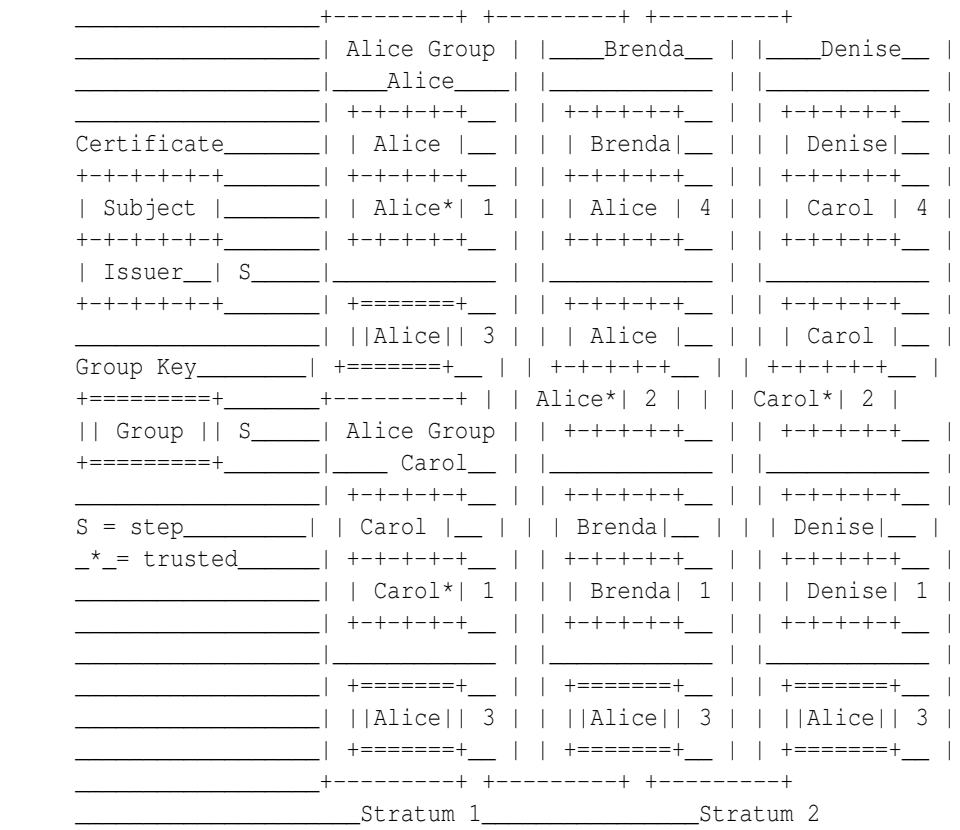
Les groupes de sécurité NTP sont utilisés pour définir des compartiments cryptographique et des hiérarchies sécurisés. Un groupe sécurisé consiste d'un nombre d'hôtes dynamiquement assemblés dans une forêt avec des racines d'hôtes de confiance (THs = Trusted Hosts) à la strate la plus basse du groupe. Les THs n'ont pas à être, mais le sont souvent, des serveurs primaires (strate 1). Une autorité de confiance (TA), pas nécessairement un hôte de groupe, génère des clé d'identité privée pour les serveurs et les clé d'identité publique pour les clients à la fin de la forêt. Le TA déploie les clés serveur aux THs et d'autres serveurs en utilisant des moyens sécurisés et poster les clés clients dans un site web publique.

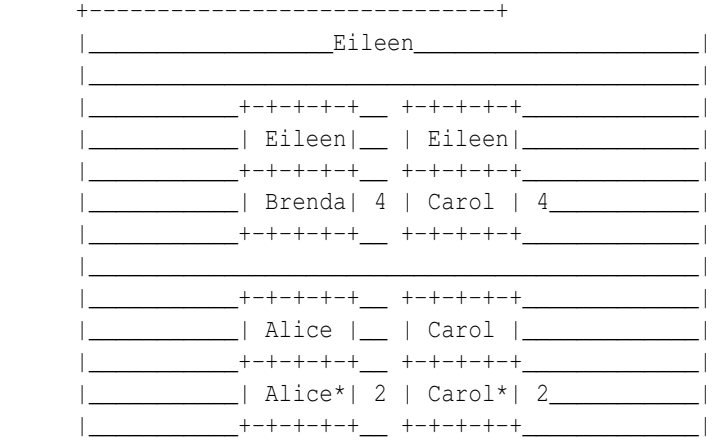
Pour Autokey, tous les hôtes appartiennent à un groupe sécurisé ayant le même nom de groupe mais différents noms d'hôte, par nécessairement liés aux noms DNS. Le nom de groupe est utilisé dans les champs subject et issuer des certificats TH ; le nom d'hôte est

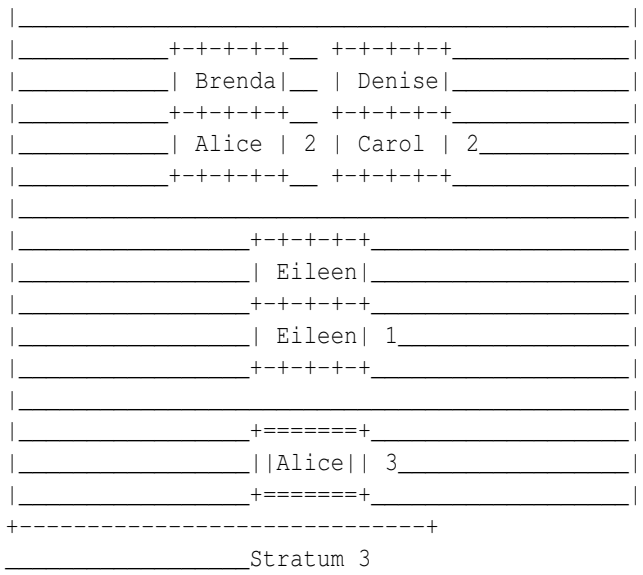
utilisé dans ces champs pour tous les hôtes. Donc, tous les certificats d'hôtes sont auto-signés. Durant l'utilisation d'Autokey, un client demande que le serveur signe son certificat et cache le résultat. Un chemin de certificat est construit par chaque hôte, possiblement via des hôtes intermédiaires et se terminant à un TH. Donc, chaque hôte dans le chemin récupère tout le chemin de ses serveurs et le fournis plus son propre certificat autosigné à ses clients.

Les groupes sécurisés peuvent être configurés comme hiérarchies où un TH d'un groupe peut être un client d'un ou plusieurs autres groupes opérant à une strate inférieure. Dans un scénario, les TH pour les groupes RED et GREEN peuvent être cryptographiquement distincts, mais sont tous 2 clients du groupe BLUE opérant à une strate inférieur. Dans un autre scénario, les TH pour le groupe CYAN peuvent être clients de plusieurs groupes YELLOW et MAGENTA, tous 2 opérants a une strate inférieure. Il y a de nombreux autres scénarios, mais tous doivent être configurés pour inclure seulement les chemins de certificat acycliques.

Dans la figure suivante, le groupe Alice consiste des TH Alice, qui est également le TA, et Carol. Les serveurs dépendants Brenda et Denise on configuré Alice et Carol, respectivement, comme sources de temps. Le serveur de strate 3 Eileen a configuré Brenda et Denise comme source de temps. Les certificats publique sont identifiés par le sujet et signés par l'émetteur. Noter que les clé du groupe de serveur a été précédemment installé sur Brenda et Denise et les clé du groupe client installés sur toutes les machines.



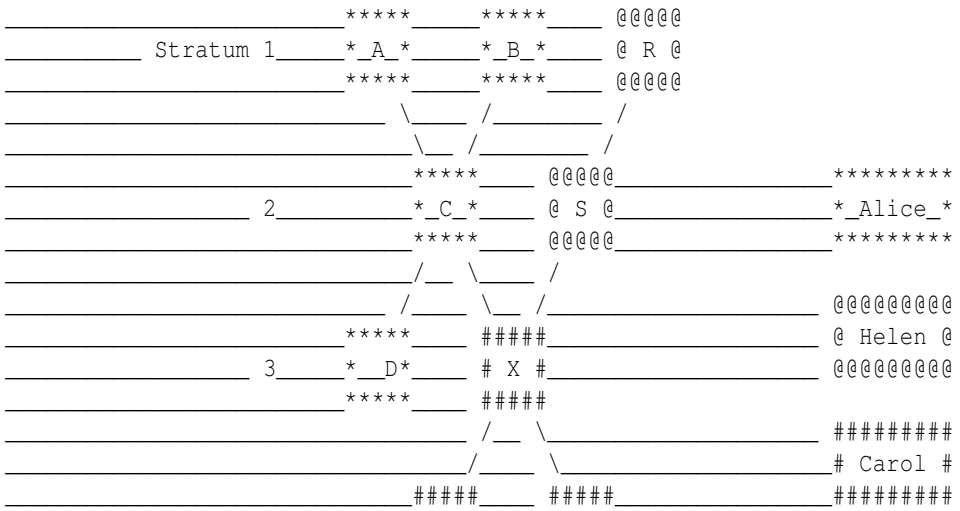


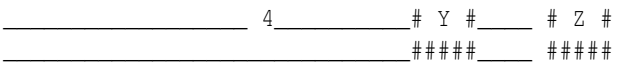


Les étapes dans le parcours des chemins de certificat et la vérification d'identité sont comme suit. Noter que le nombre d'étapes dans la description correspond au nombre d'étapes dans la figure.

1. Les filles commencent par charger la clé hôte, la clé de signature, le certificat autosigné, et la clé de groupe. Chaque client et serveur agissant comme client démarre le protocole Autokey en récupérant le nom d'hôte du serveur et le digest/signature en utilisant l'échange ASSOC.
2. Elles continuent à charger les certificats récursivement jusqu'à ce qu'un certificat de confiance auto-signé soit trouvé. Brenda et Denise trouvent immédiatement les certificats de confiance pour Alice et Carol, respectivement, mais Eileen va boucler parce que ni Brenda ni Denise n'ont leur propre certificat signé par soit Alice ou Carol. C'est fait en utilisant l'échange CERT.
3. Brenda et Denise continuent avec les schémas d'identité sélectionnés pour vérifier que Alice et Carol ont le groupe de clé correct précédemment généré par Alice. C'est fait en utilisant les schéma d'identité IFF, GQ, ou MV. En cas de réussite, continuer à l'étape 4.
4. Brenda et Denise présentent leur certificats pour la signature en utilisant l'échange SIGN. En cas de réussite, Brenda et Denise peuvent maintenant fournir ces certificats signés à Eileen. Eileen peut maintenant vérifier le chemin via soit Brenda ou Denise pour les certificats de confiance pour Alice et Carol. Une fois fait, Eileen peut compléter le protocole comme Brenda et Denise.

Pour diverses raisons, il peut être pratique pour un serveur d'avoir les clés client pour plus d'un groupe. Par exemple, la figure ci-dessous montre 3 groupes sécurisés Alice, Helen, et Carol arrangés dans une hiérarchie. Les hôtes A, B, C, et D appartiennent à Alice avec A et B comme THs. Les hôtes R et S appartiennent à Helen avec R comme TH. Les hôte X et Y appartiennent à Carol avec X comme TH. Noter que le TH pour un groupe est toujours la strate la plus faible et que les hôtes des groupes combinés forment un graphe cyclique. Noter également que le chemin de certificat pour chaque groupe se termine dans un TH pour ce groupe.





Le but de ce scénario est de fournir une séparation de sécurité, pour que les serveurs ne puissent pas se cacher comme clients dans d'autres groupes et les clients ne peut pas se cacher comme serveur. Supposons, par exemple, que Alice et Helen appartiennent aux laboratoires standards nationaux et leur clé serveur sont utilisés pour confirmer l'identité entre les membres de chaque groupe. Carol est une société de premier plan recevant des produits standards et exige une authentification cryptographique. Éventuellement sous contrat, l'hôte X appartenant à Carol a les clés client pour Alice et Helen et les clé serveurs pour Carol. Le protocole Autokey opère pour chaque groupe séparément tout en préservant la séparation de sécurité. L'hôte X peut prouver l'identité dans Carol aux clients Y et Z, mais ne peut pas prouver qu'il appartient à Alice ou Helen.

Schéma d'identité

Un schéma de signature numérique fournis une authentification sûre de serveur, mais ne fournis pas de protection contre le masquerade, sauf si l'identité du serveur est vérifié par d'autres moyens. Le modèle PKI permet au client de prouver l'identifité du serveur en validant le chemin de certificat. Bien qu'Autokey supporte ce modèle par défaut, dans un réseau hiérarchique, spécialement avec les schémas de découverte de serveur comme manycast NTP, prouver l'identité dans le chemin doit être une capacité intrinsèque de Autokey lui-même.

Le schéma d'identité décrit dans la rfc2875 est basé sur une infrastructure Diffie-Hellman, il est coûteux à générer et à utiliser comparé aux autres décrits dans l'annexe B. En principe, un schéma de clé publique ordinaire peut être conçus pour ce but, mais le design plus strict d'Autokey exige que chaque challenge, même s'il est dupliqué, résulte en une réponse différente acceptable.

1. Le schéma doit avoir une durée de vie relativement longue, certainement plus longue qu'un certificat typique, et n'a pas de durée de vie spécifique ou de date d'expiration. Au moment où le schéma est utilisé, l'hôte n'est pas encore synchronisé, donc le schéma ne peut pas dépendre du temps.
2. Comme le schéma peut être utilisé plusieurs fois où les données peuvent être exposées à des intrus potentiels, les données doivent être soit nonce ou nonce chiffré.
3. Le schéma devrait permettre aux serveurs de prouver leur identité aux clients, mais ne permet pas d'agir comme serveurs pour prouver l'identité aux clients indépendants.
4. Dans la mesure du possible, le schéma devrait représenter une preuve 0 connaissance ; c'est à dire que le client devrait être capable de vérifier que le serveur a une clé de groupe correct, mais sans connaître la clé elle-même.

Il y a 5 schémas implémentés dans l'implémentation de référence NTPv4 pour prouver l'identitié : 1. certificat privée (PC), 2. Certificat de confiance (TC), 3. un algorithme Schnorr modifié (IFF - Identify Friendly ou Foe), 4. Un algorithme Guillou-Quisquater modifié (GQ), et 5. un algorithme Mu-Varadharajan modifié (MV). Ils n'offrent pas tous le même niveau de protection, et TC ne fournis aucune protection mais est inclus pour comparaison. La suite est une brève description de chacun.

Le schéma PC implique un certificat privée comme clé de groupe. Le certificat est distribué à tous les membres du groupe par des moyen sécurisés et n'est jamais révélé en dehors du groupe. En effet, le certificat privée est utilisé comme clé symétrique. Ce schéma est utilisé principalement pour des tests et n'est pas recommandé pour une utilisation régulière et n'est pas considéré dans la suite de ce mémo.

Tous les autres schémas impliquent un chemin de certificat conventionnel comme décris dans la rfc5280. C'est le schéma par défaut quand un schéma d'identité n'est pas requis. Bien que les schéma d'identité restants incorporent TC, il n'est pas lui-même considéré dans la suite de ce mémo.

Les 3 schémas restants, IFF, GQ et MV impliquent un échange challenge/response cryptographiquement fort où un intrus ne peut pas déduire la clé serveur, même après avoir répété les observations de plusieurs échanges. De plus, le schéma MV est décris comme une preuve 0 connaissance propre, parce que le client peut vérifier que le serveur a la clé de groupe correct sans que le serveur ou le client n'ait connaissance de sa valeur. Ces schémas commencent quand le client envoie un nonce au serveur, qui envoie ensuite son propre nonce, effectue une opération mathématique et envoie le résultat au client. Le client effectue une autre opération mathématique et vérifie que les résultats sont corrects.

Timestamps et Filestamps

Bien que les signatures à clé publique fournissent une protection forte contre les mauvaises représentation de source, les calculer est très coûteux. Cela donne l'opportunité à un intrus d'obstruer le client ou le serveur en rejouant d'anciens messages ou des messages buggés. Un client recevant de tels messages peuvent être forcés de vérifier une signature invalide et consommer des ressources significatives. Pour déjouer de telles attaques, tout message Autokey gère un timestamp sous la forme de secondes NTP auquel il a été créé. Si l'horloge système est synchronisé à une source proventique, une signature est produite avec un timestamp valide. Sinon, il n'y a pas de signature et le timestamp est invalide (0). Le protocole détecte et supprime les champs d'extension avec d'anciens timestamps ou dupliqués, avant que toutes valeurs ou signature soient vérifiées.

Les signatures sont calculées seulement quand les valeurs cryptographiques sont créé ou modifiées, qui n'est pas très fréquent par design. Les champs d'extension copient ces signatures dans le messages si nécessaire, mais les signature ne sont pas recalculées. Il y a 3 types de signature :

1. **Cookie Signature/timestamp** Le cookie est signé à la création par le serveur et envoyé au client.
2. **Autokey signature/timestamp** Les valeurs autokey sont signées quand la liste de clé est créée
3. **Public signature/timestamp** La clé publique, certificat, et leapseconds sont signés au moment de la génération, qui se produit quand l'horloge système est synchronisée la première fois à une source proventique, quand les valeurs ont été changées et une fois par jours ensuite.

Le timestamp le plus récent reçus de chaque type est sauvé pour comparaison. Une fois qu'une signature avec un timestamp valid a été reçue, les messages avec des timestamp invalide ou des timestamp valides antérieur du même type sont rejetés avant de vérifier la signature. C'est plus important en mode broadcast, qui peut être vulnérable à des attaque par engorgement sans ce test.

Toutes les valeurs cryptographiques utilisées par le protocole sont sensible a temps et sont régulièrement rafraîchis. En particulier, les fichiers contenant des valeurs cryptographiques utilisées par les algorithmes de signature et de chiffrement sont regénérés de temps en temps. L'intention est que les régénérations de fichier se produisent sans avertissement préalable et sans nécessiter la distribution préalable du contenu du fichier. Bien que les fichiers de données cryptographiques ne sont pas signés spécifiquement, tout fichier est associé avec un filestamp montrant les secondes NTP depuis l'epoch de sa création.

Les filestamp et timestamp peuvent être comparés et utilisent la même convention. Il est nécessaire de les comparer de temps en temps pour déterminer lequel est ultérieure ou antérieur. Vu que ces quantités ont une granularité de seulement une seconde, de telles comparaisons sont ambiguës si les valeurs sont dans la même seconde.

Il est important que les filestamps soient des données proventique. Donc, ils ne peuvent pas être produit sauf si le producteur a été synchronisé à une source proventique. Ainsi, les filestamps dans le sous-réseau NTP représente un ordre partiel de toute créations epochs et sert de moyen d'effacer d'anciennes données et s'assurer que les nouvelles données sont consistantes. Comme les données sont envoyées du serveur au client, les filestamps sont préservés, incluant celles pour les certificats et leapseconds. Les paquets avec d'anciens filestamps sont détruits avant de vérifier la signature.

Opérations Autokey

Le protocole NTP a 3 modes principaux d'opération : client/serveur, symétrique, et broadcast et chacun a son propre programme Autokey, ou danse. Une chorégraphie Autokey est conçue pour être non-intrusive et n'exige pas de paquets additionnels autre que pour les opération NTP normales. Les protocoles NTP et Autokey opèrent simultanément et indépendamment. Quand la dase est terminée, les paquets suivants sont validés par la séquence Autokey et donc considérés comme proventique. Autokey assume que les clients NTP interrogent les serveur à fréquence relativement basse, comme une fois par minute ou moins. En particulier, il assume qu'une requête envoyée à une opportunité d'interrogation résulte normalement en une réponse avant la prochaine opportunité d'interrogation ; cependant, le protocole est robuste contre une réponse manquée ou dupliquée.

Le serveur ne conserve pas d'état pour chaque client, mais utilise un algorithme rapide et une valeur privée aléatoire 32bits pour régénérer le cookie à l'arriée d'un paquet client. Le cookie est calculé comme 32 premiers bits de l'autokey calculé depuis les adresses client et serveur, Key ID 0, et le serveur envoie le cookie. Le cookie est utilisé pour le calcul autokey par le client et le serveur et est donc spécifique à chaque client séparémment.

Dans la danse serveur, le client utilise le cookie et chaque key ID dans la liste de clé puis récupère la autokey et génère le MAC. Il génère ainsi le MAC pour la réponse en utilisant les même valeurs, mais en échangeant les adresses client et serveur. Le client génère le message digest et vérifie que le MAC corresponde. Pour déjouer les rejeux anciens, le client vérifie que le key ID correspond au dernier envoyé. Dans cette danse, la structure séquentielle de la liste de clé n'est pas exploitée, mais cela simplifie et régularise l'implémentation tout en évitant la possibilité qu'un intrus ne devine le prochain key ID.

Dans la danse broadcast, les clients n'envoient normalement pas les paquets au serveur, excepté pour le premier démarrage. À ce moment, le client lance la danse serveur pour vérifier les accréditifs du serveur et calibrer le délai de propagation. La danse nécessite l'ID d'association de l'association serveur particulier, vu qu'il y a plus d'un opérant dans le même serveur. Dans ce but, le paquet serveur inclus l'ID d'association dans chaque réponse et en envoyant le premier paquet après avoir généré une nouvelle liste de clé, il envoie les valeurs autokey. Une fois les valeurs autokey obtenues et vérifiées, aucun champ d'extension n'est nécessaire et le client vérifie les autres paquets serveur en utilisant la séquence autokey.

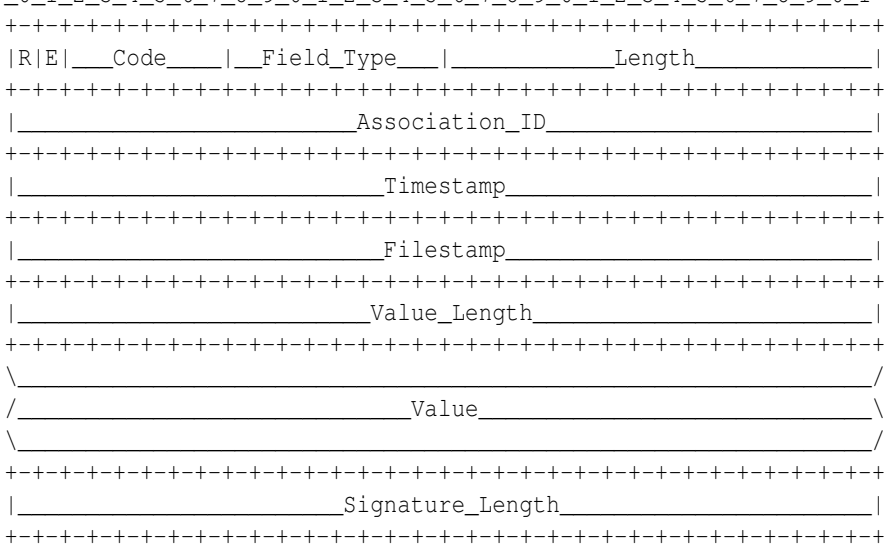
La danse symétrique est similaire à la danse serveur et nécessite seulement une petite quantité d'état entre l'arrivée d'une demande et le départ de la réponse. La liste de clé pour chaque direction est générée séparément par chaque paire et utilisé indépendamment, mais chacun est généré avec le même cookie. Le cookie est transporté de la même manière que la danse serveur, excepté que le cookie est un simple nonce. Il existe une condition possible où chaque paire envoie une demande de cookie avant de recevoir la réponse cookie de l'autre paire. Dans ce cas, chaque paire se termine par 2 valeurs, une est générée et l'autre générée par l'autre paire. L'ambiguité est résolue simplement en calculant le cookie comme le EXOR des 2 valeurs.

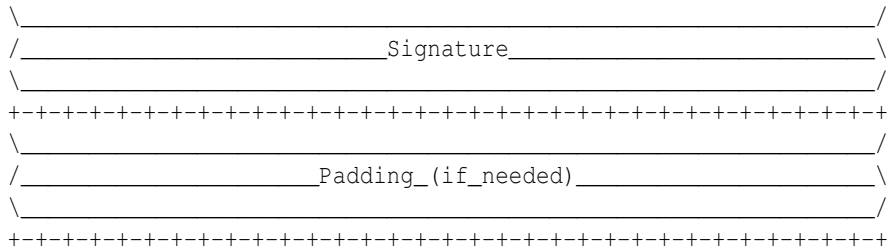
Une fois la danse Autokey completée, il est normalement dormant. Excepté pour la danse broadcast, les paquets sont normalement envoyés sans champs d'extension, sauf si le paquet est le premier envoyé après avoir généré une nouvelle liste de clé ou à moins que le client ait demandé le cookie ou les valeurs Autokey. Si pour une raison ou une autre l'horloge client est stepped au lieu d'être slewed, toutes valeurs cryptographique et de temps pour toutes les associations sont purgées et les danses dans toutes les associations redémarrent depuis le début. Celà permet de s'assurer que les valeurs périmées ne se propagent pas au-delà d'un step d'horloge.

Message de protocol Autokey

L'unité de donnée du protocole Autokey est le champ d'extension, un ou plusieurs qui peuvent être empruntés dans le paquet NTP. Un champ d'extension contient soit une requête avec des données optionnelles ou une réponse avec les données optionnelles. Pour éviter les impasses, plusieurs réponses peuvent être inclus dans un paquet, mais seulement une requête peut l'être. Une réponse est générée pour chaque requête, même si le demandeur n'est pas synchronisé à une source proventique, mais la plupart contiennent des données significatives seulement si le répondeur est synchronisé à une source proventique. Certaines requêtes et beaucoup de réponse gèrent les signatures horodatées. La signature couvre le champ d'extension entier, incluant le timestamp et filestamp, si applicable. Seulement si le paquet est correct, format, longueur et hash, la signature est vérifié.

Il y a actuellement 8 requêtes Autokey et 8 réponses correspondantes :





Un ou plusieurs champs d'extension suivent l'en-tête NTP et le dernier est suivi par le MAC. Le parser de champ d'extension initialise un pointeur au premier octet au-delà de l'en-tête de paquet NTP et calcule le nombre d'octets restant à la fin du paquet. Il le reste fait 20 (128-bit de hash + 4 octets de Key ID) ou 22 (160-bit de hash + 4 octets de Key ID), les données restantes sont le MAC. Supérieur à 22, un champ d'extension est présent. Si la longueur restante est inférieur à 8 ou n'est pas un multiple de 4, une erreur de format s'est produite et le paquet est détruit.

Dans Autokey le champ du type de champ 8-bit est interprété comme numéro de version, actuellement 2. Le champ Code 6 bits spécifie l'opération de requête ou de réponse. Il y a 2 bits de flag : le bit 0 est le flag de réponse (R) et le bit 1 est flag d'erreur (E).

Dans la plupart des opération du protocole, un client envoie une requête à un serveur avec un code d'opération spécifié dans le champ Code avec les bits R et E mis. Le serveur retourne une réponse avec le même code d'opération dans le champ Code et efface le bit R. Le serveur peut également effacer E en cas d'erreur. Noter que ce n'est pas nécessairement une erreur de protocole d'envoyer une réponse non-solicitée dans requête correspondante. Si le bit R est mis, le client définis le champ Association ID à l'id d'association du client, que le serveur retourne pour vérification. Si les 2 valeurs ne correspondent pas, la réponse est détruite. Si le bit R est mit, le champ Association ID est mis à l'Association ID du serveur obtenu dans l'échange initial. Si le champ Association ID ne correspond pas à un association ID mobilisé, la requête est détruite.

Dans certains cas, tous les champs de sont pas présents. Pour les requêtes, jusqu'à ce qu'un client se soit synchronisé à une source proventique, les signatures ne sont pas valides. Dans de tels cas, le champ Timestamp et le champ Signature Length sont à 0 et le champ Signature est absent. Certaines requêtes et messages de réponse d'erreur ne placent aucune valeur dans les champs de signature, donc dans ces messages seul les 2 premiers mots sont présent (8 octets)

Les Timestamp et Filestamp gèrent le second champ d'un timestamp NTP. Le timestamp établis la signature epoch du champ de données dans le message, alors que filestamp établis la génération epoch du fichier qui a produit la donnée qui est signée.

Une signature et timestamp sont valide seulement quand l'hôte signant est synchronisé à une source proventique ; sinon, le timestamp est 0. Un fichier de données cryptographique peut seulement être généré si une signature est possible ; sinon, le filestamp est zéro, excepté dans le messages de réponse ASSOC, où il contient le status du serveur.

No-Operation

Une requête No-operation (Code 0) ne fait rien excepté retourner une réponse vide, qui peut être utilisée comme crypo-ping.

Assiociation Message (ASSOC)

Un Association Message (Code 1) est utilisé dans l'échange de paramètre pour obtenir le nom d'hôte et le status. La requête contient le status client dans le champ Filestamp et le nom d'hôte Autokey dans le champ Value. La réponse contient le status serveur dans le champ Filestamp et le nom d'hôte Autokey dans le champ Value. Le nom d'hôte Autokey n'est pas nécessairement le nom d'hôte DNS. Une réponse valide met le bit ENAB et éventuellement d'autres dans le status d'association.

Quand plusieurs schémas d'identité sont supportés, le status détermine lesquels sont disponible. En mode serveur et symétrique, le status de réponse contient les bits correspondant aux schémas supportés. Dans tous les modes, le schéma est sélectionné en fonction des

paramètre d'identité du client qui sont chargés au démarrage.

Certificate Message (CERT)

Un Certificate Message (Code 2) est utilisé dans l'échange de certificat pour obtenir un certificat par nom du sujet. Le requête contient le nom du sujet ; la réponse contient le certificat encodé au format X.509.

Si le nom du sujet dans la réponse ne match pas le nom de l'émetteur, l'échange continue avec le nom de l'émetteur replaçant le sujet dans la requête. L'échange continue jusqu'à ce que certificat autosigné de confiance soit trouvé et met le bit CERT dans le status.

Cookie Message (COOKIE)

Le Cookie Message (Code 3) est utilisé dans en mode serveur et symétrique pour obtenir le cookie serveur. La requête contient la clé publique encodée avec ASN.1. La réponse contient le cookie chiffré par la clé publique dans la requête. Une réponse valide met le bit COOKIE dans le status d'association.

Autokey Message (AUTO)

Le Autokey Message (Code 4) est utilisé pour obtenir les valeurs autokey. La requête ne contient pas de valeur pour un client ou les valeurs Autokey pour un paire symétrique. a réponse contient 2 mots 32bits, le premier est le Key ID final, et le second est l'index du Key ID final. Une réponse valide met le bit AUTO dans le status d'association.

Leapseconds Values Message (LEAP)

Le Leapseconds Values Message (Code 5) est utilisé pour obtenir les valeurs leapseconds comme parcouru dans la table leapseconds du NIST. La requête ne contient pas de valeurs. La réponse contient 3 entiers 32bits. Le premier sont les secondes NTP du dernier évènement leap suivi par les secondes NTP quand la dernière table NIST expire puis l'offset TAI suivant l'évènement leap. Une réponse valide met le bit LEAP dans le status d'association.

Sign Message (SIGN)

Le Sign Message (Code 6) demande que le serveur signe et retourne un certificat présenté dans la requêt. La requête contient le certificat client au format X.509. La résponse contient le certificat client signé par la clé privée du serveur. Une réponse valide met de bit SIGN dans le status d'association.

Identity Messages (IFF, CQ, MV)

Les Identity Messages (Code 6 - IFF, 8 - GQ, ou 9 - MV) contiennent le challenge client, généralement un nonce 160 ou 512 bits. La réponse contient le résultat de l'opération mathématique. Une réponse valide met le bit VRFY dans le status d'association.

Autokey State Machine

Le serveur implémente un mot de status d'hôte, alors que chaque client implémente un mot de status d'association. Ces mots ont le format et le contenu définis ci-dessous. Les 16bits LSB définissent l'état de la danse Autokey, alors que les 16bits MSB spécifie le NUD tel que généré par la librairie OpenSSL de l'OID pour un des schémas de hashage/signature définis dans la rfc3279. Les valeurs NID (Numerical Identifier) pour les algorithmes de hashage/signature sont les suivants :

Algorithm	OID	NID
pkcs-1	1.2.840.113549.1.1	2
md2	1.2.840.113549.2.2	3
md5	1.2.840.113549.2.5	4
rsaEncryption	1.2.840.113549.1.1.1	6
md2WithRSAEncryption	1.2.840.113549.1.1.2	7
md5WithRSAEncryption	1.2.840.113549.1.1.4	8
id-sha1	1.3.14.3.2.26	64
sha-1WithRSAEncryption	1.2.840.113549.1.1.5	65
id-dsa-wth-sha1	1.2.840.10040.4.3	113
id-dsa	1.2.840.10040.4.1	116

Les bits 24-31 sont réservés pour le serveur, les bits 16-23 sont réservés pour le client. Dans la portion hôte, les bits 24-27 spécifient les schémas d'identité disponibles, et les bits 28-31 spécifient les capacités serveur. Il y a 2 bits additionnels implémentés séparément.

Status Word

1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1		
Digest / Signature_NID	Client	Ident	Host

Le mot de status d'hôte est inclus dans la requête ASSOC et les messages de réponse. Le client copie ce mot dans le mot de status d'association et met les bits additionnels à mesure que la danse progresse. Une fois établis, ces bits ne sont jamais effacés sauf en cas d'erreur, auquel cas le protocole est redémarré depuis le début.

Les bits de status d'hôte sont définis comme suit :

ENAB (31) Mis si le serveur implémente le protocole Autokey

LVAL (30) Mis si le serveur a installé les valeurs leapseconds

Bits (28-29) Réservés

Bits 24-27 Sélection les schémas d'identité serveur disponible

Aucun Schéma de certificat de confiance TC (défaut)

27 Schéma de certificat privée PC

26 Schéma IFF

25 Schéma GQ

24 Schéma MV

- Le schéma PC est exclus. IFF, GQ, et MV peuvent être combinés

Les bits de status d'association sont définis comme suit :

CERT (23) Mis quand le certificat d'hôte de confiance et la clé publique sont validés

VRFY (22) Mis quand les accréditifs d'identifité de l'hôte de confiance sont confirmés

PROV (21) Mis quand la signature serveur est vérifiée en utilisant sa clé publique et ses accréditifs d'identité. Également appelé le bit proventique.

COOK (20) Mis quand le cookie est reçus et validé

AUTO (19) Mis quand les valeurs autokey sont reçues et validées.

SIGN (18) Mis quand le certificat hôte est signé par le serveur

LEAP (17) Mis quand les valeurs leapseconds sont reçus et validés.

Bit 16 Réservé

Il y a 3 bits additionnels : LIST, SYNC, et PEER non inclus dans le mot de status d'association. LIST est mis quand la liste de clé est régénénée et effacée quand les valeurs d'autokey ont été transmis. SYNC est mis quand le client est synchronisé à une source proventique. PEER est mis quand le serveur a été synchronisé, comme indiqué dans l'en-tête NTP.

Variable d'état de l'hôte

Host Name Le nom de l'hôte, par défaut la chaîne retournée par la librairie Unix gethostname()

Host Status Word Initialisé quand l'hôte démarre.

Host Key Paire de clé RSA utilisé pour chiffrer/déchiffrer les cookies. C'est également la clé de signature par défaut

Sign Key La paire de clé RSA ou DSA utilisé pour les signatures quand la clé hôte n'est pas utilisée pour cela

Sign Digest Algorithme de hashage utilisé pour calculer le hash avant chiffrement

IFF Parameters Paramètres utilisés dans le schéma d'identité IFF

GQ Parameters Paramètres utilisés dans le schéma d'identité GQ

MV Paramèters Paramètres utilisés dans le schéma d'identité MV

Server Seed Valeur privée hashée avec les adresses IP et l'identifiant de clé pour construire le cookie

CIS Certificat Information Structure. Cette structure inclus certains champs d'informations d'un certificat X.509v3, avec le certificat lui-même. Les champs extraits incluent le sujet et émetteur, clé publique et algorithme de hashage, et le début et fin de validité en secondes NTP.

Le certificat lui-même est stocké comme champ d'extension pour qu'il puisse être copié tel quel dans le message. La structure est signée en utilisant la clé de signature et gère les timestamp au moment de la signature et le filestamp du fichier certificat originel. La structure est utilisé par la réponse CERT et les demande et réponse SIGN.

Un champ flags dans CIS détermine le status du certificat. Le champ est encodé comme suit :

TRUST (0x01) Le certificat a été signé par un émetteur de confiance. Si le certificat est auto-signé et contient trustRoot dans le champ d'utilisation de clé étendu, ce bit est mis.

SIGN (0x02) La signature de certificat a été vérifiée. Si le certificat est auto-signé et vérifié en utilisant la clé publique contenue, ce bit est mis.

VALID (0x04) Le certificat est valide et peut être utilisé pour vérifier les signatures. Ce bit est mis quand un certificat de confiance a été troué dans un chemin de certification.

PRIV (0x08) Le certificat est privé et ne doit pas être révélé. Si le certificat est auto-signé et contient Private dans le champ d'utilisation de clé étendue, ce bit est mis.

ERROR (0x80) Le certificat est défectueux est ne doit pas être utilisé

Certificate List Les structures CIS sont stockées dans la liste de certificat dans l'ordre d'arrivée, avec le CIS le plus récent placé en premier dans la liste. La liste est initialisée avec le CIS pour le certificat hôte, qui est lus depuis le fichier certificat d'hôte. Additionnellement les entrées CIS qui sont ajoutées à la liste comme certificats sont obtenue depuis les serveurs durant l'échange de certificat. Les entrées CIS sont supprimées si elles sont remplacées par de nouvelles.

Host Name Values C'est utilisé pour envoyer des requêtes/réponses ASSOC. Elle contient le status hôte et le nom d'hôte.

Public Key Values Utilisé pour envoyer des requêtes COOKIE. Elle contient la clé publique de chiffrement utilisé pour la réponse COOKIE

Leapseconds Values Utilisé pour envoyer le message de réponse LEAP. Elle contient les valeurs leapseconds dans la description du message LEAP.

Variables d'état client (tous les modes)

Association ID L'id d'association utilisé dans les réponses. Il est assigné quand l'association est mobilisée.

Associations Status Word Le status copié de la réponse ASSOC, éventuellement modifié par l'état machine.

Subject Name Le nom d'hôte du serveur copié depuis la réponse ASSOC

Issuer Name Le nomb d'hôte de signature de certificat. Extrait du certificat serveur

Server Public Key La clé publique utilisé pour déchiffrer les signatures. Extrait du certificat hôte serveur.

Server Message Digest La schéma de hash/signature déterminé dans l'échange de paramètres

Group Key Un jeu de valeurs utilisées par l'échange d'identité. Identifie le compartiment cryptographique partagé par le serveur et le client.

Receive Cookie Values Le cookie retourné dans une réponse COOKIE, avec ses timestamp et filestamp

Receive Autokey Values Les valeurs autokey retournées dans une réponse AUTO, avec ses timestamp et filestamp

Send Autokey Values Les valeurs autokey avec la signature et les timestamp

Key List Une séquence de Key ID commençant avec l'autokey de départ, et chacun pointant vers le suivant. Il est calculé, timestampé, et signé à la prochaine opportunité d'interrogation quand la liste de clé devient vide.

Current Key Number L'index d'entrée dans la liste de clés.

Transitions d'état du protocole

L'état machine du protocole est simple mais robuste. L'état est déterminé par le status client. Les transitions d'état des 3 danses sont affichés ci-dessous.

Danse serveur

La danse serveur commence quand le client envoie une requête ASSOC au serveur. L'horloge est mis à jours quand PREV est mis et la danse se termine quand LEAP est mis. Dans cette danse, les valeurs autokey ne sont pas utilisée, donc un échange autokey n'est pas nécessaire. Noter que les requêtes SIGN et LEAP ne sont pas émises tant que le client n'est pas synchronisé à une source proventique. Les paquets suivants sans champs d'extension sont validé par la séquence autokey. Cet exemple et d'autres assument le schéma d'identité IFF.

Danse serveur

```
while (1) {
 wait_for_next_poll;
 make_NTP_header;
 if (response_ready)
  send_response;
 if (!ENB) /*_parameter exchange_*/
  ASSOC_request;
 else if (!CERT) /*_certificate exchange_*/
  CERT_request(Host_Name);
 else if (!IFF) /*_identity exchange_*/
  IFF_challenge;
 else if (!COOK) /*_cookie exchange_*/
  COOKIE_request;
 else if (!SYNC) /*_wait for synchronization_*/
  continue;
```

```
 else if (!SIGN) /*_sign exchange_*/
  SIGN_request(Host_Certificate);
 else if (!LEAP) /*_leapsecond values exchange_*/
  LEAP_request;
 send packet;
}
```

Si le serveur rafraîchis le seed privé, le cookie devient invalide. Le serveur répond à un cookie invalide avec un message crypto-NAK, qui force le client à redémarrer le protocole depuis le début.

Danse broadcast

La dance broadcast est similaire à la danse serveur avec l'échange cookie remplacée par l'échange de valeurs autokey. La danse broadcast commence quand le client reçoit un paquet broadcast incluant une réponse ASSOC avec l'association ID serveur. Cela mobilise une association client pour proventiquer la source et calibrer le délai de propagation. La danse se termine quand le bit LEAP est mis, après quoi le client n'envoie plus de paquets. Normalement, le serveur broadcast inclus une réponse ASSOC dans chaque paquet transmis. Cependant, quand le serveur génère une nouvelle liste de clé, il inclus une réponse AUTO à la place.

Dans la danse broadcast, les champs d'extension sont utilisé avec chaque paquet, donc le cookie est toujours 0 et aucun échange de cookie n'est nécessaire. Comme dans la danse serveur, l'horloge est mise à jours quand PREV est mis et la danse se termine quand LEAP est mis. Noter que les requêtes SIGN et LEAP ne sont pas émis tant que le client n'a pas été synchronisé à une source proventique. Les paquets suivants sans champs d'extension sont validés par la séquence Autokey.

```
while (1) {
 wait_for_next_poll;
 make_NTP_header;
 if (response_ready)
  send_response;
 if (!ENB) /*_parameters exchange_*/
  ASSOC_request;
 else if (!CERT) /*_certificate exchange_*/
  CERT_request(Host_Name);
 else if (!IFF) /*_identity exchange_*/
  IFF_challenge;
 else if (!AUT) /*_autokey values exchange_*/
  AUTO_request;
 else if (!SYNC) /*_wait for synchronization_*/
  continue;
 else if (!SIGN) /*_sign exchange_*/
  SIGN_request(Host_Certificate);
 else if (!LEAP) /*_leapsecond values exchange_*/
  LEAP_request;
 send NTP_packet;
}
```

Si un paquet est perdu et la séquence autokey est cassée, le client hash l'autokey courant jusqu'à ce qu'il matche l'autokey précédent ou le nombre de hashs excède le compteur donné dans les valeurs autokey. Le client envoie plus tard une requête AUTO pour récupérer les valeurs autokey. Si le client reçoit un crypto-NAK durant la danse ou si l'association ID change, le client redémarre le protocole depuis le début.

Danse symétrique

La danse symétrique est une chorégraphie complexe. Elle commence quand le paire actif envoie une requête ASSOC au paire passif. Le paire passif mobilise une association et les 2 paires démarrent une danse 3 voies où chaque paire complète un échange de paramètre avec l'autre. Quand un des paire soit synchronisé à une source proventique et puisse signer des messages, les autres paires bouclent en attente d'un timestamp valide dans la réponse CERT.

```
while (1) {
 wait_for_next_poll;
 make_NTP_header;
 if (!ENB) /*_parameters exchange_*/
  ASSOC_request;
 else if (!CERT) /*_certificate exchange_*/
  CERT_request(Host_Name);
 else if (!IFF) /*_identity exchange_*/
  IFF_challenge;
 else if (!COOK && PEER) /*_cookie exchange_*/
  COOKIE_request);
 else if (!AUTO) /*_autokey values exchange_*/
  AUTO_request;
 else if (LIST) /*_autokey values response_*/
  AUTO_response;
 else if (!SYNC) /*_wait for synchronization_*/
  continue;
 else if (!SIGN) /*_sign exchange_*/
  SIGN_request;
 else if (!LEAP) /*_leapsecond values exchange_*/
  LEAP_request;
 send NTP_packet;
}
```

Une fois qu'un paire a été synchronisé à une source proventique, il inclus les signatures horodatées dans ses messages. l'autre paire, qui attend des horodatages valides, peuvent finir la danse. Il récupère le cookie non-zéro en utilisant un échange de cookie et met à jours les valeurs autokey en utilisant un échange autokey.

Comme dans la danse broadcast, si un paquet est perdu et la séquence autokey cassée, le paire hash l'autokey courant jusqu'à ce qu'il matche l'autokey précédent ou que le nombre de hash excède le compteur dans les valeurs autokey. Le client envoie plus tard une requête AUTO pour récupérer les valeurs autokey. Si le client reçoit un crypto-NAK durant la danse ou si l'association ID change, le client redémarre le protocole depuis le début.

Récupération d'erreur

L'état machine du protocole Autokey inclus un provisionning pour diverses conditions d'erreur qui peuvent se produire à cause de fichiers manquants, données corrompues, violations de protocole, et perte de paquet et désordonnés, sans mentionner les intrusions hostiles. Cette section décrit comme le protocole réponds aux évenements d'accessibilité et timeouts qui peuvent se produire à cause de telles erreurs.

Une association NTP persistante est mobilisée par une entrée dans le fichier de configuration, alors qu'une association éphémère est mobilisée à l'arrivée d'un paquet broadcast ou symétrique actif sans association correspondant. En conséquence, un reset général réinitialise toutes les variables d'association à l'état initial quand il est mobilisé. De plus, si l'association est éphémère, l'association est démobilisée et toutes les ressources acquises sont retournée au système.

Toute association NTP a 2 variables qui maintiennent l'état du protocole, le registre 8 bits reach et le compteur unreach. À chaque interval d'interrogation, le registre reach est décalé à gauche, et LSB est mis à 0. À ce moment, le compteur unreach est incrémenté de 1. Si un paquet arrivant passe l'authentification et les vérifications de santé, le LSB est mis et le compteur unreach est effacé. Si un bit dans le registre reach est mis, le serveur est accessible, sinon il est inaccessible.

Quand la première interrogation est envoyée depuis une association, le registre reach et le compteur unreach sont mis à 0. Si le compteur unreach atteind 16, l'interval d'interrogation est doublé. De plus, si l'association est persistante, elle est démobilisée. Cela réduit la charge réseau pour les paquets qui sont peu susceptibles de susciter une réponse.

À chaque état dans le protocole, le client attend une réponse particulière du serveur. Une requête est inclus dans le paquet NTP envoyé à chaque interval d'interrogation jusqu'à ce qu'une réponse valide soit reçue ou un reset général se produise, auquel cas le protocole redémarre depuis le début. Un reset général se produit également pour une association quand une erreur de protocole irrécupérable se produit. Un reset général se produit pour toutes les associations quand l'horloge système est d'abord synchronisée ou stepped ou quand le l'envoie serveur est rafraîchis.

Il y a des cas spéciaux conçus pour rapidement répondre à des association cassées, comme quand un serveur redémarre ou rafraîchis les clés. Vu que le cookie client est invalidé, le serveur rejète la requête client suivante et retourne un crypto-NAK. Vu que le crypto-NAK n'a pas de MAC, le problème pour le client est de déterminer s'il est légitime ou le résultat d'un intrus. Pour réduire la vulnérabilité dans de tels cas, le crypto-NAK, comme toutes les autres réponses, est validé seulement s'il est le résultat d'un paquet précédent envoyé par le client et pas un rejeux, comme confirmé par le protocole on-wire.

Il y a des situations où certains évènements se produisant causent les autokey restant dans la liste à devenir invalide. Quand une de ces situations se produit, la liste de clé est les autokeys associées dans le cache de clé sont purgés. Une nouvelle liste de clé, signature, et timestamp sont générés quand le prochain message NTP est envoyé, en supposant qu'il y en ait un. La liste de ces situation est :

1. Quand la valeur d'un cookie change pour une raison ou une autre
2. Quand l'interval d'interrogation est changé. Dans ce cas, les temps d'expiration pour les clé deviennent invalides
3. Si un problème est détecté quand une entrée est récupérée depuis la liste de clé. Cela peut se produire si la clé a été marqué non-trusted ou timeout, ce qui peut impliquer un bug logiciel.

Considérations de sécurité

Bien que le protocole n'a pas été sujet à une analyse formelle, quelques affirmations préliminaires peuvent être faites. Dans les danses client/serveur et symétrique, le protocole on-wire est résistant à la perte, duplication, et paques buggés, même si l'horloge n'est pas synchronisée, donc le protocole n'est pas vulnérable à une attaque wiretapper. Le protocole on-wire est résistant aux replays de paquet de requête client et de réponse serveur. Une attaque MITM, même si elle peut simuler un cookie valide, ne peut pas prouver l'identité.

Dans la danse broadcast, le client commence en mode client/serveur pour obtenir les valeurs autokey et signature, donc a le même niveau de protection que dans ce mode. En continuant en mode réception uniquement, un wiretapper ne peut pas produire une liste de clés avec des valeurs autokey signées valides. S'il rejoue un ancien paquet, le client le rejète par vérification du timestamp. Le mieux qu'il peut faire est de fabriquer un paquet future forçant le client à répéter les opérations de hashage autokey jusqu'à excéder le nombre maximum de clé. Si cela ce produit le client broadcast revient temporairement en mode client pour rafraîchir les valeurs autokeys.

Par supposition, un attaquant MITM qui intercepte un paquet ne peut pas casser le fil ou délai dans un paquet intercepté. Si cette supposition est supprimée, le MITM pourrait intercepter un paquet broadcast et remplacer les données et le hash sans détection par les clients.

Comme mentionné précédemment, le schéma d'identité TC est vulnérable à une attaque MITM où un intrus peut créer un chemin de certificat buggé. Pour déjouer ce type d'attaque, les schémas PC, IFF, GQ ou MV doivent être utilisés.

Un client instancie les variables cryptographiques seulement si le serveur est synchronisé à une source proventique. Un serveur ne signe pas de valeurs ou ne génère de fichiers de données cryptographiques sauf s'il est synchronisé à une source proventique. Cela soulève un problème intéressant : comment un client génère des fichiers cryptographiques proventique avant qu'il ait été synchronisé à une source proventique ? En principe, ce paradoxe est résolus en assumant que les serveurs primaires sont proventiqués par des moyens externes.

Vulnérabilité Clogging

Un incident clogging auto-induit ne peut pas se produire, vu que les signatures sont calculées seulement quand les données ont changées et les données ne changent pas très souvent. Par exemple, les valeurs autokey sont signées seulement quand la liste de clé est regénérée, se qui se produit une fois par heure, alors que les valeurs publiques sont signées seulement quand une d'entre elles est mise à jours durant une danse ou un seed serveur est rafraîchis, se qui se produit un fois par jour.

Il y a 2 vulnérabilités d'obstruction exposés dans le protocole : une attaque de chiffrement où l'intrus espère obstruer le serveur avec des calculs cryptographiques inutiles, et une attaque par déchiffrement où l'intrus tente d'obstruer le client avec des calculs cryptographiques inutiles. Autokey utilise une clé publique et les algorithmes qui effectuent ces fonctions consomment beaucoup de ressource.

Dans les danse client/serveur et paire, un hasard cryptographique existe quand un wiretapper rejoue d'anciens messages de requête de cookie rapidement. Il n'y a pas de moyen de déjouer de telles attaques, vu que le serveur ne retient pas d'état entre les requêtes. Les replays des requêtes ou réponse cookie sont détectées et supprimées par le protocole on-wire client.

En mode broadcast, un hasard de déchiffrement existe quand un wiretapper rejoue des messages réponse autokey rapidement. Une fois synchronisé à une source proventique, un champs d'extension légitime avec timestamp identique ou antérieur au plus récent reçus est immédiatement détruit. Cela met en évidente une attaque MITM cup-and-paste en utilisant une réponse antérieure, par exemple. en champ d'extension légitime avec timestamp dans le future est peu probable vu qu'il requière de prédire la séquence autokey. Cependant, cela force le client à rafraîchir et vérifier les valeurs autokey et la signature.

Un attaquant déterminé peut déstabiliser le protocole on-wire ou une danse autokey de diverses manières en rejouant d'anciens messages avant que le client ou le paire se soit synchronisé pour la première fois. Par exemple, rejouer un ancien message en mode symétrique avant que les paires se soient synchronisés empêche les paires de se synchroniser. Rejouer les messages Autokey dans le désordre durant une danse peut empêcher de completer la danse. Il n'y a rien de nouveau dans ce type d'attaques ; une vulnérabilité similaire existe dans TCP.