Groupe de travail Réseau
Request for Comments : 4226
Catégorie : Information
décembre 2005
Traduction Claude Brière de L'Isle

D. M'Raihi, VeriSign
M. Bellare, UCSD
F. Hoornaert, Vasco
D. Naccache, Gemplus
O. Ranen, Aladdin

HOTP : Algorithme de mot de passe à utilisation unique fondé sur HMAC

Statut de ce mémoire
Le présent mémoire apporte des informations pour la communauté de l'Internet. Le présent mémoire ne spécifie aucune sorte de norme de l'Internet. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Le présent document décrit un algorithme pour générer des valeurs de mot de passe à utilisation unique, fondées sur le code d'authentification de message haché (HMAC, *Hashed Message Authentication Code*). On présente une analyse de la sécurité de l'algorithme et les paramètres importants relatifs au déploiement sûr de l'algorithme. L'algorithme proposé peut être utilisé sur une large gamme d'applications réseau allant de l'accès distant aux réseaux privés virtuels (VPN, *Virtual Private Network*) à l'enregistrement sur un réseau Wi-Fi pour des applications de la Toile en mode transaction.

Ce travail est un produit conjoint des membres de OATH (Open AuTHentication) pour spécifier un algorithme qui puisse être librement distribué à la communauté technique. Les auteurs pensent qu'un algorithme commun et partagé facilitera l'adoption de l'authentification sur deux facteurs sur l'Internet en permettant l'interopérabilité entre les mises en œuvres commerciales et de source libre.

Table des matières

1. Généralités

Le document introduit d'abord le contexte de l'algorithme qui génère des valeurs de mot de passe à utilisation unique fondées sur HMAC [BCK1] et est donc nommé l'algorithme HOTP (*HMAC-Based One time Password*). La Section 4, donne la liste des exigences de l'algorithme, et la Section 5 décrit l'algorithme HOTP. Les Sections 6 et 7 se concentrent sur la sécurité de l'algorithme. La Section 8 propose des extensions et améliorations, et la Section 10 conclut le document. En Appendice A, le lecteur intéressé trouvera une analyse détaillée et complète de la sécurité de l'algorithme : une version idéalisée de l'algorithme est évaluée, et ensuite la sécurité de l'algorithme HOTP est analysée.

2. Introduction

Aujourd'hui, la portée et l'échelle du déploiement de l'authentification à deux facteurs reste extrêmement limitée. En dépit de niveaux accrus de menaces et d'attaques, la plupart des applications de l'Internet s'appuient encore sur des schémas faibles d'authentification pour réguler l'accès des utilisateurs. Le manque d'interopérabilité entre les fabricants de technologies de matériel et de logiciels a été un facteur limitant l'adoption d'une technologie d'authentification à deux facteurs. En particulier, l'absence de spécifications ouvertes a conduit à des solutions où les composants de matériel et logiciels sont étroitement couplés au travers de technologies propriétaires, résultant en des solutions coûteuses, une adoption faible, et une innovation limitée.

Ces deux dernières années, la croissance rapide des menaces sur le réseau a souligné l'inadéquation des mots de passe statiques comme moyen principal d'authentification sur l'Internet. En même temps, l'approche courante qui exige de l'utilisateur final de porter une appareil coûteux unifonctionnel utilisé seulement pour s'authentifier sur le réseau n'est clairement pas la bonne réponse. Pour que l'authentification à deux facteurs se propage sur l'Internet, elle devra être incorporée dans des appareils plus flexibles qui puissent fonctionner sur une large gamme d'applications.

La capacité à incorporer cette technologie de base tout en assurant une large interopérabilité exige qu'elle soit librement disponible à la plus large communauté technique des développeurs de matériels et logiciels. Seule une approche de source libre assurera que les primitives de base de l'authentification à deux facteurs peuvent être incorporées dans la prochaine génération d'appareils grand public comme les appareils USB de mémorisation de masse, les téléphones IP, et les tablettes numériques personnelles.

Le mot de passe à utilisation unique est certainement une des plus simples et populaires formes d'authentification à deux facteurs pour sécuriser l'accès réseau. Par exemple, dans les grandes entreprises, l'accès au réseau virtuel privé exige souvent l'utilisation de jetons de mot de passe à utilisation unique pour l'authentification de l'utilisateur distant. Les mots de passe à utilisation unique sont souvent préférés à de plus fortes formes d'authentification comme d'infrastructure de clé publique (PKI, *Public-Key Infrastructure*) ou de biométrique parce que un appareil sans fil n'exige aucune installation de logiciel client sur la machine d'utilisateur, leur permettant donc de circuler à travers plusieurs machines incluant des ordinateurs domestiques, des kiosques, et des tablettes numériques.

Le présent document propose un simple algorithme de mot de passe à utilisation unique qui peut être mis en œuvre par tout fabricant de matériel ou développeur de logiciel pour créer des appareils interopérables d'authentification et des agents logiciels. L'algorithme se fonde sur l'événement de sorte qu'il peut être incorporé dans des appareils de fort volume comme

les cartes intelligentes Java, des clés USB, et des cartes SIM du GSM. L'algorithme présenté est librement disponible à la communauté des développeurs sous les termes et conditions des droits de propriété intellectuelle de l'IETF [RFC3979].

Les auteurs du présent document sont membres de l'initiative Open AuTHentication [OATH]. L'initiative a été créée en 2004 pour faciliter la collaboration entre les fournisseurs de technologies d'authentification forte.

3. Terminologie des exigences

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].

4. Exigences d'algorithme

Cette Section présente les principales exigences qui sous-tendent la conception de l'algorithme. On a insisté sur la capacité d'utilisation du consommateur final ainsi que sur la capacité de mettre en œuvre l'algorithme sur des matériels de faible coût qui fournissent des capacités minimales à l'interface d'utilisateur. En particulier, la capacité d'incorporer l'algorithme dans des cartes SIM et Java à gros volume était un prérequis fondamental.

R1 - L'algorithme DOIT être fondé sur la séquence ou le compteur : un des buts est que l'algorithme HOTP soit incorporé dans des appareils à fort volume comme des cartes à mémoire Java, des clés USB, et des cartes SIM du GSM.

R2 - L'algorithme DEVRAIT être peu coûteux à mettre en œuvre dans le matériel en minimisant les exigences sur la batterie, le nombre de boutons, la puissance de calcul, et la taille de l'affichage LCD.

R3 - L'algorithme DOIT fonctionner avec des jetons qui ne prennent en charge aucune entrée numérique, mais PEUVENT aussi être utilisés avec des appareils plus sophistiqués comme des PIN-pads sécurisés.

R4 - La valeur affichée sur le jeton DOIT être facile à lire et à entrer par l'usager: Cela exige que la valeur HOTP soit de longueur raisonnable. La valeur HOTP DOIT être d'au moins six chiffres. Il est aussi souhaitable que la valeur HOTP soit "seulement numérique" afin qu'elle soit facilement entrée sur des appareils à fonctions restreintes comme les téléphones.

R5 - Il DOIT y avoir des mécanismes faciles pour resynchroniser le compteur. Le paragraphe 7.4 et l'Appendice E.4 détaillent le mécanisme de resynchronisation proposé dans ce document

R6 - L'algorithme DOIT utiliser un secret partagé fort. La longueur du secret partagé DOIT être d'au moins 128 bits. Le présent document RECOMMANDE un secret partagé d'une longueur de 160 bits.

5. Algorithme HOTP

Dans cette section, on introduit la notation et on décrit les blocs de base de l'algorithme HOTP -- la fonction de base pour calculer une valeur de HMAC-SHA-1 et la méthode de troncature pour extraire une valeur HOTP.

5.1 Notation et symboles

Une chaîne signifie toujours une chaîne binaire, c'est-à-dire une séquence de zéros et de uns.

Si s est une chaîne, alors |s| note sa longueur.

Si n est un nombre, alors |n| note sa valeur absolue.

Si s est une chaîne, alors s[i] note son ième bit. On commence les numéros de bits à 0, de sorte que s = s[0]s[1]...s[n-1] où n = |s| est la longueur de s.

Soit StToNum (chaîne à numéroter) pour noter la fonction qui à l'entrée d'une chaîne s retourne le nombre dont la

représentation binaire est s. (Par exemple, StToNum(110) = 6.)

Voici une liste des symboles utilisés dans ce document.

Symbole	Représente
C	valeur de compteur de 8 octets, le facteur de mouvement. Ce compteur DOIT être synchronisé avec le générateur HOTP (client) et le valideur HOTP (serveur).
K	secret partagé entre le client et le serveur ; chaque générateur HOTP a un secret différent et unique K.
T	paramètre de ralentissement d'admission : le serveur va refuser les connexions d'un usager après T tentatives d'authentification infructueuses.
s	paramètre de resynchronisation : le serveur va tenter de vérifier un authentifiant reçu sur s valeurs de compteur consécutives.
Digit	nombre de chiffres dans une valeur HOTP ; paramètre système.

5.2 Description

L'algorithme HOTP se fonde sur une valeur de compteur croissante et une clé symétrique statique connue seulement du jeton et du service de validation. Afin de créer la valeur HOTP, on va utiliser l'algorithme HMAC-SHA-1, comme défini dans la [RFC2104].

Comme le résultat du calcul de HMAC-SHA-1 fait 160 bits, on DOIT tronquer cette valeur à quelque chose qui puisse être facilement entré par un usager.

HOTP(K,C) = Truncate(HMAC-SHA-1(K,C))

où "Truncate" représente la fonction qui convertit une valeur HMAC-SHA-1 en une valeur HOTP comme défini au paragraphe 5.3.

La clé (K), le compteur (C), et les valeurs de données sont hachés avec les octets de poids fort en premier.

Les valeurs HOTP générées par le générateur HOTP sont traitées en ordre gros boutien.

5.3 Génération d'une valeur HOTP

On peut décrire les opérations en trois étapes distinctes :

Étape 1 : Générer une valeur HMAC-SHA-1. Soit HS = HMAC-SHA-1(K,C) ; HS est une chaîne de 20 octets

Étape 2 : Générer une chaîne de 4 octets (troncature dynamique). Soit Sbits = DT(HS) ; DT, défini plus loin, retourne une chaîne de 31 bits.

Étape 3: Calculer une valeur HOTP. Soit Snum = StToNum(Sbits) ; Convertit S en un nombre entre 0...2^{31}-1
Retourne D = Snum mod 10^Digit ; D est un nombre dans la gamme 0...10^{Digit}-1

La fonction Truncate effectue les étapes 2 et 3, c'est-à-dire, la troncature dynamique et ensuite la réduction modulo 10^Digit. L'objet de la technique de troncature à décalage dynamique est d'extraire un code binaire dynamique de quatre octets d'un résultat HMAC-SHA-1 de 160 bits (20 octets).

DT(String) // String = String[0]...String[19]
Soit OffsetBits les 4 bits de moindre poids de String[19]
Offset = StToNum(OffsetBits) // 0 ≤ OffSet ≤ 15
Soit P = String[OffSet]...String[OffSet+3]
Retourne les 31 derniers bits de P

La raison du masquage du bit de plus fort poids de P est d'éviter la confusion dans les calculs modulo entre signé et non signé. Des processeurs différents effectuent ces opérations de façon différente, et masquer les bits signés supprime toute ambiguïté.

Les mises en œuvre DOIVENT extraire un code de six chiffres au minimum et éventuellement un code de 7 et 8 chiffres. Selon les exigences de sécurité, Digit = 7 ou plus DEVRAIT être considéré afin d'extraire une plus longue valeur HOTP.

Le paragraphe qui suit est un exemple d'utilisation ce cette technique pour Digit = 6, c'est-à-dire, une valeur HOTP de six chiffres est calculée à partir de la valeur de HMAC.

5.4 Exemple de calcul HOTP pour Digit = 6

L'exemple de code qui suit décrit l'extraction d'un code binaire dynamique étant donné que hmac_result est une matrice d'octets avec le résultat HMAC-SHA-1 :

```
int offset = hmac_result[19] & 0xf ;
int bin_code = (hmac_result[offset]  & 0x7f) << 24
       | (hmac_result[offset+1] & 0xff) << 16
       | (hmac_result[offset+2] & 0xff) <<  8
       | (hmac_result[offset+3] & 0xff) ;
```

Octets du HMAC SHA-1 (exemple)

Numéro d'octet																			
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
Valeur d'octet																			
1f	86	98	69	0e	02	ca	16	61	85	50	ef	7f	19	da	8e	94	5b	55	5a

* Le dernier octet (octet 19) a la valeur hexadécimale 0x5a.
* La valeur des 4 bits inférieurs est 0xa (valeur du décalage).
* La valeur du décalage est l'octet 10 (0xa).
* La valeur des 4 octets commençant à l'octet 10 est 0x50ef7f19, qui est le code binaire dynamique DBC1.
* Le MSB de DBC1 est 0x50 donc DBC2 = DBC1 = 0x50ef7f19.
* HOTP = DBC2 modulo 10^6 = 872921.

On traite le code binaire dynamique comme un entier non signé de 31 bits en ordre gros boutien ; le premier octet est masqué avec un 0x7f.

On prend ensuite de nombre modulo 1 000 000 (10^6) pour générer la valeur de 6 chiffres de HOTP 872921 décimal.

6 Considérations sur la sécurité

La conclusion de l'analyse de la sécurité détaillée dans l'Appendice est que, pour tous les aspects pratiques, les résultats de la troncature dynamique (DT, *Dynamic Truncation*) sur les entrées de compteur distinctes sont des chaînes de 31 bits uniformément et indépendamment distribuées.

L'analyse de la sécurité détaille ensuite l'impact de la conversion d'une chaîne en un entier et de la réduction finale modulo 10^Digit, où Digit est le nombre de chiffres dans une valeur HOTP.

L'analyse démontre que ces étapes finales introduisent un biais négligeable, qui n'a pas d'impact sur la sécurité de l'algorithme HOTP, en ce sens que la meilleure attaque possible contre la fonction HOTP est l'attaque en force brute.

En supposant qu'un adversaire soit capable d'observer de nombreux échanges de protocole et de collecter les séquences de valeurs d'authentification réussies, cet adversaire, essayant de construire une fonction F pour générer des valeurs HOTP sur la base de ses observations, n'aura pas un avantage significatif sur un essai au hasard.

La conclusion logique est simplement que la meilleure stratégie sera là encore d'effectuer une attaque en force brute pour énumérer et essayer toutes les valeurs possibles.

En considérant l'analyse de la sécurité de l'Appendice de ce document, d'une façon générale, on peut approximer la sécurité

de l'algorithme HOTP par la formule suivante : Sec = sv/10^Digit
où :
- Sec est la probabilité de succès de l'adversaire ;
- s est la taille de la fenêtre de synchronisation vers l'avant ;
- v est le nombre de tentatives de vérification ;
- Digit est le nombre de chiffres dans les valeurs HOTP.

Évidemment, on peut jouer sur s, sur T (le paramètre de réduction de débit qui va limiter le nombre de tentatives par un attaquant) et sur Digit jusqu'à réaliser un certain niveau de sécurité, tout en préservant l'utilité du système.

7. Exigences pour la sécurité

Tout algorithme de mot de passe à utilisation unique est seulement aussi sûr que les protocoles d'application et d'authentification qui le mettent en œuvre. Donc, cette section discute des exigences critiques de sécurité qu'impose notre choix d'algorithme au protocole d'authentification et au logiciel de validation.

Les paramètres T et s discutés dans cette section ont un impact significatif sur la sécurité -- d'autres détails de la Section 6 développent les relations entre ces paramètres et leur impact sur la sécurité du système.

Il est aussi important de remarquer que l'algorithme HOTP n'est pas un substitut du chiffrement et n'assure pas la confidentialité de la transmission des données. D'autres mécanismes devraient être utilisés pour combattre les attaques visant la confidentialité des transactions.

7.1 Exigences sur le protocole d'authentification

On présente dans cette section quelques exigences pour un protocole P qui met en œuvre HOTP comme méthode d'authentification entre un prouveur et un vérifieur.

RP1 - P DOIT prendre en charge l'authentification à deux facteurs, c'est-à-dire, la communication et la vérification de quelque chose qu'on connaît (un code secret comme un mot de passe, une phrase, un code PIN, etc.) et quelque chose qu'on a (un jeton). Le code secret n'est connu que de l'usager et est généralement entré avec la valeur du mot de passe à utilisation unique pour les besoins de l'authentification (authentification à deux facteurs).

RP2 - P NE DEVRAIT PAS être vulnérable aux attaques en force brute. Cela implique qu'un schéma de réduction de débit/exclusion est RECOMMANDÉ du côté du serveur de validation.

RP3 - P DEVRAIT être mis en œuvre sur un canal sûr afin de protéger la confidentialité de l'utilisateur et éviter les attaques en répétition.

7.2 Validation des valeurs HOTP

Le client HOTP (jeton matériel ou logiciel) incrémente son compteur puis calcule la prochaine valeur HOTP du client HOTP. Si la valeur reçue par le serveur d'authentification correspond à celle calculée par le client, la valeur HOTP est alors validée. Dans ce cas, le serveur incrémente de un la valeur du compteur.

Si la valeur reçue par le serveur ne correspond pas à la valeur calculée par le client, le serveur initie le protocole de resynchronisation (fenêtre de regard vers l'avant) avant qu'il ne demande un autre mot de passe.

Si la resynchronisation échoue, le serveur demande un autre mot de passe d'authentification au protocole, jusqu'à atteindre le nombre maximum de tentatives autorisé.

Si et quand le nombre maximum de tentatives autorisé est atteint, le serveur DEVRAIT verrouiller le compte et initier une procédure pour informer l'utilisateur.

7.3 Réduction de débit chez le serveur

Tronquer la valeur du HMAC-SHA-1 à une valeur plus courte rend possible une attaque en force brute. Donc, le serveur d'authentification a besoin de détecter et arrêter les attaques en force brute.

On RECOMMANDE d'établir un paramètre de réduction de débit T, qui définit le nombre maximum de tentatives possibles de validation du mot de passe à utilisation unique. Le serveur de validation gère des compteurs individuels par appareil HOTP afin de prendre note de tout échec. On RECOMMANDE que T ne soit pas trop grand, en particulier si la méthode de resynchronisation utilisée sur le serveur est fondée sur la fenêtre, et si la taille de la fenêtre est grande. T DEVRAIT être réglé aussi bas que possible, tout en assurant que l'utilisation n'est pas impactée de façon significative.

Une autre option serait de mettre en œuvre un schéma de retard pour éviter une attaque en force brute. Après chaque échec de tentative A, le serveur d'authentification va attendre un nombre accru T*A de secondes, par exemple, disons T = 5, ensuite après 1 tentative, le serveur attend 5 secondes, au second échec, il attend 5*2 = 10 secondes, etc.

Les schémas de retard ou de verrouillage DOIVENT être maintenus à travers les sessions d'enregistrement pour empêcher les attaques fondées sur des techniques de tâtonnements multiples en parallèle.

7.4 Resynchronisation du compteur

Bien que la valeur du compteur du serveur ne soit incrémentée qu'après une authentification HOTP réussie, le compteur sur le jeton est incrémenté chaque fois qu'un nouveau HOTP est demandé par l'utilisateur. À cause de cela, les valeurs de compteur sur le serveur et sur le jeton peuvent être désynchronisées.

On RECOMMANDE d'établir un paramètre s de verrouillage sur le serveur, qui définit la taille de la fenêtre de recherche vers l'avant. Dans un très petit espace, le serveur peut recalculer les valeurs du prochain s du serveur HOTP, et les vérifier par rapport à celles reçues au client HOTP.

La synchronisation des compteurs dans ce scénario exige simplement que le serveur calcule les prochaine valeurs HOTP et détermine si il y a correspondance. Facultativement, le système PEUT exiger que l'utilisateur envoie une séquence de (disons, 2, 3) valeurs de HOTP pour les besoins de la resynchronisation, car falsifier une séquence de valeurs de HOTP consécutives est encore plus difficile que de deviner une seule valeur de HOTP.

La limite supérieure fixée par le paramètre s assure que le serveur ne va pas vérifier les valeurs de HOTP pour toujours (causant une attaque de déni de service) et aussi restreint l'espace des solutions possibles pour un attaquant qui essaye de fabriquer des valeurs de HOTP. s DEVRAIT être réglé aussi bas que possible, tout en assurant que l'utilisation n'est pas impactée.

7.5 Gestion des secrets partagés

Les opérations sur les secrets partagés utilisés pour générer et vérifier les valeurs OTP DOIVENT être effectuées de façon sûre, afin d'atténuer les risques de fuite d'informations sensibles. On décrit dans ce paragraphe différents modes de fonctionnement et techniques pour effectuer ces différentes opérations par rapport à l'état de l'art dans la sécurité des données.

On peut considérer deux voies différentes pour générer et mémoriser (en toute sécurité) les secrets partagés dans le système de validation :

* Génération déterministe : les secrets sont déduits d'un germe maître, aux deux étapes de provisionnement et de vérification et générés en vol chaque fois que nécessaire.
* Génération aléatoire : les secrets sont générés au hasard à l'étape de provisionnement et DOIVENT être mémorisés immédiatement et gardés en sécurité durant leur cycle de vie.

Génération déterministe :
Une stratégie possible est de déduire les secrets partagés d'un secret maître. Le secret maître va être mémorisé seulement chez le serveur. Un appareil résistant à l'altération DOIT être utilisé pour mémoriser la clé maîtresse et déduire les secrets partagés de la clé maîtresse et de certaines informations publiques. Le principal avantage serait d'éviter l'exposition des secrets partagés et aussi d'éviter des exigences spécifiques sur la mémorisation, car les secrets partagés pourraient être générés à la demande quand nécessaire au moment du provisionnement et de la validation.

On distingue deux cas différents :

- Une seule clé maîtresse MK est utilisée pour déduire les secrets partagés ; chaque appareil HOTP a un secret différent, K_i = SHA-1 (MK,i) où i est un élément d'information public qui identifie de façon univoque l'appareil HOTP, comme un numéro de série, un identifiant de jeton, etc. Évidemment, c'est dans le contexte d'une application ou d'un service -- différentes application ou fournisseurs de service vont avoir des secrets et réglages différents.

- Plusieurs clés maîtresses MK_i sont utilisées et chaque appareil HOTP mémorise un jeu des différents secrets dérivés, {K_i,j = SHA-1(MK_i,j)} où j note un élément d'informations public qui identifie l'appareil. L'idée serait de mémoriser SEULEMENT la clé maîtresse active au serveur de validation, dans le module de sécurité du matériel (HSM, *Hardware Security Module*) et de la garder dans un endroit sûr, en utilisant des méthodes de partage de secret comme [Shamir] par exemple. Dans ce cas, si un secret maître MK_i est compromis, il est alors possible de passer à un autre secret sans remplacer tous les appareils.

L'inconvénient du cas déterministe est que l'exposition du secret maître va évidemment permettre à un attaquant de reconstruire tout secret partagé sur la base des informations publiques correctes. La révocation de tous les secrets va être nécessaire, ou de passer à un nouveau jeu de secrets dans le cas de clés maîtresses multiples.

Par ailleurs, l'appareil utilisé pour mémoriser la ou les clés maîtresses et générer les secrets partagés DOIT être résistant à l'altération. De plus, le HSM ne sera pas exposé en dehors du périmètre de sécurité du système de validation, réduisant donc le risque de fuites.

Génération aléatoire :
Les secrets partagés sont générés au hasard. On RECOMMANDE de suivre les recommandations de la [RFC4086] et de choisir une bonne et sûre source d'aléa pour générer ces secrets. Un (vrai) générateur de nombres aléatoires exige une source d'aléa se produisant naturellement. En pratique, il y a deux voies possibles pour considérer la génération des secrets partagés :

* Générateurs fondés sur le matériel : ils exploitent les aléas qui se produisent dans les phénomènes physiques. Une bonne mise en œuvre peut se fonder sur un oscillateur et l'incorporer de telle façon qu'il soit plus difficile d'effectuer une attaque active.
* Générateurs fondés sur le logiciel : concevoir un bon générateur de nombres aléatoires logiciel n'est pas une tâche facile. Une mise en œuvre simple mais efficace devrait se fonder sur des sources diverses et appliquer à la séquence échantillonnée une fonction unidirectionnelle telle que SHA-1.

On RECOMMANDE de choisir des produits éprouvés, générateurs matériels ou logiciels, pour le calcul des secrets partagés.

On RECOMMANDE aussi de mémoriser les secrets partagés de façon sécurisée, et plus spécifiquement de les chiffrer quand ils sont mémorisés en utilisant un chiffrement de matériel résistant à l'altération et en ne les exposant que lorsque nécessaire : par exemple, le secret partagé est déchiffré quand nécessaire pour vérifier une valeur HOTP, et rechiffré immédiatement pour limiter l'exposition dans la RAM pendant un bref instant. Le magasin de données qui détient les secrets partagés DOIT être dans une zone sûre, pour éviter autant que possible une attaque directe sur le système de validation et la base de données de secrets.

En particulier, l'accès aux secrets partagés devrait être limité aux seuls programmes et processus requis par le système de validation. On n'en dit pas plus sur les différents mécanismes de sécurité à mettre en place, mais évidemment, la protection des secrets partagés est de la plus haute importance.

8. Secrets partagés composites

Il peut être souhaitable d'inclure des facteurs d'authentification supplémentaires dans le secret partagé K. Ces facteurs supplémentaires peuvent consister en toutes données connues du jeton mais pas obtenues facilement par d'autres. Des exemples de telles données incluent :

* un PIN ou mot de passe obtenu comme entrée de l'utilisateur au jeton,
* un numéro de téléphone
* un identifiant univoque disponible programmatiquement au jeton.

Dans ce scénario, le secret partagé composite K est construit durant le processus de provisionnement à partir d'une valeur de germe aléatoire combinée avec un ou plusieurs facteurs d'authentification supplémentaires. Le serveur peut soit construire à la demande soit mémoriser des secrets composites -- dans tous les cas, selon les choix de la mise en œuvre, le jeton mémorise seulement la valeur du germe. Quand le jeton effectue le calcul du HOTP, il calcule K à partir de la valeur du germe et les valeurs déduites localement ou entrées des autres facteurs d'authentification.

L'utilisation de secrets partagés composites peut renforcer les systèmes d'authentification fondés sur HOTP par l'inclusion de facteurs d'authentification supplémentaires au jeton. Dans la mesure où le jeton est un appareil de confiance, cette approche présente l'avantage supplémentaire de ne pas exiger d'exposition des facteurs d'authentification (comme le PIN entré par l'utilisateur) aux autres appareils.

9. Authentification bidirectionnelle

Il est assez intéressant de noter que le client HOTP pourrait aussi être utilisé pour authentifier le serveur de validation, qui revendique d'être une vraie entité connaissant le secret partagé.

Comme le client et le serveur HOTP sont synchronisés et partagent le même secret (ou une méthode pour le recalculer) un simple protocole en trois passes pourrait être mis en place :
1- l'utilisateur final entre l'identifiant de jeton et une première valeur d'OTP, OTP1 ;
2- le serveur vérifie si OTP1 est correct, et si oui renvoie OTP2 ;
3- l'utilisateur final vérifie OTP2 en utilisant son appareil HOTP et si il est correct, utilise le site.

Évidemment, comme indiqué précédemment, toutes les communications OTP doivent se faire sur un canal sûr, par exemple, des connexions SSL/TLS, IPsec.

10. Conclusion

Ce document décrit HOTP, algorithme de mot de passe à utilisation unique fondé sur HMAC. Il recommande aussi la mise en œuvre préférée et les modes de fonctionnement en relation pour déployer l'algorithme.

Le document présente aussi les éléments de sécurité et démontre que l'algorithme HOTP est pratique et pertinent, la meilleure attaque possible étant une attaque en force brute qui peut être empêchée par une mise en œuvre soigneuse des contre mesures au serveur de validation.

Finalement, plusieurs améliorations ont été proposées, afin d'améliorer si nécessaire la sécurité pour des applications spécifiques.

11. Remerciements

Les auteurs tiennent à remercier Siddharth Bajaj, Alex Deacon, Loren Hart, et Nico Popp de leur aide durant la conception et la rédaction de ce document.

12. Contributeurs

Les auteurs de ce document tiennent à souligner le rôle de trois personnes qui ont fait des contributions capitales au présent document :
- Laszlo Elteto qui est architecte système chez SafeNet, Inc.
- Ernesto Frutos qui est directeur de l'ingénierie chez Authenex, Inc.
- Fred McClain qui est fondateur et CTO de Boojum Mobile, Inc.
Sans leurs avis et précieux apports, le présent document ne serait pas le même.

13. Références

13.1 Références normatives

[BCK1] M. Bellare, R. Canetti and H. Krawczyk, "Keyed Hash Functions and Message Authentication", Proceedings of Crypto'96, LNCS Vol. 1109, pp. 1-15.

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

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

(MàJ par RFC8174)

[RFC3979] S. Bradner, éd., "Droits de propriété intellectuelle dans les technologies de l'IETF", mars 2005. (*MàJ par* RFC4879) (BCP0079 ; *Remplacé par* RFC8179)

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

13.2 Références pour information

[OATH] Initiative for Open AuTHentication. http://www.openauthentication.org

[PrOo] B. Preneel, P. van Oorschot, "MD-x MAC and building fast MACs from hash functions", Advances in Cryptology CRYPTO '95, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[Crack] Crack in SHA-1 code 'stuns' security gurus. http://www.eetimes.com/showArticle.jhtml?articleID=60402150

[Sha1] Bruce Schneier. "HA-1 broken". 15 février 2005.
http://www.schneier.com/blog/archives/2005/02/sha1_broken.html

[Res] "Researchers: Digital encryption standard flawed"
http://news.com.com/Researchers+Digital+encryption+standard+flawed/2100-1002-5579881.html?part=dht&tag=ntop&tag=nl.e703

[Shamir] Adi Shamir. "How to Share a Secret", dans Communications of the ACM, Vol. 22, n° 11, pp. 612-613, novembre 1979.

Appendice A. Analyse détaillée de la sécurité de l'algorithme HOTP

La présente section résume l'analyse de la sécurité de l'algorithme HOTP. On détaille d'abord les meilleures stratégies d'attaque, et on discute ensuite de la sécurité dans diverses hypothèses et de l'impact de la troncature et on fait quelques recommandations sur le nombre de chiffres.

L'analyse se concentre sur le cas où Digit = 6, c'est-à-dire, une fonction HOTP qui produit des valeurs de 6 chiffres, qui est le minimum recommandé dans ce document.

A.1 Définitions et notation

On note {0,1}^l l'ensemble de toutes les chaînes de longueur l.

Soit Z_{n} = {0,.., n - 1}.

On note IntDiv(a,b) l'algorithme de division d'entier qui prend en entrée les entiers a, b où a ≤ b ≤ 1 et retourne les entiers (q,r) respectivement le quotient et le reste de la division de a par b. (Donc, a = bq + r et 0 ≤ r < b.)

Soit H : {0,1}^k x {0,1}^c --> {0,1}^n la fonction de base qui prend une clé K de k bits et le compteur C de c bits et retourne un résultat H(K,C) de n bits. (Dans le cas de HOTP, H est HMAC-SHA-1 ; on utilise cette définition formelle pour généraliser notre preuve de sécurité.)

A.2 Algorithme idéalisé : HOTP-IDEAL

On définit maintenant une instance idéalisée de l'algorithme HOTP. Dans cet algorithme, le rôle de H est joué par une fonction aléatoire qui forme la clé.

Pour être plus précis, on note par Maps(c,n) l'ensemble de toutes les fonctions qui se transposent de {0,1}^c en {0,1}^n. L'algorithme idéalisé a un espace de clés Maps(c,n), de sorte que une "clé" pour un tel algorithme est une fonction h de {0,1}^c à {0,1}^n. On imagine cette clé (fonction) comme étant aléatoire. Il n'est pas faisable de mettre en œuvre cet

algorithme idéalisé, car la clé, étant une fonction de {0,1}^c à {0,1}^n, est trop grande pour la mémoriser. Alors, pourquoi l'examiner ?

Notre analyse de sécurité va montrer que tant que H satisfait à une certaine hypothèse bien acceptée, la sécurité des algorithmes réels et idéalisés est la même pour tous les aspects pratiques. La tâche qui se présente à nous est alors d'attester de la sécurité de l'algorithme idéalisé.

En analysant l'algorithme idéalisé, on se concentre sur d'affirmation de la qualité de la conception de l'algorithme lui-même, indépendamment de HMAC-SHA-1. C'est en fait la question importante.

A.3 Modèle de sécurité

Le modèle présente les types de menaces ou d'attaques qui sont considérées et permet de vérifier la sécurité de HOTP et de HOTP-IDEAL. On note ALG comme étant HOTP ou HOTP-IDEAL pour les besoins de cette analyse de la sécurité.

Le scénario qu'on examine est celui d'un utilisateur et d'un serveur qui partagent une clé K pour ALG. Tous deux tiennent un compteur C, initialement à zéro, et l'utilisateur s'authentifie en envoyant ALG(K,C) au serveur. Ce dernier accepte si cette valeur est correcte.

Afin de se protéger contre un incrément accidentel du compteur de l'usager, le serveur, à réception d'une valeur z, va accepter pour autant que z égale ALG(K,i) pour un certain i dans la gamme C,...,C + s-1, où s est le paramètre de resynchronisation et C est le compteur du serveur. Si il accepte avec une certaine valeur de i, il incrémente alors son compteur à i+1. Si il n'accepte pas, il ne change pas la valeur de son compteur.

Le modèle qu'on spécifie saisit ce qu'un adversaire peut faire et qu'il a besoin de réaliser afin de "gagner". D'abord, l'adversaire est supposé être capable d'espionner, ce qui signifie, de voir l'authentificateur transmis par l'usager. Ensuite, l'adversaire gagne si il peut obtenir que le serveur accepte un authentificateur relatif à une valeur de compteur pour laquelle l'usager n'a jamais transmis un authentificateur.

L'adversaire formel, qu'on note B, commence par savoir quel algorithme ALG est utilisé, connaissant la conception du système, et connaissant tous les paramètres du système. La seule et unique chose qui n'est pas donnée a priori est la clé K partagée entre l'usager et le serveur.

Le modèle donne à B le plein contrôle de l'ordre des événements. Il a accès à un oracle d'authentificateurs qui représente l'usager. En invoquant cet oracle, l'adversaire peut demander à l'utilisateur de s'authentifier et obtenir l'authentificateur en retour. Il peut invoquer cet oracle aussi souvent qu'il le veut et quand il veut, en utilisant les authentificateurs qu'il accumule pour peut-être "apprendre" comment faire des authentificateurs lui-même. À tout moment, il peut aussi invoquer un oracle de vérification, auquel il fournit un candidat authentificateur de son choix. Il gagne si le serveur accepte cet authentificateur.

Considérons le jeu suivant qui implique un adversaire B qui tente de compromettre la sécurité d'un algorithme d'authentification ALG : K x {0,1}^c --> R.

Initialisation - une clé K est choisie au hasard à partir de K, un compteur C est initialisé à 0, et la valeur booléenne win est réglée à faux.

Exécution du jeu : l'adversaire B reçoit les deux oracles suivants :

Oracle AuthO()
A = ALG(K,C)
C = C + 1
Retourne O à B

Oracle VerO(A)
i = C
Tandis que (i ≤ C + s - 1 et Win == FAUX) faire
Si A == ALG(K,i) alors Win = VRAI ; C = i + 1
Autrement i = i + 1
Retourne Win à B

AuthO() est l'oracle authentificateur et VerO(A) est l'oracle de vérification.

À l'exécution, B interroge les deux oracles à volonté. Soit Adv(B) la probabilité que win soit réglé à vrai dans le jeu ci-dessus. C'est la probabilité que l'adversaire réussisse à se faire passer pour l'utilisateur.

Notre but est d'évaluer cette valeur en fonction du nombre v d'interrogations de vérification faites par B, le nombre a d'interrogations de l'oracle d'authentificateur faites par B, et le temps courant t de B. Cela va nous dire comment régler le réducteur de débit, qui est la limite supérieure effective de v.

A.4 Sécurité de l'algorithme idéal d'authentification

Ce paragraphe résume l'analyse de sécurité de HOTP-IDEAL, en commençant par l'impact de la conversion modulo 10^Digit et ensuite en se concentrant sur les différentes attaques possibles.

A.4.1 Des bits aux chiffres

La troncature dynamique du décalage d'un chaîne binaire aléatoire de n bits donne une chaîne aléatoire de 31 bits. Qu'arrive t-il à la distribution quand elle est prise avec modulo m = 10^Digit, comme le fait HOTP ?

Le lemme suivant estime les biais des résultats dans ce cas.

Lemme 1
Soit $N \leq m \leq 1$ des entiers, et soit (q,r) = IntDiv(N,m). Pour z dans Z_{m} soit t :
P_{N,m}(z) = Pr [x mod m = z : x pris au hasard dans Z_{n}]

Alors pour tout z dans Z_{m}
P_{N,m}(z) = (q + 1) / N si $0 \leq z < r$ q / N si $r \leq z < m$

Preuve du lemme 1
Soit la variable aléatoire X uniformément distribuée sur Z_{N}. Alors :
P_{N,m}(z) = Pr [X mod m = z]
= Pr [X < mq] * Pr [X mod m = z| X < mq] + Pr [$mq \leq X < N$] * Pr [X mod m = z| $mq \leq X < N$]
= mq/N * 1/m + (N - mq)/N * 1 / (N - mq) si $0 \leq z < N - mq$
0 si $N - mq \leq z \leq m$
= q/N + r/N * 1 / r si $0 \leq z < N - mq$
0 si $r \leq z \leq m$

En simplifiant ont obtient l'équation voulue.
Soit N = 2^31, d = 6, et m = 10^d. Si x est choisi au hasard dans Z_{N} (est une chaîne aléatoire de 31 bits) alors la réduire à un nombre de 6 chiffres en prenant x mod m ne donne pas un nombre aléatoire de 6 chiffres.

Plutôt, x mod m est distribué comme le montre le tableau suivant :

Valeurs	Probabilité que chacun apparaisse comme résultat
0,1,...,483647	2148/2^31 en gros égal à 1,00024045/10^6
483648,...,999999	2147/2^31 en gros égal à 0,99977478/10^6

Si X est uniformément distribué sur Z_{2^31} (est une chaîne aléatoire de 31 bits) alors le tableau ci-dessus montre les probabilités pour les différents résultats de X mod 10^6. Le premier jeu de valeurs apparaît avec une probabilité légèrement supérieure à 10^-6, le reste avec une probabilité légèrement moindre, ce qui signifie que la distribution est légèrement non uniforme.

Cependant, comme l'indique le tableau ci-dessus, le biais est faible, et comme on le verra plus loin, négligeable : les probabilités sont très proches de 10^-6.

A.4.2 Attaques en force brute

Si l'authentificateur consiste en d chiffres aléatoires, une attaque en force brute utilisant v tentatives de vérification va réussir avec une probabilité de sv/10^Digit.

Cependant, un adversaire peut exploiter le biais dans les résultats de HOTP-IDEAL, prédit par le lemme 1, pour monter une

attaque légèrement meilleure. À savoir qu'il fait les tentatives d'authentification avec des authentificateurs qui sont les valeurs les plus probables, c'est à dire celles dans la gamme 0,...,r - 1, où (q,r) = IntDiv(2^31,10^Digit).

On spécifie ensuite qu'un adversaire dans notre modèle de sécurité monte l'attaque. Il estime la probabilité de succès comme une fonction du nombre d'interrogations de vérification.

Pour simplifier, on suppose que le nombre d'interrogations de vérification est au plus de r. Avec N = 2^31 et m = 10^6, on a r = 483 648, et la valeur de réduction de débit est certainement moins que cela, de sorte que cette hypothèse n'est pas tellement une restriction.

Proposition 1
Supposons m = 10^Digit < 2^31, et soit (q,r) = IntDiv(2^31,m). Supposons s ≤ m. L'adversaire d'attaque en force brute B-bf attaque HOTP en utilisant v ≤ r interrogations de l'oracle de vérification. Cet adversaire ne fait pas d'interrogation de l'oracle d'authentificateur, et réussit avec la probabilité Adv(B-bf) = 1 - (1 - v(q+1)/2^31)^s qui est en gros égale à sv * (q+1)/2^31.

Avec m = 10^6, on obtient q = 2 147. Dans ce cas, l'attaque en force brute utilisant v tentatives de vérification réussit avec la probabilité Adv(B-bf), en gros = sv * 2148/2^31 = sv * 1,00024045/10^6.

Comme le montre cette équation, le paramètre de resynchronisation s a un impact significatif sur la probabilité de succès de l'adversaire qui est proportionnelle à s. Cela signifie que s ne doit pas être trop grand sinon la sécurité est compromise.

A.4.3 Les attaques en force brute sont les meilleures attaques possibles

Une question centrale est de savoir si il y a des attaques meilleures que celle en force brute. En particulier, l'attaque en force brute ne tente pas de collecter les authentificateurs envoyés par l'utilisateur et d'essayer de les analyser pour tenter d'apprendre comment mieux construire les authentificateurs. Est-ce que cela aiderait ? Y a-t-il un moyen "d'apprendre" comment construire des authentificateurs qui résulte en un taux de succès plus élevé que celui donné par l'attaque en force brute ?

Ce qui suit nous dit que la réponse à ces questions est non. Quelle que soit la stratégie de l'adversaire, et même si il voit, et essaye d'exploiter, les authentificateurs provenant des tentatives d'authentification de l'utilisateur, sa probabilité de réussite ne va pas être au dessus de celle de l'attaque en force brute -- ceci est vrai tant que le nombre d'authentifications qu'il observe n'est pas incroyablement grand. C'est une information précieuse concernant la sécurité du schéma.

Proposition 2
Supposons m = 10^Digit < 2^31, et soit (q,r) = IntDiv(2^31,m). Soit B tout adversaire qui attaque HOTP-IDEAL en utilisant v interrogations d'oracle de vérification et a ≤ 2^c - s interrogations d'oracle de vérification. Alors

$$Adv(B) \leq sv * (q+1)/ 2^{31}$$

Note : Ce résultat est à condition que l'adversaire ne voit pas plus de 2^c - s authentifications effectuées par l'usager, qui est difficilement restrictive tant que c est assez grand.

Avec m = 10^6, on obtient q = 2 147. Dans ce cas, la proposition 2 dit que tout adversaire B attaquant HOTP-IDEAL et faisant v tentatives de vérification a une probabilité de réussite d'au plus :

équation 1 : sv * 2148/2^31, en gros = sv * 1,00024045/10^6

Ce qui signifie que le taux de succès de B n'est pas supérieur à ce qui est réalisé par l'attaque en force brute.

A.5 Analyse de la sécurité de HOTP

On a analysé dans les paragraphes qui précèdent la sécurité de l'instance idéalisée de HOTP-IDEAL de l'algorithme d'authentification HOTP réel. On va montrer maintenant que sous les hypothèses appropriées et bien acceptées sur H, la sécurité des algorithmes réels est essentiellement la même que celle de son instance idéalisée.

L'hypothèse en question est que H est une fonction pseudo aléatoire (PRF) sûre, ce qui signifie que ses valeurs d'entrée-sortie sont indistinguables en pratique de celles d'une fonction aléatoire.

Considérons un adversaire A qui a un oracle pour une fonction f : {0,1}^c --> {0, 1}^n et éventuellement sort un bit. On note Adv(A) comme l'avantage de PRF de A, qui représente comment l'adversaire fait pour distinguer le cas où son oracle est H(K,.) du cas où son oracle est une fonction aléatoire de {0,1}^c à {0,1}^n.

Une attaque possible se fonde sur une recherche exhaustive de la clé K. Si A fait t étapes et si T note le temps pour effectuer un calcul de H, son avantage de prf pour cette attaque se trouve être (t/T)2^-k. Un autre attaque possible est celle de l'anniversaire [PrOo], par laquelle A peut obtenir l'avantage p^2/2^n sur p interrogations de l'oracle et un temps courant d'environ pT.

Notre hypothèse est que ce sont les meilleures attaques possibles. Ceci se traduit par ce qui suit.

Hypothèse 1
Soit T qui note le temps pour effectuer un calcul de H. Alors si A est un adversaire avec le temps courant d'au plus t et faisant au plus p interrogations de l'oracle, Adv(A) ≤ (t/T)/2^k + p^2/2^n.

En pratique, cette hypothèse signifie que H est très sûre comme PRF. Par exemple, avec k = n = 160, un attaquant avec un temps courant de 2^60 et faisant 2^40 interrogations d'oracle a un avantage d'au plus (environ) 2^-80.

Théorème 1
Supposons m = 10^Digit < 2^31, et soit (q,r) = IntDiv(2^31,m). Soit B tout adversaire attaquant HOTP en utilisant v interrogations de l'oracle de vérification, a ≤ 2^c - s interrogations de l'oracle d'authentification, et le temps courant t. Soit T qui note le temps pour effectuer un calcul de H.
Si l'hypothèse 1 est vraie, alors Adv(B) ≤ sv * (q + 1)/2^31 + (t/T)/2^k + ((sv + a)^2)/2^n

En pratique, le terme (t/T)2^-k + ((sv + a)^2)2^-n est beaucoup plus petit que le terme sv(q + 1)/2^n, de sorte que cela dit que pour tous les aspects pratiques le taux de succès d'un adversaire qui attaque HOTP est sv(q + 1)/2^n, juste comme pour HOTP-IDEAL, ce qui signifie que l'algorithme HOTP est en pratique essentiellement aussi bon que son instance idéale.

Dans le cas de m = 10^6 d'un résultat à six chiffres, cela signifie qu'un adversaire qui fait v tentatives d'authentification va avoir un taux de succès d'au plus celui de l'équation 1.

Par exemple, considérons un adversaire avec un temps courant d'au plus 2^60 qui voit au plus 2^40 tentatives d'authentification de l'utilisateur. Ces deux choix sont très généreux pour l'adversaire, qui ne va normalement pas avoir ces ressources, mais on dit que même un adversaire aussi puissant n'aura pas plus de succès qu'indiqué par l'équation 1.

On peut en toute sécurité supposer sv ≤ 2^40 du fait de la réduction de débit et des limites sur s. Donc :

(t/T)/2^k + ((sv + a)^2)/2^n ≤ 2^60/2^160 + (2^41)^2/2^160, en gros ≤ 2^-78.

ce qui est beaucoup plus petit que la probabilité de succès de l'équation 1 et négligeable par rapport à elle.

Appendice B. Attaques sur SHA-1

Cette Section traite de l'impact de récentes attaques sur SHA-1 contre la sécurité du HOTP fondé sur HMAC-SHA-1. On commence par une discussion de la situation de SHA-1 et ensuite on discute de sa pertinence pour HMAC-SHA-1 et HOTP. Les références citées sont à la Section 13.

B.1 État de SHA-1

Une collision pour une fonction de hachage h signifie une paire x,y d'entrées différentes telles que h(x)=h(y). Comme les résultats de SHA-1 sont de 160 bits, une attaque de l'anniversaire trouve une collision dans 2^{80} essais. (Un essai signifie un calcul de la fonction.) Ceci a été jugé être le mieux possible jusqu'à ce que Wang, Yin, et Yu annoncent le 15 février 2005, qu'ils avaient eu une attaque trouvant des collisions en 2^{69} essais.

SHA-1 est-il cassé ? Pour la plupart des objets pratiques, on dirait probablement non, car les ressources nécessaires pour monter l'attaque sont énormes. On a un moyen pour en avoir une idée : on peut estimer que c'est à peu près le même délai que ce qu'il faudrait pour factoriser un module RSA de 760 bits, et ceci est actuellement considéré comme hors de portée.

Burr du NIST est cité dans [Crack] comme disant "Les grandes agences nationale de renseignement pourraient faire cela

dans un délai raisonnable avec quelques millions de dollars de temps d'ordinateur". Cependant, le calcul peut être hors de la portée de tout ce qui n'est pas une agence bien dotée en moyens.

On devrait aussi se demander quel impact a réellement le fait de trouver des collisions de SHA-1 sur la sécurité des applications réelles comme les signatures. Pour exploiter une collision x,y à falsifier des signatures, on a besoin d'obtenir une signature de x et ensuite on peut falsifier une signature de y. Les dommages qui peuvent en résulter dépendent du contenu de y : le y créé par l'attaque peut n'être pas significatif dans le contexte de l'application. Aussi, on a besoin d'une attaque de message choisi pour obtenir la signature de x. Cela semble possible dans certains contextes, mais pas dans d'autres. Globalement, il n'est pas clair que l'impact sur la sécurité des signatures soit significatif.

Bien sûr, on peut lire dans la presse que SHA-1 est "cassé" [Sha1] et que le chiffrement et SSL sont "cassés" [Res]. La presse a tendance a surévaluer les événements : il serait assez peu intéressant d'annoncer dans les nouvelles qu'une équipe de cryptanalystes a fait un travail théorique très intéressant en attaquant SHA-1.

Les cryptographes sont aussi excités. Mais principalement parce que c'est une percée théorique importante. Les attaques ne peuvent que s'améliorer avec le temps : il est donc important de surveiller tous les progrès des cryptanalyses de fonctions de hachage et être prêt pour toute percée pratique réelle avec un plan réfléchi de migration pour l'avenir.

B.2 État de HMAC-SHA-1

Les nouvelles attaques sur SHA-1 n'ont pas d'impact sur la sécurité de HMAC-SHA-1. La meilleure attaque sur ce dernier reste qu'on a besoin qu'un envoyeur authentifie 2^{80} messages avant qu'un adversaire puisse créer un message falsifié. Pourquoi ?

HMAC n'est pas une fonction de hachage. C'est un code d'authentification de message (MAC) qui utilise une fonction de hachage en interne. Un MAC dépend d'une clé secrète, ce qui n'est pas le cas d'une fonction de hachage. Ce dont on a besoin de se soucier pour un MAC est d'une falsification, pas d'une collision. HMAC a été conçu afin que des collisions dans la fonction de hachage (ici SHA-1) ne donnent pas de falsification pour HMAC.

On se rappelle que HMAC-SHA-1(K,x) = SHA-1(K_o,SHA-1(K_i,x)) où les clés K_o,K_i sont déduites de K. Supposons que l'attaquant trouve une paire x,y telle que SHA-1(K_i,x) = SHA-1(K_i,y) (on appelle cela une collision de clé cachée.) Alors si il peut obtenir le MAC de x (lui-même de grand ordre) il peut falsifier le MAC de y. (Ces valeurs sont les mêmes.) Mais trouver des collisions de clé cachée est plus difficile que de trouver des collisions, parce que l'attaquant ne connaît pas la clé cachée K_i. Tout ce qu'il peut avoir sont quelques résultats de HMAC-SHA-1 avec la clé K. Aujourd'hui, il n'y a pas de revendication ni de preuve que les récentes attaques sur SHA-1 aient réussi à trouver des collisions de clé cachée.

Historiquement, le concept de HMAC a déjà fait ses preuves à cet égard. MD5 est considéré comme cassé en ce que des collisions dans cette fonction de hachage peuvent être trouvées relativement facilement. Mais il n'y a toujours pas d'attaque sur HMAC-MD5 meilleure que celle triviale de 2^{64} de l'anniversaire. (MD5 sort 128 bits, non 160.) On voit que cette force de HMAC est encore en jeu dans le contexte de SHA-1.

B.3 État de HOTP

Comme aucune nouvelle faiblesse n'est apparue dans HMAC-SHA-1, il n'y a pas d'impact sur HOTP. Les meilleures attaques sur HOTP restent celles décrites dans ce document, à savoir d'essayer de deviner les valeurs de résultat.

La preuve de sécurité de HOTP exige que HMAC-SHA-1 se comporte comme une fonction pseudo aléatoire. La qualité de HMAC-SHA-1 comme fonction pseudo aléatoire n'est pas impactée par les nouvelles attaques sur SHA-1, et donc aucune de ces garanties n'est prouvée.

Appendice C. Mise en œuvre de référence de l'algorithme HOTP

```
/* OneTimePasswordAlgorithm.java
* OATH Initiative,
*Algorithme HOTP de mot de passe à utilisation unique */

/* Copyright (C) 2004, OATH. Tous droits réservés.
* Licence de copie et d'utilisation de ce logiciel est accordée pourvu qu'il soit identifié comme "Algorithme OATH HOTP"
    dans tous les matériels qui mentionnent ou font référence à ce logiciel ou cette fonction. Licence est aussi accordée de
```

faire et utiliser des travaux dérivés pourvu que de tels travaux soient identifiés comme "dérivés de l'algorithme OATH HOTP" dans tous les matériaux qui mentionnent ou font référence au travail dérivé. OATH (Open AuTHentication) et ses membres ne font aucune représentation concernant la possibilité de commercialiser le logiciel ni la convenance de ce logiciel pour aucun objet particulier. Il est fourni "tel qu'il est" sans garantie exprimée ou implicite d'aucune sorte et OATH et ses membres renoncent expressément à toute garantie ou responsabilité d'aucune sorte relative à ce logiciel. Ces notices DOIVENT être conservées dans toutes copies de toute partie de ce document et/ou logiciel. */

```
package org.openauthentication.otp;

  import java.io.IOException;
  import java.io.File;
  import java.io.DataInputStream;
  import java.io.FileInputStream ;
  import java.lang.reflect.UndeclaredThrowableException;

  import java.security.GeneralSecurityException;
  import java.security.NoSuchAlgorithmException;
  import java.security.InvalidKeyException;

  import javax.crypto.Mac;
  import javax.crypto.spec.SecretKeySpec;

/** Cette classe contient des méthodes statiques qui sont utilisées pour calculer le mot de passe à utilisation unique (OTP)
    utilisant JCE pour fournir le HMAC-SHA-1.    *
* @author Loren Hart@version 1.0  */

  public class OneTimePasswordAlgorithm {
     private OneTimePasswordAlgorithm() {}

     // Ceci est utilisé pour calculer les chiffres de la somme de contrôle.
     //                          0  1  2  3  4  5  6  7  8  9
     private static final int[] doubleDigits =
                  { 0, 2, 4, 6, 8, 1, 3, 5, 7, 9 };

/**Calcule la somme de contrôle en utilisant l'algorithme de la carte de crédit. Cet algorithme présente l'avantage de
    détecter tout chiffre mal tapé et toute transposition de chiffres adjacents. *
* @param num : le nombre pour calculer la somme de contrôle pour param nombre de chiffres de place significative dans
    le nombre retourne la somme de contrôle de num */

     public static int calcChecksum(long num, int digits) {
        boolean doubleDigit = true;
        int     total = 0;
        while (0 < digits--) {
           int digit = (int) (num % 10);
           num /= 10;
           si (doubleDigit) {
              digit = doubleDigits[digit];
           }
           total += digit;
           doubleDigit = !doubleDigit;
        }
        int result = total % 10;
        si (result > 0) {
           result = 10 - result;
        }
        retourne le résultat ;
     }

/** Cette méthode utilise le JCE pour fournir l'algorithme HMAC-SHA-1. HMAC calcule un code d'authentification de
    message haché et dans ce cas, SHA1 est l'algorithme de hachage utilisé. *
* @param keyBytes : les octets à utiliser pour la clé HMAC-SHA-1. *
* @param text : le message ou texte à authentifier. *
```

```
* sort NoSuchAlgorithmException si aucun fournisseur ne rend disponible d'algorithme de résumé HmacSHA1 ou HMAC-
    SHA-1.
* sort InvalidKeyException : le secret fourni n'est pas une clé HMAC-SHA-1 valide. */

     public static byte[] hmac_sha1(byte[] keyBytes, byte[] text)
        throws NoSuchAlgorithmException, InvalidKeyException
     {
  //       try {
            Mac hmacSha1;
            try {
               hmacSha1 = Mac.getInstance("HmacSHA1");
            } catch (NoSuchAlgorithmException nsae) {
               hmacSha1 = Mac.getInstance("HMAC-SHA-1");
            }
            SecretKeySpec macKey =
        new SecretKeySpec(keyBytes, "RAW");
            hmacSha1.init(macKey);
            return hmacSha1.doFinal(text);
  //       } catch (GeneralSecurityException gse) {
  //          throw new UndeclaredThrowableException(gse);
  //       }
     }

     private static final int[] DIGITS_POWER
     // 0  1    2     3       4         5         6           7           8
    = {1,10,100,1000,10000,100000,1000000,10000000,100000000};

/** Cette méthode génère une valeur d'OTP pour l'ensemble de paramètres en question.
* @param secret : le secret partagé.
* @param movingFactor : le compteur, temps, ou autre valeur qui change sur la base de l'utilisation.
* @param codeDigits : nombre de chiffres dans l'OTP, non inclus la somme de contrôle, si il en est.
* @param addChecksum : fanion qui indique si un chiffre de somme de contrôle devrait être ajouté à l'OTP.
* @param truncationOffset : décalage dans le résultat du MAC pour commencer la troncature. Si cette valeur est hors de la
    gamme de 0 à 15, la troncature dynamique sera alors utilisée.
* La troncature dynamique est quand les 4 derniers bits du dernier octet du MAC sont utilisés pour déterminer le début du
    décalage.
* sort NoSuchAlgorithmException si aucun fournisseur ne rend disponible d'algorithme de résumé HmacSHA1 ou HMAC-
    SHA-1.
* sort InvalidKeyException : le secret fournit n'est pas une clé HMAC-SHA-1 valide.
* retourne une chaîne numérique en base 10 qui inclut {@link codeDigits} chiffres plus le chiffre de somme de contrôle
    facultatif si nécessaire. */

     static public String generateOTP(byte[] secret,
              long movingFactor,
           int codeDigits,
              boolean addChecksum,
           int truncationOffset)
        throws NoSuchAlgorithmException, InvalidKeyException
     {
        // met la valeur de movingFactor en un dispositif d'octets de texte
    String result = null;
    int digits = addChecksum ? (codeDigits + 1) : codeDigits;
        byte[] text = new byte[8];
        for (int i = text.length - 1; i ≤ 0; i--) {
           text[i] = (byte) (movingFactor & 0xff);
           movingFactor >>= 8;
        }

        // calcule le hachage de hmac
        byte[] hash = hmac_sha1(secret, text);

        // met les octets choisis dans le résultat int
```

```
        int offset = hash[hash. length - 1] & 0xf;
    si ( (0<=truncationOffset) &&
          (truncationOffset<(hash.length -4)) ) {
       offset = truncationOffset;
    }
        int binary =
           ((hash[offset] & 0x7f) << 24)
           | ((hash[offset + 1] & 0xff) << 16)
           | ((hash[offset + 2] & 0xff) << 8)
           | (hash[offset + 3] & 0xff);

        int otp = binary % DIGITS_POWER[codeDigits];
    si (addChecksum) {
       otp =  (otp * 10) + calcChecksum(otp, codeDigits);
    }
    result = Integer.toString(otp);
    while (result.length () < digits) {
       result = "0" + result;
    }
    retourne le résultat ;
      }
  }
```

Appendice D. Valeurs d'essai de l'algorithme HOTP

Les données d'essai suivantes utilisent la chaîne ASCII "12345678901234567890" pour le secret :

Secret = 0x31323334353637383930313233343536373839 30

Le Tableau 1 détaille pour chaque compte, la valeur HMAC intermédiaire.

Compte	HMAC-SHA-1 hexadécimal (secret, compte)
0	cc93cf18508d94934c64b65d8ba7667fb7cde4b0
1	75a48a19d4cbe100644e8ac1397eea747a2d33ab
2	0bacb7fa082fef30782211938bc1c5e70416ff44
3	66c28227d03a2d5529262ff016a1e6ef76557ece
4	a904c900a64b35909874b33e61c5938a8e15ed1c
5	a37e783d7b7233c083d4f62926c7a25f238d0316
6	bc9cd28561042c83f219324d3c607256c03272ae
7	a4fb960c0bc06e1eabb804e5b397cdc4b45596fa
8	1b3c89f65e6c9e883012052823443f048b4332db
9	1637409809a679dc698207310c8c7fc07290d9e5

Le Tableau 2 détaille pour chaque compte les valeurs tronquées (en hexadécimal et en décimal) et ensuite la valeur HOTP.

	Tronqué		
Compte	Hexadécimal	Décimal	HOTP
0	4c93cf18	1284755224	755224
1	41397eea	1094287082	287082
2	82fef30	137359152	359152
3	66ef7655	1726969429	969429
4	61c5938a	1640338314	338314
5	33c083d4	868254676	254676
6	7256c032	1918287922	287922
7	4e5b397	82162583	162583
8	2823443f	673399871	399871
9	2679dc69	645520489	520489

Appendice E - Extensions

On introduit dans cette section plusieurs améliorations à l'algorithme HOTP. Ce ne sont pas des extensions recommandées ni des parties de l'algorithme standard, mais de simples variantes qui pourraient être utilisées pour personnaliser une mise en œuvre.

E.1 Nombre de chiffres

Une simple amélioration en termes de sécurité serait d'extraire plus de chiffres de la valeur de HMAC-SHA-1.

Par exemple, calculer la valeur HOTP modulo 10^8 pour construire une valeur HOTP de huit chiffres réduirait la probabilité de succès de l'adversaire de sv/10^6 à sv/10^8.

Ceci pourrait donner l'opportunité d'améliorer l'utilisabilité, par exemple, en augmentant T et/ou s, tout en réalisant encore une meilleure sécurité globale. Par exemple, s = 10 et 10v/10^8 = v/10^7 < v/10^6 qui est l'optimum théorique pour le code à 6 chiffres quand s = 1.

E.2 Valeurs alphanumériques

Une autre option est d'utiliser les valeurs A-Z et 0-9; ou plutôt un sous ensemble de 32 symboles tirés de l'alphabet alphanumérique afin d'éviter toute confusion entre caractères : 0, O, et Q ainsi que l, 1, et I sont très similaires, et peuvent paraître les mêmes sur un petit affichage.

La conséquence immédiate est que la sécurité est maintenant de l'ordre de sv/32^6 pour une valeur HOTP de 6 chiffres et sv/32^8 pour une valeur HOTP de huit chiffres.

32^6 > 10^9 de sorte que la sécurité d'un code HOTP de 6 alphanumérique est légèrement meilleure qu'une valeur HOTP de neuf chiffres, qui est la longueur maximum d'un code HOTP pris en charge par l'algorithme proposé.

32^8 > 10^12 de sorte que la sécurité d'un code HOTP de 8 alphanumériques est significativement meilleure qu'une valeur HOTP de neuf chiffres.

Selon l'application et le jeton/interface utilisé pour afficher et entrer la valeur HOTP, le choix de valeurs alphanumériques pourrait être une façon simple et efficace d'améliorer la sécurité pour un coût et impact réduits pour les usagers.

E.3 Séquence de valeurs HOTP

Comme on a suggéré d'entrer une courte séquence (disons 2 ou 3) de valeurs HOTP pour la resynchronisation, on pourrait généraliser le concept au protocole, et ajouter un paramètre L qui définirait la longueur de la séquence HOTP à entrer.

Par défaut, la valeur L DEVRAIT être réglée à 1, mais si la sécurité doit être augmentée, les utilisateurs pourraient demander (éventuellement pour une courte période, ou sur une opération spécifique) d'entrer des valeurs L de HOTP.

C'est une autre façon, sans augmenter la longueur HOTP ou utiliser des valeurs alphanumériques, d'augmenter la sécurité.

Note : le système PEUT aussi être programmé à demander la synchronisation de façon régulière (par exemple, chaque nuit, deux fois par semaine, etc.) et pour ce faire, de demander une séquence de valeurs L de HOTP.

E.4 Méthode de resynchronisation fondée sur le compteur

Dans ce cas, on suppose que le client peut accéder et envoyer non seulement la valeur HOTP mais aussi d'autres informations, plus spécifiquement, la valeur du compteur.

Une méthode plis efficace et sûre pour la resynchronisation est possible dans ce cas. L'application du client ne va pas seulement envoyer de valeur de client HOTP, mais la valeur du client HOTP et la valeur du compteur de client C qui s'y rapporte, la valeur HOTP agissant comme un code d'authentification de message du compteur.

Protocole de resynchronisation fondé sur le compteur (RCP, *Resynchronization Counter-based Protocol*)

Le serveur accepte si ce qui suit est vrai, où C-serveur est sa propre valeur courante de compteur :
1) C-client ≥ C-serveur
2) C-client - C-serveur ≥ s
3) Vérifier que le client HOTP est un HOTP(K,C-Client) valide
4) si c'est vrai, le serveur règle C à C-client + 1 et le client est authentifié

Dans ce cas, il n'est pas nécessaire de gérer une fenêtre de regard vers l'avant. La probabilité de succès de l'adversaire est seulement v/10^6 ou en gros v sur un million. Un autre avantage est évidemment d'être capable d'augmenter "à l'infini" et donc d'améliorer l'utilisabilité du système sans impacter la sécurité.

Ce protocole de resynchronisation DEVRAIT être utilisé chaque fois que l'impact attendu sur les applications de client et de serveur est réputé acceptable.

E.5 Champ Données

Une autre option intéressante est l'introduction d'un champ Données, qui pourrait être utilisé pour générer les valeurs de mot de passe à utilisation unique : HOTP (K, C, [Données]) où Données est un champ facultatif qui peut être l'enchaînement de diverses informations de pièces d'identité, par exemple, Données = adresse | PIN.

On pourrait aussi utiliser un temporisateur, soit comme seul facteur mouvant ou en combinaison avec le compteur -- dans ce cas, par exemple, Données = temporisateur, où temporisateur pourrait être l'heure UNIX (secondes GMT depuis le 1/1/1970) divisé par un facteur (8, 16, 32, etc.) afin de donner une étape spécifique de temps. La fenêtre de temps pour le mot de passe à utilisation unique est alors égale à l'étape de temps multipliée par le paramètre de resynchronisation comme défini ci-dessus. Par exemple, si on prend 64 secondes comme étape de temps et 7 pour le paramètre de resynchronisation, on obtient une fenêtre d'acceptation de +/- 3 minutes.

Utiliser un champ Données offre plus de souplesse dans la mise en œuvre de l'algorithme, pourvu que le champ Données soit clairement spécifié.

Adresse des auteurs

David M'Raihi
VeriSign, Inc.
685 E. Middlefield Road
Mountain View, CA 94043
USA
téléphone : 1-650-426-3832
mél : dmraihi@verisign.com

Mihir Bellare
Dept 0114
University of California at San Diego
9500 Gilman Drive
La Jolla, CA 92093
USA
mél : mihir@cs.ucsd.edu

Frank Hoornaert
VASCO Data Security, Inc.
Koningin Astridlaan 164
1780 Wemmel,
Belgium
mél : frh@vasco.com

David Naccache
Gemplus Innovation
34 rue Guynemer, 92447,
Issy les Moulineaux, FranceTel Aviv, Israel 61110
mél : david.naccache@gemplus.com

Ohad Ranen
Aladdin Knowledge Systems Ltd.
15 Beit Oved Street
Tel Aviv, Israel 61110
mél : Ohad.Ranen@ealaddin.com

Déclaration de droits de reproduction

Propriété intellectuelle

L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourrait être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr .

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf- pr@ietf.org.

Remerciement

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