Protocoles d'authentification

Authentification mutuelle

- Utilisé pour convaincre les parties de l'identité l'une de l'autre et pour échanger des clés de session
- Protocole mutuel
- Faire attention
  - À la confidentialité – protection des clés de session
  - Au paramètre de temps – empêcher le rejeu

Approche symétrique

- comme discuté précédemment, on peut employer une hiérarchie à deux niveaux des clefs
  - Transfert confidentiel de données
  - Transfert confidentiel de clé
- habituellement avec un centre de distribution de clé (KDC)
  - Chaque partie partage une clé principale avec le KDC
  - Le KDC produit des clefs de session utilisées pour les communications entre les parties
  - Les clés principales sont utilisées pour distribuer ces clés de session aux parties

Approche symétrique

- Needham-Schroeder
  - protocole de distribution de clé avec un tiers médiateur
  - pour une session entre A et B, négocié par le KDC
    1. $A \rightarrow KDC : ID_A \,||\, ID_B \,||\, N_1$
    2. $KDC \rightarrow A : E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks \,||\, ID_A]]$
    3. $A \rightarrow B : E_{Kb}[Ks||ID_A]$
    4. $B \rightarrow A : E_{Ks}[N_2]$
    5. $A \rightarrow B : E_{Ks}[f(N_2)]$

Approche symétrique

- utilisé pour distribuer sûrement une nouvelle clé de session entre A et B
- mais est vulnérable à une attaque par rejeu si une vieille clé de session a été compromise
  - Rejeu du message 3 qui convainc B que A communique avec lui
- modifications pour contrer ce risque :
  - Timestamp (Denning 81)
  - Employer un nonce supplémentaire (Neuman 93)

Denning :

1. $A \rightarrow KDC$: $ID_A||ID_B$
2. $KDC \rightarrow A$: $E_{Ka}[Ks||ID_B||T||E_{Kb}[Ks||ID_A||T]]$
3. $A \rightarrow B$: $E_{Kb}[Ks||ID_A||T]$
4. $B \rightarrow A$: $E_{Ks}[N_1]$
5. $A \rightarrow B$: $E_{Ks}[f(N_1)]$

Attention à la synchronisation des horloges….

1. $A \rightarrow B$: $ID_A||N_a$
2. $B \rightarrow KDC$: $ID_B||N_b||E_{Kb}[ID_A||N_a||T_b]$
3. $KDC \rightarrow A$: $E_{Ka}[ID_B||N_a||Ks||T_b]||E_{Kb}[ID_A||Ks||T_b]||N_b$
4. $A \rightarrow B$: $E_{Kb}[ID_A||Ks||T_b]||E_{Ks}[N_b]$

Approche par clé publique

- On dispose de différentes alternatives basées sur l'utilisation du chiffrement à clé publique
- Nécessité de s'assurer que les clefs publiques dont on dispose correspondent aux bonnes personnes
- Utilisation d'un serveur central d'authentification (AS)
  - (plutôt que KDC puisque plus responsable de la distribution des clés)
- Divers protocoles existent utilisant des horodateurs ou des nonces

Approche par clé publique

- Protocole de Denning utilisant un AS :
  - $A \rightarrow AS : ID_A \,||\, ID_B$
  - $AS \rightarrow A : E_{KRas}[ID_A||KUa||T] \,||\, E_{KRas}[ID_B||KUb||T]$
  - $A \rightarrow B : E_{KRas}[ID_A||KUa||T] \,||\, E_{KRas}[ID_B||KUb||T] \,||\, E_{KUb}[E_{KRas}[Ks||T]]$
- Remarques :
  - la clef de session est choisie par A, par conséquent, il n'est pas nécessaire de faire confiance à l'AS pour la protéger
  - les horodateurs empêchent le rejeu mais exigent des horloges synchronisées

Authentification par passage unique

- requis quand l'expéditeur et le récepteur ne sont pas en communication en même temps (p.e. email)
- L'en-tête est en clair, ainsi il peut être compris par le système d'email
- On souhaite que le contenu du corps du texte soit protégé et que l'expéditeur soit authentifié

Solution pour la désynchronisation [Woo & Lam 92]: (on fonctionne avec des certificats)

- $A \rightarrow KDC :$ $ID_1||ID_B$
- $KDC \rightarrow A :$ $E_{KRauth}[ID_B||KUb]$
- $A \rightarrow B :$ $E_{KUb}[N_a||ID_A]$
- $B \rightarrow KDC :$ $ID_B||ID_A \,||\, E_{KUauth}[N_a]$
- $KDC \rightarrow B :$ $E_{KRauth}[ID_A||KUa] \,||\, E_{KUb}[E_{KRauth}[Na \,||\, Ks \,||\, ID_B]]$
- $B \rightarrow A :$ $E_{KUa}[E_{KRauth}[Na||Ks||ID_B]||Nb]$
- $A \rightarrow B :$ $E_{Ks}[Nb]$

Renforcement [WOO & LAM 92]

1. $A \rightarrow KDC :$ $ID_A||ID_B$
2. $KDC \rightarrow A :$ $E_{KRauth}[ID_B||KUb]$
3. $A \rightarrow B :$ $E_{KUb}[Na||ID_A]$
4. $B \rightarrow KDC :$ $ID_B||ID_A||E_{KUauth}[Na]$
5. $KDC \rightarrow B :$ $E_{KRauth}[ID_A||KUa] \,||\, E_{KUb}[E_{KRauth}[Na||Ks||ID_B]]$
6. $B \rightarrow A :$ $E_{KUa}[E_{KRauth}[Na||Ks||ID_A||ID_B]||Nb]$
7. $A \rightarrow B :$ $E_{Ks}[Nb]$

Approche symétrique

- Raffinement de l'utilisation de KDC :
  1. A $\rightarrow$ KDC : $ID_A \,||\, ID_B \,||\, N_1$
  2. KDC $\rightarrow$ A : $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]]$
  3. A $\rightarrow$ B : $E_{Kb}[Ks||ID_A] \,||\, E_{Ks}[M]$
- Garanti un certain niveau d'authentification
- Ne protège pas du rejeu
  - On pourrait utiliser des timestamps mais les délais des mails rendent ce système problématique

Approche par clé publique

- Plusieurs alternatives ont déjà été présentées
- Si la confidentialité est le souci principal :
  - A $\rightarrow$B: $E_{KUb}[Ks] \,||\, E_{Ks}[M]$
    - La clé de session est chiffrée avec la clé publique de B, le message est chiffré avec cette clé de session
- Si l'authentification est importante : utiliser une signature numérique avec un certificat numérique :
  - A$\rightarrow$B: $M \,||\, E_{KRa}[H(M)] \,||\, E_{KRas}[T||ID_A||KUa]$
    - message, signature, certificat sont fournis

Kerberos

Kerberos

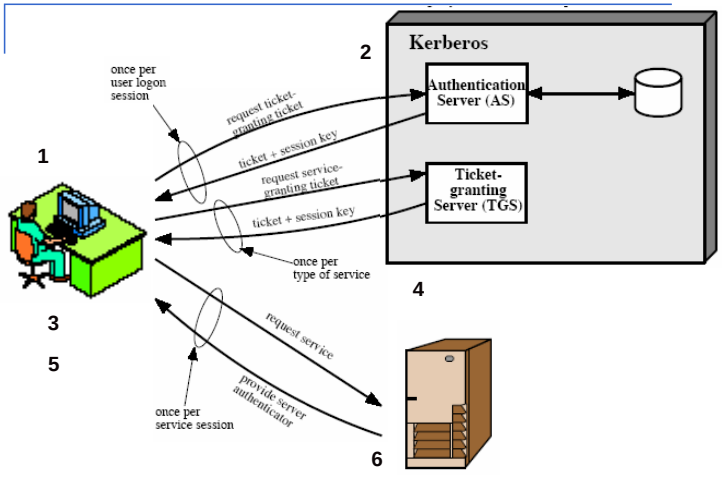
- système de serveur de distribution de clé proposé par le MIT
- fournit un authentification centralisée clé privée dans un réseau distribué
  - permet l'accès pour les utilisateurs aux services distribués proposé par le réseau
  - sans devoir faire confiance à tous les postes de travail
  - mais plutôt en faisant confiance à un serveur central d'authentification
- deux versions en service : 4 et 5

Conditions à remplir

- Conditions à remplir :
  - sécurité
  - fiabilité
  - transparence
  - « scalabilité »
- utilise un protocole d'authentification basé sur Needham-Schroeder

Kerberos 4 – vue générale

- Schéma d'authentification basique utilisant un tiers de confiance
- Le KDC
  - Dispose d'un serveur d'authentification (AS)
    - Les utilisateurs négocient avec l'AS pour s'identifier
    - AS fournit un ticket d'authentification non corruptible (Ticket Granting Ticket - TGT)
  - Dispose d'un serveur de ticket (Ticket Granting Server – TGS)
    - Les utilisateurs demandent un accès à des services proposés par le TGS sur base de leur TGT



1. L'utilisateur se logge sur poste de travail et demande un service à l'AS.
2. L'AS vérifie les droits d'accès de l'utilisateur dans la base de données, crée un ticket (TGT) et une clé de session. Ces résultats sont chiffrés en utilisant une clé dérivée du mot de passe de l'utilisateur.
3. Le poste de travail demande le mot de passe de l'utilisateur et l'utilise pour déchiffrer le message venant du serveur, puis il envoie le TGT et un authentificateur qui contient le nom de l'utilisateur, l'adresse réseau du poste de travail et un horodateur au TGS.
4. Le TGS déchiffre le ticket et l'authentificateur, vérifie la demande, puis crée le ticket pour le serveur (service) demandé.
5. Le poste de travail envoie ce ticket et l'authentificateur au serveur concerné.
6. Le serveur vérifie que ce ticket et l'authentificateur correspondent et accorde alors l'accès au service. Si l'authentification mutuelle est exigée, le serveur renvoie un authentificateur.

Messages échangés

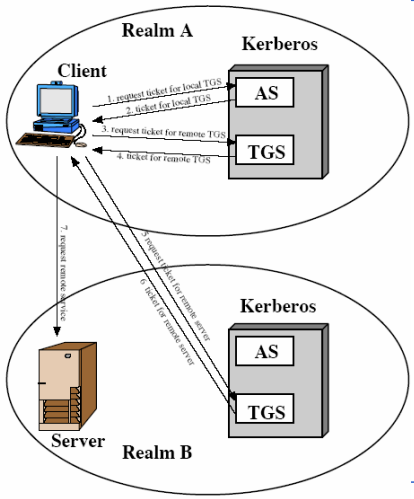
(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) C → AS: $ID_c \| ID_{tgs} \| TS_1$
(2) AS → C: $E_{K_c}[K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}]$
$Ticket_{tgs} = E_{K_{tgs}}[K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket
(3) C → TGS: $ID_v \| Ticket_{tgs} \| Authenticator_c$
(4) TGS → C: $E_{K_{c,tgs}}[K_{c,v} \| ID_v \| TS_4 \| Ticket_v]$
$Ticket_{tgs} = E_{K_{tgs}}[K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2]$
$Ticket_v = E_{K_v}[K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4]$
$Authenticator_c = E_{K_{tgs}}[ID_C \| AD_C \| TS_3]$
(c) Client/Server Authentication Exchange: to obtain service
(5) C → V: $Ticket_v \| Authenticator_c$
(6) V → C: $E_{K_{c,v}}[TS_5 + 1]$ (for mutual authentication)
$Ticket_v = E_{K_v}[K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4]$
$Authenticator_c = E_{K_{c,v}}[ID_C \| AD_C \| TS_5]$

K_C : clé calculée à partir du mot de passe de C

$K_{c,tgs}$: est chiffrée par K_C (chiffrement DES) et donc seul C peut la lire

Domaines Kerberos

- un environnement Kerberos se compose de :
  - un serveur de Kerberos
  - un certain nombre de clients, tout inscrits au serveur
  - serveurs d'application, partageant des clés avec le serveur
- est appelé domaine Kerberos (realm)
  - Typiquement un domaine d'administration simple
- si on a des domaines Kerberos multiples, les différents serveurs Kerberos doivent partager des clés et se faire confiance



1. $C \rightarrow AS$: $ID_c \,||\, ID_{tgs} \,||\, TS_1$
2. $AS \rightarrow C$: $E_{Kc}[K_{c,tgs} \,||\, ID_{tgs} \,||\, TS_2 \,||\, Lifetime_2 \,||\, Ticket_{tgs}]$
3. $C \rightarrow TGS$: $ID_{tgsrem} \,||\, Ticket_{tgs} \,||\, Authenticator_c$
4. $TGS \rightarrow C$: $E_{Kc,tgs}[K_{c,tgsrem} \,||\, ID_{tgsrem} \,||\, TS_4 \,||\, Ticket_{tgsrem}]$
5. $C \rightarrow TGS_{rem}$: $ID_{vrem} \,||\, Ticket_{tgsrem} \,||Authenticator_c$
6. $TGS_{rem} \rightarrow C$: $E_{Kc,tgsrem}[K_{c,vrem} \,||\, ID_{vrem} \,||\, TS_6 \,||\, Ticket_{vrem}]$
7. $C \rightarrow V_{rem}$: $Ticket_{vrem} \,||\, Authenticator_c$

Kerberos Version 5

- développé dans le milieu des années 90
- fournit des améliorations de v4
  - corrige des imperfections environnementales
    - alg de chiffrage, protocole réseau, ordre de byte, temps de vie du ticket, transfert d'authentification, authentification inter-domaines
  - et des insuffisances techniques
    - double chiffrement , mode non-STD, clefs de session, attaques de mot de passe
- Repris dans la RFC 1510

(a) Authentication Service Exchange: to obtain ticket-granting ticket

(1) C → AS: $Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$

(2) AS → C: $Realm_c \| ID_C \| Ticket_{tgs} \| E_{K_c}[K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}]$

$Ticket_{tgs} = E_{K_{tgs}}[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times]$

(b) Ticket-Granting Service Exchange: to obtain service-granting ticket

(3) C → TGS: $Options \| ID_v \| Times \| \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C: $Realm_c \| ID_C \| Ticket_v \| E_{K_{c,tgs}}[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_V]$

$Ticket_{tgs} = E_{K_{tgs}}[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times]$

$Ticket_v = E_{K_v}[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times]$

$Authenticator_c = E_{K_{c,tgs}}[ID_C \| Realm_c \| TS_1]$

(c) Client/Server Authentication Exchange: to obtain service

(5) C → V: $Options \| Ticket_v \| Authenticator_c$

(6) V → C: $E_{K_{C,V}}[TS_2 \| Subkey \| Seq\#]$

$Ticket_v = E_{K_v}[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times]$

$Authenticator_c = E_{K_{c,V}}[ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#]$

(1)

•Realm : indique le domaine de l'utilisateur

•Options : utilisé pour demander que certains 'flags' soient positionnés dans le ticket obtenu en retour

•Times : permet de préciser les balises de durée de vie du ticket : from, till, rtime

•Nonce : valeur aléatoire assurant de la fraîcheur de l'information obtenue

(5)

•Subkey : proposition du client pour l'utilisation d'une clé de chiffrement à utiliser les informations qui transiteront dans cette session. Si le paramètre est omis, c'est $K_{c,v}$ qui sera utilisée

•Séquence number : précise le numéro de séquence à partir duquel seront numérotés les messages échangés pendant la session (évite les rejeux)

Sécurité WEB - Réseaux

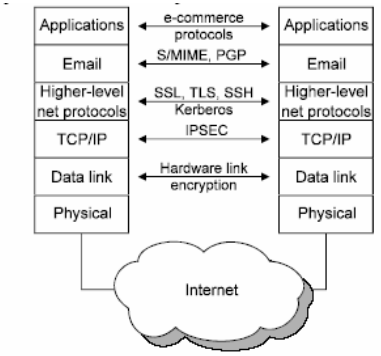
Comparaisons des menaces sur le Web

	Menaces	Conséquences	Contre-mesures
Intégrité	▪ modification des données utilisateur ▪Cheval de Troie ▪Modification de la mémoire	▪Perte d'information ▪Machine compromise ▪Vulnérabilités aux autres menaces	Checksum cryptographiques
Confidentialité	▪Écoute sur le net ▪Vol d'informations (client, serveur, configuration du réseau, …)	▪ perte d'information ▪Perte d'intimité	Chiffrement, proxies internet

Comparaisons des menaces sur le Web

	Menaces	Conséquences	Contre-mesures
Déni de service	▪ destruction des threads utilisateurs ▪Flooding de machines, ▪Saturation de disques	▪Interruption ▪Empêche le travail effectif ▪Ralentissement	Difficile à empêcher
Authentification	▪Mascarade ▪Fabrication/ modification d'informations	▪ mauvaise présentation de l'utilisateur ▪Utilisation d'informations fausses	Techniques cryptographiques

Localisation de la sécurité



Applications d'authentification

SSL

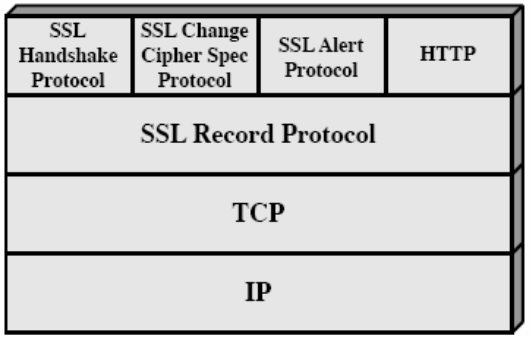
SSL (Secure Socket Layer)

- Service de sécurité de la couche transport
- Développé à l'origine par Netscape
- Protocole : SSL
- Norme : TLS (Transport Layer Security)
- Utilisations de TCP pour fournir un service bout à bout fiable

Objectifs

- Authentifier un serveur SSL et un client SSL (optionnel)
- Echanger une clé de session
- Etablir une session confidentielle et intègre au sein de laquelle plusieurs connexions pourront avoir lieu.

SSL Architecture



Ce protocole est composé de 2 sous-couches :

Une couche qui définit la façon dont seront traitées les données à chiffrer

Une couche qui gère l'établissement de la connexion sécurisée et la négociation des paramètres de la session

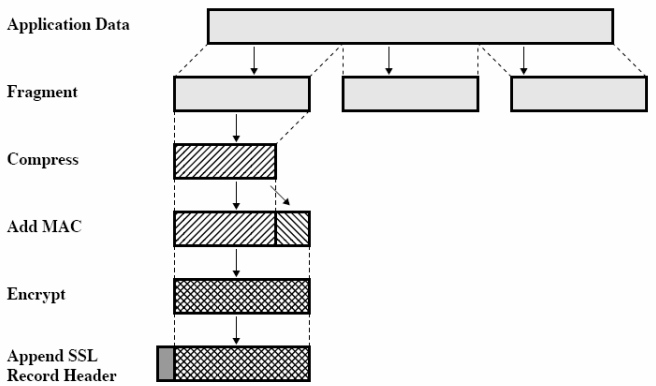
SSL - Architecture

- Session SSL
  - association entre le client et le serveur
  - créé par le « handshake protocol »
  - définit un ensemble de paramètres cryptographiques
  - peut être partagé par des connexions SSL multiples
- Connexion SSL
  - liaison peer to peer
  - liée à 1 session de SSL

SSL Record Protocol

- Fournit 2 services :
  - confidentialité
    - Via un chiffrement symétrique avec une clef secrète partagée définie dans le « Handshake Protocol »
    - IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128
    - Compression du message avant chiffrement
  - intégrité de message
    - utilisation d'un MAC avec la clef secrète partagée
    - semblable à HMAC mais avec un padding différent

SSL Record Protocol



Fragmentation : chaque message est fragmenté en blocs de 2^{14} bytes ou moins.

Compression : optionnel : doit être sans pertes et ne doit pas augmenter la taille du contenu de plus de 1024 bytes

MAC : on utilise une clé secrète partagée des 2 parties (voir page suivante)

Chiffrement : symétrique : ne doit pas augmenter la longueur du contenu de plus de 1024 bytes et donc la longueur totale ne doit pas excéder 2^{14} + 2048 bytes. Les algorithmes permis sont : DES, 3DES, IDEA, …

Padding : après le mac

Le header contient :

Content type (8 bits) : le protocole utilisé pour manipuler le fragment : change_cipher_spec, alert, handshake, application_data

Major version (8 bits) : version ssl utilisée : souvent 3

Minor version (8 bits) : si sslv3 : valeur = 0

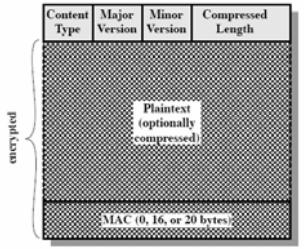
Compressed length (16 bits) : longueur du fragment de texte clair

SSL Record Protocol

- Calcul du MAC :

Hash (MAC_write_secret || pad_2 || hash (MAC_write_secret || pad_1 || seq_num || SSLCompressed.type || SSLCompressed.length || SSLCompressed.fragment))

- Format du SSL Record :



Calcul du MAC :

•MAC_write_secret : la clé secrete partagée

•Hash : algorithme MD5 ou SHA-1

•Pad_1 : byte 0x36 (0011 0110) répété 48 fois (384 bits) pour le MD5 et 40 fois pour le SHA-1

•Pad_2 : byte 0x5C (0101 1100) répété 48 fois pour le MD5 et 40 fois pour le SHA-1

•Seq_num : numéro de séquence pour ce message

•SSLCompressed.type : le protocole de haut niveau utilisé pour manipuler ce fragment

•SSLCompressed.length : longueur du fragment compressé

•SSLCompressed.fragment : le fragment compressé

Algorithme similaire à HMAC excepté que les paddings sont concaténés ici et pas manipulés par le biais d'un XOR

SSL Change Cipher Spec Protocol

- Lancé au cours du handshake
- But : synchroniser les stratégies de chiffrement utilisées par le client et le serveur.
- Etapes :
  - Au cours du handshake, les paramètres de chiffrement sont négociés et calculés
  - Ces nouveaux paramètres constituent une stratégie 'temporaire'
  - A l'envoi du message 'Change_cipher_specs', les nouveaux paramètres sont copiés dans la stratégie 'courante' pour être directement utilisés par la couche SSL Record

SSL Alert Protocol

- transmet des alertes connexes à SSL
- sévérité
  - avertissement ou fatal e
- alerte spécifique
  - message inattendu, mauvais MAC, échec de décompression, échec de handshake, paramètre illégal
  - la fin annoncent, aucun certificat, mauvais certificat, certificat non supporté, certificat révoqué, certificat a expiré, certificat inconnu
- comprimé et chiffré comme toutes les données de SSL

SSL Handshake Protocol

- permet au serveur et au client de :
  - s'authentifier
  - négocier les algorithmes de chiffrement et de MAC
  - négocier les clés cryptographiques à employer
- comporte 4 phases d'échanges de messages
  - Etablissement des possibilités en matière de sécurité. Cette phase détermine entre autres :
    - La version, la session, les algorithmes de chiffrement et compression, des nombres aléatoires
  - Authentification du serveur et échange de clé
  - Authentification du client et échange de clé
  - Clôture

SSL Handshake Protocol

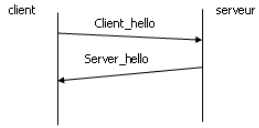
Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Phase 1 : « hello »

- Lors du démarrage de ce protocole, les paramètres de chiffrement sont initialisés à 0 (c-à-d pas d'algorithme de chiffrement, de hash, de signature définis)
  - Client_hello contient la combinaison des algorithmes cryptographiques supportés par le client, par ordre décroissant de préférence.
  - Server_hello contient la méthode d'échange de clé et les paramètres de chiffrement sélectionnés parmi ceux proposés par le client.

Phase 1 : « hello »

- Client_hello est composé de :
  - N° de version SSL (2 ou 3)
  - N° aléatoire du client
  - N° de session (optionnel) (permet de récupérer les paramètres de sécurité d'une connexion antérieure)
  - Liste des stratégies de chiffrements supportées, par ordre décroissant de « préférence » (DES, 3DES, IDEA, …)
  - Algorithmes de compression supportés. (MD5, SHA-1, …)



Phase 1 : « hello »

- Server_hello comporte :
  - N° de version SSL (en fonction de celui supporté par le client)
  - N° aléatoire du serveur
  - N° de session (soit ancien, soit nouveau)
  - Stratégie de chiffrement choisie (en fonction de celles du client)
  - Algorithme de compression choisi (en fonction de ceux du client)
- Remarque : si le serveur accepte de réutiliser une session antérieure, il renvoie le même session ID au client et le protocole de handshake se termine automatiquement.

Méthodes d'échange de clé

- basées sur une version de l'algorithme Diffie-Hellman et/ou du RSA
  - DH anonyme : il s'agit de l'algorithme de base.
    - Il est sujet à l'attaque man-in-the-middle car les paramètres DH ne sont pas authentifiés
  - DH fixé : les paramètres de DH sont fixés une fois pour toutes et publiés au moyen d'un certificat.
    - contre l'attaque man-in-the-middle mais pas une attaque par force brute

Méthodes d'échange de clé

- DH éphémère :
  - les paramètres DH peuvent changer d'une session à l'autre.
    - rend une attaque par force brute bien plus difficile à réaliser.
  - Les paramètres DH sont en plus signés grâce à la clé RSA privée de l'expéditeur pour contrer l'attaque man-in-the-middle.
    - implique que l'expéditeur dispose d'une paire de clé RSA
- RSA :
  - La clé secrète est chiffrée grâce à la clé publique du receveur.
    - Cela implique qu'un certificat du receveur soit disponible afin d'obtenir sa clé publique de façon fiable.
    - Cette méthode assure la confidentialité des clés mais n'assure pas l'authentification de l'expéditeur mais ce n'est pas le but

Phase 2 : authent. serveur et échange de clé

- Remarque : les échanges concernant les certificats sont directement liés aux paramètres cryptographiques de l'échange de clé.
- Certificate envoie le certificat au client.
  - Il n'est pas envoyé s'il s'agit d'un DH anonyme.
  - Il contient aussi les paramètres de la clé publique DH s'il s'agit d'un DH fixé.

Pour l'envoi des paramètres, utilisation d'une signature :

Sign = hash(ClientHello.random || ServerHello.random || ServerParams)

Phase 2 : authent. serveur et échange de clé

- Server_key_exchange envoie les paramètres relatifs à la clé de session.
  - N'est pas envoyé si DH fixé (car déjà fournis dans le certificat) ou RSA sont utilisés
- Certificate_request demande le certificat du client.
  - Ne sera pas envoyé si un DH anonyme est utilisé.
- Server_hello_done marque la fin de la phase de hello pour le serveur

Phase 3 : authent. client et échange de clé

- Certificate : le client vérifie d'abord le certificat du serveur s'il l'a reçu ensuite envoie son certificat au serveur si celui-ci l'a demandé
- Client_key_exchange est similaire à Server_key_exchange. En fonction du mécanisme d'échange de clé utilisé, le client génère une clé maîtresse primaire (PMK) (Nbre aléatoire de 48 bytes) et l'envoie au serveur.
  - ! La méthode d'échange choisie

PMK (pre master key) : 48 bytes : cette pmk est utilisée pour calculer la clé de session (master key)

Phase 3 : authent. client et échange de clé

- Client_certificate_verify est envoyé après la génération de la clé maîtresse de session (MK).
  - Il ne sera pas envoyé si un DH fixé est utilisé.
- Ce message est utilisé par le client pour prouver qu'il dispose de la clé privée associé au certificat.

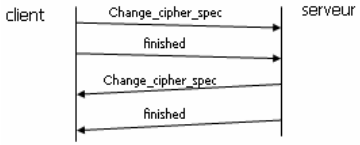
Génération des clés

- Valeur de 48 bytes générée pour la session
- 2 étapes :
  - Echange d'une clé pre master (PMK)
  - Calcul de la master key (MK)
- Echange de la PMK
  - RSA : chiffrement d'une clé avec la clé publique du serveur
  - DH : échange de clé publique DH

Génération des clés – calcul de MK

- X : ClientHello.random
- Y : ServerHello.random
- MK =
  - MD5(PMK || SHA (‘A’ || PMK || X || Y))
    || MD5(PMK || SHA (‘BB’ || PMK || X || Y))
    || MD5(PMK || SHA (‘CCC’ || PMK || X || Y))

Phase 4 : « finish »



- Change_cipher_spec fait partie du protocole Cipher Spec. Il a pour but de transformer la stratégie temporaire en stratégie courante.
- Le message Finished vérifie que l’échange de clé et les processus d’authentification se sont correctement déroulés.

Phase 4 : « finish »

- Il s'agit également des premiers messages utilisant la stratégie de chiffrement négociée au cours du « handshake ».
- Après le message Finished, le client et le serveur s'échangent des messages venant des couches applicatives sur base des paramètres contenus dans l'état de la session.

TLS (Transport Layer Security)

- standard IETF (RFC 2246) semblable à SSLv3
- avec des différences mineures
  - dans le nombre de version de format d'enregistrement
  - utilisations HMAC pour le calcul de MAC
  - une fonction pseudo aléatoire intervient dans le calcul de clé
  - codes alertes additionnels
  - changements des chiffrements supportés
  - changements des négociations de certificat
  - changements dans l'utilisation du padding

Applications d'authentification

SET

Secure Electronic Transactions (SET)

- ouvrir les spécifications de chiffrage et de sécurité
- pour protéger des transactions liées aux cartes de crédit sur Internet
- développé en 1996 par Mastercard, visa etc..
- pas un système de paiement
- plutôt un ensemble de protocoles et de formats de sécurité
  - Communications sécurisée entre les parties
  - Utilisation des certificats X.509v3
  - intimité en restreignant l'information à ceux qui en ont besoin

Spécification détaillée dans 3 livres (voir www.setco.org)

SET – vue générale (contraintes)

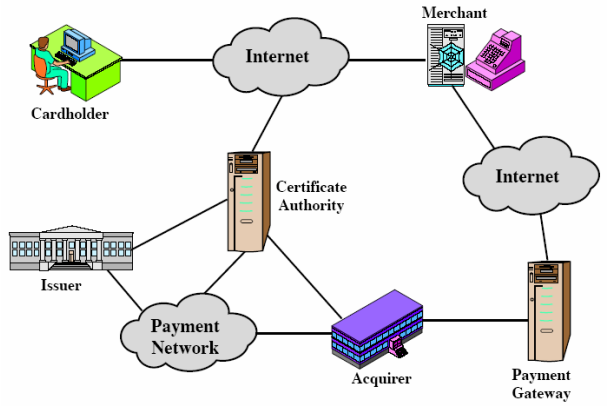
- Fournir la confidentialité des paiements et des informations liées
- Assurer l'intégrité des données transférées
- Fournir l'authentification permettant de vérifier la légitimité de l'utilisateur d'une carte
- Fournir l'authentification du marchand
- Protection optimale de toutes les parties de la transaction
- Indépendant des protocoles, plate-formes, OSs, …

SET – vue générale (points importants)

- Confidentialité de l'information
  - Chiffrement DES, seule la banque peut lire le n° de carte
- Intégrité des données
  - Signatures digitales RSA, SHA 1+ HMAC
- Authentification du propriétaire de la carte
  - Certificat X.509v3 + signatures RSA
- Authentification du marchand
  - Certificat X.509v3 + signatures RSA

⇒ choix des algorithmes imposé

SET – composants



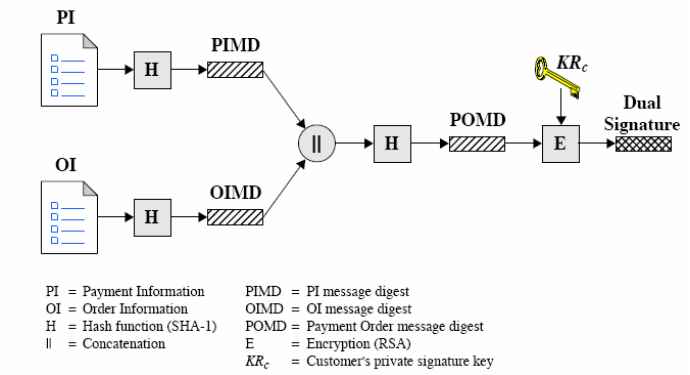
SET Transaction

1. le client ouvre le compte
2. le client reçoit un certificat
3. les négociants ont leurs propres certificats
4. le client passe une commande
5. le négociant est vérifié
6. l'ordre et le paiement sont envoyés
7. le négociant demande l'autorisation de paiement
8. le négociant confirme l'ordre
9. le négociant fournit des marchandises ou le service
10. le négociant demande le paiement

Signature duale

- le client crée les messages duaux
  - l'information de commande (OI) pour le négociant
  - l'information de paiement (PI) pour la banque
- ni l'une ni l'autre partie n'a besoin des détails de l'autre
- mais doit savoir qu'ils sont liés
- employer une signature duale pour ceci
  - Signature des condensés enchaînés d'OI et de PI

Fonctionnement d'une signature duale



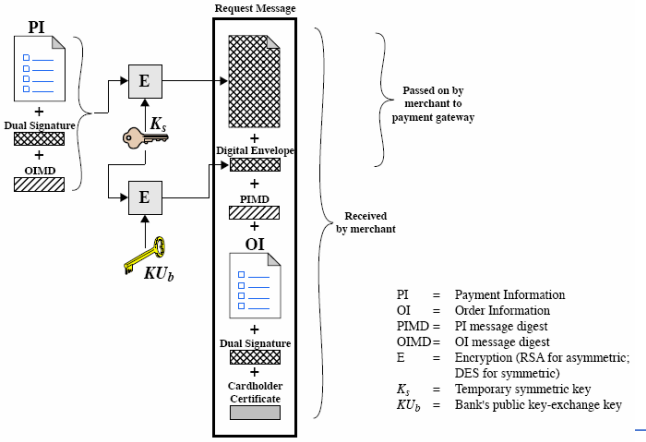
PI = Payment Information
OI = Order Information
H = Hash function (SHA-1)
|| = Concatenation

PIMD = PI message digest
OIMD = OI message digest
POMD = Payment Order message digest
E = Encryption (RSA)
KR_c = Customer's private signature key

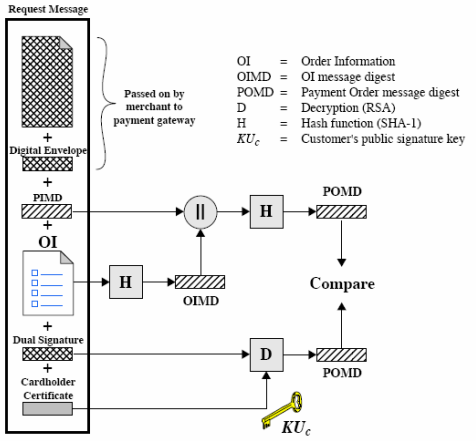
Fonctionnement du paiement

- Nombreuses transactions possibles (voir tables)
- Principales :
  - Commande
  - Autorisation de paiement
  - Capture du paiement

Commande - client



Commande – marchand (vérification)



Commande - Autorisation – marchand

1. Vérifie des certificats de détenteur de carte en utilisant des signatures de CA
2. Vérifie la signature duale en utilisant la clef publique de signature du client pour s'assurer que la commande n'a pas été manipulée pendant son transit et qu'elle a été signée en utilisant la clef privée de signature du détenteur de la carte
3. Fait suivre à l'information de paiement au gateway de paiement pour l'autorisation (décrite plus tard) et effectue la commande
4. Envoie une réponse d'achat au détenteur de carte

Autorisation du gateway de paiement

1. Vérifie tous les certificats
2. Déchiffre l'enveloppe numérique du bloc d'autorisation pour obtenir la clef symétrique et déchiffre alors le bloc d'autorisation
3. Vérifie la signature du négociant sur le bloc d'autorisation
4. Déchiffre l'enveloppe numérique du bloc de paiement pour obtenir la clef symétrique et déchiffre alors le bloc de paiement
5. Vérifie la signature duale sur le bloc de paiement
6. Vérifie que l'identificateur de transaction reçu du marchand correspond à celui reçu (indirectement) dans le PI du client
7. Demande et reçoit une autorisation de l'issuer
8. Renvoie la réponse d'autorisation au négociant

Paiement

1. le négociant envoie à gateway de paiement une demande de paiement
2. Le gateway vérifie la demande
3. Effectue le transfert de fond vers le compte du négociant
4. informe le négociant du transfert

Questions

- Protocoles d'authentification
  - Approche symétrique, à clés publiques
- Kerberos
- SSL
- SET

Références

- [Stallings] – ch 13,14,17
- www.setco.org
- http://web.mit.edu/kerberos/www/
- http://rambit.qc.ca/plamondon/crypto.htm
- http://www.cryptography.com/resources/researchlinks.html#ssl
- http://www.openssl.org/