La sécurisation des protocoles de Transport

Bernard Cousin

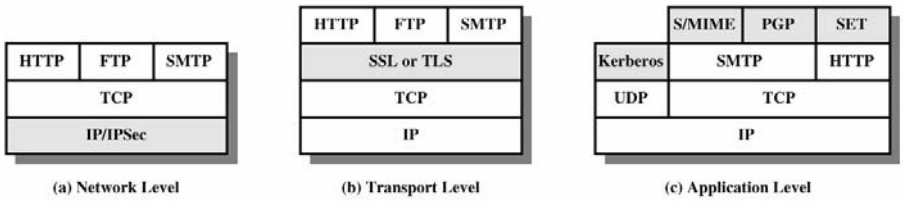
UNIVERSITE DE RENNES 1

Plan

- Introduction
- Architecture et Services de SSL/TLS
- Les protocoles de SSL
- Le protocole "Handshake"
- L'environnement SSL/TLS

Sécurité et pile protocolaire

- Au niveau réseau : IPsec
  - transparent aux couches supérieures
  - filtrage du trafic au niveau du réseau
- Au niveau supérieur :
  - seulement dans les entités d'extrémité
  - au niveau Transport+ : SSL ou TLS (driver intégré au système)
  - au niveau application : très spécifique



Historique de SSL et TLS

- SSL proposé par Netscape
  - v1, v2 (9 fev. 1995), v3.0 (18 nov. 1996)
- TLS normalisé par l'IETF,
  - rfc 2246 (janv. 1999)
  - extensions : rfc 3546 (juin 2003)
- TLS est équivalent à SSL v3.1

Services de sécurité

- TLS/SSL offrent les services suivants :
  - Intégrité
  - Authentification (utilisation de certificats)
  - Confidentialité
- Optimisation :
  - Compression des messages
  - Mise en commun des paramètres de sécurité
  - Modification possible des paramètres de sécurité
  - Basé sur un secret partagé
  - Puis la génération de clefs, etc.
- Adaptation :
  - Négociation, paramétrisation

L'architecture de SSL

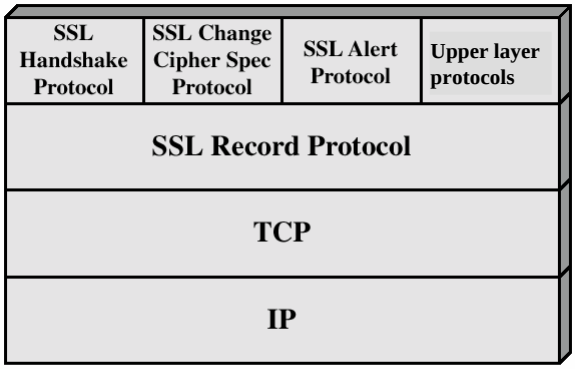


Figure 7.2 SSL Protocol Stack

Connexion et session SSL

- Connexion :
  - Offre certains services de sécurité entre deux entités.
  - Éphémère mais établie dans le cadre d'une Session SSL
- Session :
  - Association entre un client et un serveur
  - Etabli lors du protocole de "Handshake"
  - Un ensemble de paramètres de sécurité, dans chaque sens, est négocié une seule fois et partagé par les connexions
  - L'état courant défini les paramètres courants, l'état en attente ("Session pending state") défini les paramètres suivants (cf. "Change Cipher Spec Protocol")

  Nota : il peut y a avoir plusieurs sessions établies entre 2 mêmes partenaires (rarement).

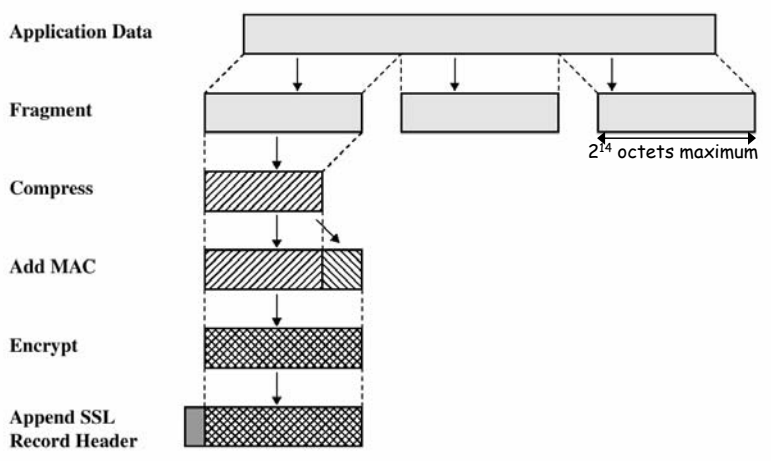
Paramètres de la Session

- "Session identifier"
- "Peer certificate" :
  - certificat X509 ou rien
- "Compression method"
- "Cipher specification" :
  - algo. de chiffrement et algo. d'authentification et leurs paramètres
- "Master secret" :
  - 48 octets partagés entre le client et le serveur
- "Is resumable":
  - Les paramètres de la session peuvent être utilisés pour établir une connexion

Paramètres de la Connexion

- "Server and client random" :
  - Choisie à chaque nouvelle connexion.
- "Server write MAC secret" et "Client write MAC secret "
  - Un secret utilisé pour vérifier l'intégrité des messages (un pour chaque sens).
- "Server write key" et "Client write key" :
  - Pour le chiffrement, dans le sens serveur-client et vice-versa.
- "Initialization vectors" :
  - Utilisés en mode CBC. Le premier IV est issu du protocole "Handshake", puis le dernier bloc chiffré sert d'IV pour la prochaine transmission chiffrée, au sein de la même connexion.
- "Sequence numbers" :
  - Un numéro de séquence est mémorisé, un pour chaque sens de transmission.

Opérations du "SSL Record protocol"



Fragmentation et compression

- Fragmentation :
  - Des blocs de 2^{14} octets, au plus
- Compression :
  - Doit être sans perte
    - Lempel-Ziv-Stac (rfc 3943)
    - Par défaut, la fonction de compression est l'identité

Intégrité des messages

- Le contrôle d'intégrité est optionnel
- "Message Authentication Code" :

  hash(MAC_write_secret || pad2 || hash(MAC_write_secret || pad1 || seq_num || Compression.type || Compression.length || Compressed. Fragment))
    - "pad1" = 0x36 (n fois), "pad2" = 0x5C(n fois)
  - Fonctions de hachage :
    - rien ou MD5 (16 octets) ou SHA-1 (20 octets)
  - Procédé de calcul similaire à HMAC

    HMAC-hash(MAC_write_secret , seq_num || Compression.type || Compresssion.version ||Compression.length || Compressed. Fragment))
    - Utilisé par TLS

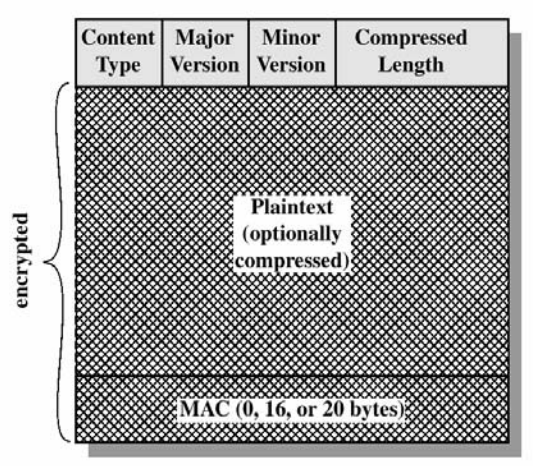
Algorithmes de chiffrement

Algorithme	Longueur de la clef	Commentaires
IDEA	128	Mode par blocs
RC2	40	Mode par blocs
DES	56	Mode par blocs
3DES	168	Mode par blocs
Fortezza	80	Mode par blocs, (chiffrement des cartes bancaires), seulement SSL
RC4	40	"stream"
RC4	128	"stream"

Bourrage

- Un "padding" est ajouté avant le chiffrement en mode par blocs, pour que la quantité de données soit un multiple de la taille du bloc.
- Un octet de fin de bourrage est toujours ajouté et indique la longueur du bourrage.
- Exemple :
  - Texte de 58 octets, sans compression, avec signature SHA-1 (20 octets), chiffré avec DES (bloc de 64 bits).
  - Longueur du bourrage :
    - Un octet de bourrage + l'octet de fin de bourrage valant 1.
- Pour SSL, on utilise le plus petit bourrage possible; pour TLS on utilise n'importe lequel qui convient
  - Longueur inférieure à 255,
  - Cela améliore la résistance aux attaques.

SSL Record Format

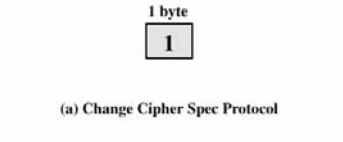


Champs du "SSL record"

- "Content Type" (8 bits):
  - "change_cipher_spec" = 20, "alert" = 21, "handshake" = 22,"application_data" = 23
- Versions (2 x 8 bits)
  - SSL : "major" = 3, "minor" = 0
  - TLS : "major" = 3, "minor" = 1
- "Compressed length" (16 bits)
  - Au maximum 2^{14} + 2048 octets

"Change Cipher Spec Protocol"

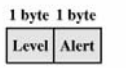
- Lorsque la "cipher suite" doit être changée :
  - à la réception de ce message d'un octet, on substitue à la spécification courante la spécification en attente ("pending state").



(a) Change Cipher Spec Protocol

"Alert Protocol "

- Transmission d'un message d'alerte aux entités.
  - Ces messages peuvent être compressés et chiffrés comme les autres messages de la même session.
- Le champ "Level" :
  - "warning" = 1
  - "fatal" = 2 : la connexion est close et il n'y aura plus de nouvelles connexions pour cette Session
- Le champ "Alert" spécifie l'erreur :
  - "unexpected_message", "bad_record_MAC", "decompression_failure", "handshake_failure", "illegal_parameter" // "close_notify", "no_certificate' "bad_certifcate", "unsupported_certificate", "certificate_expired", "certificate_unknown"



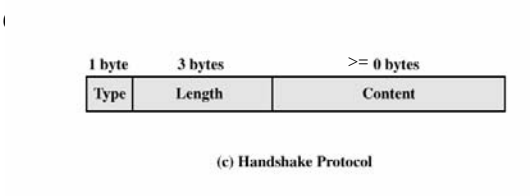
(b) Alert Protocol

"Handshake Protocol "

- Services du "Handshake Protocol":
  - Négociation des algorithmes et des clefs
  - Authentification du client et du serveur

Préalable à toutes transmission de données.

- Format général
  - format TLV



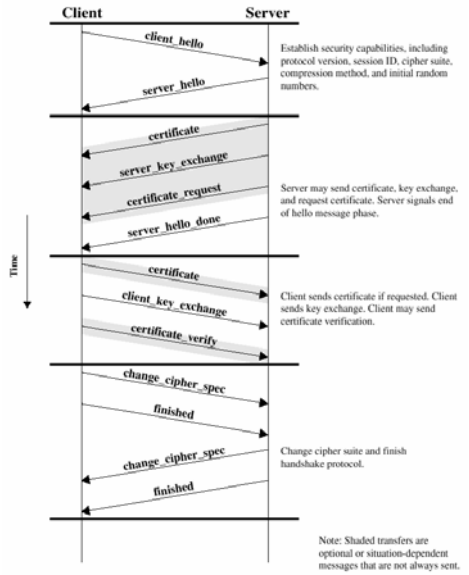
(c) Handshake Protocol

"Upper Layer Protocol "

- N'importe quel protocole applicatif :
  - par ex. HTTP, FTP, POP, IMAP, etc.

(d) Other Upper-Layer Protocol (e.g., HTTP)

Les phases du "Handshake Protocol"



Phase 1 : établissement des capacités de sécurité

- Capacités de sécurité :
  - "CipherSuite"
    - Méthode d'échange des clefs
    - "CipherSpec"
  - Méthode de compression
- La phase 1
  - Échange d'un message "client-hello" suivi d'un message "server-hello"

Message "client_hello"

- Paramètres du message "client_hello" :
  - Version :
    - Négocie la version commune la plus élevée possible
  - "Random"
    - Utilisé comme "nonce" pour lutter contre le rejeu : horodateur sur 32 bits + 28 bits aléatoires
  - "Session ID"
    - 0 lors de la création d'une nouvelle connexion dans une nouvelle session
  - "Cipher Suite" :
    - Une liste en ordre de préférence décroissante de "cipher suites"
    - Chacune définie un algo. d'échange de clefs et une "CipherSpec"
  - "Compression method" :
    - Une liste ordonnée de méthodes

Message "server-hello"

- Paramètres du message "server_hello" :
  - Version :
    - La version la plus élevée possible compatible entre le client et le serveur
  - "Random"
    - Les bits aléatoires sont propres au serveur
  - "Session ID"
    - Le numéro de la (nouvelle) session
  - "Cipher Suite" :
    - La "cipher suite" sélectionnée
  - "Compression method" :
    - La méthode sélectionnée

Méthodes d'échange de clefs

- Méthodes d'échange de clefs :
  - RSA :
    - La clef secrète est chiffrée par la clef publique du récepteur. Nécessite un certificat de cette clef publique.
  - Diffie-Hellman fixe :
    - Produit un secret partagé construit à partir de paramètres de clefs publiques de D-H qui on été certifiés.
    - L'authentifcation du client est optionnelle.
  - Diffie-Hellman éphémère :
    - Un secret partagé temporaire est créé. Les paramètres de clefs publiques de D-H sont signés par la clef privée (RSA ou DSS) de l'émetteur.
    - Un certificat authentifie la clef publique utilisée pour vérifiée la clef D-H.
  - Diffie-Hellman anonyme :
    - L'algo. D-H standard (pas d'authentification) : attaque de type MiM possible
  - Fortezza

"CipherSpec"

- Les paramètres du "CipherSpec" :
  - Cipher algorithm :
    - RC4, RC2, DES, 3DES, IDEA, Fortezza.
  - "MAC algorithm" :
    - MD5, SHA-1
  - "Cipher Type" :
    - "Stream" ou mode par blocs
  - "IsExportable" (booléen)
  - "HashSize" :
    - 0, 16 (MD5), 20 (SHA-1) octets
  - "Key materials" :
    - une séquence d'octets
  - "IV size" :
    - pour le chiffrement en mode CBC

Phase 2 : authentification du serveur et échange des clefs

- Message "Certificate" (optionnel) :
  - Une liste d'un ou plusieurs X.509 certificats
    - Ceux des paramètres D-H publics du serveur (sauf D-H anonyme)
    - … ou ceux de la clef publique RSA du serveur
- Message "server_key_exchange" (optionnel) :
  - inutile pour D-H fixe et RSA
  - Contient les paramètres D-H (plus leur signature si nécessaire)
  - Les paramètres D-H sont concaténées aux "randoms" des messages "hello"
- Message certificat_request" (optionnel) :
  - inutile pour D-H anonyme
  - "Certificat_type" :
    - Algo. RSA ou DSA, pour signature seulement ou pour D-H fixe ou pour D-H éphémère
  - "Certificat_authorities" :;
    - Liste d'autorités de certification acceptables
- Message "server_done" :
  - Le serveur attend la réponse du client

Phase 3 : authentification du client et échange des clefs

- Le client doit vérifier la validité d'un des certifcats reçus et les paramètres du message "server-hello"
- Message "Certificate" (optionnel) :
  - idem
- Message "client_key_exchange" :
  - RSA : contient la "pre-master key" chiffrée
  - D-H éphémère ou anonyme : contient les paramètres D-H du client
  - D-H fixe : rien (les paramètres sont dans le certificat)
- Message certificat_verify" (optionnel) :
  - Utilisé lorsque le certificat a été utilisé pour signer (donc inutile pour fixe D-H anonyme)
  - Vérification que le client possède la clef privée pour le certificat du client
  - SIGN(master_secret || pad2 || SIGN(handshake_messages || master_secret || pad1))
    - SIGN = {MD5 et/ou SHA-1}
    - Tous les messages "Handshake" à partir du "Client-hello" sauf celui-ci

Phase 4 : modification de la "cipher suite" et terminaison

- Le client envoie d'un message "change-cipher-spec"
  - N'est pas élément du protocole de "Handshake"
- Puis le message "finished" utilise les nouveaux algorithmes, clefs et secrets
  - Permet de vérifier l'échange des clefs et l'authentification.
  - Le message contient
    - SIGN(master_secret || pad2 || SIGN(handshake_messages || Sender || master_secret || pad1))
      - SIGN = {MD5 et SHA-1},
      - sender = 1 si client, 2 sinon
      - Tous les messages "Handshake" à partir du "Client-hello" sauf celui-ci
- Idem par le serveur

Création du "master secret"

- Le "master secret"
  - 48 octets
  - Permet l'échange sécurisé de clefs
  - 1ère phase : échange du "pre-master-secret"
    - Soit générée par le client puis transmis chiffré avec la clef publique du serveur
    - Soit par D-H
  - 2ème phase : calcul du "master secret"

    Master-secret = MD5(pre-master-secret||SHA('A'||pre-master-secret||Client.random||Server.random))|| MD5(pre-master-secret||SHA('BB'||pre-master-secret||Client.random||Server.random))||MD5(pre-master-secret||SHA('CCC'||pre-master-secret||Client.random||Server.random))

Génération des autres paramètres secrets

- Les paramètres secrets suivants :
  - client-write-MAC-secret, server-write-MAC-secret, client-write-key, server-write-key, client-write-IV, server-write-IV
- … sont générés dans cette ordre à partir du master-secret
- Utilise la méthode proposée pour générer le master-secret
  - celle-ci est répétée autant de fois que nécessaire pour obtenir une longueur suffisante
  - Master-secret = MD5(pre-master-secret||SHA('A'||pre-master-secret||Client.random||Server.random))|| MD5(pre-master-secret||SHA('BB'||pre-master-secret||Client.random||Server.random))||MD5(pre-master-secret||SHA('CCC'||pre-master-secret||Client.random||Server.random)) || …

SSL versus TLS

- Le même format de message.
- L'un est propriétaire, l'autre publique (IETF).
- Les différences mineures résident dans :
  - version number
  - message authentication code
  - pseudorandom function (TLS : fonction PRF)
  - alert codes (TLS en ajoute plusieurs)
  - cipher suites
  - client certificate types
  - certificate_verify and finished message
  - cryptographic computations
  - padding

HTTPS

- HTTPs = HTTP standard sur le port 443 (à la place de 80) + TLS + TCP/IP
- Généralement le driver SSL/TLS est intégré au navigateur

Open SSL

The OpenSSL ssl library implements the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. It provides a rich API which is documented here.

At first the library must be initialized; see SSL_library_init(3).
Then an SSL_CTX object is created as a framework to establish TLS/SSL enabled connections (see SSL_CTX_new(3)). Various options regarding certificates, algorithms etc. can be set in this object.

When a network connection has been created, it can be assigned to an SSL object. After the SSL object has been created using SSL_new(3), SSL_set_fd(3) or SSL_set_bio(3) can be used to associate the network connection with the object.

Then the TLS/SSL handshake is performed using SSL_accept(3) or SSL_connect(3) respectively. SSL_read(3) and SSL_write(3) are used to read and write data on the TLS/SSL connection. SSL_shutdown(3) can be used to shut down the TLS/SSL connection.

Cf. http://www.openssl.org
http://www.rtfm.com/openssl-examples

TLS et les mobiles

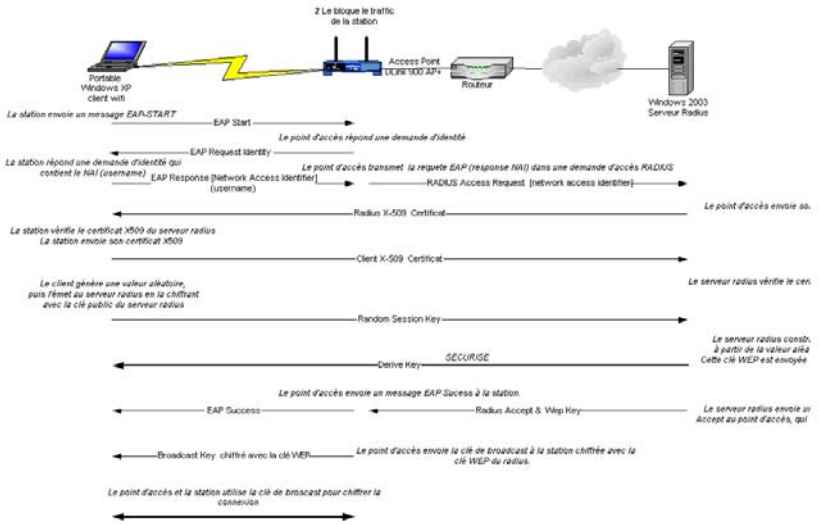
- TLS et les réseaux sans fil
- EAP – TLS
  - "Extensible Authentification Protocol" : un "framework" qui permet à des adaptateurs pour des clients sans fils d'utiliser différents types d'authentification pour communiquer avec des serveurs "back-end" (par ex. Radius)
  - Le point d'accès (adaptateur) est transparent à la méthode d'authentification

Exemple

Before sending a credit card number to buy a book at Amazon.com, a customer must verify that the Web site he or she entered is indeed Amazon.com. Also, a secured tunnel between the customer and Amazon must be established to send the credit card number safely. SSL provides this capability. In this case, the customer (using SSL) authenticates Amazon; but note that Amazon does not authenticate the customer. This is called server-side authentication (only the server is authenticated). With EAP-TLS, the RADIUS server authenticates the user, and the user authenticates the RADIUS server. This is called mutual authentication. EAP-TLS authentication will be examined in detail later.

There are two means to verify that Amazon is Amazon. If Amazon and the customer share a secret (a shared secret known only to the customer and to Amazon), the customer is then able to challenge Amazon and to verify that Amazon is holding the shared secret. The problem with this model is that it is impossible for everyone in the world to have a shared secret with everyone else. PKI was invented for this reason. PKI eliminates the need for a shared secret between you and Amazon. Digital certificates are used instead.

EAP



TLS et les RFC

- "TLS", rfc 2246, rfc 3546, janv 1999, june 2003
- "Secure SMTP over TLS", rfc 3207,
- "Using TLS with IMAP, POP3", rfc 2595, june 1999
- "PPP EAP TLS Authentication Protocol", rfc 2716, oct 1999
- "HTTP over TLS", rfc 2817, may 2000
- "Securing FTP with TLS", rfc 4217, oct 2005
- "Common Open Policy Service (COPS) over TLS", rfc 4261, dec 2005

Secure Electronic Transactions

- An open encryption and security specification.
- Protect credit card transaction on the Internet.
- Companies involved:
  - MasterCard, Visa, IBM, Microsoft, Netscape, RSA, Terisa and Verisign
- Not a payment system.
- Set of security protocols and formats.

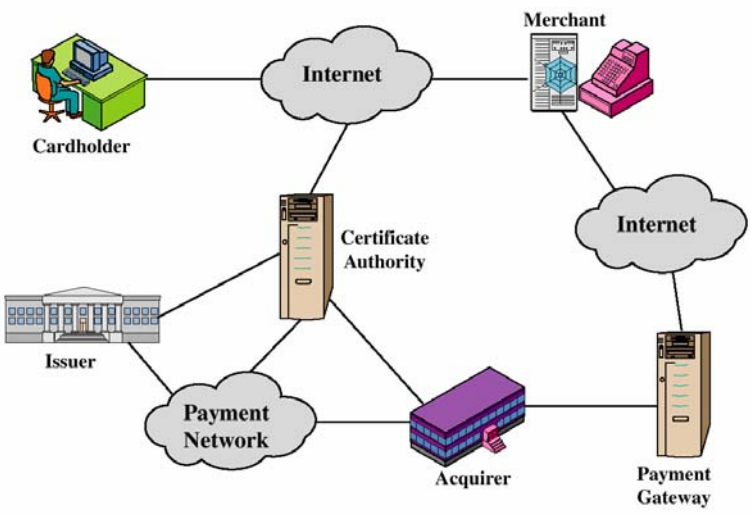
SET Services

- Provides a secure communication channel in a transaction.
- Provides tust by the use of X.509v3 digital certificates.
- Ensures privacy.

SET Overview

- Key Features of SET:
  - Confidentiality of information
  - Integrity of data
  - Cardholder account authentication
  - Merchant authentication

SET Participants

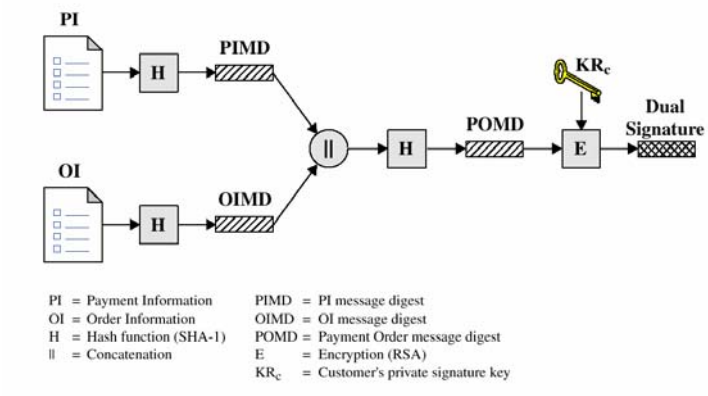


Sequence of events for transactions

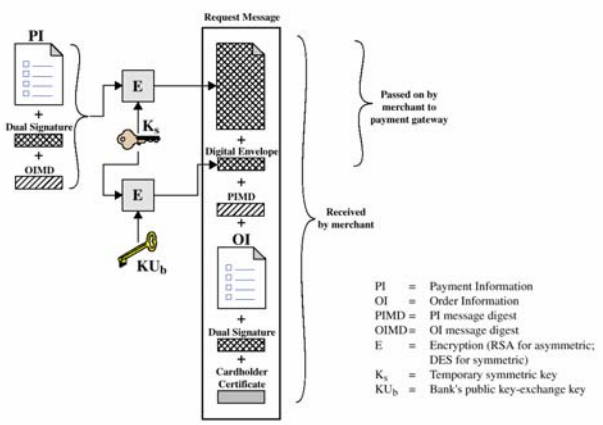
1. The customer opens an account.
2. The customer receives a certificate.
3. Merchants have their own certificates.
4. The customer places an order.
5. The merchant is verified.
6. The order and payment are sent.
7. The merchant request payment authorization.
8. The merchant confirm the order.
9. The merchant provides the goods or service.
10. The merchant requests payments.

Dual Signature

$$DS = E_{KR_c}[H(H(PI) \| \mathrm{H(OI)})]$$

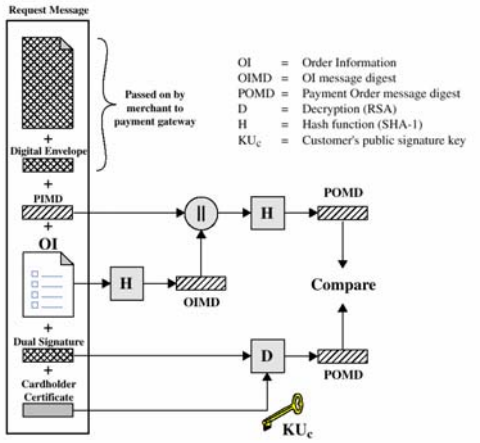


Payment processing



Cardholder sends Purchase Request

Payment processing



Merchant Verifies Customer Purchase Request

Payment processing

- Payment Authorization:
  - Authorization Request
  - Authorization Response
- Payment Capture:
  - Capture Request
  - Capture Response

Bibliographie

- Drew, G. *Using SET for Secure Electronic Commerce*. Prentice Hall, 1999
- Garfinkel, S., and Spafford, G. Web Security & Commerce. O'Reilly and Associates, 1997
- Eric Rescola. SSL and TLS : Design and Building Secure Systems. Addison Wesley, 2001