Security of network applications

Antonio Lioy
< lioy @ polito.it >

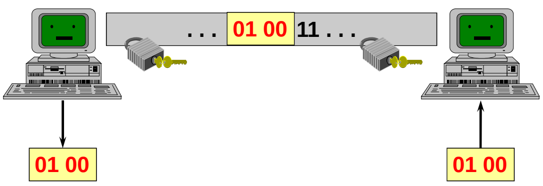
Politecnico di Torino
Dip. Automatica e Informatica

Standard situation

- **authentication and authorisation based on username and password**
  - problem: password snooping
- **authentication based on IP address (server web; R commands = rsh, rlogin, rcp, ...)**
  - problem: IP spoofing
- **general problems:**
  - data snooping / forging
  - shadow server / MITM

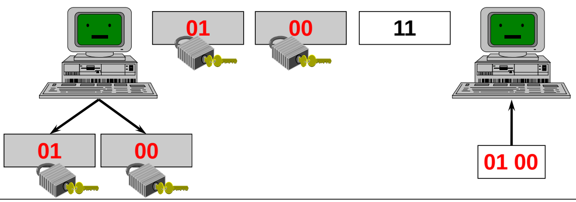
Channel security

- **authentication (single or mutual), integrity and privacy only during the transit inside the communication channel**
- **no possibility of non repudiation**
- **requires no (or small) modification of applications**

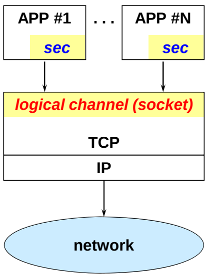


Message / data security

- authentication (single), integrity and privacy self-contained in the message
- possibility of non repudiation
- requires modification of applications

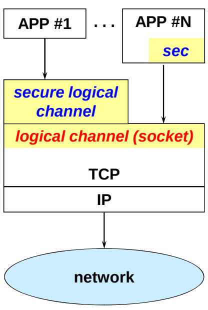


Security internal to the applications



- each application implements security internally
- the common part is limited to the communication channels (socket)
- possible implementation errors (inventing security protocols is not simple!)
- does not guarantee interoperability

Security external to applications



- the session level would be the ideal one to be used to implement many security functions
- … but it does not exist in TCP/IP!
- a "secure session" level was proposed:
  - it simplifies the work of application developers
  - it avoids implementation errors
  - it is up to the application to select it (or not)

Secure channel protocols

- **SSL / TLS**
  - the most widely used !
- **SSH**
  - it was a successful product (especially in the period when export of USA crypto products was restricted), but today it is a niche product
- **PCT**
  - proposed by MS as an alternative to SSL
  - one of the few fiascos of MS!

SSL (Secure Socket Layer)

- **proposed by Netscape Communications**
- **secure transport channel (session level):**
  - peer authentication (server, server+client)
  - message confidentiality
  - message authentication and integrity
  - protection against replay and filtering attacks
- **easily applicable to all protocols based on TCP:**
  - HTTP, SMTP, NNTP, FTP, TELNET, ...
  - e.g. the famous secure HTTP (https://....) = 443/TCP

Official ports for SSL applications

nsiiops	**261/tcp # IIOP Name Service over TLS/SSL**
https	**443/tcp # http protocol over TLS/SSL**
smtps	**465/tcp # smtp protocol over TLS/SSL (was ssmtp)**
nntps	**563/tcp # nntp protocol over TLS/SSL (was snntp)**
imap4-ssl	**585/tcp # IMAP4+SSL (use 993 instead)**
sshell	**614/tcp # SSLshell**
ldaps	**636/tcp # ldap protocol over TLS/SSL (was sldap)**
ftps-data	**989/tcp # ftp protocol, data, over TLS/SSL**
ftps	**990/tcp # ftp protocol, control, over TLS/SSL**
telnets	**992/tcp # telnet protocol over TLS/SSL**
imaps	**993/tcp # imap4 protocol over TLS/SSL**
ircs	**994/tcp # irc protocol over TLS/SSL**
pop3s	**995/tcp # pop3 protocol over TLS/SSL (was spop3)**
msft-gc-ssl	**3269/tcp # MS Global Catalog with LDAP/SSL**

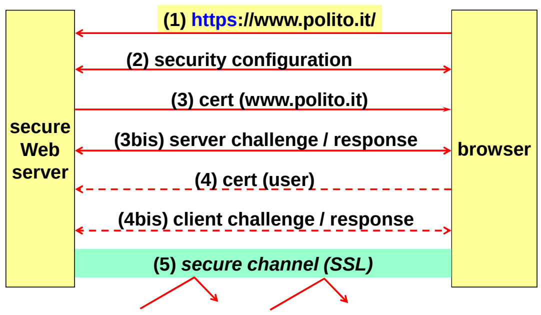
SSL – authentication and integrity

- **peer authentication at channel setup:**
  - the server authenticates itself by sending its public key (X.509 certificate) and by responding to an asymmetric challenge
  - the client authentication (with public key and X.509 certificate) is optional
- **for authentication and integrity of the data exchanged over the channel the protocol uses:**
  - a keyed digest (MD5 or SHA-1)
  - an MID to avoid *replay* and cancellation

SSL - confidentiality

- **the client generates a session key used for symmetric encryption of data (RC2, RC4, DES, 3DES or IDEA)**
- **the key is sent to the server after having encrypted it with the public key of the server (RSA, Diffie-Hellman or Fortezza-KEA)**

SSL



SSL-3 architecture

<table>
<tr><td>SSL handshake protocol</td><td>SSL change cipher spec protocol</td><td>SSL alert protocol</td><td>application protocol (e.g. HTTP)</td></tr>
<tr><td colspan="4">SSL record protocol</td></tr>
<tr><td colspan="4">reliable transport protocol (e.g. TCP)</td></tr>
<tr><td colspan="4">network protocol (e.g. IP)</td></tr>
</table>

Session-id

Tipical web transaction:

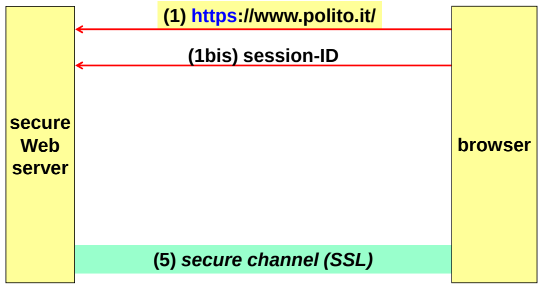
- **1. open, 2. GET page.htm, 3. page.htm, 4. close**
- **1. open, 2. GET home.gif, 3. home.gif, 4. close**
- **1. open, 2. GET logo.gif, 3. logo.gif, 4. close**
- **1. open, 2. GET back.jpg, 3. back.jpg, 4. close**
- **1. open, 2. GET music.mid, 3. music.mid, 4. close**

If the SSL cryptographic parameters must be negotiated every time, then the computational load becomes high.

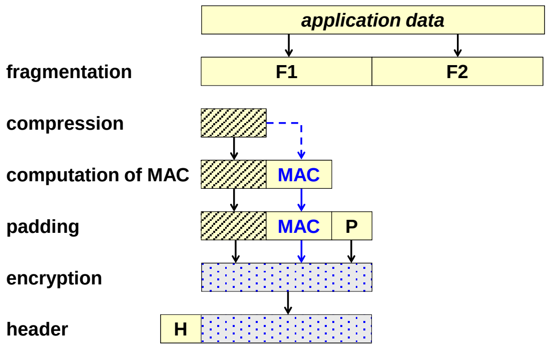
Session-id

- **in order to avoid re-negotiation of the cryptographic parameters for each SSL connection, the SSL server can send a session identifier (that is, more connections can be part of the same logical session)**
- **if the client, when opening the SSL connection, sends a valid *session-id* then the negotiation part is skipped and data are immediately exchanged over the secure channel**
- **the server can reject the use of session-id (always or after a time passed after its issuance)**

SSL with session-ID



SSL-3 record protocol



TLS-1.0 record format

- uint8 type = change_cipher_spec (20), alert (21), handshake (22), application_data (23)
- uint16 version = major (uint8) + minor (uint8)
- uint16 length:
  - <= 2**14 (record not compressed) for compatibility with SSL-2
  - <= 2**14 + 1024 (compressed records)

type	
major	minor
length	
. . . fragment [length] . . .	

SSL – computation of MAC

```
MAC = message_digest ( key, seq_number ||
type || version || length || fragment )
```

- `message_digest`
  - depends on the chosen algorithm
- `key`
  - sender-write-key or receiver-read-key
- `seq_number`
  - 32-bit integer

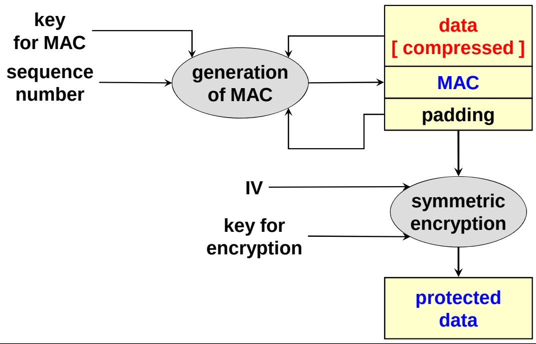
SSL-3: new features with respect to SSL-2

- **data compression:**
  - optional
  - before encryption (after it's not useful anymore …)
- **data encryption is optional:**
  - in order to have only authentication and integrity
- **possibility to re-negotiate the SSL connection:**
  - periodical change of keys
  - change of the algorithms

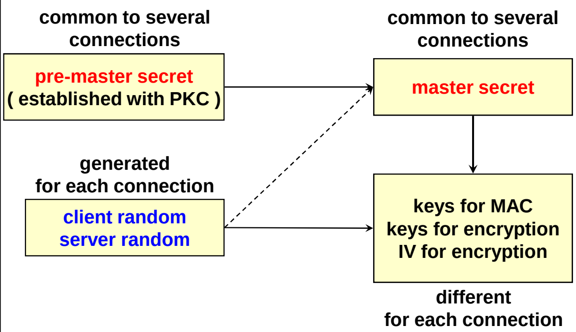
SSL-3 handshake protocol

- **agree on a set of algorithms for confidentiality and integrity**
- **exchange random numbers between the client and the server to be used for the subsequent generation of the keys**
- **establish a symmetric key by means of public key operations (RSA, DH or Fortezza)**
- **negotiate the session-id**
- **exchange the necessary certificates**

Confidentiality of data

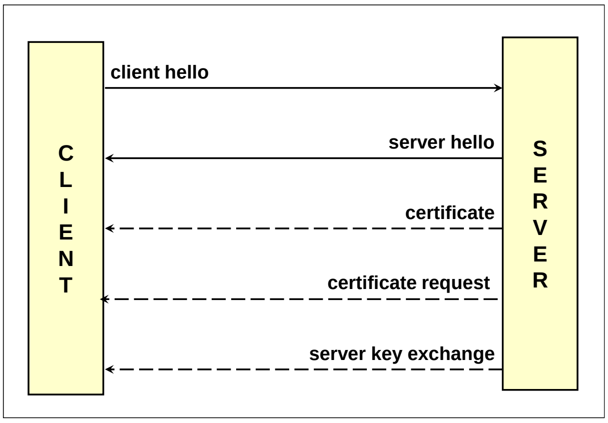


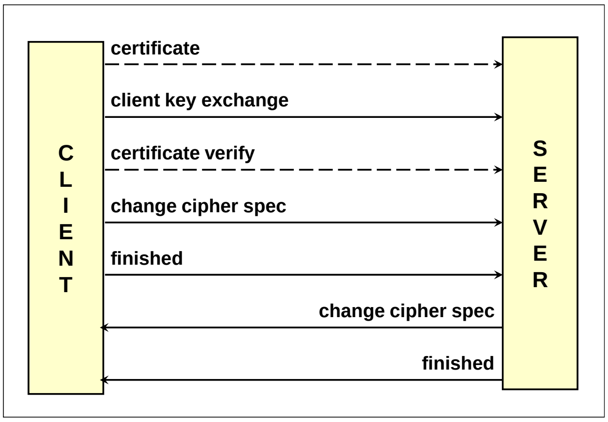
Relationship among keys and sessions



“Ephemeral ” mechanisms

- **one-time keys generated on the fly:**
  - to provide authentication they must be signed (e.g. an X.509 certificate must be available)
  - DH suitable, RSA slow
    - compromise for RSA = re-use N times
- **perfect forward secrecy:**
  - who knows the private key can decrypt all the SSL sessions
  - with ephemeral mechanisms the server’s private key is used only for signing





Client hello

- **SSL version preferred by the client**
- **28 bytes generated in a pseudo-random manner**
- **a session identifier (session-id)**
  - 0 to start a new session
  - different from 0 to ask to resume a previous session
- **list of "cipher suite" (=alg of encryption + key exchange + integrity) supported by the client**
- **list of compression methods supported by the client**

Server hello

- **SSL version chosen by the server**
- **28 bytes generated in a pseudo-random manner**
- **a session identifier (session-id)**
  - new session-id if session-id=0 in the client-hello or reject the session-id proposed by the client
  - session-id proposed by the client if the server accepts to resume the session
- **"cipher suite" chosen by the server**
  - should be the strongest one in common with the client
- **compression method chosen by the server**

Cipher suite

- **key exchange algorithm**
- **symmetric encryption algorithm**
- **hash algorithm (for MAC)**

- **examples:**
  - SSL_NULL_WITH_NULL_NULL
  - SSL_RSA_WITH_NULL_SHA
  - SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
  - SSL_RSA_WITH_3DES_EDE_CBC_SHA

Certificate (server)

- **certificate for server authentication**
  - the subject / subjectAltName must be the same as the identity of the server (DNS name, IP address, ...)
- **can be used only for signing or (in addition) also for encryption**
  - described in the field keyUsage
  - if it is only for signing then it is required also the phase for server-key exchange

Certificate request

- **used for client authentication**
- **specifies also the list of CAs considered trusted by the server**
  - the browsers show to the users (for a connection) only the certificates issued by trusted CAs

Server key exchange

- **carries the server public key for encryption**
- **needed only in the following cases:**
  - the RSA server certificate can be used only for signature
  - anonymous or ephemeral DH is used to establish the master-secret
  - there are export problems that force the use of ephemeral RSA/DH keys
  - Fortezza
- **important: this is the only message esplicitly signed by the server**

Certificate (client)

- **carries the certificate for client authentication**
- **the certificate must have been issued from one CA in the trusted CA list in the Certificate Request message**

Client key exchange

- the client generates symmetric keys and send them to the server
- various ways
  - pre-master secret encrypted with the server RSA public key (ephemeral or from its X.509 certificate)
  - public part of DH
  - Fortezza

Certificate verify

- explicit test signature done by the client
- hash computed over all the handshake messages before this one and encrypted with the client private key

Change cipher spec

- trigger the change of the algorithms to be used for message protection
- allows to pass from the previous unprotected messages to the protection of the next messages with algorithms and keys just negotiated
- theoretically is a protocol on its own and not part of the handshake
- some analysis suggest that it could be eliminated

Finished

- **first message protected with the negotiated algorithms**
- **very important to authenticate the whole handshake sequence:**
  - contains a MAC computed over all the previous handshake messages (but change cipher spec) using as a key the master secret
  - useful to avoid man-in-the-middle attacks of the rollback type
  - different for client and server

TLS

- **Transport Layer Security**
- **standard IETF:**
  - TLS-1.0 = RFC-2246 (jan 1999)
  - TLS-1.1 = RFC-4346 (apr 2006)
- **TLS-1.0 = SSL-3.1 (99% coincident with SSL-3)**
- **emphasis on standard (i.e. not proprietary) digest and asymmetric crypto algorithms; mandatory:**
  - DH + DSA + 3DES
  - HMAC
  - ... that is the ciphersuite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

TLS-1.1

- **RFC-4346**
- **the implicit IV is replaced with an explicit IV to protect against CBC attacks**
- **handling of padding errors is changed to use the bad_record_mac rather than the decryption_failed alert to protect against CBC attacks**
- **IANA registries defined for protocol parameters**
- **premature closes no longer cause a session to be nonresumable**
- **additional notes added for various new attacks**

TLS evolution

- **ciphersuites:**
  - (RFC-2712) Kerberos ciphersuites for TLS
  - (RFC-3268) AES ciphersuites for TLS
  - (RFC 4492) ECC cipher suites for TLS
  - (RFC-4132) Camellia ciphersuites for TLS
  - (RFC-4279) pre-shared key ciphersuites for TLS
- **compression:**
  - (RFC-3749) TLS compression methods
  - (RFC-3943) TLS protocol compression using LZS
- **other:**
  - (RFC-4366) TLS extensions

DTLS

- **Datagram Transport Layer Security (RFC-4347)**
- **applies the TLS concepts to datagram security (e.g. UDP)**
- **doesn't offer the same properties as TLS**
- **competition with IPsec and application security**
- **example – SIP security:**
  - with IPsec
  - with TLS (only for SIP_over_TCP)
  - with DTLS (only for SIP_over_UDP)
  - with secure SIP

HTTP security

- **security mechanisms defined in HTTP/1.0:**
  - "address-based" = the server performs access control based on the IP address of the client
  - "password-based" (or Basic Authentication Scheme) = access control based on username and password, Base64 encoded
- **both schemas are highly insecure (because HTTP assumes a secure channel!)**
- **HTTP/1.1 introduces "digest authentication" based on a symmetric challenge**
- **RFC-2617 "HTTP authentication: basic and digest access authentication"**

HTTP - basic authentication scheme

```
GET /path/to/protected/page
  HTTP/1.0 401 Unauthorized - authentication failed
  WWW-Authenticate: Basic realm="RealmName"
Authorization: Basic B64_encoded_username_password
  HTTP/1.0 200 OK
  Server: NCSA/1.3
  MIME-version: 1.0
  Content-type: text/html
  <HTML> protected page ... </HTML>
```

HTTP and SSL/TLS

- **two approaches:**
  - "TLS then HTTP"
    (RFC-2818 – HTTP over TLS)
  - "HTTP then TLS"
    (RFC-2817 – upgrading to TLS within HTTP/1.1)
  - note: "SSL then HTTP" is in widespread use but it is undocumented
- **the two approaches are not equivalent and have an impact over applications, firewall and IDS**
- **concepts generally applicable to all protocols:**
  - "SSL/TLS then proto" vs. "proto then TLS"

WWW security

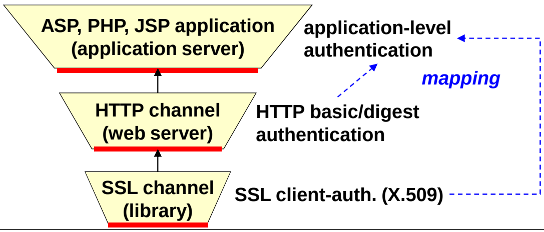
- **SSL channel:**
  - protection of the transactions
  - protection of the application passwords
- **password (for Basic/Digest Authentication):**
  - of the HTTP service
  - of the OS hosting the server (e.g. XP or UNIX)
- **ACL for document access control:**
  - depending on the authentication performed (OS users, X.509 DN)

SSL client authentication at the application level

- via client authentication it's possible to identify the user that opened the channel (without asking for his username and password)
- some web servers support a (semi-)automatic mapping between the credentials extracted from the X.509 certificate and the users of the web server and/or the OS

Authentication in web applications

- the earlier the access control, the smaller the attack surface
- no need to repeat the authentication (the id may be propagated)



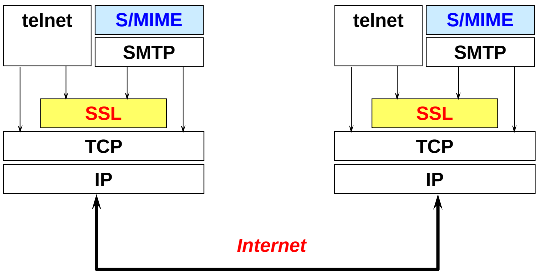
What about forms requesting user/pwd?

- the actual security depends on the URI of the method used to send username and password to the server
- technically speaking, it's not important the security of the page containing the form
- psicologically, it is very important the security of the page containing the form because few users have the technical knowledge to verify the URI of the HTTP method used to send user/pwd

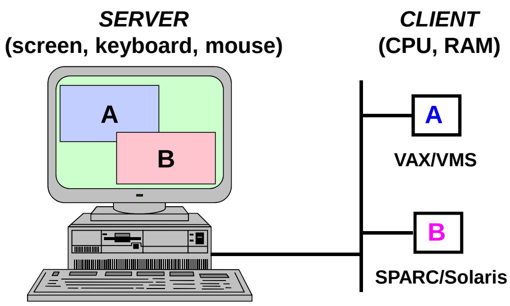
S-HTTP

- new version of the HTTP-1.0 protocol developed by EIT-Terisa
- HTTP queries and replies are encapsulated inside a secure envelope (PEM, PGP or PKCS-7)
- key negoziation: in-line, OOB or Kerberos
- certificates: X.509 o PKCS-6
- digital signature: RSA o DSA
- digest: MD2, MD5 o SHA-1
- encryption: DES, IDEA, RC2, RC4
- RFC-2660 “The Secure HTTP”
- RFC-2659 “Security extensions for HTML”

Coexistence of SSL with other protocols



The X-windows system (X11)



Security of X-windows

- **authentication:**
  - IP address (`xhost +host`)
  - cookie (since X11R4)
  - Kerberos V5 (since X11R6)
  - weak authentication implies that by using X11 primitives (e.g. xwindump) we may capture the whole I/O of the graphic terminal without any need for sniffing
- **X11 forwarding over secure channel:**
  - SSH
  - IPsec

Security of remote DBMS access

- **protection heavily depends on the access type (note that there is not a thing such as "network SQL"):**
  - terminal emulation
    = channel protection (SSL-telnet, SSH, ...)
  - web-based DBMS front-end
    = web protection (SSL, ...)
  - client-server query environment
    = proprietary security solution, or IPsec for protecting the transactions

E-payment systems

- **failure of the digital cash, for technical and political problems (e.g. the DigiCash failure)**
- **currently the most widely used approach is transmittting a credit card number over a SSL channel ...**
- **... but this is no guarantee against fraud: VISA Europe declares that Internet transactions generate about 50% of the fraud attempts, although they are just 2% of its total transaction amount!**

Security of the credit card transactions

- STT
  (Secure Transaction Technology)
  VISA + Microsoft
- SEPP
  (Secure Electronic Payment Protocol)
  Mastercard, IBM, Netscape, GTE, CyberCash
- SET = STT + SEPP
  (Secure Electronic Transaction)

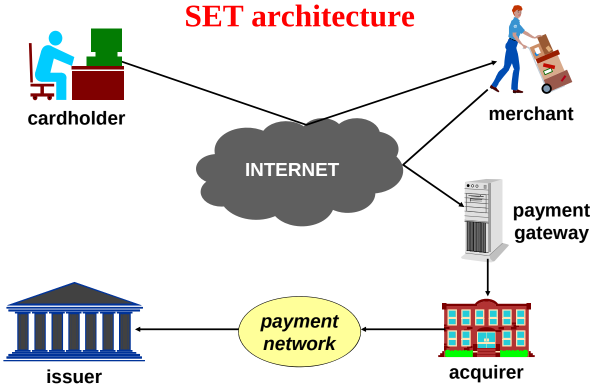
SET

- SET is not a payment system but a set of protocols to use inside an open untrusted network the existing infrastructure for credit card payments
- uses X.509v3 certificates dedicated only to SET transactions
- protects the user privacy because it shows to each part only the relevant and pertinent data

Features of SET

- version 1.0 (may 1997)
- digest: SHA-1
- symmetric encryption: DES
- key exchange: RSA
- digital signature: RSA con SHA-1
- www.setco.org (doesn't exist any more)

SET architecture



SET actors (I)

- cardholder
  proper owner of a SET-enabled credit card
- merchant
  seller of a product via Internet (Web or e-mail)
- issuer
  financial institute that issued the credit card of the user

SET actors (II)

- acquirer
  financial institute that has a relatin with the merchant and interfaces it with one or more payment networks
- payment gateway
  system thattranslates the SET transactions to the format accepted by the payment network of the acquirer
- certification authority
  creates X.509v3 certificates and CRL for all the SET actors

SET double signature

- to protect the user privacy towards the merchant and the financial entities (acquirer + issuer) SET uses a double signature
- the merchant has no information about the payment details
- the financial entities have no information about the goods
- only the user can prove the association between the goods and the payment

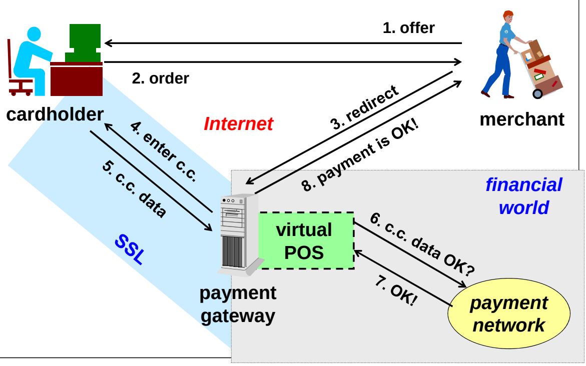
SET double signature: details

- PI: Purchase Information (payment)
- OI: Order Information (goods)
- DS = E (H(H(PI),H(OI)), Ukpri)
- DS+H(PI) to the merchant
- the merchant knows OI and thus can compute H(H(PI),H(OI)) verifying that it matches the value extracted from the signature
- DS+H(OI) to the acquirer
- the acquirer knows PI and thus can compute H(H(PI),H(OI)) verifying that it matches the value extracted from the signature

Problems of SET

- software very expensive (for the CA, the merchant and the acquirer)
- a special client-side application is needed (the SET wallet)
- complex procedure to issue the public-key certificates for the user
- a version 2.0 of SET was planned to get rid of the wallet (should have used the browser for user interface)

A web-based payment architecture



Web-based payment architecture

- **baseline:**
  - the buyer owns a credit card
  - the buyer has a SSL-enabled browser
- **consequences:**
  - the effective security depends upon the configuration of both the server and the client
  - the payment gateway has all the information (payment + goods) while merchant knows only info about the goods

PCI DSS

- **Payment Card Industry Data Security Standard**
- **required by all credit card issuers for Internet-based transactions**
- **much more detailed technical presciptions compared to other security standards (e.g. HIPAA = Health Insurance Portability and Accountability Act)**
- **https://www.pcisecuritystandards.org**

PCI DSS prescriptions (I)

- **design, build and operate a protected network:**
  - R1 = install and maintain a configuration with firewall to protect access to the cardholders' data
  - R2 = don't use pre-defined system passwords or other security parameters set by the manufacturer
- **protect the cardholders' data:**
  - R3 = protect the stored cardholders' data
  - R4 = encrypt the cardholders' data when transmitted across an open public network

PCI DSS prescriptions (II)

- **establish and follow a program for vulnerability management**
  - R5 = use an antivirus and regularly update it
  - R6 = develop and maintain protected applications and systems
- **implement strong access control**
  - R7 = limit the access to the cardholders' data only to those needed for a specific task
  - R8 = assign a single unique ID to each user
  - R9 = limit physical access to the cardholders' data

PCI DSS prescriptions (III)

- **regularly monitor and test the networks**
  - R10 = monitor and track all accesses to network resources and cardholders' data
  - R11 = periodically test the protection systems and procedures
- **adopt a Security Policy**
  - R12 = adopt a Security Policy