

## Sicurezza delle applicazioni di rete

Antonio Lioy  
< lioy @ polito.it >

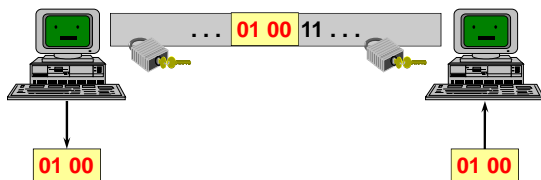
Politecnico di Torino  
Dip. Automatica e Informatica

## Situazione standard

- autenticazione ed autorizzazione basate su username e password
  - problema: password snooping
- autenticazione basata su indirizzo IP (server web; comandi R = rsh, rlogin, rcp, ...)
- problemi generali:
  - data snooping / forging
  - shadow server / MITM

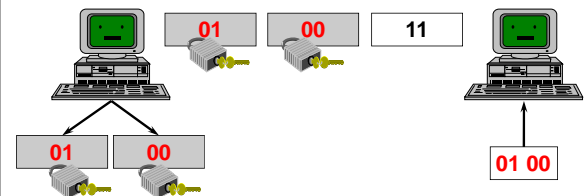
## Sicurezza di canale

- autenticazione (singola o mutua), integrità e segretezza **solo durante il transito nel canale**
- nessuna possibilità di non ripudio
- richiede poca (o nulla) modifica alle applicazioni

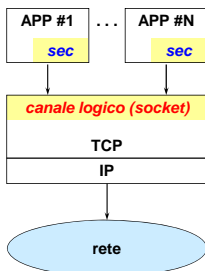


## Sicurezza di messaggio (o dei dati)

- autenticazione (singola), integrità e segretezza **auto-contenute nel messaggio**
- possibilità di non ripudio
- richiede modifica alle applicazioni

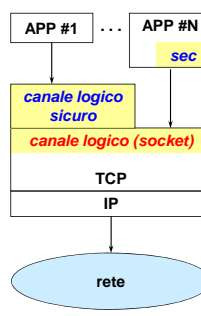


## Sicurezza interna alle applicazioni



- ogni applicazione implementa la sicurezza al proprio interno
- la parte in comune si limita ai canali di comunicazione (socket)
- possibili errori di implementazione (inventare protocolli di sicurezza non è semplice!)
- non garantisce l'interoperabilità

## Sicurezza esterna alle applicazioni



- il livello sessione sarebbe ideale per implementare molte funzioni di sicurezza
- ... ma non esiste in TCP/IP!
- è stato proposto un livello "sessione sicura":
  - semplifica il lavoro degli sviluppatori applicativi
  - evita possibili errori di implementazione
  - a scelta dell'applicazione

### Protocolli di sicurezza orientati al canale di comunicazione

- **SSL / TLS**
  - il più diffuso al mondo
- **SSH**
  - ha avuto un momento di gloria (legato ai divieti di esportazione USA), ma oggi è una soluzione di nicchia
- **PCT**
  - proposto da MS come alternativa a SSL
  - uno dei pochi fiaschi di MS!

### SSL (Secure Socket Layer)

- proposto da Netscape Communications
- protocollo di trasporto sicuro (circa livello sessione):
  - autenticazione (server, server+client)
  - riservatezza dei messaggi
  - autenticazione ed integrità dei messaggi
  - protezione da replay e da filtering
- applicabile facilmente a tutti i protocolli basati su TCP:
  - HTTP, SMTP, NNTP, FTP, TELNET, ...
  - es. famoso HTTP sicuro (https://...) = 443/TCP

### Porte ufficiali per applicazioni SSL

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 smtp)
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

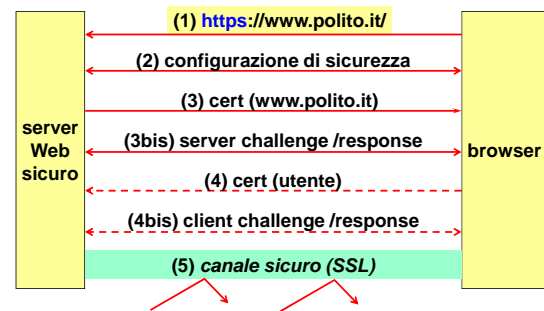
### SSL - autenticazione e integrità

- peer authentication all'apertura del canale:
  - il server si autentica presentando la propria chiave pubblica (certificato X.509) e subendo una sfida asimmetrica
  - l'autenticazione del client (con chiave pubblica e certificato X.509) è opzionale
- per l'autenticazione e l'integrità dei dati scambiati il protocollo prevede:
  - un keyed digest (MD5 o SHA-1)
  - un MID per evitare *replay* e cancellazione

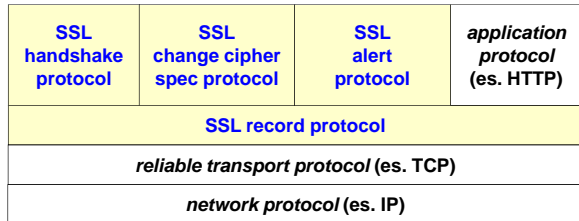
### SSL - riservatezza

- il client genera una session key utilizzata per la cifratura simmetrica dei dati (RC2, RC4, DES, 3DES o IDEA)
- la chiave viene comunicata al server cifrandola con la chiave pubblica del server (RSA, Diffie Hellman o Fortezza-KEA)

### SSL



### Architettura di SSL-3



### Session-id

Tipica transazione Web:

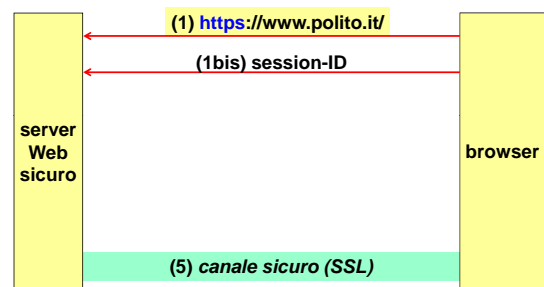
- 1. open, 2. GET page.htm, 3. page.htm, 4. close
- 1. open, 2. GET home.gif, 3. home.gif, 4. close
- 1. open, 2. GET logo.gif, 3. logo.gif, 4. close
- 1. open, 2. GET back.jpg, 3. back.jpg, 4. close
- 1. open, 2. GET music.mid, 3. music.mid, 4. close

Se ogni volta si devono rinegoziare i parametri crittografici per SSL, il collegamento si appesantisce molto.

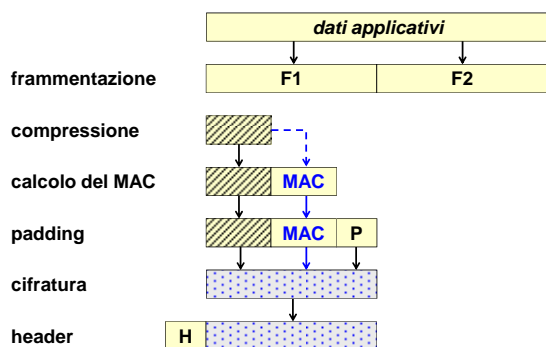
### Session-id

- per evitare di ri-negoziare ad ogni connessione i parametri crittografici, il server SSL può offrire un session identifier (ossia più connessioni possono far parte della stessa sessione logica)
- se il client, all'apertura della connessione, presenta un *session-id* valido si salta la fase di negoziazione e si procede subito col dialogo SSL
- il server può rifiutare l'uso del *session-id* (in assoluto o dopo un certo tempo dalla sua emissione)

### SSL con session-ID

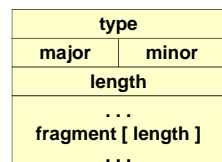


### SSL-3 record protocol



### 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:
  - $\leq 2^{14}$  (record non compressi)
  - $\leq 2^{14} + 1024$  (record compressi)



### TLS - calcolo del MAC

MAC = message\_digest ( key, seq\_number || type || version || length || fragment )

- message\_digest
  - dipende dall'algorithmo scelto
- key
  - sender-write-key o receiver-read-key
- seq\_number
  - intero a 32 bit

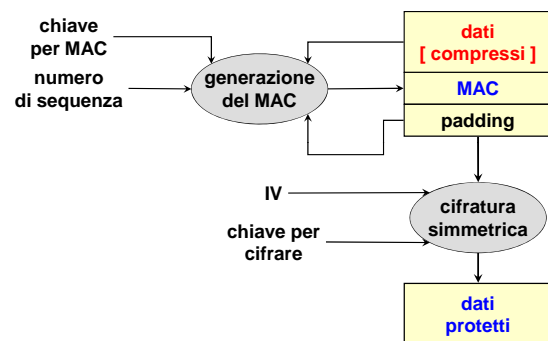
### SSL-3: novità rispetto a SSL-2

- compressione dei dati:
  - opzionale
  - prima della cifratura (dopo non serve ...)
- opzionalità della cifratura dei dati:
  - per avere solo autenticazione e integrità
- possibilità di rinegoziare la connessione:
  - cambio periodico delle chiavi
  - cambio degli algoritmi

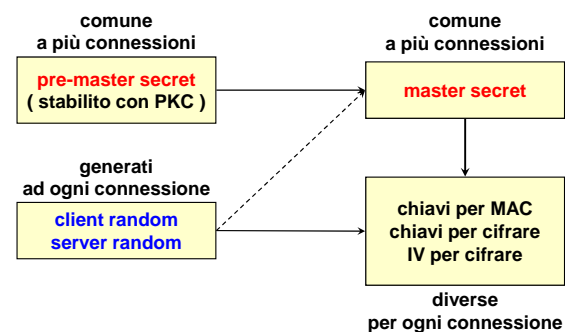
### SSL-3 handshake protocol

- concordare una coppia di algoritmi per confidenzialità ed integrità
- scambiare dei numeri casuali tra client e server per la successiva generazione delle chiavi
- stabilire una chiave simmetrica tramite operazioni a chiave pubblica (RSA, DH o Fortezza)
- negoziare il session-id
- scambiare i certificati necessari

### Confidenzialità dei dati

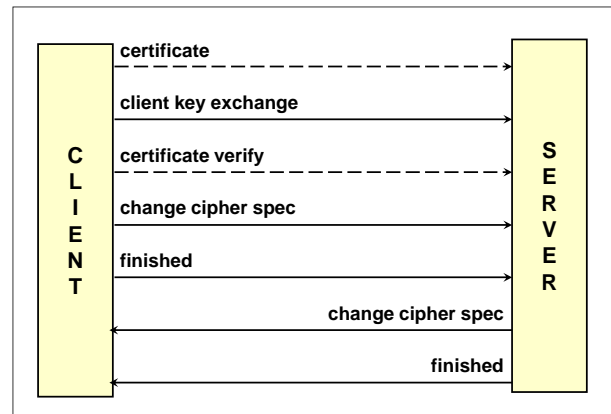
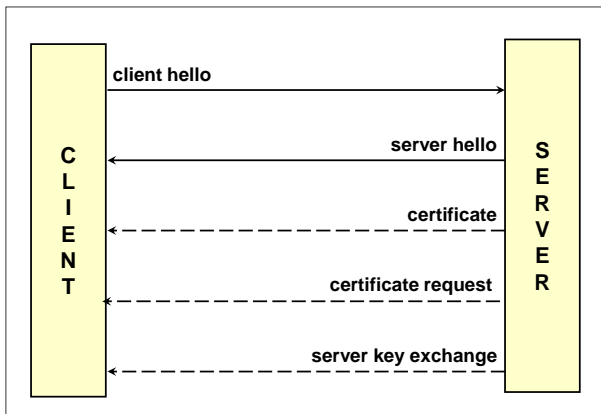


### Rapporto tra chiavi e sessioni



### Meccanismi "effimeri"

- chiavi usa-e-getta generate al volo:
  - per avere autenticazione devono essere firmate (es. devo avere un certificato X.509)
  - DH adatto, RSA lento
    - compromesso per RSA = riuso N volte
- perfect forward secrecy:
  - chi conosce la chiave privata può decifrare tutte le sessioni
  - con meccanismi effimeri la chiave privata del server viene usata solo per firma



### Client hello

- versione di SSL preferita dal client
- 28 byte generati in modo pseudo-casuale
- un identificatore di sessione (session-id)
  - 0 per iniziare una nuova sessione
  - diverso da 0 per chiedere di riesumare una sessione precedente
- elenco delle "cipher suite" (=algoritmo di cifratura + scambio chiavi + integrità) supportate dal client
- elenco dei metodi di compressione supportati dal client

### Server hello

- versione di SSL scelta dal server
- 28 byte generati in modo pseudo-casuale
- un identificatore di sessione (session-id)
  - nuovo session-id se session-id=0 nel client-hello oppure rifiuta il session-id proposto dal client
  - session-id proposto dal client se il server accetta di riesumare la sessione
- "cipher suite" scelta dal server
  - dovrebbe essere la più forte in comune col client
- metodo di compressione scelto dal server

### Cipher suite

- algoritmo di scambio chiavi
- algoritmo di cifratura simmetrico
- algoritmo di hash (per il MAC)
- esempi:
  - 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)

- certificato per server authentication
  - il subject / subjectAltName deve coincidere con l'identità del server (nome DNS, indirizzo IP, ...)
- può essere solo di firma od anche per cifratura
  - descritto nel campo keyUsage
  - se è solo per firma allora è poi necessaria la fase server-key exchange

### Certificate request

- serve per autenticazione del client
- specifica anche la lista delle CA che il server ritiene fidate
  - i browser mostrano all'utente per il collegamento solo i certificati emessi da CA fidate

### Server key exchange

- contiene la chiave pubblica di cifratura del server
- richiesto solo nei seguenti casi:
  - server ha certificato RSA solo per firma
  - si usa DH anonimo o effimero per stabilire il master-secret
  - ci sono problemi di export e si devono usare chiavi RSA/DH temporanee
  - Fortezza
- importante: unico messaggio con firma esplicita del server

### Certificate (client)

- trasporta il certificato per la client authentication
- il certificato deve essere stato emesso da una delle CA fidate elencate dal server nel messaggio Certificate Request

### Client key exchange

- varie possibilità
  - pre-master secret cifrato con la chiave pubblica RSA del server (temporanea o dal certificato)
  - parte pubblica di DH
  - Fortezza

### Certificate verify

- prova esplicita di firma
- hash calcolato su tutti i messaggi di handshake che precedono questo e firmato con la chiave privata del client

### Change cipher spec

- provoca l'aggiornamento degli algoritmi da usare per la protezione
- permette di passare dai messaggi precedenti in chiaro all'uso nei messaggi successivi degli algoritmi appena negoziati
- a rigore fa parte di un protocollo specifico e non dell'handshake
- varie analisi indicano come sia eliminabile

### Finished

- primo messaggio protetto con gli algoritmi negoziati
- fondamentale per autenticare tutto lo scambio
  - contiene un MAC calcolato su tutti i messaggi di handshake precedenti (escluso change cipher spec) tramite l'uso del master secret
  - evita attacchi man-in-the-middle di tipo rollback
  - differenziato per client e server

### TLS 1.0 (SSL 3.1)

- 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 (al 99% coincide con SSL-3)
- enfasi su algoritmi crittografici e di digest standard (non proprietari); supporto obbligatorio:
  - DH + DSA + 3DES
  - HMAC
  - ... ossia la ciphersuite  
TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA

### TLS-1.1 (SSL 3.2)

- RFC-4346
- per proteggere da attacchi contro CBC:
  - IV implicito rimpiazzato da IV esplicito
  - errori di padding ora usano il messaggio di alert bad\_record\_mac (invece di decryption\_failed)
- registrazione IANA dei parametri del protocollo
- una chiusura prematura del canale non implica più che non si possa riprendere la sessione
- note aggiuntive per chiarire vari possibili attacchi

### TLS-1.2 (SSL 3.3)

- RFC-5246
- ciphersuite specifica anche la PRF (pseudo-random function)
- uso esteso di SHA-256 (es. in Finished, HMAC)
- supporto per authenticated encryption (AES in modo GCM o CCM)
- incorpora le estensioni (RFC-4366) e le AES ciphersuite (RFC-3268)
- default ciphersuite  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- deprecate le ciphersuite con IDEA e DES

### Evoluzione di TLS

- 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
- compressione:
  - (RFC-3749) TLS compression methods
  - (RFC-3943) TLS protocol compression using LZS
- altro:
  - (RFC-4366) TLS extensions

### DTLS

- Datagram Transport Layer Security (RFC-4347)
- applica i concetti di TLS alla sicurezza dei datagrammi (es. UDP)
- non offre tutti i servizi di sicurezza di TLS
- in competizione con IPsec e sicurezza applicativa
- esempio – sicurezza di SIP:
  - con IPsec
  - con TLS (solo per SIP\_over\_TCP)
  - con DTLS (solo per SIP\_over\_UDP)
  - con secure SIP

### Sicurezza di HTTP

- meccanismi di sicurezza definiti in HTTP/1.0:
  - “address-based” = il server controlla l'accesso in base all'indirizzo IP del client
  - “password-based” (o Basic Authentication Scheme) = accesso limitato da username e password, codificate con Base64
- entrambi gli schemi sono altamente insicuri (perché HTTP suppone che sia sicuro il canale!)
- HTTP/1.1 introduce “digest authentication” basata su sfida simmetrica
- RFC-2617 “HTTP authentication: basic and digest access authentication”

### HTTP - basic authentication scheme

```
GET /path/alla/pagina/protetta HTTP/1.0
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> pagina protetta ... </HTML>
```

### HTTP e SSL/TLS

- due alternative:
  - “TLS then HTTP” (RFC-2818 – HTTP over TLS)
  - “HTTP then TLS” (RFC-2817 – upgrading to TLS within HTTP/1.1)
  - nota: “SSL then HTTP” usato ma non documentato
- non equivalenti e con impatto su applicazioni, firewall e IDS
- concetti applicabili in generale a tutti protocolli:
  - “SSL/TLS then proto” vs. “proto then TLS”

### Sicurezza del WWW

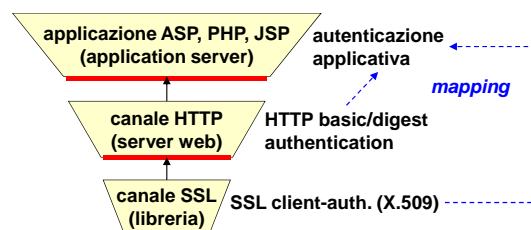
- canale SSL:
  - protezione delle transazioni
  - protezione delle password applicative
- password (per Basic/Digest Authentication):
  - del servizio HTTP
  - del S.O. che ospita il server (es. NT o UNIX)
- ACL per accedere ai documenti:
  - in funzione dell'autenticazione effettuata (utenti del S.O., DN per X.509)

### Client authentication SSL a livello applicativo

- tramite la client authentication è possibile identificare l'utente che ha aperto un canale (senza richiederli username e password)
- alcuni server web permettono di fare un mapping (semi-)automatico tra credenziali estratte dal certificato X.509 e utenti del server web e/o del S.O.

### Autenticazione nelle applicazioni web

- più in basso si fa il controllo e meno parti si espongono agli attacchi
- inutile ripetere l'autenticazione (id propagabile)





### Come introdurre username e password?

- la sicurezza effettiva dipende dalla URI del metodo usato per inviare username e password al server
- tecnicamente, non importa la sicurezza della pagina in cui si introducono i dati
- psicologicamente è importante la sicurezza della pagina in cui si introducono i dati perché pochi utenti hanno le conoscenze tecniche necessarie a verificare la URI del metodo usato per l'invio

### S-HTTP

- nuova versione del protocollo HTTP-1.0 sviluppato da EIT-Terisa
- incapsula comandi e risposte HTTP in una busta sicura (PEM, PGP o PKCS-7)
- negoziazione chiavi: in-line, OOB o con Kerberos
- certificati: X.509 o PKCS-6
- firme: RSA o DSA
- digest: MD2, MD5 o SHA-1
- crittografia: DES, IDEA, RC2, RC4
- RFC-2660 "The Secure HTTP"
- RFC-2659 "Security extensions for HTML"

### Sistemi di pagamento elettronico

- fallimento della moneta elettronica per problemi tecnici e normativi (es. bancarotta di DigiCash)
- attualmente il metodo più usato è trasmissione del numero di carta di credito su canale SSL ...
- ... che però non garantisce contro le frodi: VISA Europa dichiara che il 50% dei tentativi di frode nascono da transazioni Internet, che però sono solo il 2% del suo volume d'affari!

### Sicurezza delle transazioni basate su carta di credito

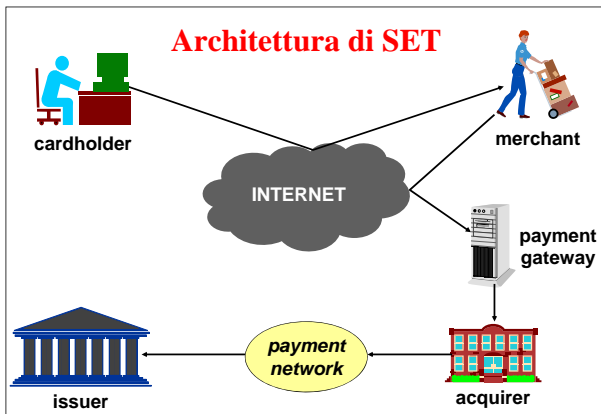
- STT (Secure Transaction Technology)  
VISA + Microsoft
- SEPP (Secure Electronic Payment Protocol)  
Mastercard, IBM, Netscape, GTE, CyberCash
- SET = STT + SEPP  
(Secure Electronic Transaction)

### SET

- SET non è un sistema di pagamento ma un insieme di protocolli per usare in rete aperta in modo sicuro un'infrastruttura esistente basata su carte di credito
- usa certificati X.509v3 dedicati esclusivamente alle transazioni SET
- assicura la privacy degli utenti perché mostra a ciascuna parte solo i dati che le competono

### Caratteristiche di SET

- versione 1.0 (maggio 1997)
- digest: SHA-1
- crittografia simmetrica: DES
- scambio chiavi: RSA
- firma digitale: RSA con SHA-1
- [www.setco.org](http://www.setco.org)



### Attori in SET (I)

- **cardholder**  
legittimo possessore di una carta di credito aderente a SET
- **merchant**  
venditore di un prodotto via Internet (Web o e-mail)
- **issuer**  
istituto finanziario che ha emesso la carta di credito dell'utente

### Attori in SET (II)

- **acquirer**  
organizzazione finanziaria che stabilisce un rapporto col mercante e lo interfaccia con uno o più circuiti di pagamento
- **payment gateway**  
sistema che traduce transazioni SET nel formato richiesto dal circuito di pagamento dell'acquirer
- **certification authority**  
emette certificati X.509v3 e CRL X.509v2 per tutti gli altri attori di SET

### Doppia firma SET

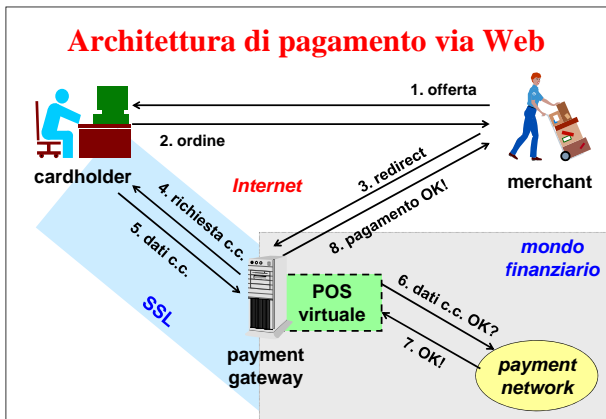
- per garantire privacy all'utente nei confronti di merchant e del sistema finanziario (acquirer + issuer) si usa una **doppia firma**
- al merchant non vengono fatti sapere gli estremi del pagamento
- alla banca non viene fatto sapere quale merce si è acquistata
- solo l'utente può dimostrare l'associazione tra merce e pagamento

### Doppia firma SET: dettagli

- **PI**: Purchase Information (pagamento)
- **OI**: Order Information (merce)
- $DS = E ( H ( H(PI), H(OI) ), Ukpri)$
- **OI+DS+H(PI)** al merchant
- merchant conosce OI e può calcolare  $H(H(PI), H(OI))$  verificando che corrisponda al valore estratto dalla firma
- **PI+DS+H(OI)** all'acquirer
- acquirer conosce PI e può calcolare  $H(H(PI), H(OI))$  verificando che corrisponda al valore estratto dalla firma

### Problemi di SET

- software molto caro (sia per le CA, sia per il merchant e l'acquirer)
- necessita di un'applicazione speciale dal lato utente (SET wallet)
- procedura di rilascio del certificato a chiave pubblica per gli utenti complessa
- in sviluppo la versione 2.0 di SET che farà a meno del wallet (userà un browser)



### Pagamento con carta di credito via Web

- **ipotesi base:**
  - l'acquirente possiede una carta di credito
  - l'acquirente ha un browser con SSL
- **conseguenze:**
  - la sicurezza effettiva dipende dalla configurazione sia del server sia del client
  - il payment gateway ha tutte le informazioni (pagamento + merce) mentre il merchant ha solo le informazioni sulla merce

### PCI DSS

- **Payment Card Industry Data Security Standard**
- oggi richiesto da tutte le carte di credito per transazioni Internet
- molto più prescrittivo rispetto ad altre norme di sicurezza (es. HIPAA = Health Insurance Portability and Accountability Act)
- <https://www.pcisecuritystandards.org>

### Requisiti PCI DSS (I)

- **costruire e mantenere una rete protetta:**
  - R1 = installare e mantenere una configurazione con firewall per proteggere i dati dei titolari delle carte
  - R2 = non usare password di sistema predefinite o altri parametri di sicurezza impostati dai fornitori
- **proteggere i dati dei titolari delle carte:**
  - R3 = proteggere i dati dei titolari delle carte memorizzati
  - R4 = cifrare i dati dei titolari delle carte quando trasmessi attraverso reti pubbliche aperte

### Requisiti PCI DSS (II)

- **rispettare un programma per la gestione delle vulnerabilità**
  - R5 = usare e aggiornare con regolarità l'antivirus
  - R6 = sviluppare e mantenere applicazioni e sistemi protetti
- **implementare misure forti per il controllo accessi**
  - R7 = limitare l'accesso ai dati dei titolari delle carte solo se effettivamente indispensabili per lo svolgimento dell'attività commerciale
  - R8 = dare un ID univoco ad ogni utente del SI
  - R9 = limitare la possibilità di accesso fisico ai dati dei titolari delle carte

### Requisiti PCI DSS (III)

- **monitorare e testare le reti con regolarità**
  - R10 = monitorare e tenere traccia di tutti gli accessi effettuati alle risorse della rete e ai dati dei titolari delle carte
  - R11 = eseguire test periodici dei processi e dei sistemi di protezione
- **adottare una Politica di Sicurezza**
  - R12 = adottare una Politica di Sicurezza