# Security of network applications

**Antonio Lioy**

**< lioy @ polito.it >**

*Politecnico di Torino*

*Dip. Automatica e Informatica*

# Standard situation

- **weak authentication:**
  - username and password
    - problem: password snooping
  - IP address (server web; R commands = rsh, rlogin, rcp, ...)
    - problem: IP spoofing
- **other frequent problems:**
  - data snooping / forging
  - shadow server / MITM
  - replay, filtering

# Channel security

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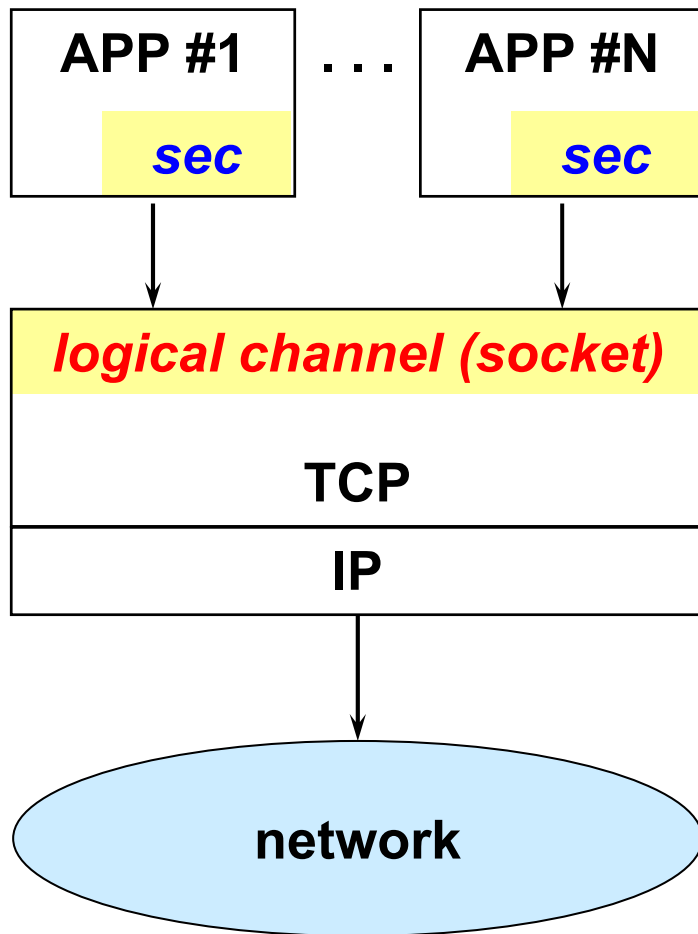- **requires no (or small) modification of applications**



. . . john 123456  Hi dad! . . .

john
123456

john
123456

# Message / data security

- **authentication (single), integrity and privacy self-contained in the message**

- **possibility of non repudiation**

- **requires modification of applications**
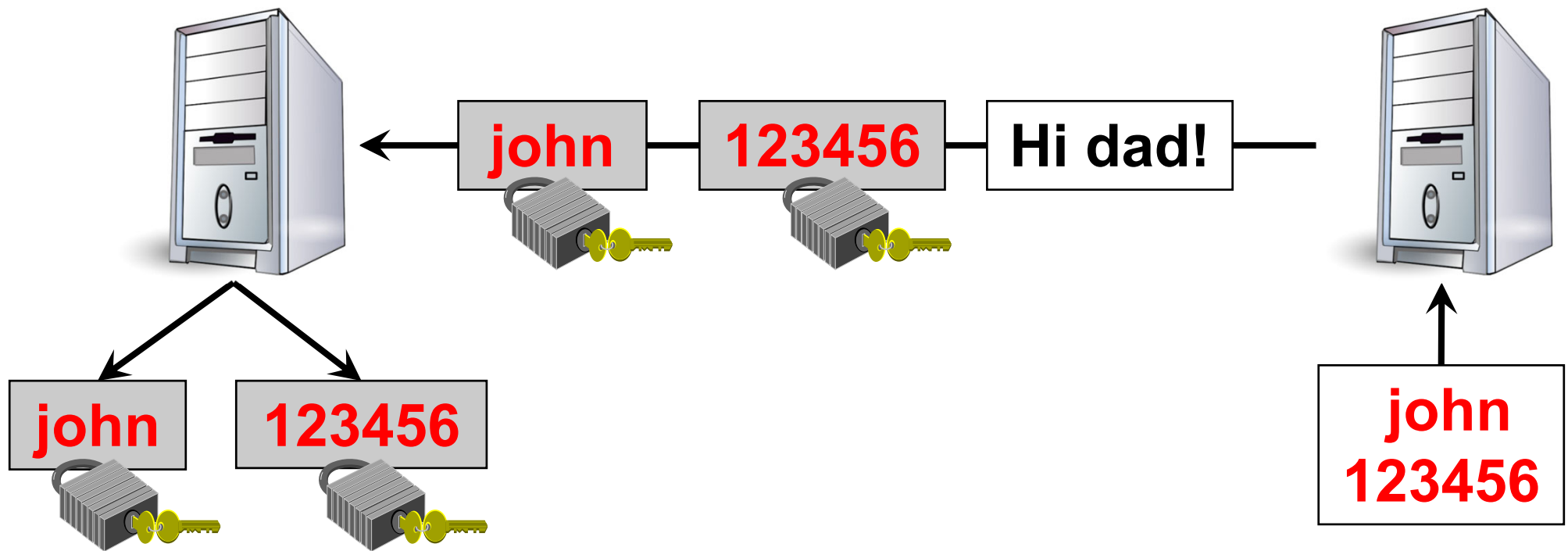
# Security internal to applications

```
APP #1  . . .  APP #N
   sec           sec
```

logical channel (socket)

TCP

IP

network

- 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

APP #1 . . . APP #N

*sec*

*secure logical channel*

*logical channel (socket)*

**TCP**

**IP**

**network**
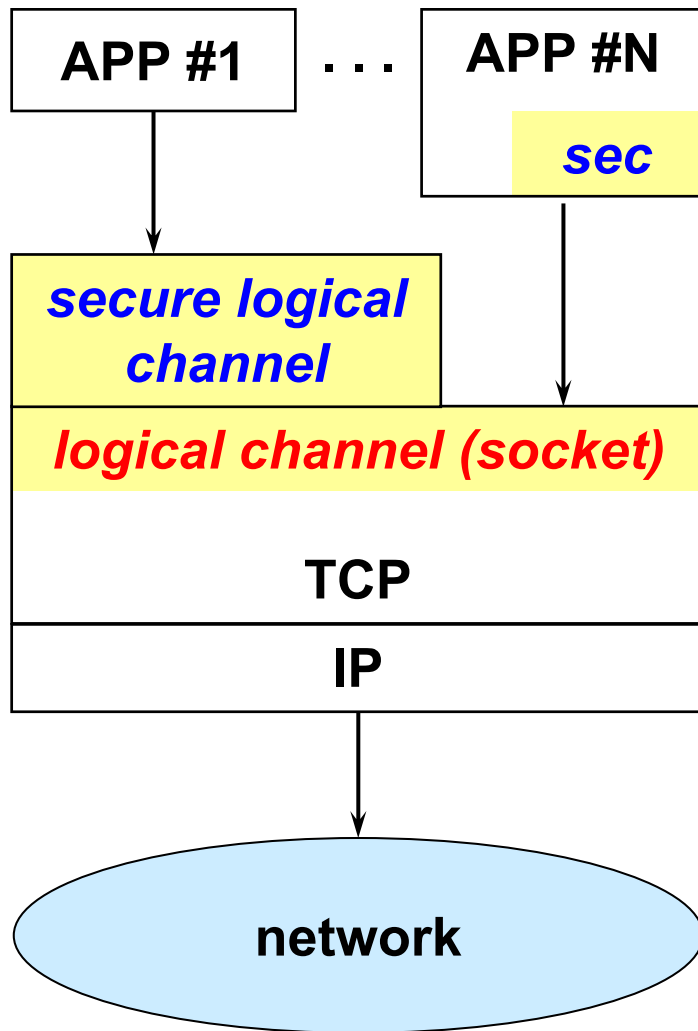
- **the session level would be the ideal one to be used to implement many security functions**

- **… 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 TLS
  - one of the few fiascos of MS!

# TLS (was 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
- **SSL-2, SSL-3, then TLS (Transport layer Security) … TLS-1.0, TLS-1.1, TLS-1.2, TLS-1.3**
- **everything below TLS-1.2 is insecure and deprecated**

# Official ports for SSL/TLS 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
```

# TLS – 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 implicit asymmetric challenge
  - the client authentication (with public key, X.509 certificate, and explicit challenge) is optional
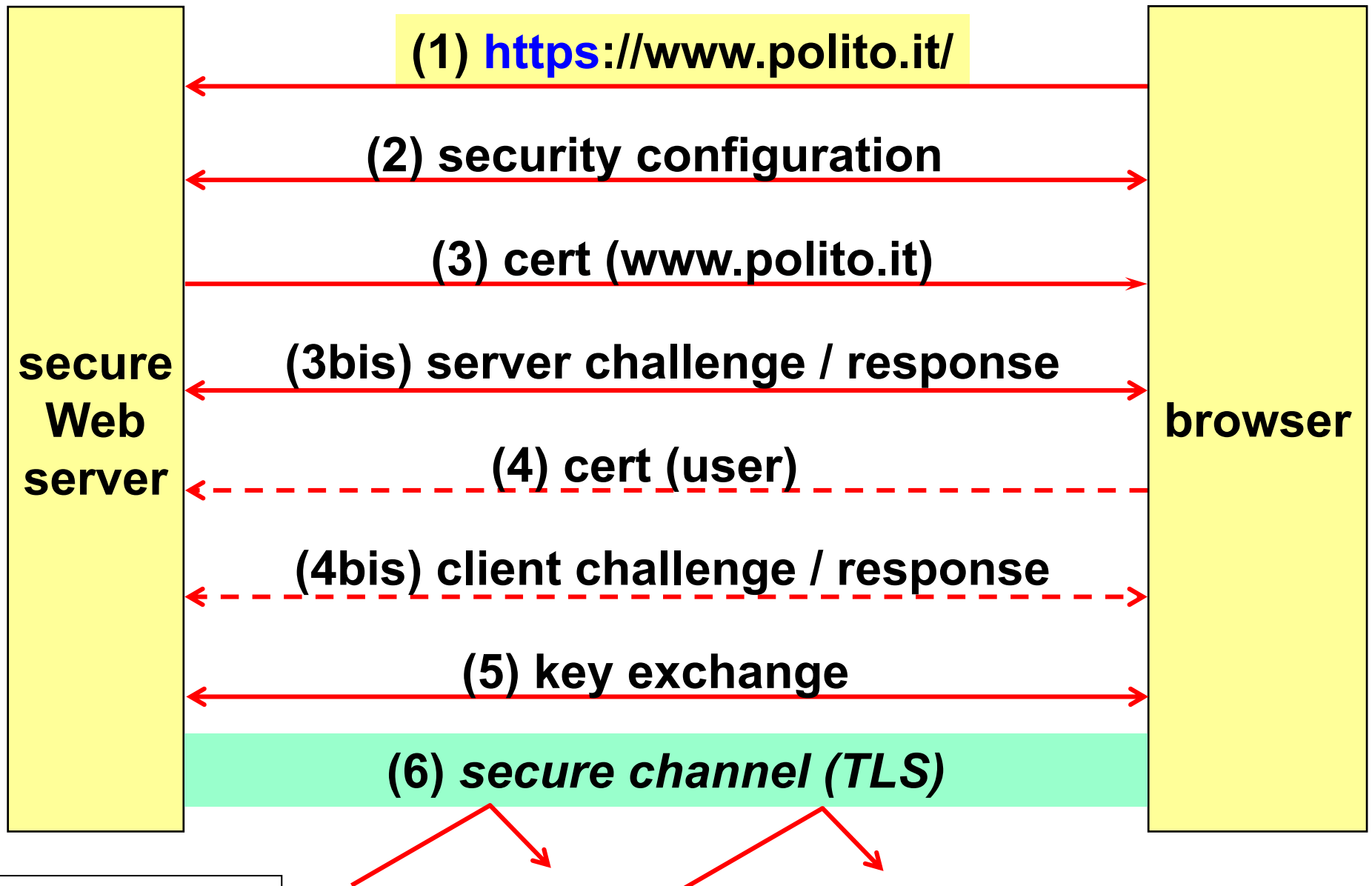- **for authentication and integrity of the data exchanged over the channel the protocol uses:**
  - a keyed digest (SHA-1 or better)
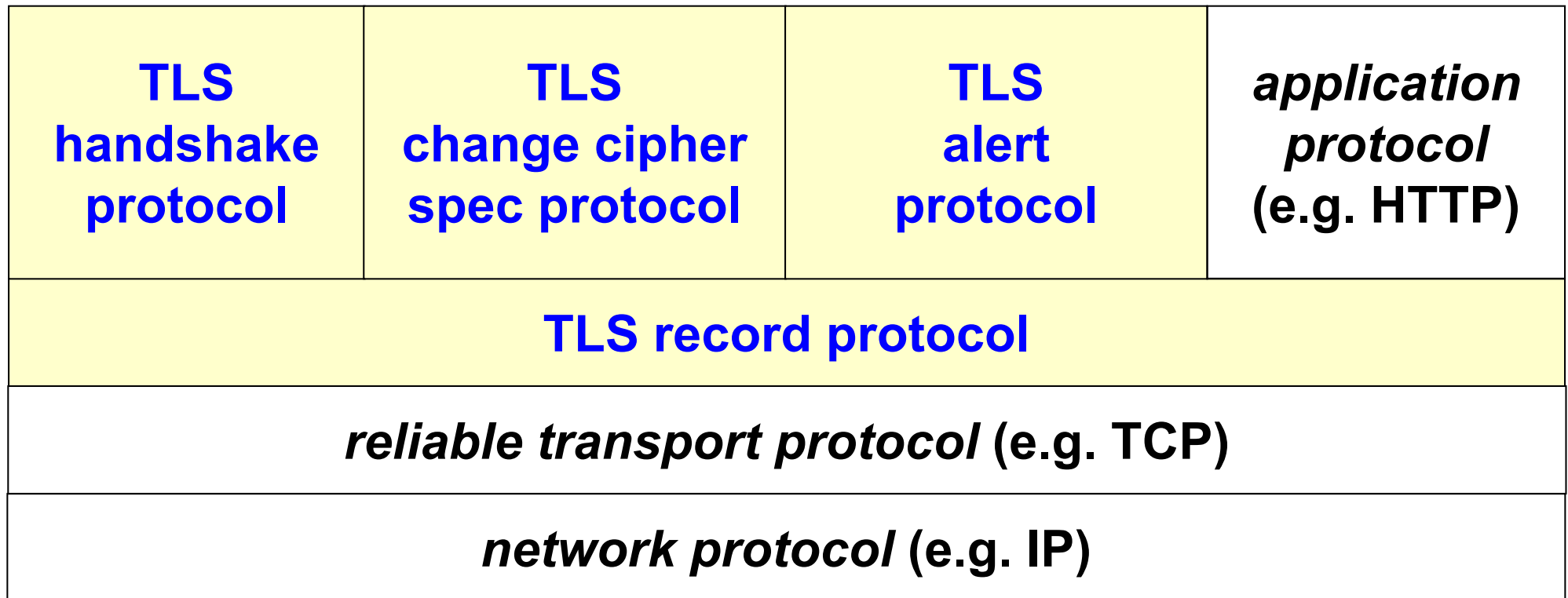  - an implicit MID to avoid replay and cancellation

# TLS - confidentiality

- **the client generates a session key used for symmetric encryption of data (RC4, 3DES, IDEA, AES, …)**

- **key exchange with the server occurs via public-key cryptography (RSA, Diffie-Hellman, or Fortezza-KEA)**

- **since TLS-1.2 authenticated encryption is also available**

# TLS handshake (simplified view)

**secure Web server**

**browser**

(1) **https**://www.polito.it/

(2) security configuration

(3) cert (www.polito.it)

(3bis) server challenge / response

(4) cert (user)

(4bis) client challenge / response

(5) key exchange

(6) *secure channel (TLS)*

# TLS architecture

| TLS handshake protocol | TLS change cipher spec protocol | TLS alert protocol | *application protocol* (e.g. HTTP) |
|---|---|---|---|
| TLS record protocol | | | |

*reliable transport protocol* (e.g. TCP)

*network protocol* (e.g. IP)

# TLS session-id

**Typical 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 TLS cryptographic parameters must be negotiated every time, then the computational load becomes high.**

# TLS session-id

- **in order to avoid re-negotiation of the cryptographic parameters for each TLS connection, the TLS server can send a session identifier (that is, more connections can be part of the same logical session)**

- **if the client, when opening the TLS 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)**

# TLS handshake with session-ID

**secure Web server**

**browser**

(1) **https**://www.polito.it/

(1bis) session-ID
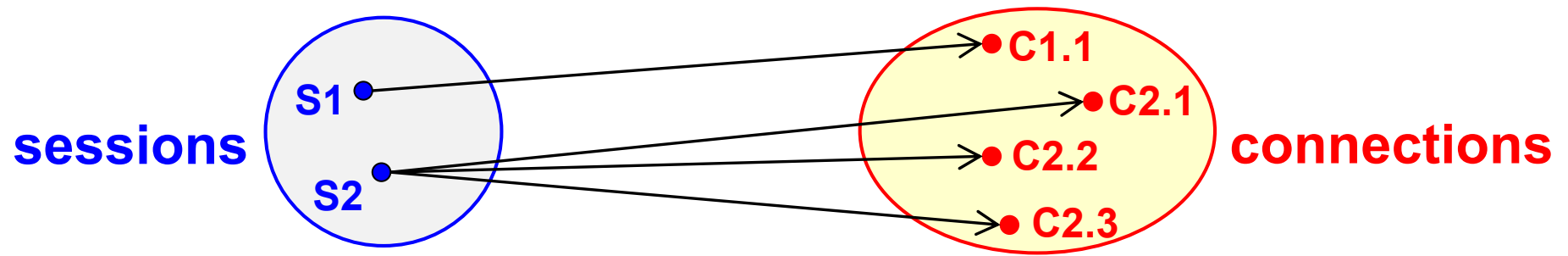
(5) *secure channel (TLS)*
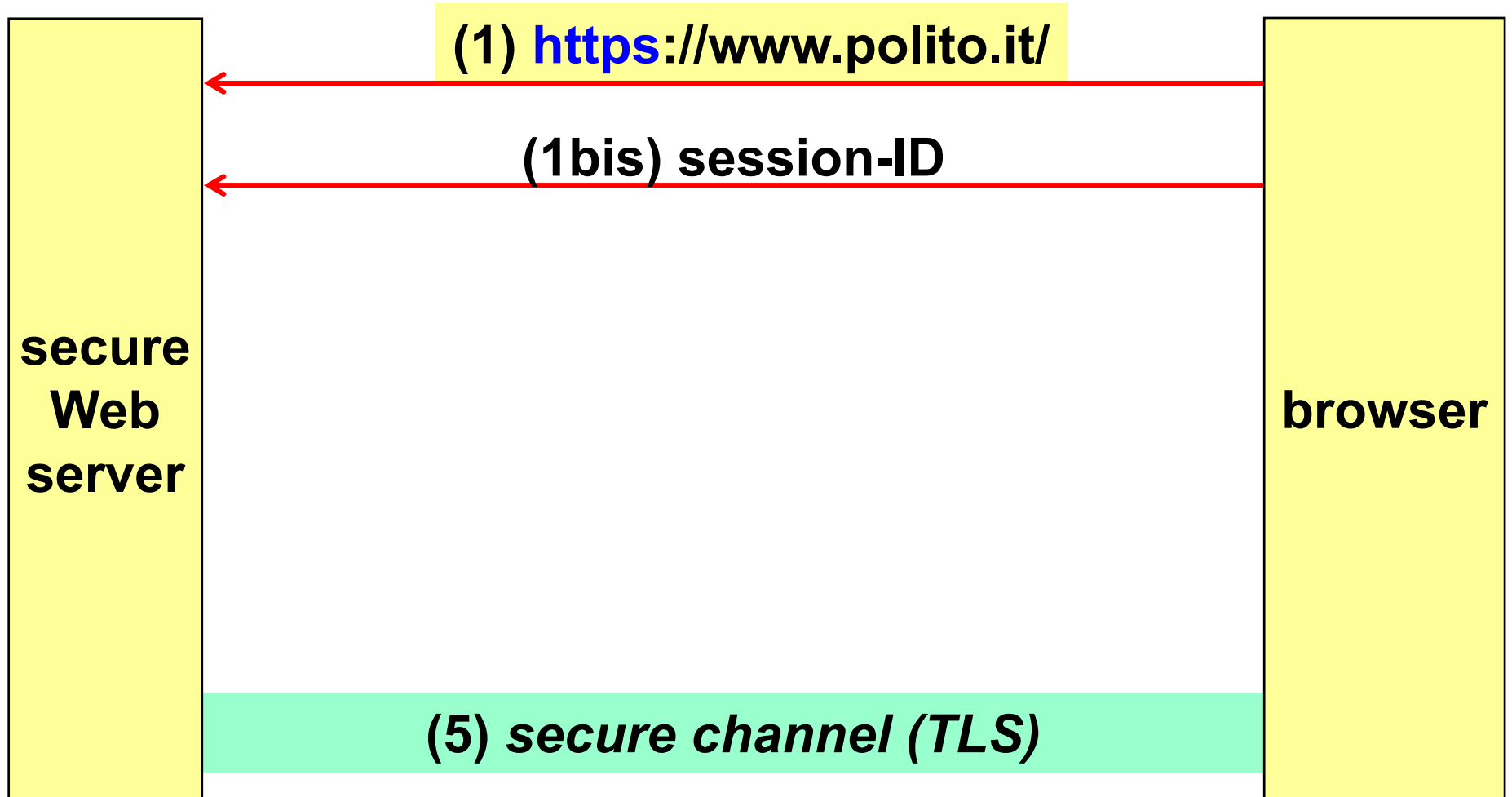
# TLS sessions and connections
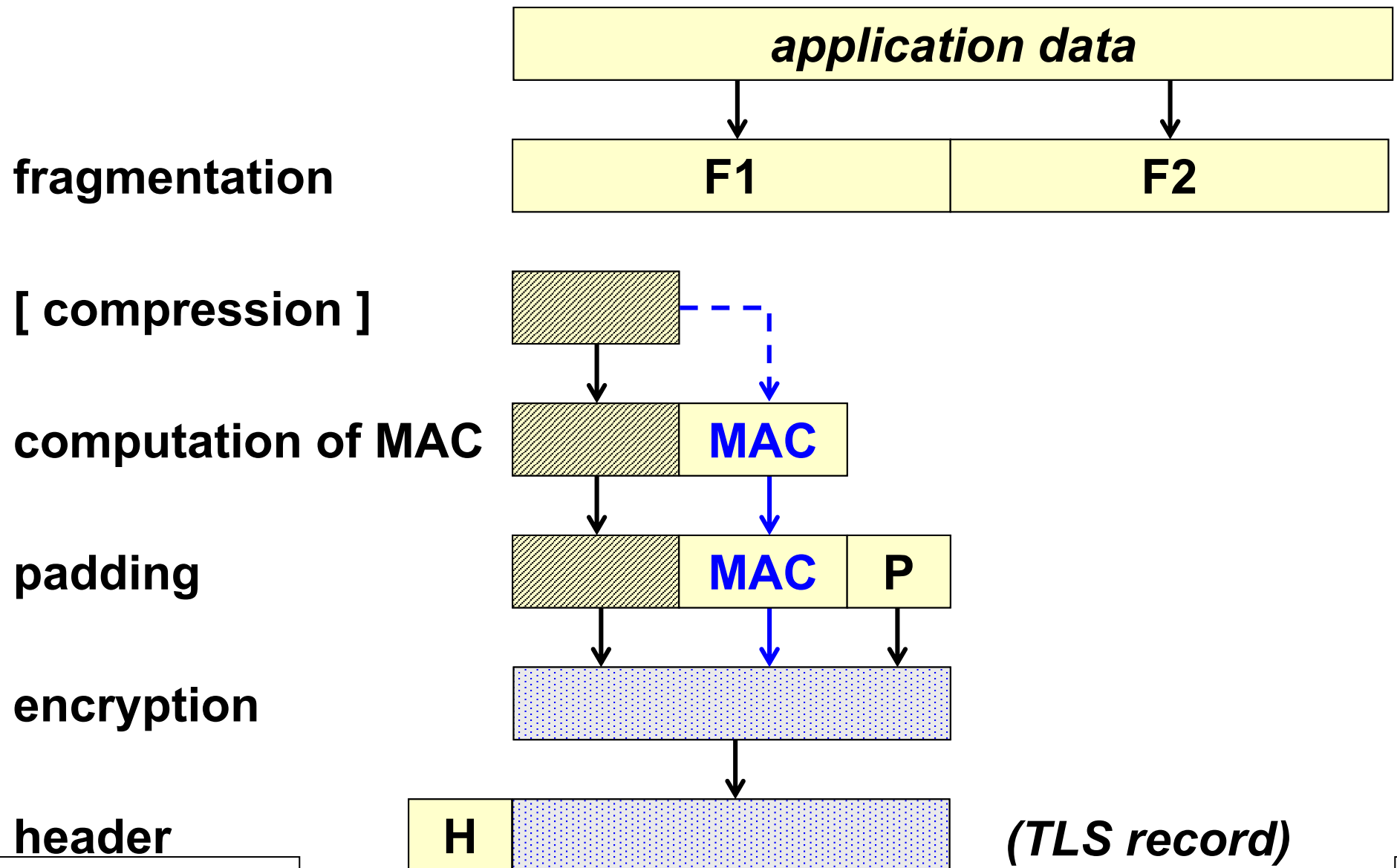
- **TLS session**
  - a logical association between client and server
  - created by the Handshake Protocol
  - defines a set of cryptographic parameters
  - is shared by one or more TLS connections (1:N)
- **TLS connection**
  - a transient TLS channel between client and server
  - associated to one specific TLS session (1:1)

sessions     S1   S2     C1.1   C2.1   C2.2   C2.3     connections

# TLS record protocol (authenticate-then-encrypt)

**application data**

fragmentation

| F1 | F2 |

[ compression ]

computation of MAC    MAC

padding    MAC    P

encryption

header    H    *(TLS record)*

# TLS – computation of MAC

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

- **message_digest**
    - depends on the chosen algorithm
- **key**
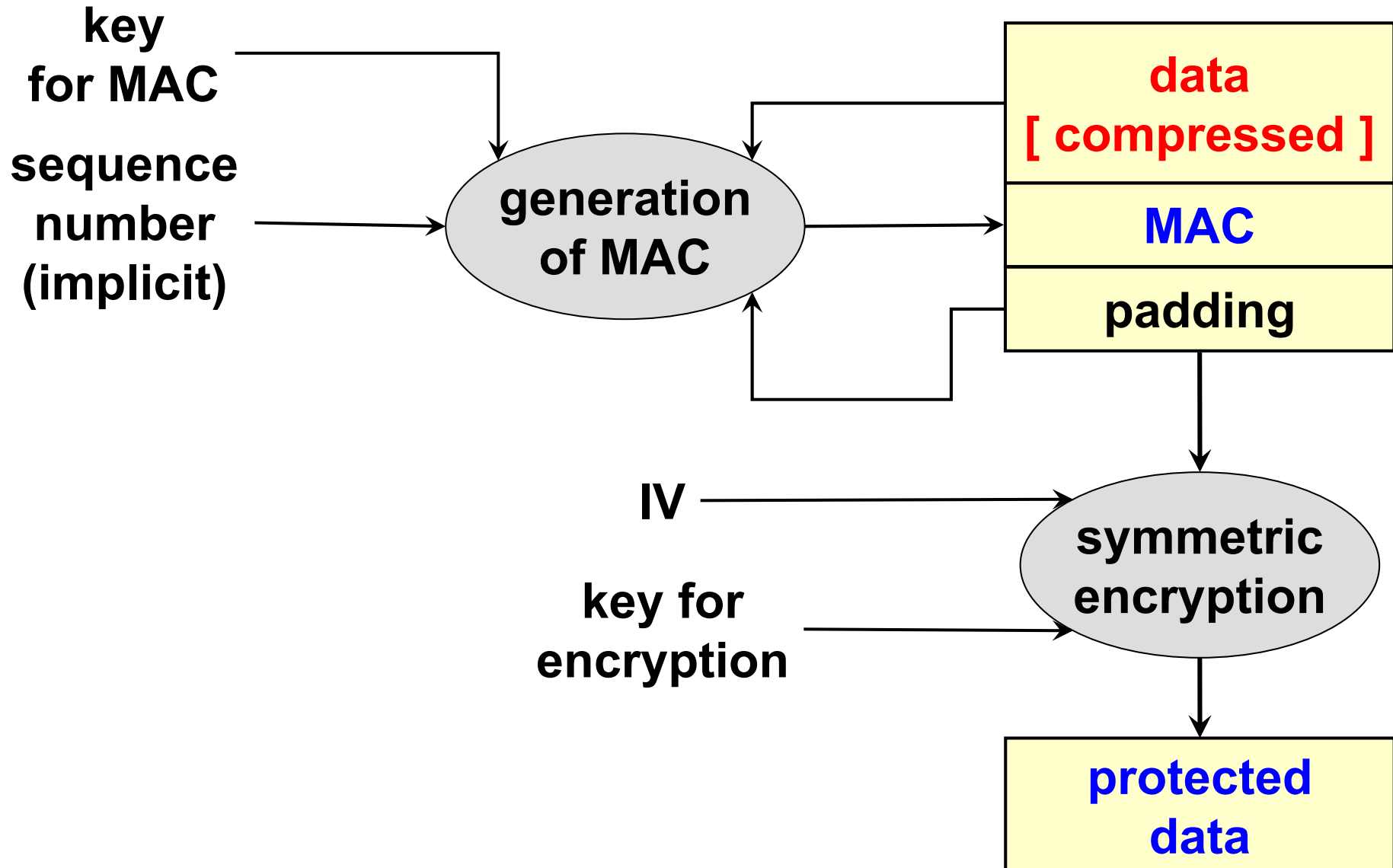    - sender-write-key or receiver-read-key
- **seq_number**
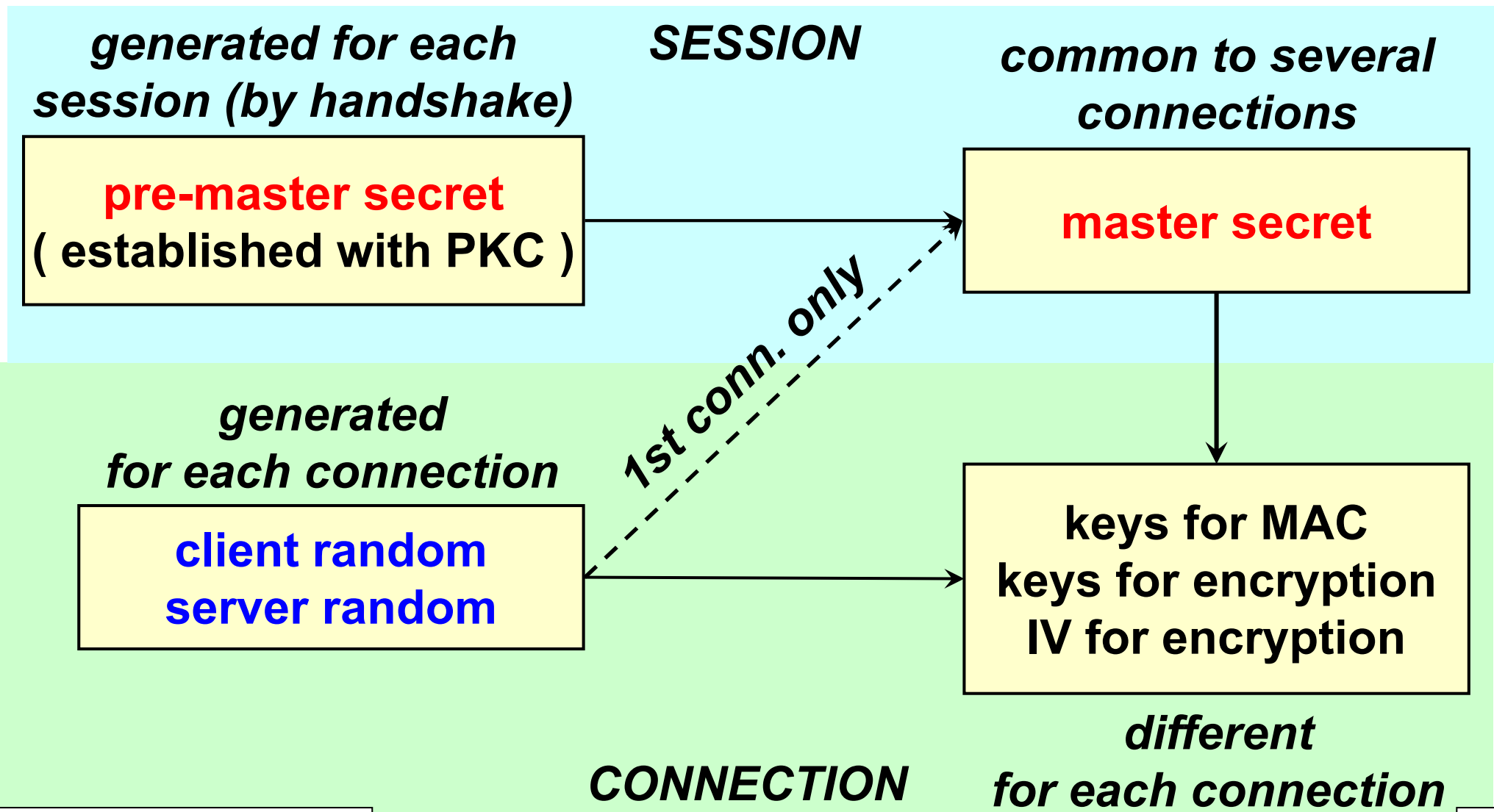    - 64-bit integer (never transmitted but computed implicitly)

# TLS 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**

# Data protection (authenticate-then-encrypt)

# Relationship among keys and sessions (between a server and the same client)

**SESSION**

*generated for each session (by handshake)*

*common to several connections*

| pre-master secret ( established with PKC ) | → | master secret |

*generated for each connection*

| client random server random | → | keys for MAC keys for encryption IV for encryption |

*1st conn. only* (dashed arrow to master secret)

**CONNECTION**

*different for each connection*

# Perfect forward secrecy

- **if a server has a certificate valid for both signature and encryption**

- **... then it can be used both for authentication (via a signature) and key exchange (asymmetric encryption of the session key)**

- **... but if**
    - an attacker copies all the encrypted traffic
    - and later discovers the (long term) private key

- **... then the attacker can decrypt all the traffic, past, present, and future**

- **perfect forward secrecy:**
    - the compromise of a private key compromises only the current (and eventually future) traffic but not the past one

# "Ephemeral " mechanisms

- **one-time key generated on the fly:**

  - for authenticity it must be signed (but cannot have an associated X.509 certificate because the CA process is slow and often not on-line)

  - DH suitable, RSA slow

    - compromise for RSA = re-use N times

- **in this case the server's private key is used only for signing**

- **... so we obtain perfect forward secrecy:**

  - if the (temporary or short-lived) private key is compromised then the attacker can decrypt only the related traffic

  - compromise of the long-term private key is an issue for authentication but not for confidentiality

- **examples: ECDHE**

# TLS 1.3

- **RFC-8446 (August 2018)**
- **only (EC)DHE for key-exchange**
- **only AEAD with modern ciphers (AES, ChaCha20)**
- **handshake redesigned for faster and more secure setup**
- **various other improvements**

# TLS and virtual servers: the problem

- **virtual server (frequent case with web hosting)**
    - different logical names associated to the same IP address
    - e.g. home.myweb.it=1.2.3.4, food.myweb.it=1.2.3.4
- **easy since HTTP/1.1**
    - the client uses the Host header to specify the server it wants to connect to
- **… but difficult in HTTPS**
    - because TLS is activated before HTTP
    - which certificate should be provided? (must contain the server's name)

# TLS and virtual servers: solutions

- **collective (wildcard) certificate**
    - e.g. CN=*.myweb.it
    - private key shared by all servers
    - different treatment by different browsers
- **certificate with a list of servers in subjectAltName**
    - private key shared by all servers
    - need to re-issue the certificate at any addition or cancellation of a server
- **use the SNI (Server Name Indication) extension**
    - in ClientHello (permitted by RFC-4366)
    - limited support by browsers and servers

# ALPN extension (Application–Layer Protocol Negotiation)

- **RFC-7301**

- **application protocol negotiation (for TLS-then-proto) to speed up the connection creation, avoiding additional round-trips for application negotiation**

  - (ClientHello) ALPN=true + list of supported app. protocols

  - (ServerHello) ALPN=true + selected app. protocol

- **important to negotiate HTTP/2 and QUIC**

  - Chrome & Firefox support HTTP/2 only over TLS

- **useful also for those servers that use different certificates for the different application protocols**

- **some possible values: http/1.0, http/1.1, h2, h2c**

# 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

# The TLS downgrade problem (I)

- **client sends (in ClientHello) the highest supported version**

- **server notifies (in ServerHello) the version to be used (highest in common with client)**

- **normal version negotiation:**

  - agreement on TLS-1.2

    - (C > S) 3,3

    - (S > C) 3,3

  - fallback to TLS-1.1 (e.g. no TLS-1.2 at server)

    - (C > S) 3,3

    - (S > C) 3,2

# The TLS downgrade problem (II)

- **(insecure) downgrade:**
    - some servers do not send the correct response, rather they close the connection …
    - then the client has no choice but to try again with a lower protocol version
- **downgrade attack:**
    - attacker sends fake server response, to force repeated downgrade until reaching a vulnerable version (e.g. SSL-3) …
    - then execute a suitable attack (e.g. Poodle)
- **not always an attack (e.g. connection with the server closed due to a network problem)**

# TLS Fallback Signaling Cipher Suite Value (SCSV)

- **RFC-7507**

- **to prevent protocol downgrade attacks**

- **new (dummy) ciphersuite TLS_FALLBACK_SCSV**

  - SHOULD be sent by the client when opening a downgraded connection (as last in ciphersuite list)

- **new fatal Alert value "inappropriate_fallback"**

  - MUST be sent by the server when receiving TLS_FALLBACK_SCSV and a version lower than the highest one supported

  - then the channel is closed and the client should retry with its highest protocol version

# SCSV - notes

- **many servers do not yet support SCSV**

- **… but most servers have fixed their bad behaviour when the client requests a version higher than the supported one**

- **… so browsers can now disable insecure downgrade**

  - Firefox (from 2015) and Chrome (from 2016)

# 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

```
GET /path/to/protected/page/ HTTP/1.0
   HTTP/1.0 401 Unauthorized - authentication failed
   WWW-Authenticate: Basic realm="POLITO - didattica"
Authorization: Basic czEyMzQ1NjpTZWdyZXRpc3NpbWE=
   HTTP/1.0 200 OK
    Server: Apache/1.3
    Content-type: text/html

   <html> ... protected page ... </html>
```

```
$ echo czEyMzQ1NjpTZWdyZXRpc3NpbWE= | openssl enc –a –d
s123456:Segretissima
```

# HTTP digest authentication

- **RFC-2069**
  - technically obsoleted
  - but considered as base case in RFC-2617
- **keyed-digest computation:**
  - HA1 = md5 ( A1 ) = md5 ( *user* ":" *realm* ":" *pwd* )
  - HA2 = md5 ( A2 ) = md5 ( *method* ":" *URI* )
  - response = md5 ( HA1 ":" nonce ":" HA2 )
- **server uses a nonce to avoid replay attacks**
- **the authentication server may insert a field "opaque" to transport state informations (e.g. a SAML token) towards the content server**

# HTTP Digest Authentication

```
GET /private/index.html HTTP/1.1
   HTTP/1.0 401 Unauthorized - authentication failed
   WWW-Authenticate: Digest realm="POLITO",
      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
      opaque="5ccc069c403ebaf9f0171e9517f40e41"
Authorization: Digest username="lioy",
   realm="POLITO",
   nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
   uri="/private/index.html",
   response="32a8177578c39b4a5080607453865edf",
   opaque="5ccc069c403ebaf9f0171e9517f40e41"
   HTTP/1.1 200 OK
   Server: NCSA/1.3
   Content-type: text/html
   <HTML> pagina protetta ... </HTML>
```

**pwd=antonio**

# 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**
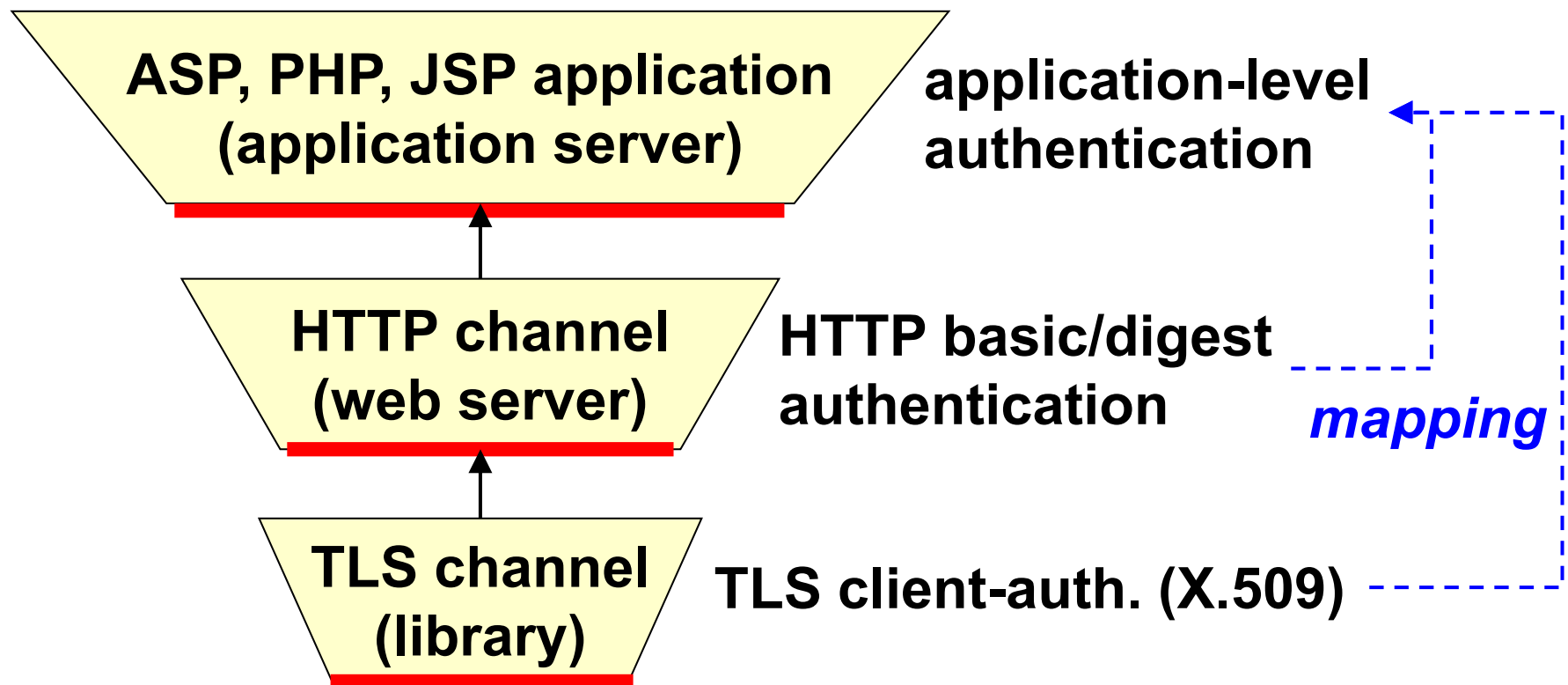- **concepts generally applicable to all protocols:**
  - "SSL/TLS then proto" vs. "proto then TLS"

# TLS 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 HTTP service 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)**



ASP, PHP, JSP application (application server) — application-level authentication

HTTP channel (web server) — HTTP basic/digest authentication

TLS channel (library) — TLS client-auth. (X.509)

*mapping*

# What about forms requesting user/pwd?

- **technically speaking, it's not important the security of the page containing the form**

  - http://www.ecomm.it/login.html

- **… because the actual security depends on the URI of the method used to send username and password to the server**

  - <form … action="https://www.ecomm.it/login.php">

- **… but psychologically, 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**

# HTTP Strict Transport Security (HSTS)

- **RFC-6797**

- **HTTP server declares that its interaction with UA must only be via HTTPS**

  - prevents protocol downgrade and cookie hijacking

- **valid only in HTTPS response**

- **expiration renewed at every access**

- **may include subdomains (recommended)**

- **may be pre-loaded**

  - dangerous (HTTPS or nothing) and difficult removal

  - preload list maintained by Google and used by many browsers = https://hstspreload.org/

# HSTS – syntax and examples

```
Strict-Transport-Security:

   max-age = <expire-time-in-seconds>

   [ ; includeSubDomains ]

   [ ; preload ]
```

```
$ curl -s -D- https://www.paypal.com/ | fgrep -i strict
strict-transport-security: max-age=63072000
$ curl -s -D- https://accounts.google.com/ | grep -i strict
strict-transport-security: max-age=31536000; includeSubDomains
```

# HTTP Public Key Pinning (HPKP)

- **RFC-7469**

- **HTTPS site specifies the digest of its own public key and/or one or more CAs in its chain (but the root!)**

- **UA caches this key and refuses connecting to a site with a different key**

  - it's a TOFU (Trust On First Use) technique

  - dangerous when losing control of the key

  - problems with key updates

    - always include at least one backup key

- **URI to report violations**

- **enforcing or report-only mode**

# HPKP – syntax and examples

```
Public-Key-Pins:
   pin-sha256 = " <base64-sha256-of-public-key> ";
   max-age = <expireTime-in-seconds>
   [; includeSubDomains]
   [; report-uri = " <reportURI> "]
```

```
Public-Key-Pins-Report-Only:
   pin-sha256 = " <base64-sha256-of-public-key> ";
   max-age = <expireTime-in-seconds>
   [; includeSubDomains]
   [; report-uri = " <reportURI> "]
```

```
$ $ curl -s -D - https://scotthelme.co.uk/
    | fgrep -i public-key
```

public-key-pins:
pin-sha256="9dNiZZueNZmyaf3pTkXxDgOzLkjKvI+Nza0ACF5IDwg=";
pin-sha256="X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="V+J+7lHvE6X0pqGKVqLtxuvk+0f+xowyr3obtq8tbSw=";
pin-sha256="9lBW+k9EF6yyG9413/fPiHhQy5Ok4UI5sBpBTuOaa/U=";
pin-sha256="ipMu2Xu72A086/35thucbjLfrPaSjuw4HIjSWsxqkb8=";
pin-sha256="+5JdLySIa9rS6xJM+2KHN9CatGKln78GjnDpf4WmI3g=";
pin-sha256="MWfCxyqG2b5RBmYFQuLllhQvYZ3mjZghXTRn9BL9q10=";
includeSubDomains; max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
    enforce"

public-key-pins-report-only:
pin-sha256="X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="Vjs8r4z+80wjNcr1YKepWQboSIRi63WsWXhIMN+eWys=";
pin-sha256="IQBnNBEiFuhj+8x6X8XLgh01V9Ic5/V3IRQLNFFc7v4=";
max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
    reportOnly"

# E-payment systems

- **failure of the digital cash, for technical and political problems (e.g. the DigiCash failure)**

- **failure of a dedicated payment protocol (SET, Secure Electronic Transactions) due to technical and organizational problems**

- **currently the most widely used approach is transmitting a credit card number over a TLS channel ...**

- **... but this is no guarantee against fraud: in the past VISA Europe declared that Internet transactions generate about 50% of the fraud attempts, although they are just 2% of its total transaction amount!**

# A web-based payment architecture
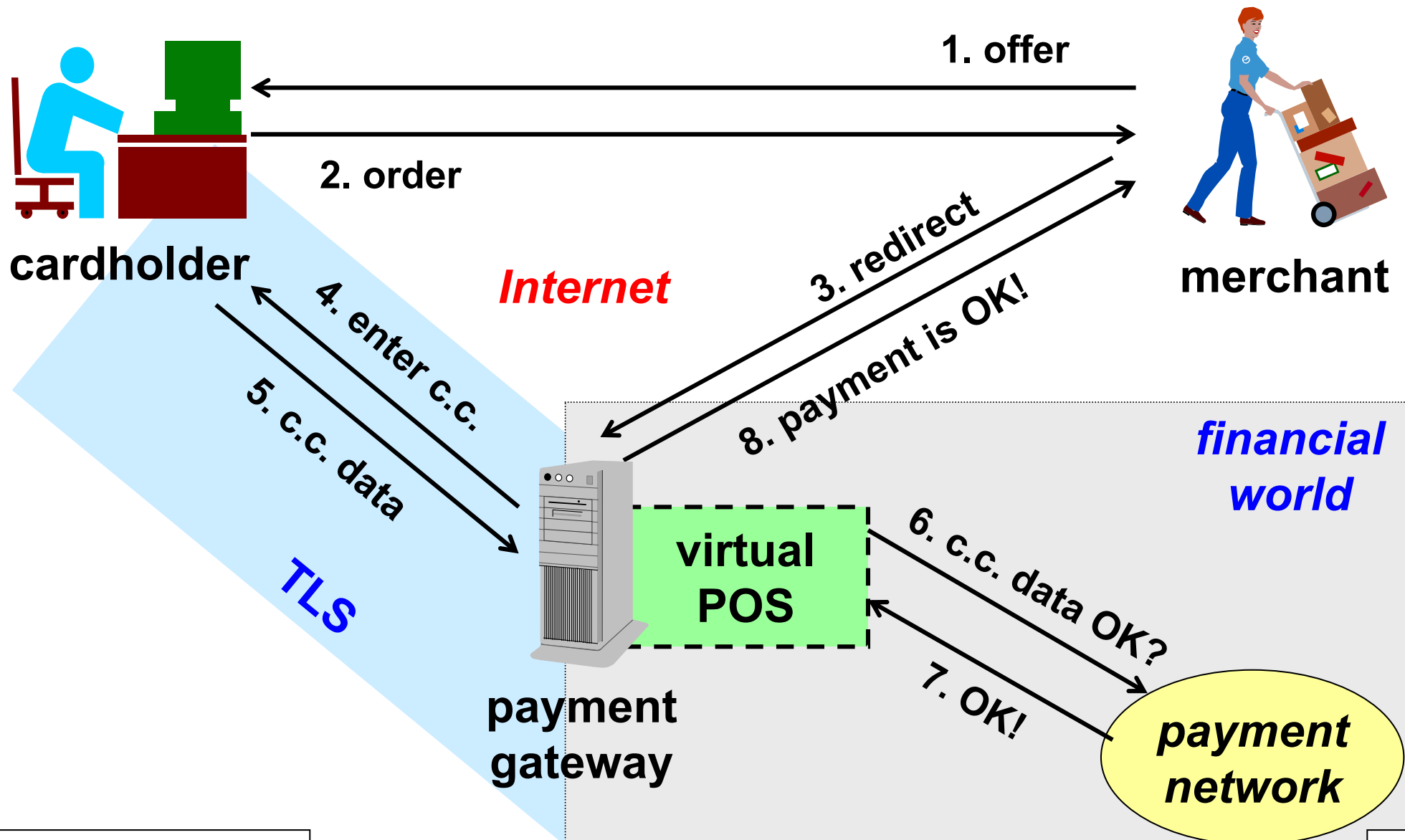
# Web-based payment architecture

- **baseline:**
  - the buyer owns a credit card
  - the buyer has a TLS-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**

- **very detailed technical prescriptions compared to other security standards (e.g. HIPAA = Health Insurance Portability and Accountability Act)**

- **https://www.pcisecuritystandards.org**

  - v2.0 = oct 2010

  - v3.0 = nov 2013

  - v3.1 = apr 2015 (no TLS or "old" TLS)

  - v3.2 = apr 2016 (MFA, test functions/procedures, describe architecture, …)

  - v3.2.1 = may 2018 (minor updates and clarifications)

# 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