



# SICUREZZA AL LIVELLO TRASPORTO: “SECURE SOCKET LAYER” – SSL

---

Corso di Laurea Magistrale in Ingegneria Informatica

A.A. 2015/2016

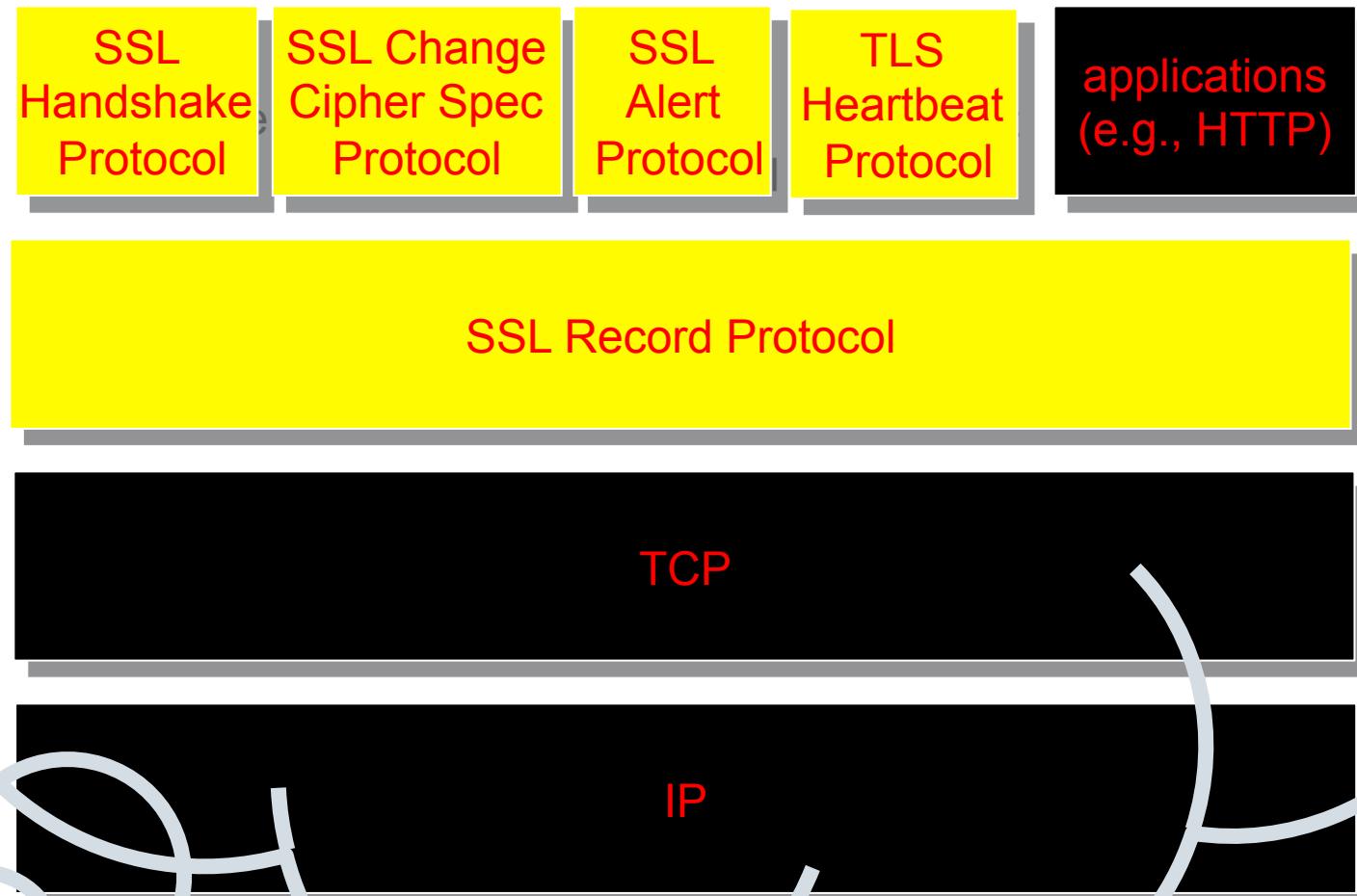
Prof. Simon Pietro Romano

[sromano@unina.it](mailto:sromano@unina.it)

---



## L'ARCHITETTURA SSL (DÉJÀ VU)





# SSL E PROTOCOLLI: SINTESI (DÉJÀ VU)

1 byte



(a) Change Cipher Spec Protocol

1 byte

3 bytes

 $\geq 0$  bytes

(c) Handshake Protocol

1 byte



(b) Alert Protocol

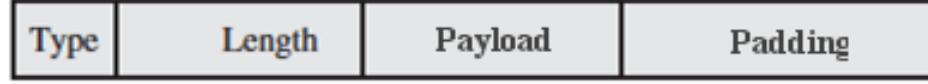
 $\geq 1$  byte

Opaque content

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

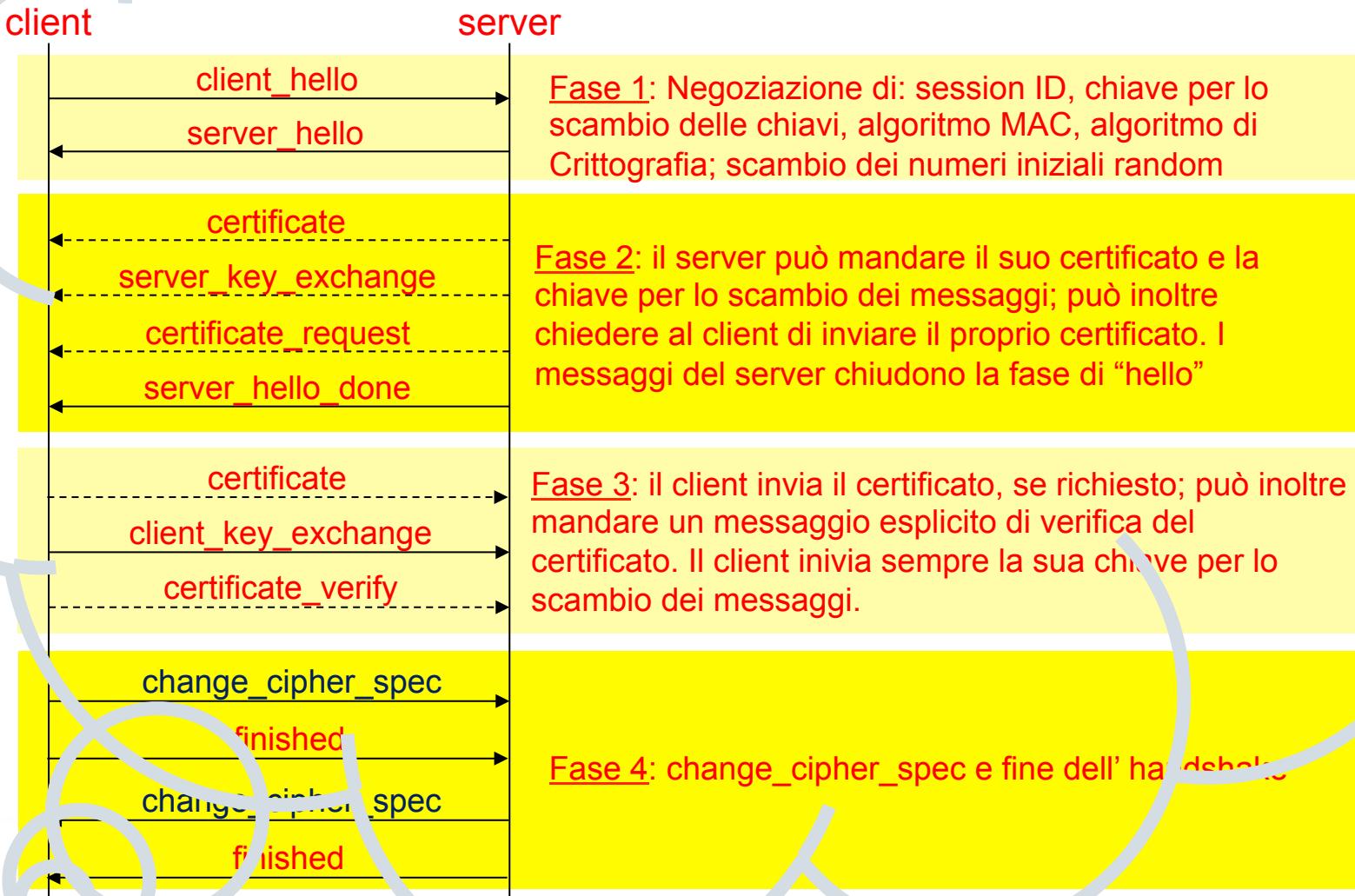
1 byte

2 bytes

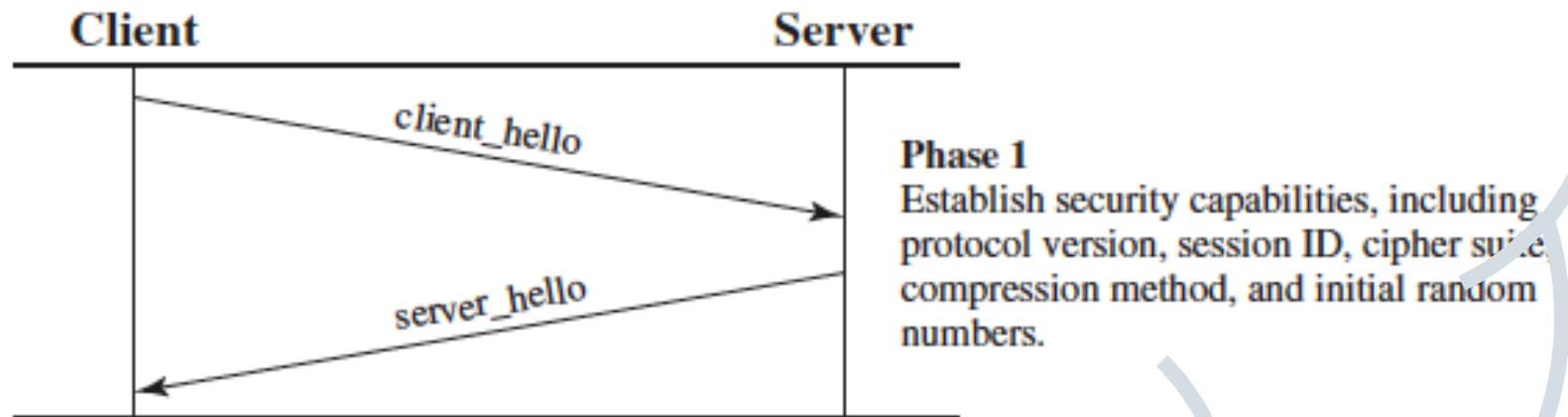
 $\geq 0$  bytes $\geq 16$  bytes

(e) Heartbeat Protocol

## IL PROTOCOLLO DI HANDSHAKE IN AZIONE (DÉJÀ VU)

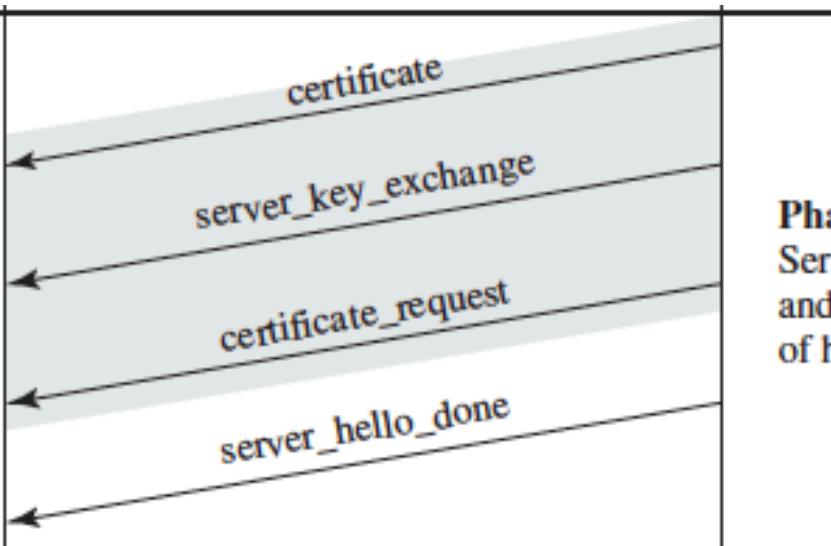


## IL PROTOCOLLO DI HANDSHAKE: FASE 1 (DÉJÀ VU)





## IL PROTOCOLLO DI HANDSHAKE: FASE 2 (DÉJÀ VU)



### Phase 2

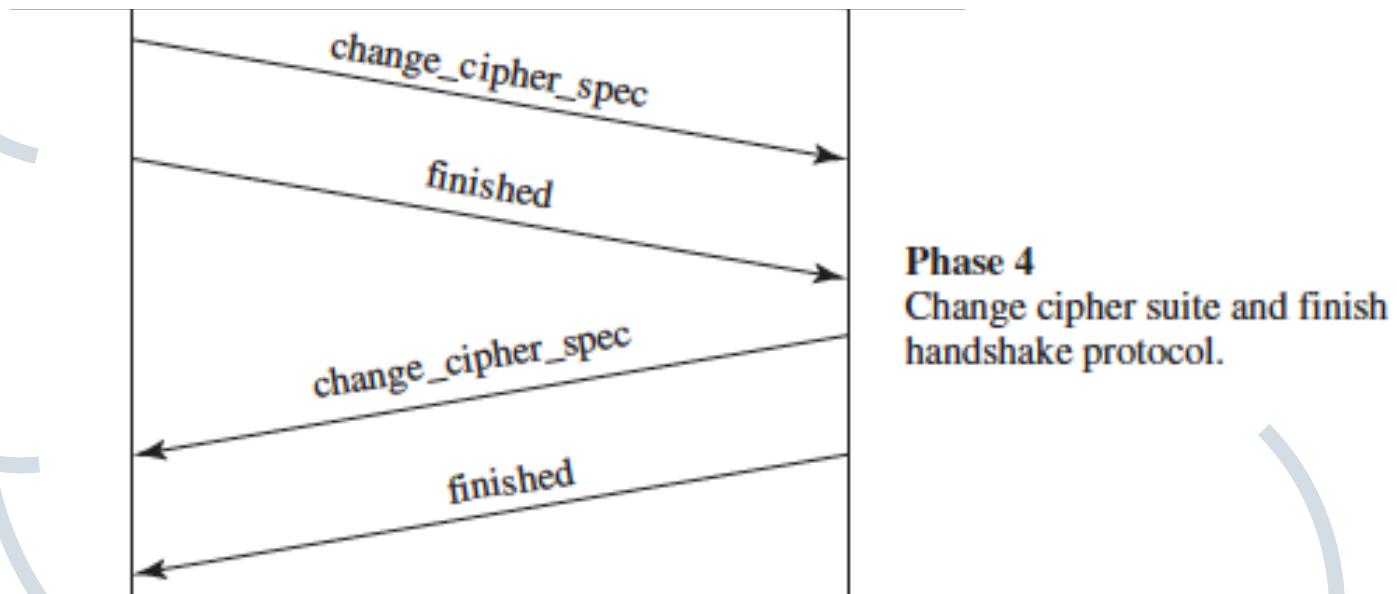
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

## IL PROTOCOLLO DI HANDSHAKE: FASE 3 (DÉJÀ VU)

*certificate*  
*client\_key\_exchange*  
*certificate\_verify*

**Phase 3**  
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

## IL PROTOCOLLO DI HANDSHAKE: FASE 4 (DÉJÀ VU)





# CREAZIONE DEL MASTER SECRET (1/2)

- “master secret”:
  - valore monouso di 48 byte
  - generato per una sessione tramite scambio di chiavi sicuro
- La creazione si svolge in due fasi
  - Scambio di “*pre\_master\_secret*”
  - Calcolo del valore “*master\_secret*” da entrambe le parti
- Per lo scambio del *pre\_master\_secret* ci sono due alternative
  - RSA:
    - il client genera un valore di *pre\_master\_secret* crittografato con la chiave RSA pubblica del server
  - Diffie-Hellman:
    - il client e il server generano un chiave pubblica D-H
    - Dopo essersi scambiati questa chiave ognuno esegue il calcolo D-H per creare il valore del *pre\_master\_secret* condiviso



## CREAZIONE DEL MASTER SECRET (2/2)

- Il valore del *master\_secret* viene calcolato applicando la seguente funzione:

```
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 ) )
```

- ...dove *client\_random* e *server\_random* sono i valori di “nonce” scambiati all'inizio



# Generazione parametri crittografici

- Per generare i segreti MAC di scrittura, le chiavi di scrittura e l'IV di scrittura sia per il client che per il server si utilizza il valore del segreto master secondo la seguente funzione hash:

*Key\_block =*

```
MD5(master_secret || SHA( "A" || master_secret || client_random || server_random) )
|| MD5(master_secret || SHA( "BB" || master_secret || client_random || server_random) )
|| MD5(master_secret || SHA("CCC" || master_secret || client_random || server_random) )
|| ...
```

- ...fino a generare un output di dimensioni sufficienti
- In effetti, possiamo interpretare il “master\_secret” come il seme del generatore pseudo-casuale sopra descritto!



# TLS E SSL (1/3)

- Numero di versione
  - TLS standard 1.0 usa nel campo version major.minor 3.1
  - Il numero di versione corrente 3.3 indica standard TLS 1.2 (TLS1.3 è Draft ad Ottobre 2015)
- MAC
  - TLS utilizza HMAC (keyed-Hash Message Authentication Code)
- TLS definisce più codici di alert
- Suite crittografiche
  - TLS non utilizza lo scambio delle chiavi Fortezza e l'algoritmo di crittografia Fortezza
- Messaggio “certificate\_verify”
  - I codici hash sono calcolati soltanto sul messaggio “handshake\_message”
  - In SSL i calcoli hash includono anche il valore del “master secret” e i bit di riempimento



## TLS E SSL (2/3)

- TLS utilizza una funzione pseudocasuale chiamata PRF per espandere i valori segreti in blocchi di dati per la generazione o la convalida della chiave
- Si basa sulla funzione di espansione cosiddetta “P\_hash”:

$$\begin{aligned} P\_hash(secret, seed) = \\ HMAC\_hash(secret, A(1) \parallel seed) \parallel \\ HMAC\_hash(secret, A(2) \parallel seed) \parallel \\ HMAC\_hash(secret, A(3) \parallel seed) \parallel \dots \end{aligned}$$

dove:

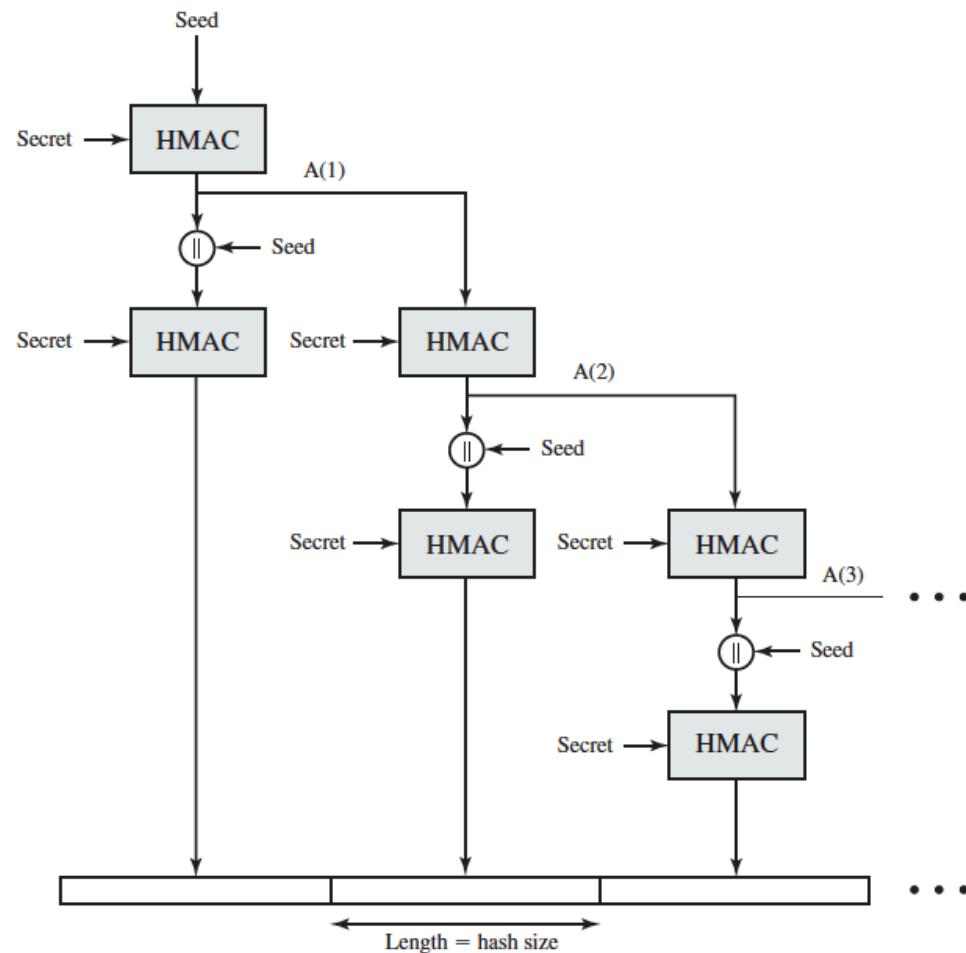
$$\begin{aligned} A(0) &= seme \\ A(i) &= HMAC\_hash(secret, A(i-1)) \end{aligned}$$

- In particolare:

$$\begin{aligned} PRF(secret, label, seed) = \\ P\_MD5(secret\_left, label \parallel seed) \oplus P\_SHA(secret\_right, label \parallel seed) \end{aligned}$$



# LA FUNZIONE P\_HASH





## TLS E SSL (3/3)

- Messaggio “finished”

$$\text{PRF}(\text{master\_secret}, \text{"client finished"}, \text{MD5}(\text{handshake\_messages}) \parallel \text{SHA}(\text{handshake\_messages}))$$

- Calcoli crittografici

- Il “pre\_master\_secret” è calcolato nello stesso modo di SSL
- Il calcolo del “master\_secret” utilizza la seguente formula:

$$\text{PRF}(\text{pre\_master\_secret}, \text{"master secret"}, \text{client\_random} \parallel \text{server\_random})$$

- Il calcolo del “key-block” utilizza la seguente formula:

$$\text{PRF}(\text{master\_secret}, \text{"key expansion"}, \text{server\_random} \parallel \text{client\_random})$$

- Riempimento

- In TLS sono consentiti riempimenti di lunghezza variabile, fino ad un massimo di 255 byte



## TLS 1.0 -> 1.1 -- RFC4346 (2006)

- I.V.隐式现在是显式 (CBC 攻击防护)
- 错误的 padding 通过“bad\_record\_mac”警报而不是“decryption\_failed”传达
- 定义 IANA 的注册表参数协议
- 早期关闭不再意味着会话成为非恢复性的
- 关于新 TLS 攻击 (CBC, Timing) 的注释



## TLS 1.1 -> 1.2 – RFC5246 (2008)

- Molte modifiche negli algoritmi di autenticazione (Pseudo-random-function *specificate* nella cipher-suite, ...)
- Cambi nell'insieme di algoritmi di cifratura supportati/obbligatori
- Maggiori restrizioni su alcuni prerequisiti
- Descrizione di difesa da attacchi Bleichenbacher(1998)/Klima(2003)



## TLS 1.2 -> 1.3 – IN DEFINIZIONE (DRAFT 10/2015)

- Ulteriori modifiche negli algoritmi supportati / obbligatori
- No RC4 support (non sicuro)
- No compression support (CRIME exploit)
- No SSL support (downgrade attack: tutte le vulnerabilità note)
- No ChangeCipherSpec protocol !!!
- ...



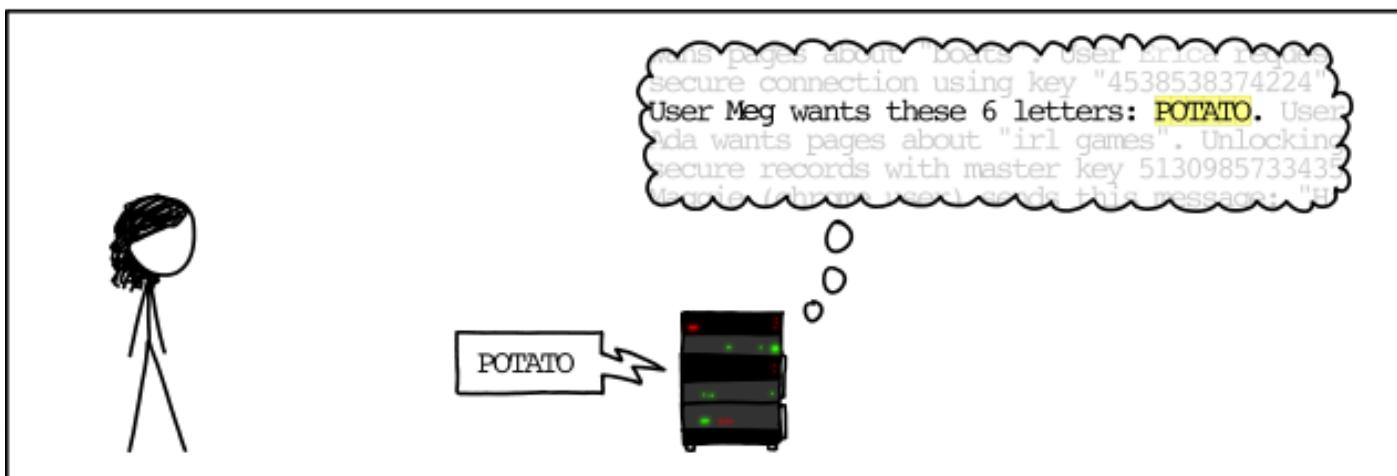
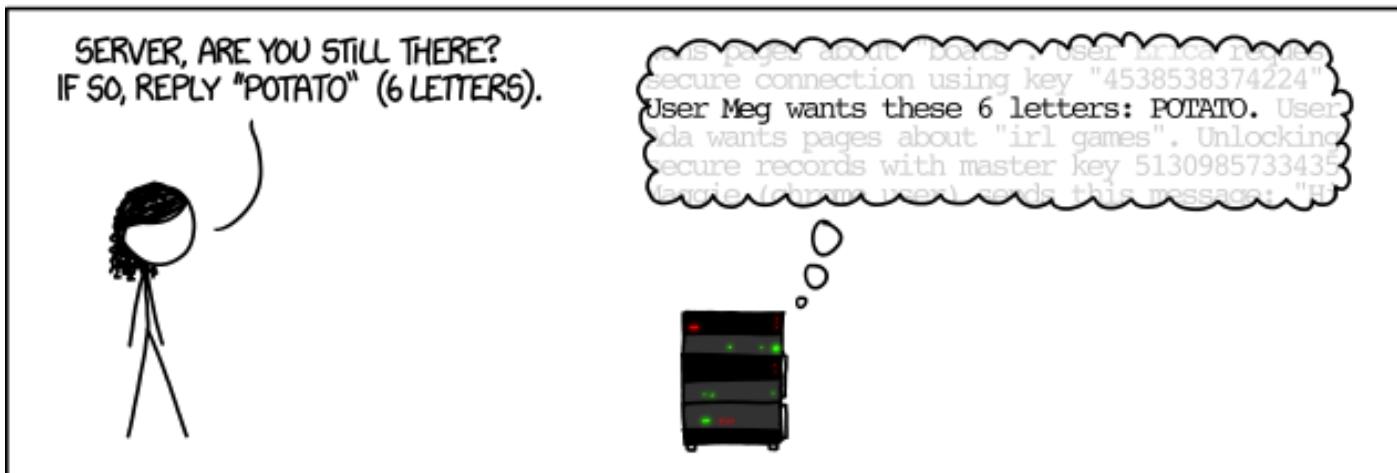
## IL PROTOCOLLO HEARTBEAT – RFC6520 (2012)

- Formato:
  - Type (1 byte): 1 request, 2 response
  - Payload Length (2 bytes)
  - Payload ( $0, 2^{16}-1$ ), arbitrario
  - Padding ( $\geq 16$ bytes), random
- Consente di mantenere attiva una connessione anche in assenza di trasmissione dati applicazione
- Usato per MTU discovery di DTLS (Datagram-TLS)



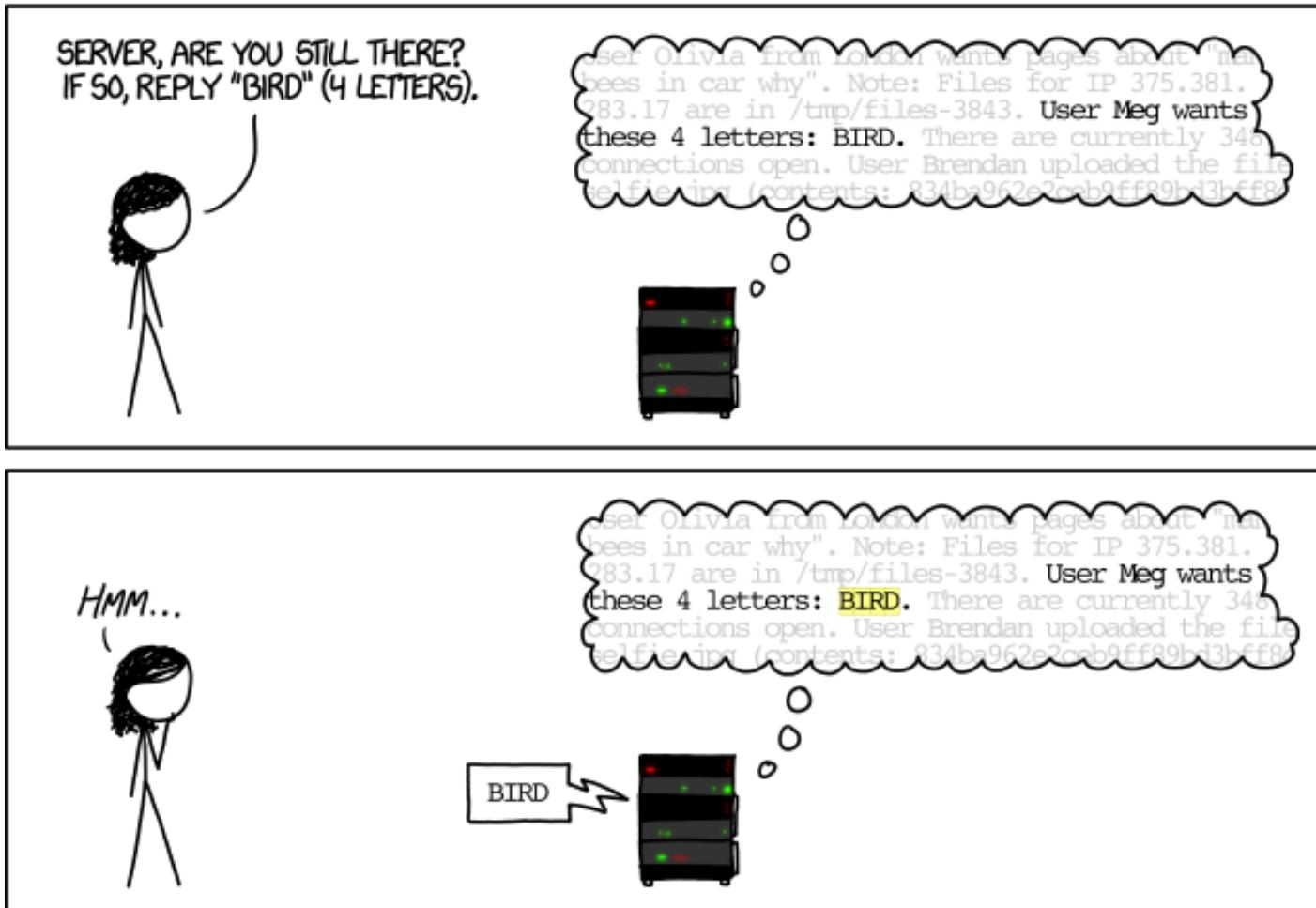
# XKCD.COM - THE HEARTBLEED BUG (1/3)

## HOW THE HEARTBLEED BUG WORKS:



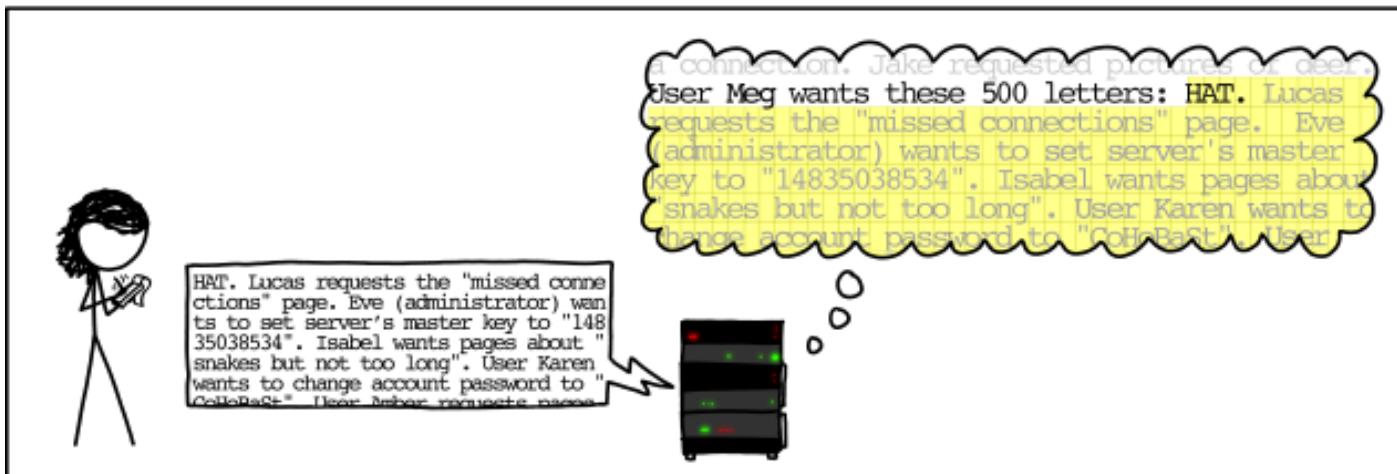
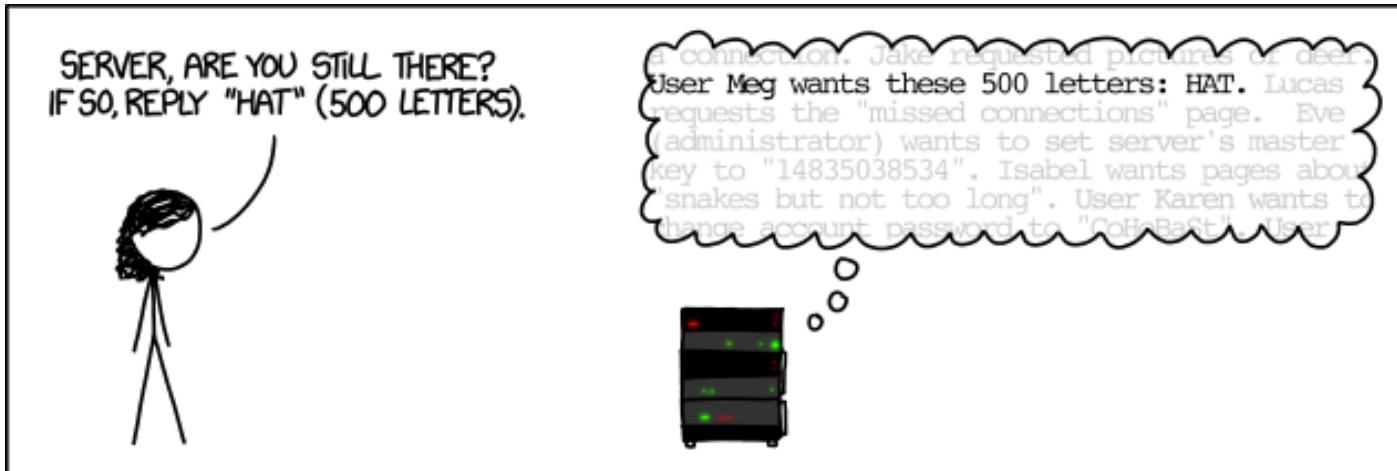


## XKCD.COM - THE HEARTBLEED BUG (2/3)





## XKCD.COM -THE HEARTBLEED BUG (3/3)





# HTTPS: HTTP OVER SSL



## HTTPS: HTTP su SSL



- Uso combinato di HTTP ed SSL per implementare comunicazioni sicure tra un browser ed un server Web
- Funzionalità disponibile in tutti i browser moderni
- Quasi trasparente per gli utenti finali:
  - unica differenza “visiva”: *https://* al posto di *http://* negli URL di navigazione
- Servizio associato, di default, alla porta 443 ed all’impiego di SSL
- Specificato nell’RFC RFC 2818 (“*HTTP Over TLS*”)
  - Non esistono differenze sostanziali nell’impiego di HTTP su SSL o su TLS
    - ...entrambe le implementazioni sono associate all’acronimo HTTPS
- Quando si usa HTTPS, i seguenti elementi della comunicazione sono crittografati:
  - URL del documento richiesto
  - Contenuto del documento
  - Contenuto di eventuali form web gestite dal browser
  - “Cookies” inviati dal browser al server e viceversa
  - Contenuto dell’header dei messaggi HTTP scambiati



# HTTPS in AZIONE: HTTP su TLS su TCP...

SSL\_Handshake\_unina.pcapng [Wireshark 1.12.8 (v1.12.8-0-g5b6e543 from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
26425	106.27	143.225.28.167	192.132.34.41	TCP	78	61482-443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSecr=1034020727 TSecr=0 SACK_PERM=1
26429	106.28	192.132.34.41	143.225.28.167	TCP	74	61482-443 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSecr=365353100 TSecr=1034020727 WS=128
26430	106.28	143.225.28.167	192.132.34.41	TCP	66	61482-443 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSecr=365353100 TSecr=1034020732
26433	106.28	143.225.28.167	192.132.34.41	TLSv1	280	Client Hello
26435	106.28	192.132.34.41	143.225.28.167	TCP	66	443-61482 [ACK] Seq=1 Ack=215 Win=6912 Len=0 TSecr=1034020732 TSecr=365353100
26438	106.29	192.132.34.41	143.225.28.167	TLSv1	1514	Server Hello
26439	106.29	192.132.34.41	143.225.28.167	TLSv1	1360	Certificate
26440	106.29	143.225.28.167	192.132.34.41	TCP	66	61482-443 [ACK] Seq=215 Ack=2743 Win=129760 Len=0 TSecr=365353103 TSecr=1034020745
26441	106.29	143.225.28.167	192.132.34.41	TLSv1	200	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
26446	106.31	192.132.34.41	143.225.28.167	TLSv1	316	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
26447	106.31	143.225.28.167	192.132.34.41	TCP	66	61482-443 [ACK] Seq=349 Ack=2993 Win=130816 Len=0 TSecr=365353108 TSecr=1034020761
26461	106.43	143.225.28.167	192.132.34.41	TLSv1	940	Application Data, Application Data
26462	106.43	192.132.34.41	143.225.28.167	TLSv1	519	Application Data
26463	106.43	143.225.28.167	192.132.34.41	TCP	66	61482-443 [ACK] Seq=1223 Ack=3446 Win=130592 Len=0 TSecr=365353137 TSecr=1034020868
26464	106.43	143.225.28.167	192.132.34.41	TLSv1	940	Application Data, Application Data

> Frame 26433: 280 bytes on wire (2240 bits), 280 bytes captured (2240 bits) on interface 0

> Ethernet II, Src: Apple\_d1:73:ee (98:5a:eb:d1:73:ee), Dst: CiscoInc\_b3:c4:00 (00:17:df:b3:c4:00)

> Internet Protocol Version 4, Src: 143.225.28.167 (143.225.28.167), Dst: 192.132.34.41 (192.132.34.41)

> Transmission Control Protocol, Src Port: 61482 (61482), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 214

> Secure Sockets Layer

  TLSv1 Record Layer: Handshake Protocol: Client Hello

    Content Type: Handshake (22)

    Version: TLS 1.0 (0x0301)

    Length: 209

    Handshake Protocol: Client Hello

      Handshake Type: Client Hello (1)

      Length: 205

      Version: TLS 1.2 (0x0303)

      Random

        GMT Unix Time: Mar 12, 2034 00:47:02.000000000 CET

        Random Bytes: df0011597bd15d5d2177bedaac0f2b7216156ec6d84fe43b...

        Session ID Length: 0

        Cipher Suites Length: 32

        Cipher Suites (16 suites)

          Cipher Suite: TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 (0xc02b)

          Cipher Suite: TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0xc02f)

          Cipher Suite: TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0x009e)

          Cipher Suite: TLS\_ECDHE\_ECDSA\_WITH\_CHACHA20\_POLY1305\_SHA256 (0xcc14)

          Cipher Suite: TLS\_ECDHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256 (0xcc13)

          Cipher Suite: TLS\_DHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256 (0xcc15)

          Cipher Suite: TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA (0xc00a)



# LA SEQUENZA DI HANDSHAKE

Comment
TCP: 61482→443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1034020727 TSecr=0 SACK_PERM=1
TCP: 443→61482 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=365353100 TSecr=1
TCP: 61482→443 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=1034020732 TSecr=365353100
TLsv1: Client Hello
TCP: 443→61482 [ACK] Seq=1 Ack=215 Win=6912 Len=0 TSval=365353100 TSecr=1034020732
TLsv1: Server Hello
TLsv1: Certificate
TCP: 61482→443 [ACK] Seq=215 Ack=2743 Win=129760 Len=0 TSval=1034020745 TSecr=365353103
TLsv1: Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
TLsv1: New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
TCP: 61482→443 [ACK] Seq=349 Ack=2993 Win=130816 Len=0 TSval=1034020761 TSecr=365353108
TLsv1: Application Data, Application Data
TLsv1: Application Data
TCP: 61482→443 [ACK] Seq=1223 Ack=3446 Win=130592 Len=0 TSval=1034020868 TSecr=365353137
TLsv1: Application Data, Application Data
TLsv1: Application Data
TCP: 61482→443 [ACK] Seq=2097 Ack=3963 Win=130528 Len=0 TSval=1034020901 TSecr=365353146
TLsv1: Application Data, Application Data
TCP: [TCP segment of a reassembled PDU]
TCP: [TCP segment of a reassembled PDU]
TCP: 61482→443 [ACK] Seq=2987 Ack=6859 Win=129600 Len=0 TSval=1034020936 TSecr=365353155
TLsv1: Application Data
TCP: 61482→443 [ACK] Seq=2987 Ack=7408 Win=130496 Len=0 TSval=1034020936 TSecr=365353155
TLsv1: Application Data, Application Data
TCP: 443→61482 [ACK] Seq=7408 Ack=4117 Win=15488 Len=0 TSval=365354782 TSecr=1034027307
TLsv1: Application Data
TCP: 61482→443 [ACK] Seq=4117 Ack=8005 Win=130464 Len=0 TSval=1034027411 TSecr=365354798
TLsv1: Application Data, Application Data
TCP: 443→61482 [ACK] Seq=8005 Ack=5071 Win=17792 Len=0 TSval=365354798 TSecr=1034027412
TLsv1: Application Data



# INIZIALIZZAZIONE DELLA CONNESSIONE

L'agente che agisce da client HTTP, agisce anche da client TLS

Il client inizializza una connessione verso il server e poi invia il messaggio TLS ClientHello per dare inizio all'handshake TLS

Al termine dell'handshake, il client può inviare la prima richiesta HTTP

Tutti i dati HTTP sono inviati come dati applicativi su TLS

Tre distinti livelli di "connessione" in HTTPS:

Al livello HTTP, un client richiede una connessione ad un server inviando una richiesta di connessione al livello inferiore nello stack protocolare (normalmente TCP, ma eventualmente, anche TLS/SSL)

Al livello TLS, si crea una sessione tra un client TLS ed un server TLS. Tale sessione può fare da supporto ad una o più "connessioni" contemporanee

Una richiesta TLS per stabilire una connessione inizia con la creazione di una connessione TCP tra l'entità TCP sul client e l'entità TCP sul server



# CHIUSURA DELLA CONNESSIONE

- Client e server HTTP possono indicare la chiusura della connessione includendo la riga “*Connection: close*” nell’header di un messaggio HTTP
- La chiusura di una connessione HTTPS richiede che TLS chiuda il canale di comunicazione con l’entità TLS remota
  - ciò implica, tra l’altro, la chiusura della connessione TCP sottostante
- Le implementazioni TLS devono scambiare appositi messaggi “alert” (messaggio “close\_notify”) prima di chiudere una connessione
- Una specifica implementazione può decidere di chiudere una connessione, dopo aver inviato l’alert di chiusura, senza aspettare che l’entità remota invii a sua volta l’alert in questione
  - in questo caso si parlerà di “incomplete close”
- Una chiusura TCP non annunciata potrebbe rappresentare un’evidenza di un potenziale attacco alla sicurezza, motivo per cui il client HTTPS dovrebbe inviare qualche sorta di “warning” quando tale evento si verifica



# HTTPS UX PROBLEM

Utente è anello debole:

- abituato a cliccare conferme e via...
- non ha tempo da perdere né voglia di leggere messaggi allarmanti
- tecnofobo, tecnoindifferenti: è **utente**, lo strumento è già un fastidio necessario

(im)possibili soluzioni:

- “obblighiamo il browser a chiudere la connessione insicura senza chiedere”
  - Utente: “il browser è rotto, uso InternetExploder che mi fa connettere”
- “obblighiamo il server ad accettare solo standard/algoritmi più sicuri”
  - Utente: “il sito è rotto: non compro, non vedo pubblicità, uso siti dei concorrenti, che non mi danno problemi”



# SSH: SECURE SHELL



## SECURE SHELL – SSH

- Un protocollo per la comunicazione sicura in rete, progettato con requisiti di semplicità di implementazione
- Applicazioni client e server per SSH sono ampiamente diffuse per la maggior parte dei sistemi operativi
- La prima scelta per operazioni di login remoto e di “tunneling” di sessioni di terminale (“X tunneling”)
  - una delle applicazioni maggiormente pervasive nel mondo della crittografia, insieme alle soluzioni basate sull’impiego di sistemi embedded
- Capace di fornire funzioni (sicure) generiche di tipo client/server:
  - trasferimento file
  - posta elettronica
  - ...

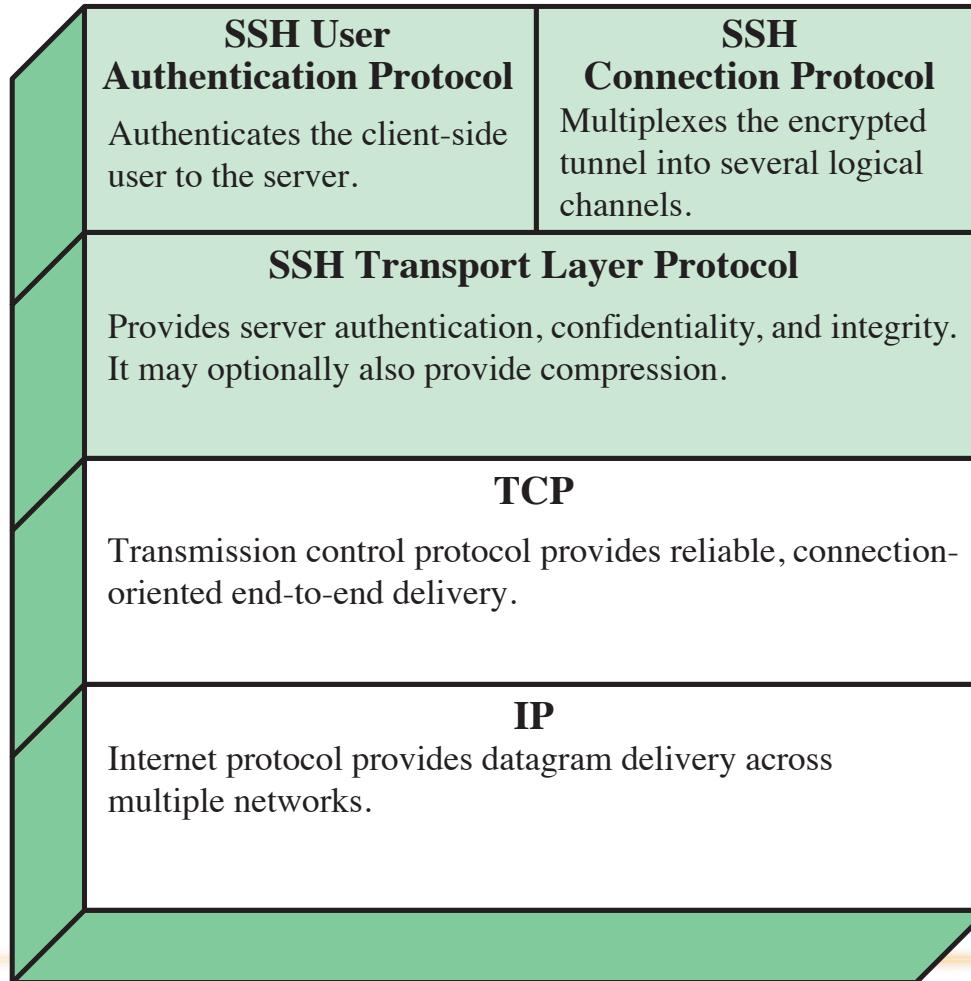


## SSH: VERSIONI

- SSHv1:
  - login remoto sicuro
    - in sostituzione di rsh (remote shell), rlogin, telnet e di analoghi approcci ***privi di sicurezza***
- SSHv2:
  - un approccio più strutturato alla sicurezza
    - rimuove alcuni seri difetti di progettazione del precedente schema protocollare
    - documentato in una nutrita serie di RFC di Internet (da RFC4250 ad RFC4256)



# SSH: ARCHITETTURA





# SSH TRANSPORT LAYER PROTOCOL

- L'autenticazione con il server avviene al livello trasporto ssh, mediante l'elaborazione di una coppia di chiavi (pubblica/privata)
- Un server può adottare molteplici chiavi di host (“host key”), ciascuna associata ad un differente algoritmo di crittografia asimmetrica
- Host diversi possono condividere un'unica host key
- La host key del server viene utilizzata durante il processo di scambio delle chiavi per autenticare l'identità dell'host
  - per questo motivo, il client deve conoscere a priori la chiave pubblica del server
- Lo standard RFC 4251 (architettura di SSH) specifica due modelli di “trust” alternativi:
  - Il client ha un database locale che associa ciascun nome di host alla corrispondente chiave pubblica
  - L'associazione “nome host  $\leftrightarrow$  chiave host” è certificata da una Certification Authority (CA) fidata
    - il client conosce soltanto la chiave “root” della CA e può verificare la validità di qualsiasi host key certificata da una qualsiasi CA riconosciuta



# SSH TRANSPORT LAYER PROTOCOL (!)

- L'autenticazione con il server avviene attraverso la generazione di una coppia di chiavi (pubblica/privata)
- Un server può adottare molteplici host key, e per ogni host key un differente algoritmo di crittografia a seconda delle capacità del server
- Host diversi possono condividere un'unica host key
- La host key del server viene utilizzata durante il primo passo di scambio delle chiavi per autenticare l'identità dell'host
  - per questo motivo, il client deve conoscere a priori la chiave pubblica del server
- Lo standard RFC 4251 (architettura di SSH) specifica due modelli di “trust” alternativi:
  - Il client ha un database locale che associa ciascun nome di host alla corrispondente chiave pubblica
  - L'associazione “nome host ↔ chiave host” è certificata da una Certification Authority (CA) fidata
    - il client conosce soltanto la chiave “root” della CA e può verificare la validità di qualsiasi host key certificata da una qualsiasi CA riconosciuta

Se non è presente nel database (e.g. la prima volta che ci si connette da quel client) viene mostrata la fingerprint della chiave del server: l'utente deve verificare la corrispondenza out-of-band



# SSH TRANSPORT LAYER PROTOCOL (!!)

- L'autenticazione con il server avviene con una coppia di chiavi (pubblica/privata)
- Un server può adottare molteplici chiavi pubbliche con differente algoritmo di crittografia e chiavi
- Host diversi possono condividere la stessa host key
- La host key del server viene utilizzata per autenticare l'identità dell'host
  - per questo motivo, il client deve avere un database locale delle host keys
- Lo standard RFC 4251 (architettura) definisce:
  - Il client ha un database locale delle host keys e le associa alle chiavi pubblica
  - L'associazione "nome host → host key" è garantita da una certificazione (CA) fidata
    - il client conosce soltanto la chiave "root" della CA e può verificare la validità di qualsiasi host key certificata da una qualsiasi CA riconosciuta

Se non è presente nel database

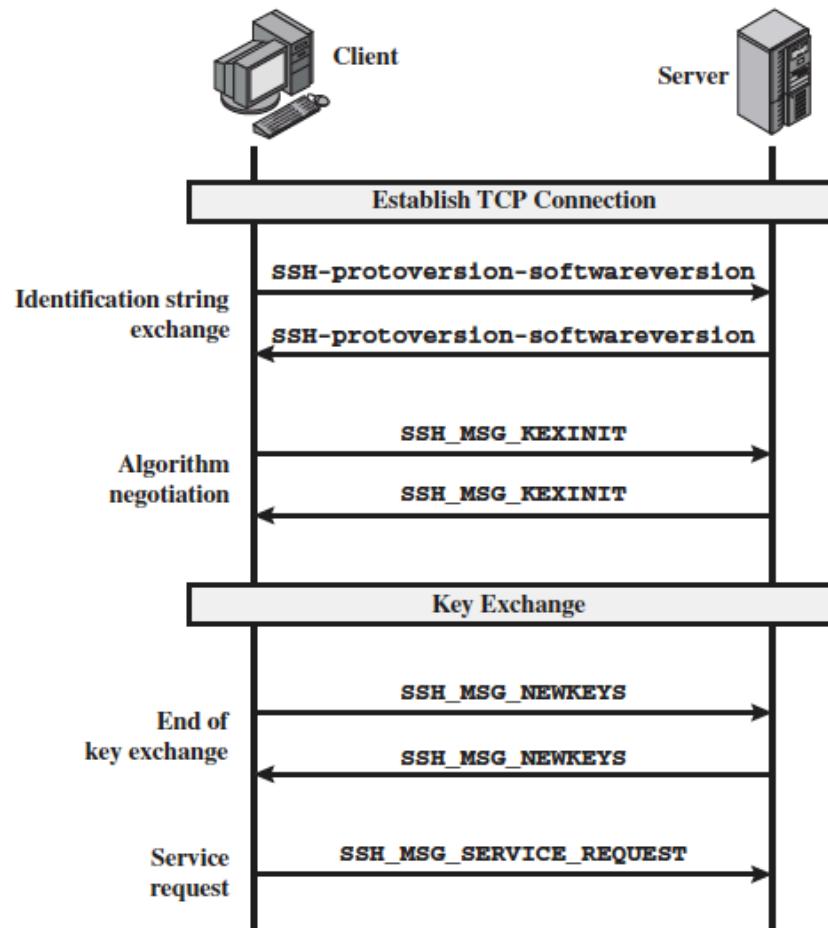
(e.g. la prima volta che ci si connette da quel client)  
viene mostrata la fingerprint della chiave del server:  
**l'utente** deve verificare la corrispondenza out-of-band  
“un aiutino”

-o VisualHostKey=yes:

```
Host key fingerprint is cf:2c:2e:f6:d4:d4:50:0f:bb:25:89:ec:  
84:c0:93:f2  
+-- [ RSA 2048]----+  
| ... .o |  
| . +. o.. + |  
| + .. +o+ o |  
| E o. .+ |  
| .s . |  
| ..o |  
| . o |  
| o o . |  
| . +oo. |
```

# SSH TRANSPORT LAYER PROTOCOL

- Il client stabilisce una connessione TCP con il server
- A connessione stabilita, client e server scambiano dati encapsulati in segmenti TCP
  - identificazione reciproca
  - negoziazione degli algoritmi di crittografia
  - scambio delle chiavi
  - richieste di servizio





# SSH IN AZIONE

1. Connessione al server
  - ...con memorizzazione in locale della fingerprint RSA dello stesso
2. Impiego del terminale remoto sicuro
3. Disconnessione

```
Simons-iMac:~ ssh spromano$ ssh root@143.225.229.134
The authenticity of host '143.225.229.134 (143.225.229.134)' can't be established.
RSA key fingerprint is ca:5a:db:0f:b6:c6:70:e1:da:11:b3:00:6d:63:18:46.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '143.225.229.134' (RSA) to the list of known hosts.
root@143.225.229.134's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation: https://help.ubuntu.com/
Last login: Wed Oct 28 14:27:05 2015 from 143.225.28.167
root@srv134:~# ls
janus-gateway  libsrtp-1.5.0  libsrtp-1.5.0.tar.gz  meetecho  rabbitmq-c  ssh_keys  std.err  std.out  testPPTconversion  weblite-recordings
root@srv134:~# exit
logout
Connection to 143.225.229.134 closed. 3
Simons-iMac:~ ssh spromano$ more known_hosts
143.225.229.134 ssh-rsa AAAAAB3NzaC1yc2EAAAQABAAQDFb4+VZvKfE0EMxnehh/g/k9cj7/WmMwXQMBZPHCgcorE2C+i5FBtf6zAZRURJCbofjc2Qtzw+EcAKK7Gq7jden3jEjIx9YFz5
hl6maXTQfGfokpsKaNkpSfchVPJpsZmjyyh8sCS7yLafq4+j9a16UFNd8i6b7uD98dQkuIee3amDLCIyIG6pv/PBr0cXJYDg2nc/UJKQPrCd89ldBEoj+CFCC003YNg9j8WHCF
Simons-iMac:~ ssh spromano$
```



# DIETRO LE QUINTE...

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0003	143.225.229.134	143.225.229.134	TCP	78	53020-22 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TStamp=1045235432 TSectr=0 SACK_PERM=1
2	0.0003	143.225.229.134	143.225.28.167	TCP	74	22-53020 [SYN, ACK] Seq=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TStamp=3224104178 TSectr=1045235432 WS=128
3	0.0003	143.225.28.167	143.225.229.134	TCP	66	53020-22 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TStamp=1045235432 TSectr=3224104178
4	0.0004	143.225.28.167	143.225.229.134	SSHv2	87	Client: Protocol (SSH-2.0-OpenSSH_6.2)
5	0.0007	143.225.229.134	143.225.28.167	TCP	66	22-53020 [ACK] Seq=1 Ack=22 Win=29056 Len=0 TStamp=3224104178 TSectr=1045235432
6	0.0120	143.225.229.134	143.225.28.167	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_6.6.1p1_Ubuntu-2ubuntu2)
7	0.0121	143.225.28.167	143.225.229.134	TCP	66	53020-22 [ACK] Seq=22 Ack=42 Win=131712 Len=0 TStamp=1045235443 TSectr=3224104181
8	0.0123	143.225.28.167	143.225.229.134	TCP	1514	[TCP segment of a reassembled PDU]
9	0.0123	143.225.28.167	143.225.229.134	SSHv2	210	Client: Key Exchange Init
10	0.0129	143.225.229.134	143.225.28.167	TCP	66	22-53020 [ACK] Seq=42 Ack=1614 Win=34816 Len=0 TStamp=3224104181 TSectr=1045235443
11	0.0132	143.225.229.134	143.225.28.167	TCP	1514	[TCP segment of a reassembled PDU]
12	0.0132	143.225.229.134	143.225.28.167	SSHv2	266	Server: Key Exchange Init
13	0.0133	143.225.28.167	143.225.229.134	TCP	66	53020-22 [ACK] Seq=1614 Ack=1690 Win=130048 Len=0 TStamp=1045235444 TSectr=3224104181
14	0.0134	143.225.28.167	143.225.229.134	SSHv2	90	Client: Diffie-Hellman Group Exchange Request
15	0.0164	143.225.229.134	143.225.28.167	SSHv2	218	Server: Diffie-Hellman Group Exchange Group

V Transmission Control Protocol, Src Port: 53020 (53020), Dst Port: 22 (22), Seq: 1470, Ack: 42, Len: 144

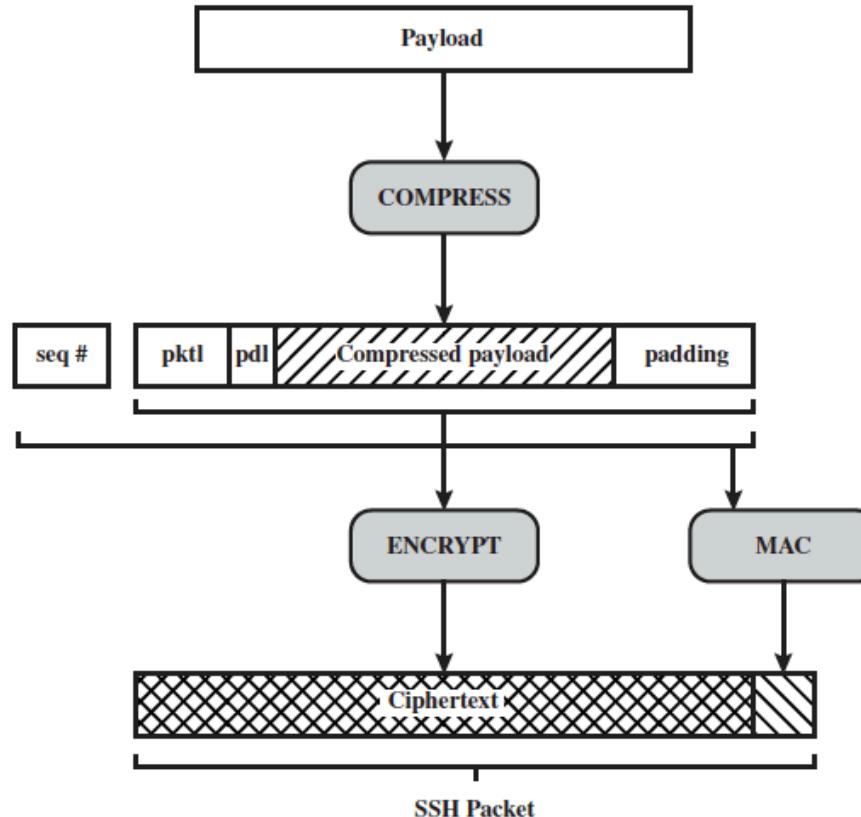
▷ [2 Reassembled TCP Segments (1592 bytes): #8(1448), #9(144)]

▼ SSH Protocol

- ▷ SSH Version 2 (encryption: aes128-ctr mac:hmac-md5-eth@openssh.com compression:none)
  - Packet Length: 1588
  - Padding Length: 6
  - ▽ Key Exchange
    - Message Code: Key Exchange Init (20)
    - ▽ Algorithms
      - Cookie: 3edbdb9b6ddedcc91ec67a32a2d6f8a
      - kex\_algorithms length: 126
      - kex\_algorithms string: diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group1-sha1
      - server\_host\_key\_algorithms length: 131
      - server\_host\_key\_algorithms string: ssh-rsa-cert-v01@openssh.com,ssh-dss-cert-v01@openssh.com,ssh-rsa-cert-v00@openssh.com,ssh-dss-cert-v00@openssh.com,ssh-rsa,ssh-dss
      - encryption\_algorithms\_client\_to\_server length: 203
      - encryption\_algorithms\_client\_to\_server string [truncated]: aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour,rijn
      - encryption\_algorithms\_server\_to\_client length: 203
      - encryption\_algorithms\_server\_to\_client string [truncated]: aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour,rijn
      - mac\_algorithms\_client\_to\_server length: 402
      - mac\_algorithms\_client\_to\_server string [truncated]: hmac-md5-eth@openssh.com,hmac-sha1-eth@openssh.com,umac-64-eth@openssh.com,umac-128-eth@openssh.com,hmac-sha2-256-eth@openssh.com,hmac-sha2-512-eth@openssh.com,hmac-ripemd160-eth@openssh.
      - mac\_algorithms\_server\_to\_client length: 402
      - mac\_algorithms\_server\_to\_client string [truncated]: hmac-md5-eth@openssh.com,hmac-sha1-eth@openssh.com,umac-64-eth@openssh.com,umac-128-eth@openssh.com,hmac-sha2-256-eth@openssh.com,hmac-sha2-512-eth@openssh.com,hmac-ripemd160-eth@openssh.
      - compression\_algorithms\_client\_to\_server length: 26
      - compression\_algorithms\_client\_to\_server string: none,zlib@openssh.com,zlib
      - compression\_algorithms\_server\_to\_client length: 26
      - compression\_algorithms\_server\_to\_client string: none,zlib@openssh.com,zlib
      - languages\_client\_to\_server length: 0
      - languages\_client\_to\_server string: [Empty]
      - languages\_server\_to\_client length: 0
      - languages\_server\_to\_client string: [Empty]
    - KEX First Packet Follows: 0
    - Reserved: 00000000
    - Payload: <MISSING>
    - Padding String: 000000000000



# IL PACCHETTO SSH



**pktl** = packet length  
**pdl** = padding length



# FORMATO DEI PACCHETTI SSH (1/2)

- Packet length:
  - lunghezza del pacchetto in byte, senza considerare i campi “packet length” e MAC
- Padding length:
  - numero di byte generati casualmente ed impiegati come riempimento
- Payload:
  - contenuto utile del pacchetto
  - prima della negoziazione dell'algoritmo:
    - non compresso
  - a negoziazione avvenuta:
    - compresso, qualora negoziato dalle parti
- Random padding:
  - a negoziazione avvenuta, contiene byte random di padding, necessari per assicurarsi che la lunghezza totale del pacchetto (campo MAC escluso) sia:
    - un multiplo della dimensione del “cypher block”
    - 8 byte nel caso di stream cypher



## FORMATO DEI PACCHETTI SSH (2/2)

- Message Authentication Code (MAC):
  - in caso di negoziazione della autenticazione dei messaggi, contiene il valore del MAC
  - Calcolato sull'intero pacchetto (MAC stesso escluso), più un numero di sequenza
- Sequence Number:
  - un numero su 32 bit
    - inizializzato a 0 per il primo pacchetto
    - incrementato di 1 per ogni pacchetto successivo
  - NON inviato sulla connessione TCP!



# ALGORITMI CRITTOGRAFICI SUPPORTATI

Cipher		MAC algorithm		Compression algorithm	
<b>3des-cbc*</b>	Three-key 3DES in CBC mode	<b>hmac-sha1*</b>	HMAC-SHA1; digest length = key length = 20	<b>none*</b>	No compression
<b>blowfish-cbc</b>	Blowfish in CBC mode	<b>hmac-sha1-96**</b>	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20	<b>zlib</b>	Defined in RFC 1950 and RFC 1951
<b>twofish256-cbc</b>	Twofish in CBC mode with a 256-bit key	<b>hmac-md5</b>	HMAC-SHA1; digest length = key length = 16		
<b>twofish192-cbc</b>	Twofish with a 192-bit key	<b>hmac-md5-96</b>	First 96 bits of HMAC-SHA1; digest length = 12; key length = 16		
<b>twofish128-cbc</b>	Twofish with a 128-bit key				
<b>aes256-cbc</b>	AES in CBC mode with a 256-bit key				
<b>aes192-cbc</b>	AES with a 192-bit key				
<b>aes128-cbc**</b>	AES with a 128-bit key				
<b>Serpent256-cbc</b>	Serpent in CBC mode with a 256-bit key				
<b>Serpent192-cbc</b>	Serpent with a 192-bit key				
<b>Serpent128-cbc</b>	Serpent with a 128-bit key				
<b>arcfour</b>	RC4 with a 128-bit key				
<b>cast128-cbc</b>	CAST-128 in CBC mode				

\* = Required

\*\* = Recommended



# METODI DI AUTENTICAZIONE

- Publickey
  - Il client invia un messaggio al server
    - contenente la sua chiave pubblica
    - firmato con la sua chiave privata
  - Alla ricezione del messaggio, il server
    - verifica che la chiave sia accettabile per l'autenticazione
    - in caso affermativo, verifica che la firma sia corretta
- Password
  - Il client invia un messaggio contenente una password in chiaro, ma protetta crittograficamente dal protocollo TLS sottostante
- Hostbased:
  - L'autenticazione viene effettuata sull'host del client piuttosto che sul client (cioè, sull'utente) vero e proprio
  - Il client invia una firma creata con la chiave privata dell'host da cui si collega al server
  - Piuttosto che verificare l'identità dell'utente, il server SSH verifica l'identità dell'host



# METODI DI AUTENTICAZIONE(!)

- Publickey
  - Il client invia un messaggio al server
    - contenente la sua chiave pubblica
    - firmato con la sua chiave privata
  - Alla ricezione del messaggio, il server
    - verifica che la chiave sia accettabile per la comunicazione
    - in caso affermativo, verifica che la firma sia corretta
- Password
  - Il client invia un messaggio contenente una password in chiaro, ma protetta crittograficamente dal protocollo TLS sottostante
- Hostbased:
  - L'autenticazione viene effettuata sull'host del client piuttosto che sul client (cioè, sull'utente) vero e proprio
  - Il client invia una firma creata con la chiave privata dell'host da cui si collega al server
  - Piuttosto che verificare l'identità dell'utente, il server SSH verifica l'identità dell'host

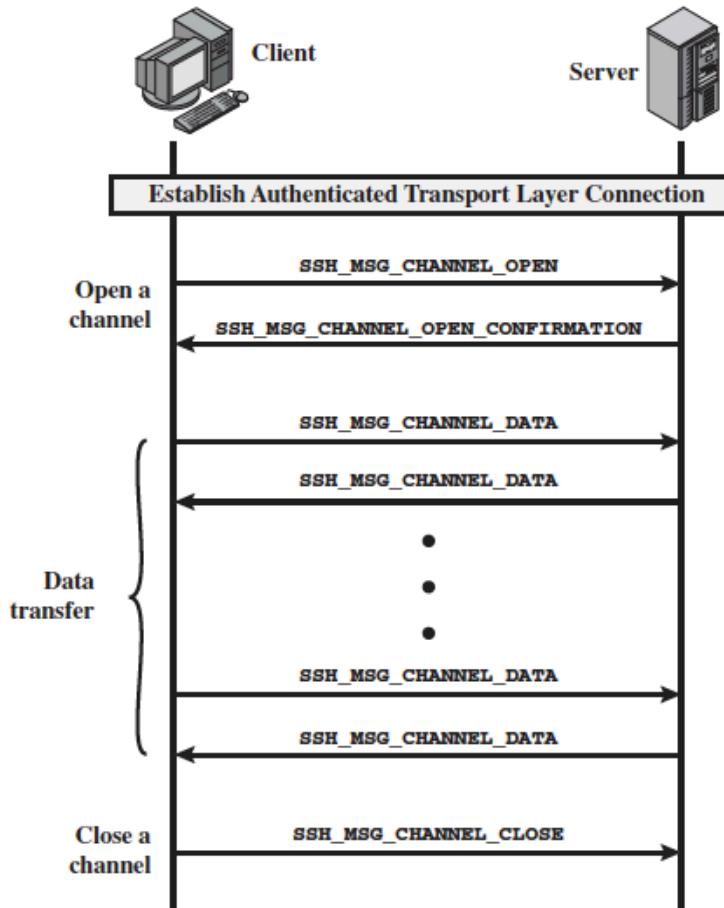
Molto usato ma più aperto ad attacchi.  
Si consiglia l'uso della password solo la prima volta, per upload di chiavi pubbliche, poi autenticazione a chiave pubblica.



# PROTOCOLLO DI CONNESSIONE

- Lavora sul protocollo di trasporto
- Assume che una connessione sicura ed autenticata sia attiva
  - Tale connessione, detta “tunnel”, viene utilizzata per il multiplexing di molteplici canali logici
- Meccanismo dei “canali”:
  - tutti i tipi di comunicazione sono supportati mediante canali separati
  - entrambe le parti possono aprire un canale
  - ad ogni canale, ciascuna parte associa un identificativo univoco
  - i canali sono sottoposti a controllo di flusso mediante un meccanismo “a finestra”
    - nessun dato può essere inviato su di un canale in assenza di un esplicito messaggio che indichi lo spazio disponibile nella finestra di controllo
  - ogni canale ha un suo ciclo di vita:
    - apertura, trasferimento dati, chiusura

# PROTOCOLLO DI CONNESSIONE: UN ESEMPIO





# TIPI DI CANALI

## Session

- Esecuzione remota di un programma
- Tipici programmi: shell, applicazioni quali trasferimento file ed e-mail, comandi di sistema, ecc.
- Una volta aperto un canale di sessione, richieste successive sono utilizzate per lanciare in esecuzione il programma remoto

## X11

- Fa riferimento al sistema “X Window”, un modulo software di sistema ed un protocollo di comunicazione che forniscono un’interfaccia grafica per nodi di rete
- X consente alle applicazioni di essere eseguite su di un server di rete, ma di essere “mostrate” su un desktop remoto

## Forwarded-tcpip

- Abilita la funzionalità di “port forwarding” remoto (cfr. prossima slide...)

## Direct-tcpip

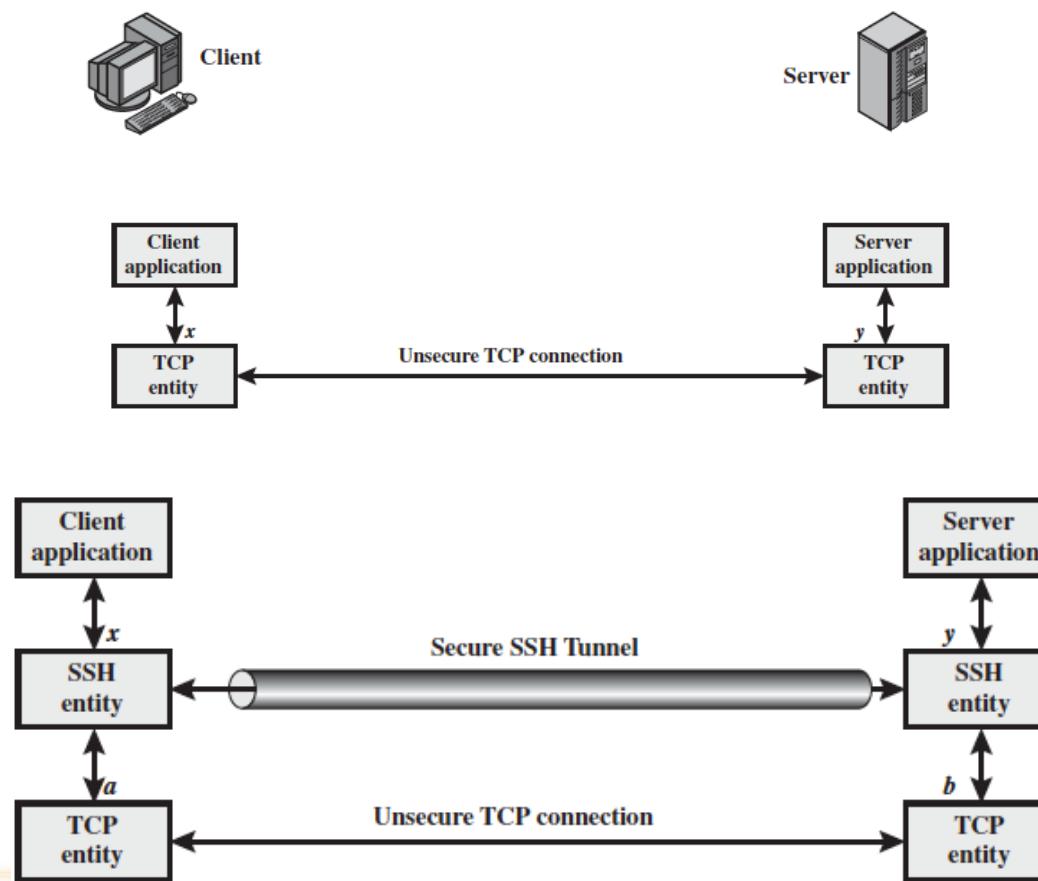
- Funzionalità di port forwarding locale (cfr. prossima slide...)



# PORT FORWARDING

- Una delle funzioni più utili di SSH
- Fornisce la possibilità di convertire qualsiasi connessione TCP non sicura in una corrispondente sessione SSH
  - il cosiddetto “SSH tunneling”
- Traffico TCP in ingresso al nodo viene consegnato all'applicazione appropriata sulla base del numero di porta
  - es:
    - SMTP → porta TCP 25
      - in presenza di tunnel SSH, i pacchetti giunti a destinazione sono “riconosciuti” da TCP ed inoltrati all'applicazione interessata (in questo caso, il mail server) in base al campo port number dell'header TCP...
- Una stessa applicazione può fare uso di più numeri di porta

# IL TUNNELING COME STRUMENTO DI SECURITY





# LOCAL vs REMOTE PORT FORWARDING

- Local:
  - connessioni dal client SSH sono inoltrate tramite il server SSH verso un server SSH remoto
    - il client configura un processo “hijacker”, il quale:
      - intercetta il traffico dell’applicazione non sicura
      - lo redirige dalla connessione TCP (non sicura) verso un tunnel SSH (sicuro)
    - all’altro capo del tunnel, il server SSH smista il traffico in ingresso alla porta destinazione indicata dall’applicazione client
- Remote:
  - connessioni dal server SSH sono inoltrate tramite il client SSH verso un server SSH remoto
    - il client SSH dell’utente funge da server
      - riceve traffico con una determinata porta destinazione
      - lo “assegna” alla porta corretta
      - lo invia alla destinazione scelta dall’utente
  - consente, ad esempio, di collegarsi, dal proprio server SSH, ad un computer nella propria Intranet aziendale!



## SSH POST-SNOWDEN ERA

- In base a leaks di Snowden (analisi SPIEGEL online\* nel Dicembre 2014) risulta che NSA è capace di intercettare comunicazioni SSH
  - Sembra tramite sfruttamento di debolezze (imposte per legge, o scoperte in seguito) di algoritmi di crittografia
    - Algoritmi con vulnerabilità
    - Algoritmi con chiavi troppo piccole
  - Infiltrazione degli host (acquisizione di credenziali)
- Soluzione / mitigazione:
  - Forzare l'uso di algoritmi+chiavi sicuri
  - Autenticazione con chiave pubblica



# TOOLS

- SSLsniff -- <http://www.thoughtcrime.org/software/sslSniff/>
  - https-to-http rewriting
  - CA flaws exploitation
- M-i-t-m proxy -- <https://mitmproxy.org/>
- Firesheep -- <http://codebutler.com/firesheep/>
- TestSSL -- <https://testssl.sh/>



# TESTSSL CHECKS

- Heartbleed (CVE-2014-0160)
- CCS (CVE-2014-0224)
- Secure Renegotiation (CVE-2009-3555)
- Secure Client-Initiated Renegotiation
- CRIME, TLS (CVE-2012-4929)
- BREACH (CVE-2013-3587)
- POODLE, SSL (CVE-2014-3566)
- TLS\_FALLBACK\_SCSV (RFC 7507)
- FREAK (CVE-2015-0204)
- LOGJAM (CVE-2015-4000)
- BEAST (CVE-2011-3389)
- RC4 (CVE-2013-2566, CVE-2015-2808)



# DOMANDE?

