Università degli Studi "Roma Tre"

Dipartimento di Informatica ed Automazione

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesina per il corso di Elementi di Crittografia

Protocollo SSH (Secure Shell)

Prof.ssa

Maria Rosaria Rota

Cesario Daniele

Cangemi Carlo

Ruggeri Chiara

Motivazioni

- Protocollo che consente di stabilire connessioni sicure tra due sistemi tramite un'architettura client-server

- Fornisce:
  – Una infrastruttura per connessioni crittografate
  – Autenticazione forte tra host e host e tra utente e host
    - login remoto sicuro
  – Possibilità di creare un canale di comunicazione sicuro attraverso il quale veicolare qualsiasi connessione TCP/IP

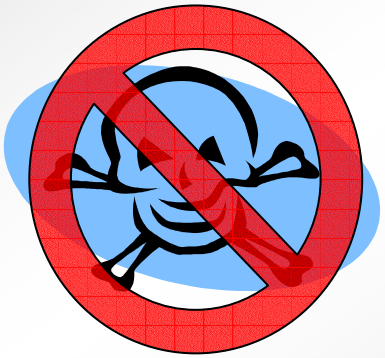
- Nato per rimpiazzare i comandi Berkeley r* (**rsh, rlogin, rcp**) ,con le rispettive versioni sicure (**ssh, slogin, scp**)

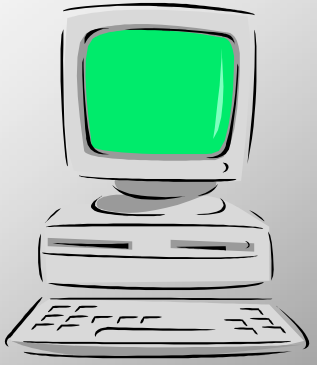
- Risolve alcuni noti problemi di sicurezza dei protocolli TCP/IP come:
  – L'***IP spoofing*** (falsificazione dell'indirizzo IP del mittente)
  – Il ***DNS spoofing*** (falsificazione delle informazioni contenute nel DNS)
  – ***Routing spoofing*** (falsificazione delle rotte intraprese dai pacchetti e instradamento su percorsi diversi)

Connessioni crittografate

Il server accetta il collegamento se:
-**La richiesta proviene da una porta TCP privilegiata**
-**L'host remoto è fidato, cioè presente nel file /etc/hosts.equiv**

Protocolli classici di UNIX testo in chiaro, vulnerabili ad attacchi di sniffing





RSH

Rlogin

RCP

SSH

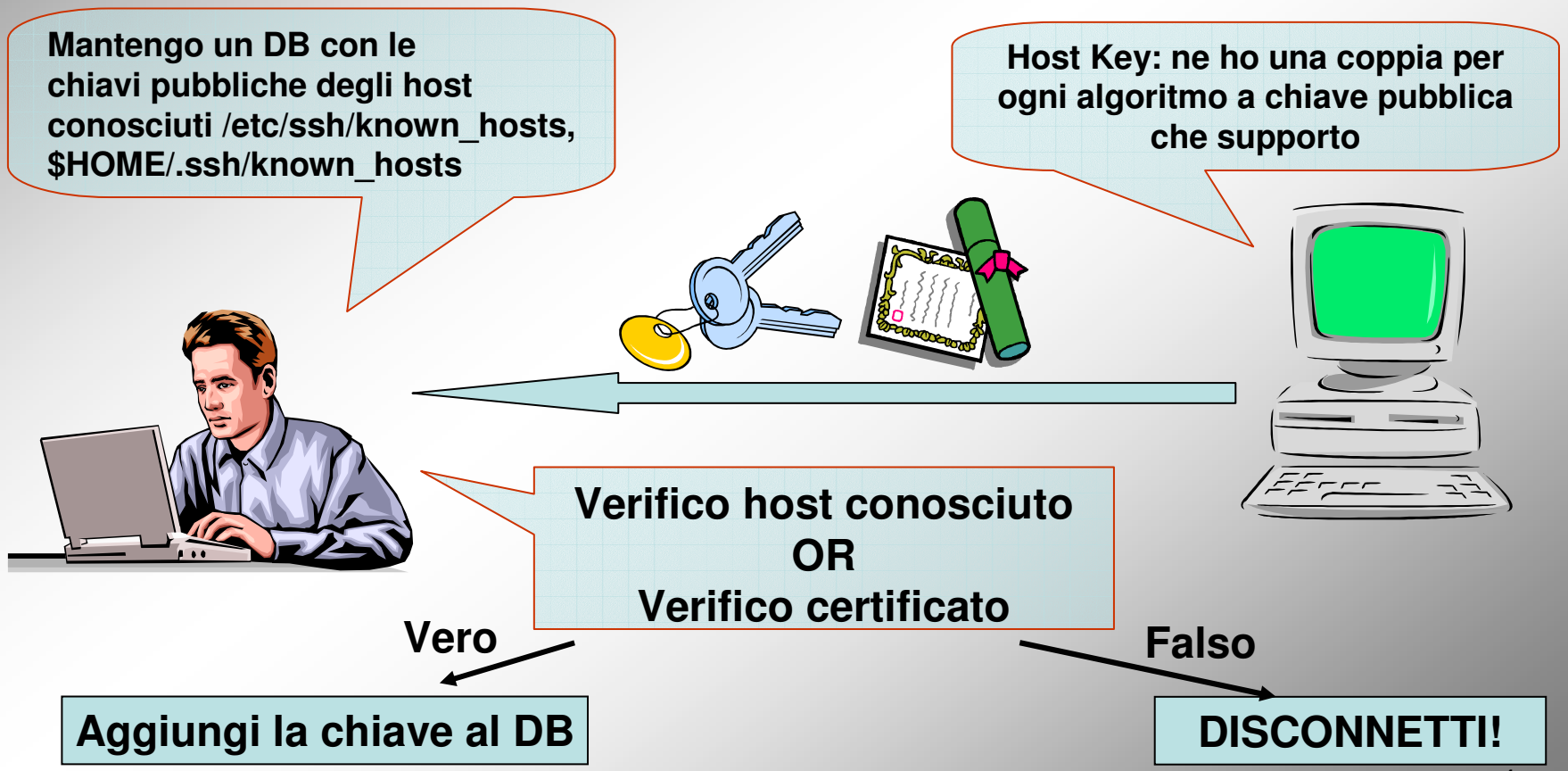
Slogin

SCP

Protocolli SSH: testo cifrato con algoritmi a chiave simmetrica

Autenticazione forte

***Esempio di verifica dell'identità del server*: chiave + certificati**

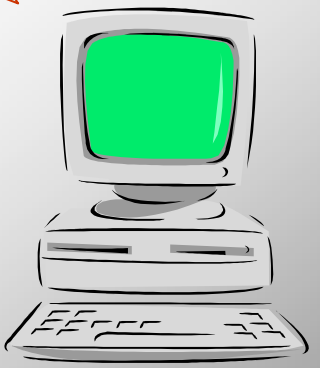


Autenticazione forte

Esempio di verifica dell'identità del server: chiave

Mantengo un DB con le chiavi pubbliche degli host conosciuti /etc/ssh/known_hosts, $HOME/.ssh/known_hosts

Host Key: ne ho una coppia per ogni algoritmo a chiave pubblica che supporto



Conosco questo host?

Sì

PROSEGUI!

No

Mi fido?

No

DISCONNETTI!

Sì

Aggiungo la chiave pubblica in $HOME/.ssh/known_hosts e PROSEGUI!

Autenticazione client → server

- **PROBLEMA**: Metodo *classico* di autenticazione *rhosts* di UNIX vulnerabile ad attacchi di *spoofing*

Autenticazione RHOSTS:
I files /etc/hosts.equiv e $HOME/.ssh/known_hosts Contengono IP o nomi host dei client ammessi al login senza password

- Un noto attacco**: IP Spoofing**

Azioni per un tipico attacco IP SPOOFING da parte di un hacker (H):

1. **Scegliere l'host target(B)**
2. **Scoprire l'indirizzo IP di un host fidato(A)**
3. **Disabilitare A (con TCP SYN flooding)**
4. **Utilizzare l'indirizzo IP di B per ottenere una shell su B cercando di indovinare i Sequence Number**
   - N.B.: H potrebbe catturare i pacchetti in uscita da B falsificando la rotta di ritorno (ROUTING SPOOFING)
5. **Aggiungere il proprio indirizzo in /etc/hosts.equiv su B**

Autenticazione client → server

- Un'altro attacco: **DNS Spoofing**

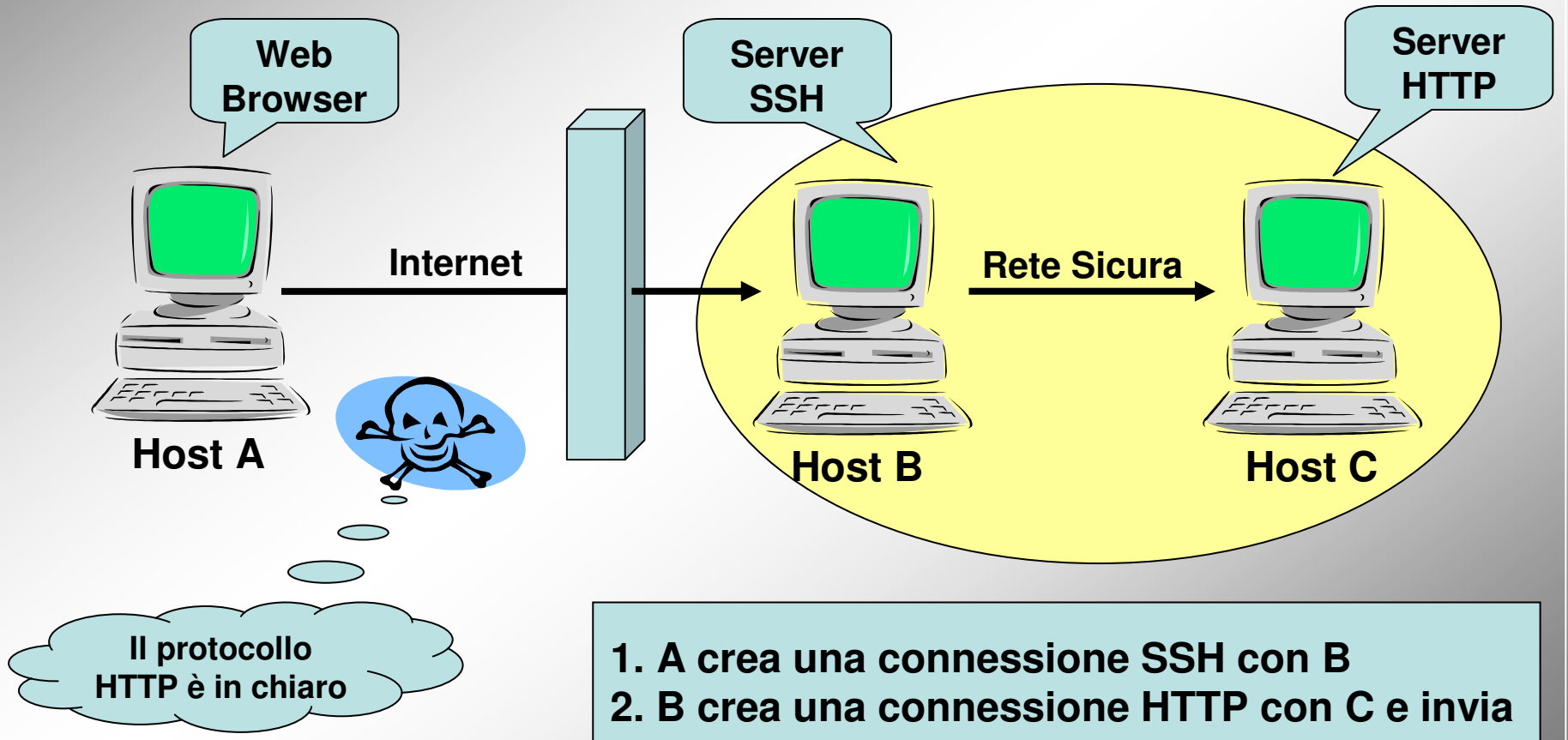
Azioni per un tipico attacco DNS SPOOFING da parte di un hacker (H):
1. **Scegliere l'host target (B)**
2. **Scoprire il nome host di un host fidato(A)**
3. **Falsificare il DNS di B facendo in modo che il nome host di A venga risolto con l'indirizzo IP di H**
4. **Stabilire una connessione con B, fingendosi A**

- **SOLUZIONE**: SSH aggiunge un controllo sull'host più rigoroso: Autenticazione ***rhosts + chiave pubblica****:*

➢ Il client invia un pacchetto firmato con la chiave privata del proprio host
➢ Il server verifica la firma con la chiave pubblica dell'host del client

TCP port forwarding

- **Una qualsiasi connessione TCP può essere resa sicura:**



1. **A crea una connessione SSH con B**
2. **B crea una connessione HTTP con C e invia ad A i dati cifrati con SSH**

Credits e Versioni

- 2 Versioni non compatibili tra loro
  - SSH1 (è un protocollo integrato)
  - SSH2 (ridefinisce la precedente versione in 3 “layers”)
- Gestito da 2 società finlandesi:
  - **SSH Communications Security** (http://www.ssh.org) ha sviluppato i protocolli SSH1 e SSH2 e mantiene le distribuzioni principali di SSH
  - **Data Fellows** (http://www.datafellows.com) ha acquistato dalla prima la licenza di vendere e supportare SSH

Licenze d'uso

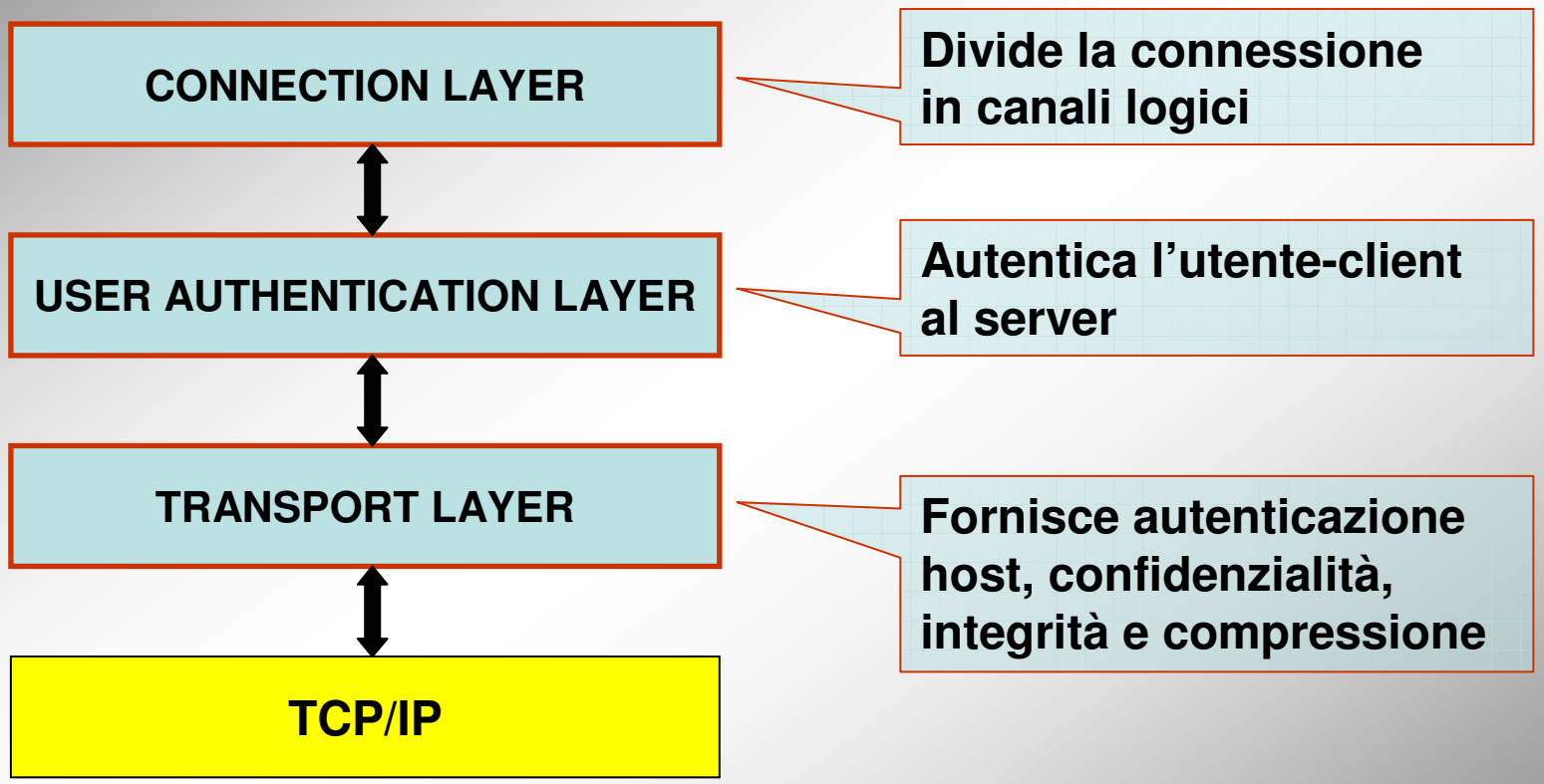
- Data Fellows ha imposto i limiti di utilizzo per il software:
  - SSH1 può essere utilizzato liberamente a patto che non se ne ricavi guadagno
  - SSH2 può essere utilizzato liberamente solo privatamente o in ambito educational
- Protocolli SSH1 e SSH2 pubblicati come Internet Drafts
- OpenSSH (http://www.openssh.com)
  - Open Source
  - nato per far parte del sistema operativo OpenBSD
  - utilizza solo algoritmi di crittografia senza restrizioni di utilizzo

Caratteristiche
Politiche Locali

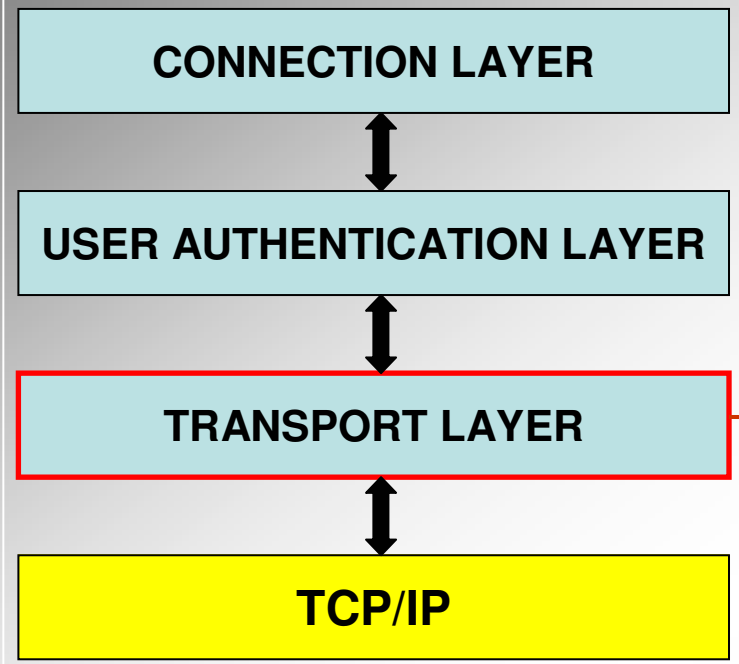
- Piena negoziazione degli algoritmi di crittografia, integrità, scambio delle chiavi e compressione
- Per ogni categoria le politiche locali specificano un algoritmo preferito
- A seconda dell'utente e della sua locazione possono essere richiesti differenti modalità di autenticazione o addirittura autenticazioni multiple
- E' possibile limitare od estendere l'insieme delle operazioni consentite a determinati utenti

Architettura

Tre componenti primari:



Transport Layer Protocol



Esegue

- **Autenticazione dell'host**
- **Scambio di chiavi**
- **Cifratura**
- **Protezione dell'integrità**
- **Calcolo di un unico *session_id*, utilizzato dai protocolli di livello superiore**

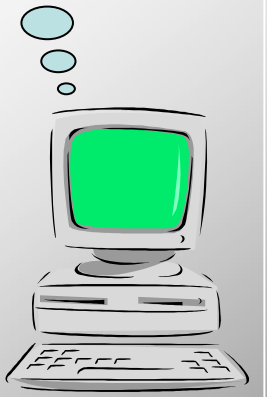
• Progettato per essere utilizzato su un livello di trasporto affidabile (TCP/IP)
• Autenticazione dell'utente non implementata a questo livello, bensì dal protocollo al livello più alto
• Progettato per essere semplice e flessibile: sono necessari in media solo 2 (max 3) round trip per una operazione di connessione

Connessioni crittografate

CONNESSIONE TCP/IP

PORTA 22

“SSH-protoversion-softwareversion-commenti”



Scambio di stringa della versione di SSH

- Entrambe le parti inviano una stringa di versione con la seguente forma: “*SSH-protoversion-softwareversion comments*”

- Utilizzata per indicare le capacità di un’implementazione

- La versione attuale del protocollo è 2.0

- Tutti i pacchetti a seguire adottano la convezione del Binary Packet Protocol

Transport Layer Protocol

Negoziazione algoritmi

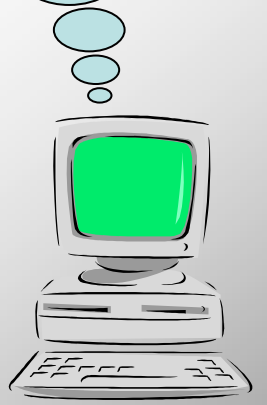
Spero che S usi questi stessi algoritmi

Contiene:
1) Lista di algoritmi supportati per ogni Categoria (algoritmo a chiave pubblica, cifratura, MAC, compressione) in ordine di preferenza
2) Cookie: valore casuale utilizzato per lo scambio delle chiavi

Spero che C usi questi stessi algoritmi

Client

SSH_MSG_KEXINIT



Server

Tutti gli algoritmi preferiti coincidono

Qualche algoritmo non coincide

OK
Prosegui

Successivo algoritmo compatibile nella lista del Client
(se non esiste disconnetti)

Transport Layer Protocol
Scambio di Chiavi

- **Obiettivo**: ottenere dei valori H e K, che verranno utilizzati per il calcolo delle chiavi di sessione
- Il valore H è usato anche come identificatore di sessione
- L'identificatore di sessione (*session_id*)
  - Univoco per connessione
  - Una volta computato non cambia, anche se le chiavi sono rinegoziate in seguito
- **DH-sha1** è l'unico algoritmo richiesto
- La verifica dell'identità del server avviene in questa fase

Transport Layer Protocol

Scambio di Chiavi (DH-sha1)

Obiettivo:
ottenere valori H e K

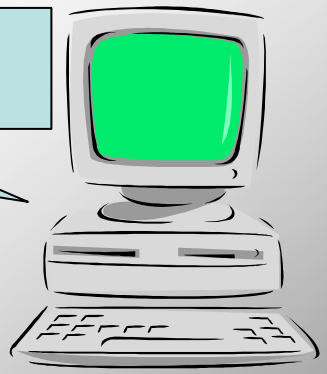
p = numero primo grande
g = generatore di sottogruppo di GF(p)
q = ordine del sottogruppo

Genera y
1<y<q

Genera x
1<x<q

$e = g^x \bmod p$

$K = e^y \bmod p$; $f = g^y \bmod p$
$H = Hash(versionC, versionS, kexinitC, kexinitS, +KS, e, f, K)$

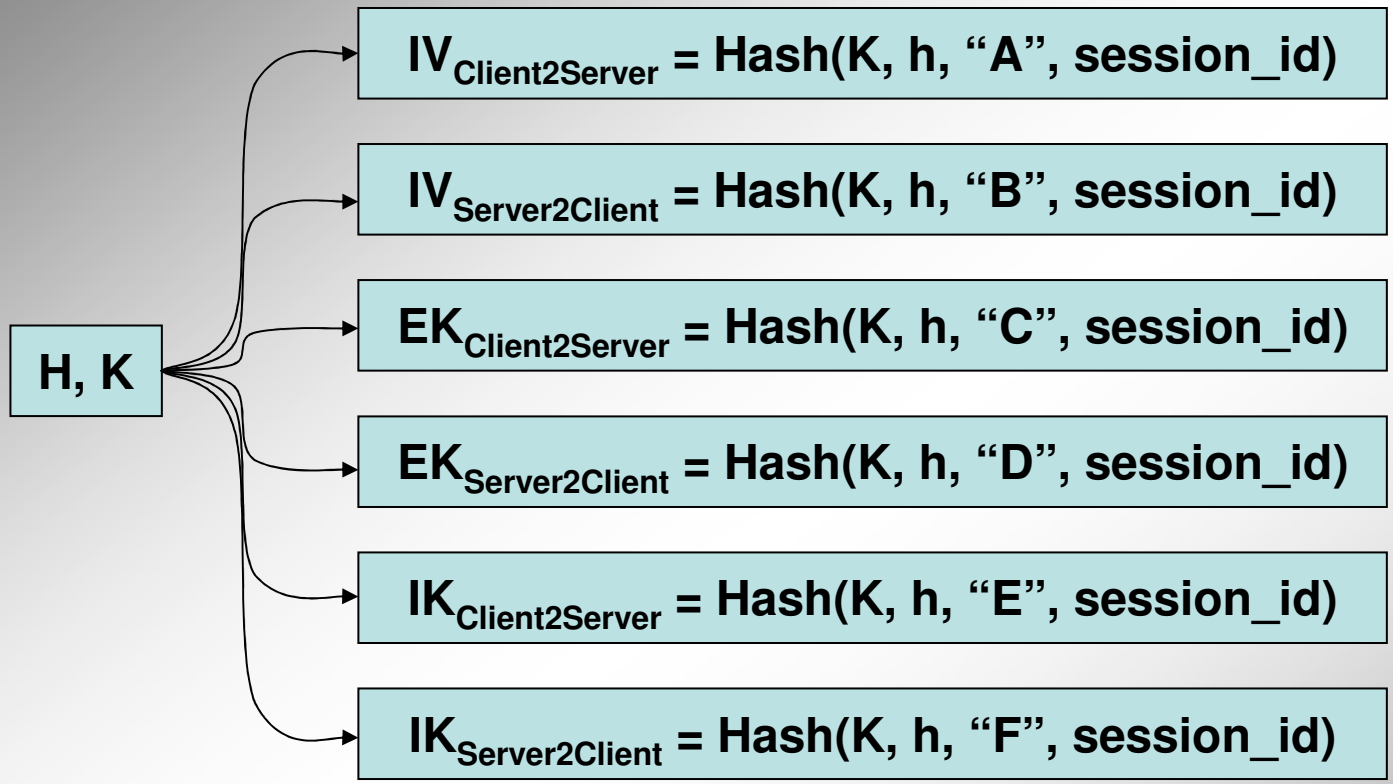


$+K_S$
$f = g^y \bmod p$
$s = sign(H)$ con $-K_S$

Verifica $+K_S$(opzionale) - Autenticazione del Server
con certificato o su DB
$K = f^x \bmod p$
$H = Hash(versionC, versionS, kexinitC, kexinitS, +K_S, e, f, K)$
Verifica della firma s su H (l'host ha $-K_S$)

Transport Layer Protocol

Scambio di Chiavi



- La chiave viene ricavata dai primi n byte dell'hash
- Se la chiave risulta troppo corta si applica la seguente procedura:

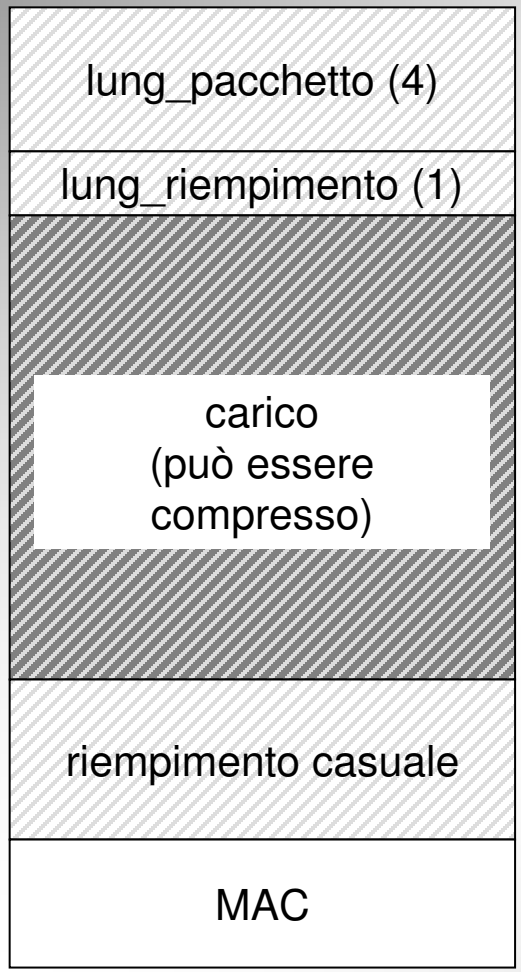
K1 = Hash(K, h, x, session_id)
K2 = Hash(K, h, K1)
K3 = Hash(K, h, K1, K2)
...
chiave = K1 • K2 • K3 •...

Transport Layer Protocol
Rinegoziazione delle chiavi

- Entrambe le parti posso inizializzare una nuova negoziazione di chiavi
  - Viene inviato un pacchetto SSH_MSG_KEXINIT
- Il nuovo processo di scambio di chiavi è identico a quello iniziale
  - la session_id rimane la stessa
- Possono essere cambiati gli algoritmi
- Le chiavi ed i vettori vengono ricalcolati
- I contesti di cifratura vengono riavviati
- E' raccomandabile effettuare uno scambio di chiavi dopo ogni gigabyte trasmesso o dopo un'ora di connessione

Transport Layer Protocol

Formato dei pacchetti



- *lung_pacchetto*:
  lunghezza in byte del pacchetto, MAC escluso
- *lung_riempimento*:
  lunghezza in byte del riempimento
- *carico:*
  contenuto utile del pacchetto
- *riempimento*:
  ottetti casuali per riempire il carico affinché abbia lunghezza multipla di 8 o della lunghezza del blocco del cifrario (scelto il maggiore dei due)
- *MAC*: Message Authentication Code
  calcolato sul pacchetto in chiaro

Criptazione (se già negoziata)

Compressione (se negoziata)

Transport Layer Protocol
Compressione

- Se la compressione è stata negoziata, il campo *payload* sarà compresso utilizzando l'algoritmo stabilito
- I campi *packet_lenght* e *mac* saranno computati in base al campo *payload* compresso
- La cifratura avviene dopo la compressione
- L'algoritmo di compressione viene scelto indipendentemente per ogni direzione
- Sono correntemente definiti i seguenti metodi di compressione:

-nessuno	**RICHIESTO**	**nessuna compressione**
-zlib	**OPZIONALE**	**Compressione GNU ZLIB**

Transport Layer Protocol
Integrità

- Il campo mac è calcolato come segue
- *mac* = **MAC(key, sequence_number • pacchetto_non_cifrato)**
  - ***sequence_number:*** inizializzato a 0 e incrementato per ogni pacchetto
  - ***pacchetto_non_cifrato:*** campi lunghezza, *payload* e *padding*
- L'algoritmo **MAC** viene scelto indipendentemente per ogni direzione
- Sono correntemente supportati i seguenti algoritmi MAC

hmac-sha1	**RICHIESTO**	**HMAC-SHA1 (digest length = key length = 20)**
hmac-sha1-96	**RACCOMANDATO**	**primi 96 bits of HMAC-SHA1 (digest length = 12, key length = 20)**
hmac-md5	**OPZIONALE**	**HMAC-MD5 (digest length = key length = 16)**
hmac-md5-96	**OPZIONALE**	**primi 96 bits of HMAC-MD5 (digest length = 12, key length = 16)**
nessuno	**OPZIONALE**	**nessun MAC; NON RACCOMANDATO**

Transport Layer Protocol
Cifratura

- Sono soggetti a cifratura i campi: *packet_length*, *padding_length*, *payload* e *padding*
- Tutti gli algoritmi utilizzano chiavi di lunghezza > 128 bit
- L'algoritmo di cifratura viene scelto indipendentemente per ogni direzione

Transport Layer Protocol
Cifratura

- Sono correntemente supportati i seguenti cifrari

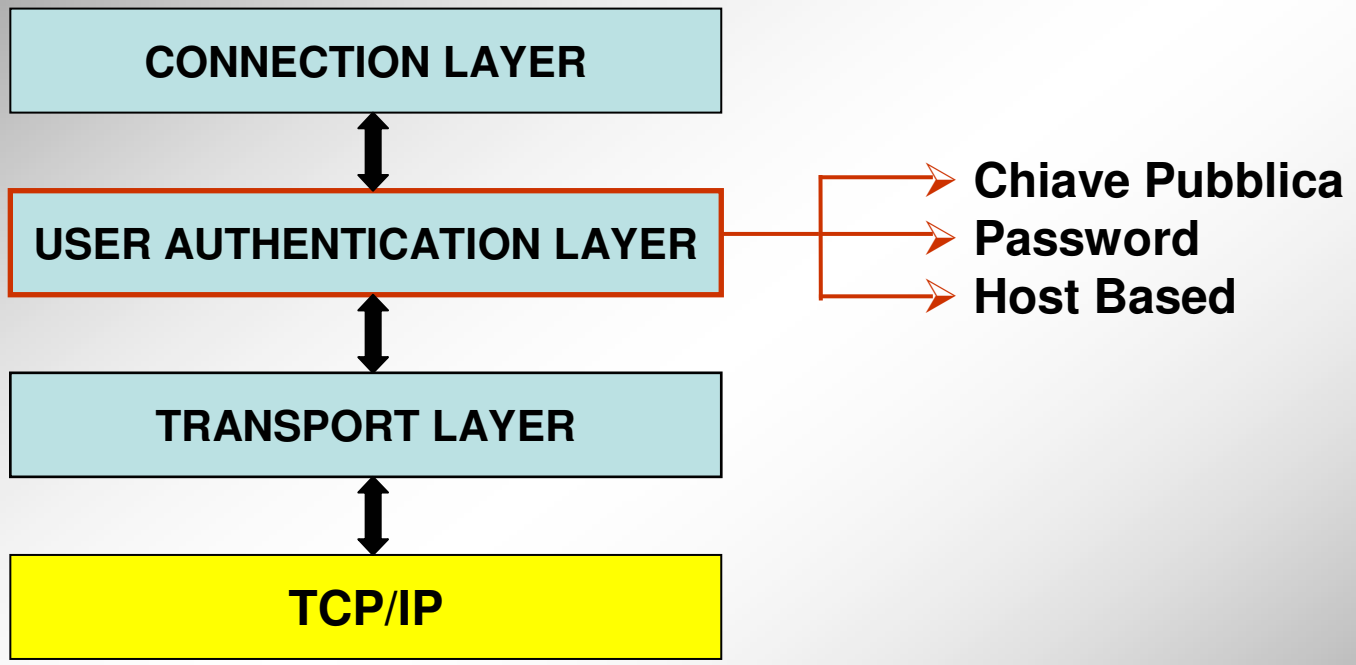
3des-cbc	RICHIESTO	three-key 3DES in modalità CBC
blowfish-cbc	RACCOMANDATO	Blowfish in modalità CBC
twofish256-cbc	OPZIONALE	Twofish in modalità CBC, con chiave a 256-bit
twofish-cbc	OPZIONALE	alias per "twofish256-cbc"
twofish192-cbc	OPZIONALE	Twofish con chiave a 192-bit
twofish128-cbc	RACCOMANDATO	Twofish con chiave a 128-bit
aes256-cbc	OPZIONALE	AES in modalità CBC con chiave a 256-bit
aes192-cbc	OPZIONALE	AES con chiave a 192-bit
aes128-cbc	RACCOMANDATO	AES con chiave a 128-bit
serpent256-cbc	OPZIONALE	Serpent in modalità CBC, con chiave a 256-bit
serpent192-cbc	OPZIONALE	Serpent con chiave a 192-bit
serpent128-cbc	OPZIONALE	Serpent con chiave a 128-bit
arcfour	OPZIONALE	cifrario ARCFOUR per stream
idea-cbc	OPZIONALE	IDEA in modalità CBC
cast128-cbc	OPZIONALE	CAST-128 in modalità CBC
nessuno	OPZIONALE	nessuna cifratura; NON RACCOMANDATO

Transport Layer Protocol
Algoritmi a chiave pubblica

- Il protocollo è stato progettato per operare con quasi tutti i formati di chiave, codifiche ed algoritmi
- Sono supportati i seguenti formati di chiavi pubbliche e di certificati

ssh-dss	**RICHIESTO**	**sign Simple DSS**
ssh-rsa	**RACCOMANDATO**	**sign Simple RSA**
x509v3-sign-rsa	**RACCOMANDATO**	**sign X.509 certificates (RSA key)**
x509v3-sign-dss	**RACCOMANDATO**	**sign X.509 certificates (DSS key)**
spki-sign-rsa	**OPZIONALE**	**sign SPKI certificates (RSA key)**
spki-sign-dss	**OPZIONALE**	**sign SPKI certificates (DSS key)**
pgp-sign-rsa	**OPZIONALE**	**sign OpenPGP certificates (RSA key)**
pgp-sign-dss	**OPZIONALE**	**sign OpenPGP certificates (DSS key)**

Authentication Protocol



- Autentica l'utente-client al server
- All'avvio del protocollo, lo strato di trasporto gli fornisce l'identificatore di sessione
(Il valore H del primo pacchetto di scambio di chiavi)

Authentication Protocol

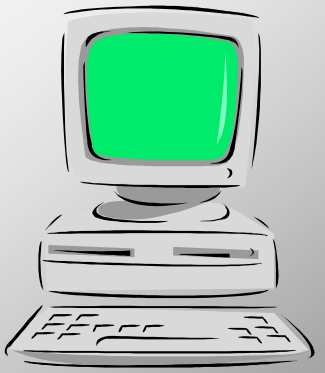
- Si assume che il protocollo sottostante fornisca protezione dell'**integrità** e **confidenzialità**
- 3 metodi di autenticazione:
  - **Chiave Pubblica**
  - **Password**
  - **Host Based**
- Il server guida l'autenticazione:

Metodi di autenticazione consentiti

Scelgo il più conveniente tra quelli che supporto

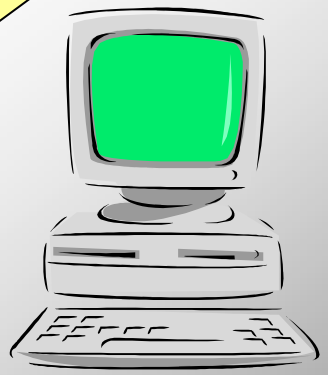
Posso andare in *sleep* dopo un certo num di autenticazioni fallite

Ho stabilito un timeout ed un numero max di tentativi



Authentication Protocol
Schema generale

byte SSH_MSG_USERAUTH_REQUEST
string user name
string service name //servizio da avviare successivamente all' autenticazione
string method name (US-ASCII)
The rest of the packet is method-specific



No — Accetta Autenticazione? — Si

Forse

byte SSH_MSG_USERAUTH_SUCCESS

byte SSH_MSG_USERAUTH_FAILURE
string auth can continue
boolean partial success = TRUE

byte SSH_MSG_USERAUTH_FAILURE
string auth can continue
boolean partial success = FALSE

DISCONNECT

Successo parziale: altri passi richiesti

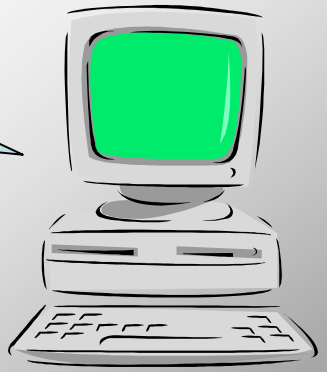
"Auth can continue" è una lista di metodi che possono proseguire produttivamente il dialogo dell'autenticazione

Authentication Protocol
Autenticazione a Chiave Pubblica

- Questo metodo deve essere supportato da ogni implementazione
- Autenticazione basata su algoritmi a chiave pubblica

byte SSH_MSG_USERAUTH_REQUEST
string user name
string service
string "publickey"
string public key algorithm name
string public key blob
(puo' contenere certificati)

Verifica:
Chiave conosciuta
Certificati



SSH_MSG_USERAUTH_PK_OK

Campi del pacchetto SSH_MSG_USERAUTH_REQUEST
string signature

signature = firma digitale del session_id seguito dai campi precedenti

SSH_MSG_USERAUTH_SUCCESS

Authentication Protocol
Autenticazione a Chiave Pubblica

Metodo RSAAuthentication

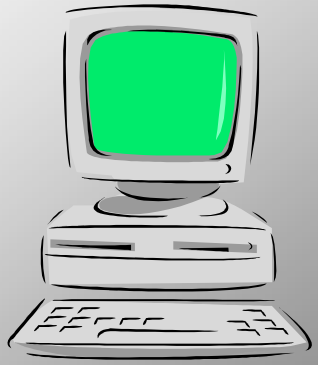
Ogni utente possiede una coppia di chiavi pubblica e privata:
- **$HOME/.ssh/identity (-rw-------) contiene la chiave privata**
- **$HOME/.ssh/identity.pub (-rw-rw-r--) contiene la chiave pubblica**

Il server conosce le chiavi pubbliche dei clienti autorizzati a collegarsi:
- **$HOME/.ssh/authorized_keys (-rw-r--r--) contiene le chiavi degli utenti autorizzati**

Challenge-response

Manda una challenge cifrata con la chiave pubblica dell'utente

L'utente decifra la challenge con la sua chiave privata e la rimanda in chiaro

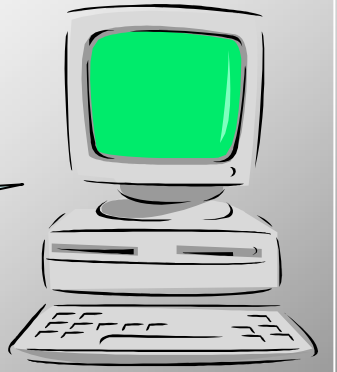


Authentication Protocol
Autenticazione con Password

- Tutte le implementazioni dovrebbero supportare questo metodo dato che non è obbligatorio il possesso di una chiave pubblica

byte	**SSH_MSG_USERAUTH_REQUEST**
string	**user name**
string	**service**
string	**"password"**
boolean	**FALSE**
string	**plaintext password (1)**

Verifica password



SSH_MSG_USERAUTH_SUCCESS (2)

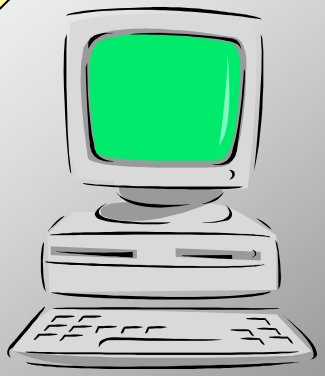
NOTE
(1) E' compito del livello sottostante fare in modo che la password non viaggi in chiaro
(2) Il server può richiedere che la password venga cambiata

Authentication Protocol
Autenticazione Host Based

- Questo tipo di autenticazione è opzionale (è simile a Rhosts di UNIX)
- Si basa su **nome host**, sulla **host key** del client e sul **nome utente**

byte	SSH_MSG_USERAUTH_REQUEST
string	user name
string	service
string	"hostbased"
string	public key algorithm for host key
string	public host key and certificates
string	client host name
string	client user name on the remote host
string	signature

firma digitale del session_id seguito dai campi precedenti



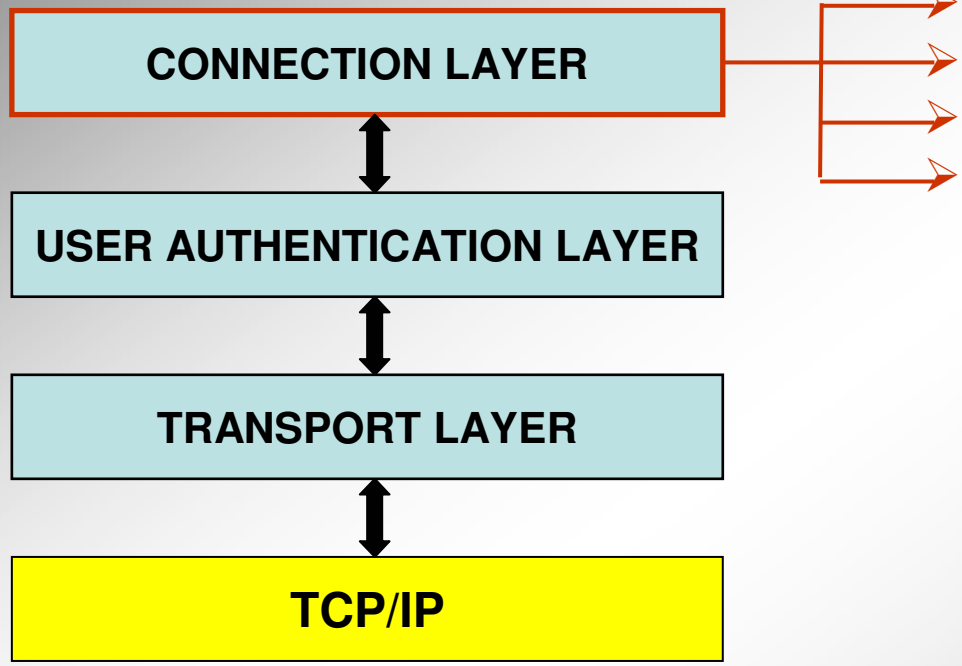
Controlli:
Chiave conosciuta o certificati
Verifica della firma
Matching nome host – chiave
Matching nome utente sul client - nome utente sul server

SSH_MSG_USERAUTH_SUCCESS

Authentication Protocol
Considerazioni sulla sicurezza

- Si assume che il protocollo di trasporto sicuro abbia già
  – Autenticato la macchina server
  – Stabilito un canale di comunicazione cifrato
  – Computato un identificatore di sessione
- Il server può andare in uno stato di *sleep* dopo ripetute autenticazioni fallite
- Se il livello di trasporto non utilizza il MAC, non deve essere possibile cambiare la password per evitare attacchi del tipo *denial of service*
- La politica locale decide quali metodi accettare per ogni utente
- I messaggi di debug possono essere disabilitati per evitare che forniscano troppe informazioni sull'host

Connection Protocol

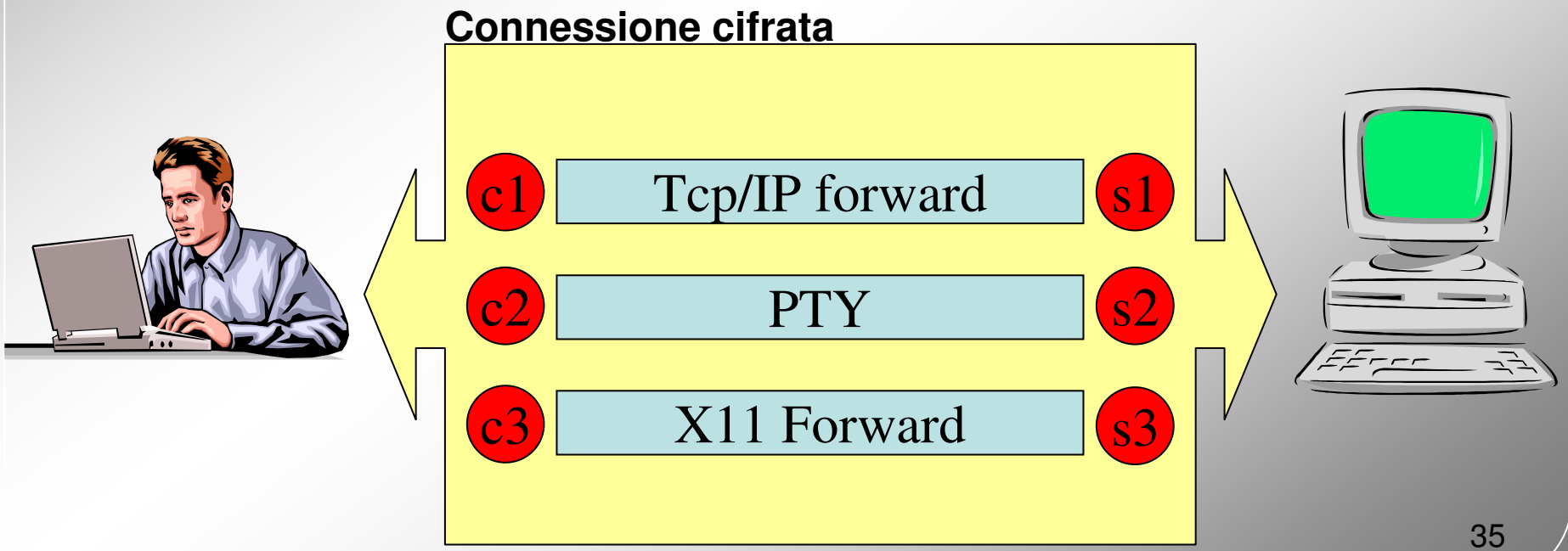


Fornisce:
Sessioni interattive di login
Esecuzione remota di comandi
Inoltro di connessioni TCP/IP
Inoltro di connessioni X11

Divide la connessione in canali logici;
tutti questi canali sono *multiplexati* in un singolo tunnel cifrato,
facente uso di una connessione con protocollo di trasporto protetto
e con meccanismo di integrità

Connection Protocol
Meccanismo dei canali

- Tutte gli pseudo-terminali, connessioni inoltrate, ecc. sono canali
- I canali sono identificati da numeri (ad entrambe le parti), che possono essere differenti tra client e server
- Sono tipizzati: “session”, “x11”, “forwarded-tcpip”,“direct-tcpip”...
- I canali hanno un meccanismo di controllo del flusso attuato tramite finestre e nessun dato può essere inviato attraverso un canale fin quando non c’è sufficiente “window space”



Meccanismo dei canali: Schema generale

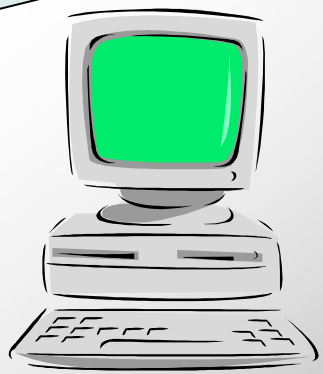
Session, X11, Forwarded-tcpip, direct-tcpip

Identificatore locale scelto dal richiedente del canale

Specifica quanti Bytes possono essere mandati

Max dimensione del pacchetto accettata dal canale

```
byte SSH_MSG_CHANNEL_OPEN
string      channel type
uint32      sender channel
uint32      initial window size (in bytes)
uint32      maximum packet size (in bytes)
... channel type specific data follows
```



byte SSH_MSG_CHANNEL_OPEN_CONFIRMATION
... Seguono altri campi

oppure

byte SSH_MSG_CHANNEL_OPEN_FAILURE
... Seguono altri campi

...

```
byte        SSH_MSG_CHANNEL_DATA
uint32      recipient channel
string      data ...
```

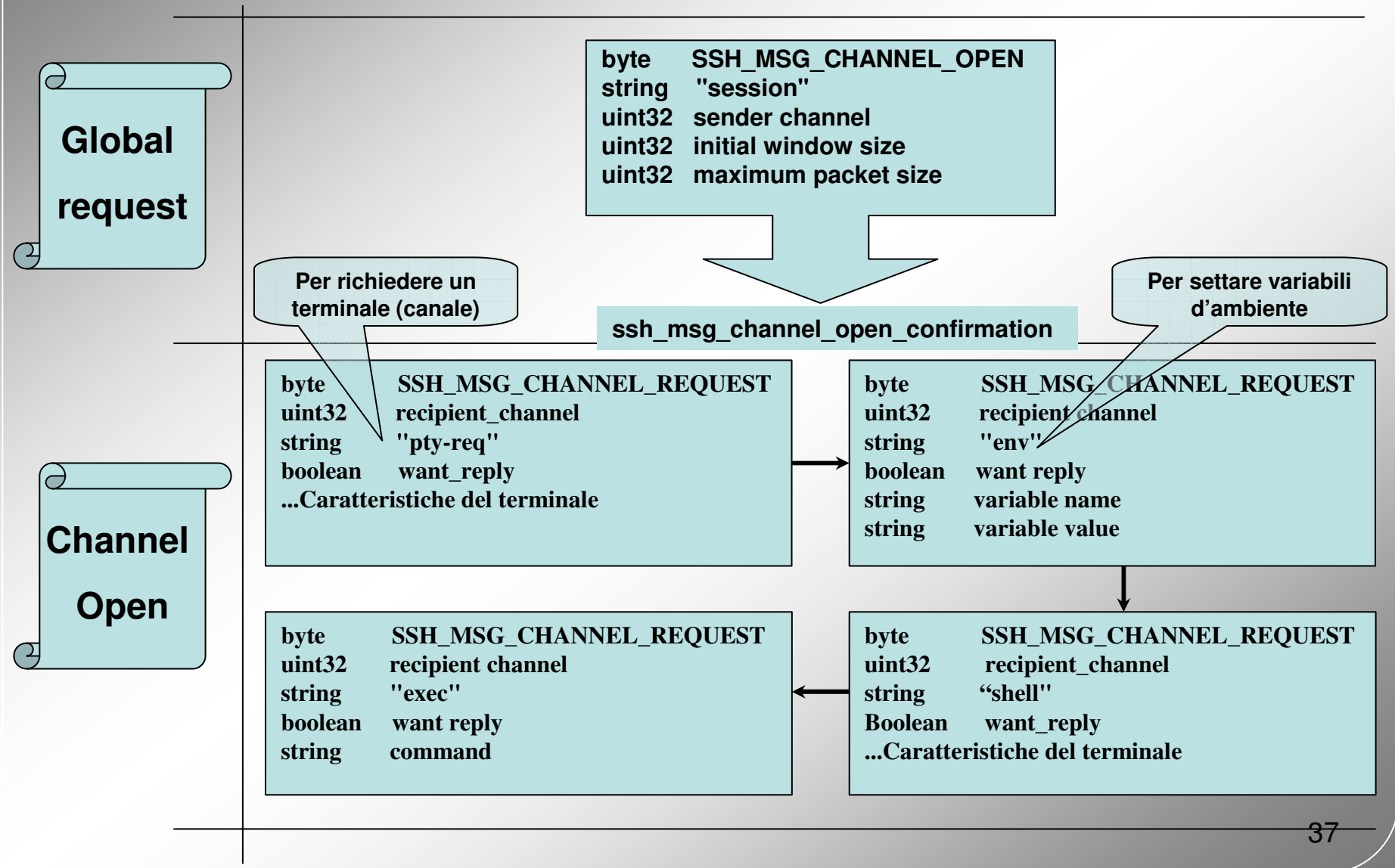
```
byte        SSH_MSG_CHANNEL_EOF
uint32      recipient_channel
```

```
byte        SSH_MSG_CHANNEL_CLOSE
uint32      recipient_channel
```

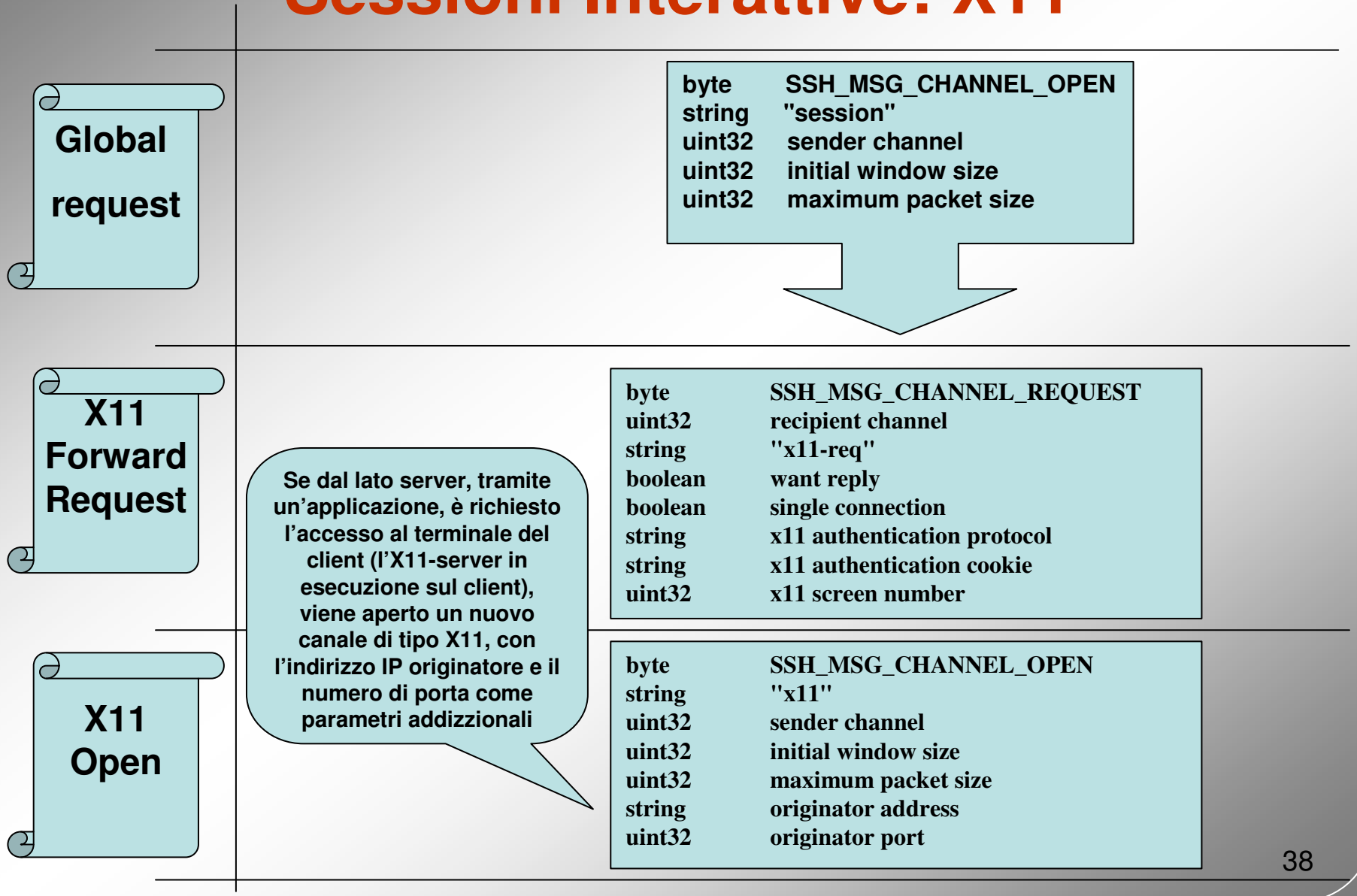
```
byte        SSH_MSG_CHANNEL_CLOSE
uint32      recipient_channel
```

NOTE:
1) Quando una delle parti non vuole più utilizzare un canale dovrebbe inviare un EOF
2) Dopo un EOF il canale resta aperto e possono essere spediti messaggi in direzione opposta
3) Un canale è considerato chiuso quando è stato sia inviato che ricevuto un messaggio di chiusura

Connection Protocol
Sessioni Interattive & avvio shell

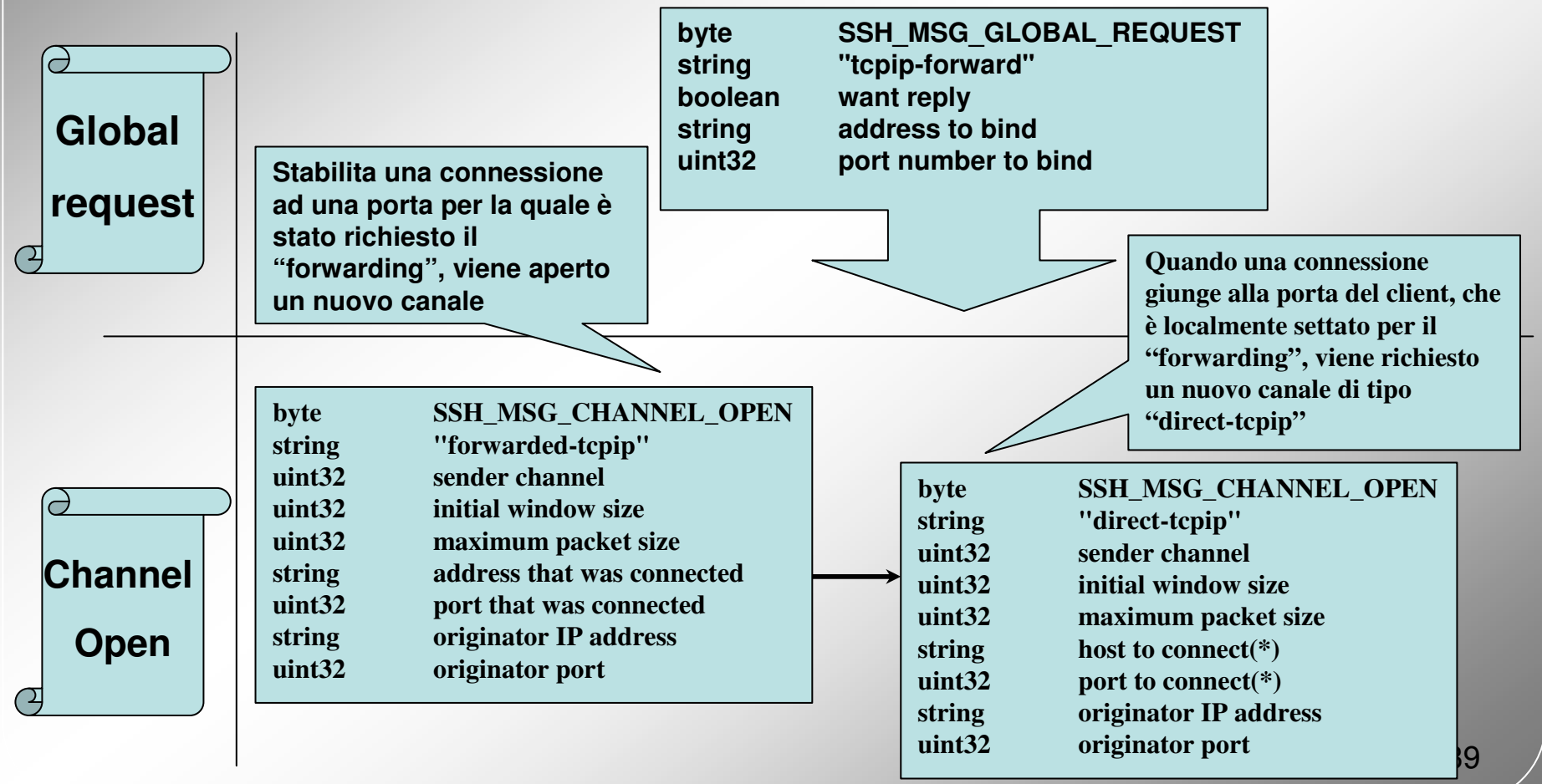


Connection Protocol
Sessioni Interattive: X11



Connection Protocol
TCP/IP Port Forwarding

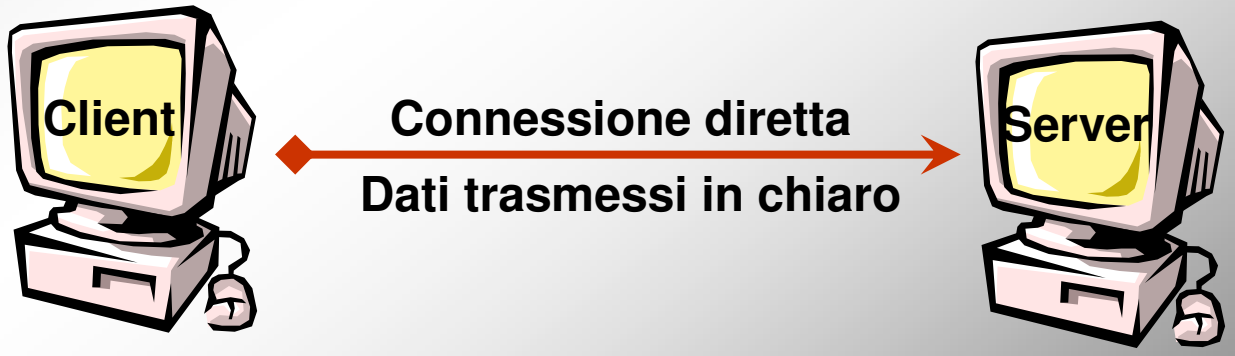
- permette di creare un canale di comunicazione sicuro attraverso il quale veicolare qualsiasi connessione TCP



Port Forwarding, tunneling

- **Port Forwarding è la tecnica utilizzata per intercettare i pacchetti destinati ad una specifica porta e macchina TCP o UDP , e reindirizzarli (Forward) ad una differente porta e/o macchina. Questo viene fatto in modo trasparente, ovvero i clients in rete non possono vedere che è stato fatto un port forwarding. Essi si connettono ad una porta su di una macchina quando in realtà i pacchetti vengono reindirizzati altrove.**
- **SSH abilita il "forwarding", di ogni flusso di dati TCP, attraverso un *tunnel* nell'ambito di una sessione SSH.**
- **Questo significa che il flusso di dati, invece di essere direttamente gestito dalle porte del client e del server, è incapsulato in un tunnel creato a tempo di connessione (sessione).**
- **Ciò è reso particolarmente agevole dal protocollo X11, con gestione trasparente delle finestre e permette una propagazione continua quando si è in presenza di salti multipli.**

Connessione diretta tra Client-Server



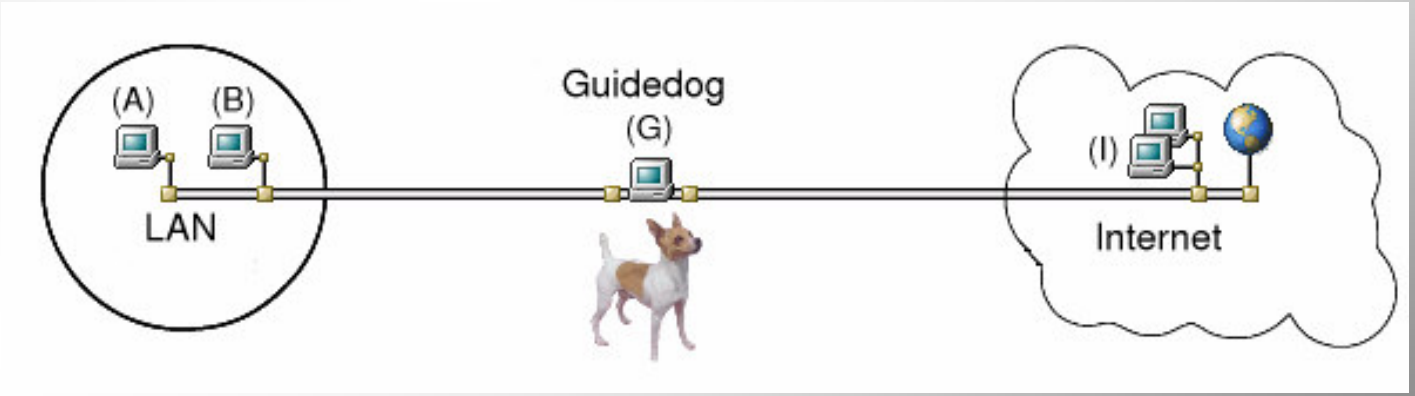
Port Forwarding, tunneling

Abbiamo 2 macchine (A e B) su una rete locale, sono connesse ad internet attraverso un gateway (G) il quale ha in esecuzione "Guidedog".

Immaginiamo si voler reindirizzare il traffico della porta TCP 80 sulla macchina G verso la porta 80 di un altro computer su internet (I). I pacchetti provenienti dalla LAN (A e B) dovrebbero andare alla porta 80 sul Gateway(G), e poi Guidedog dovrebbe reindirizzarli sulla nuova destinazione su internet (I).

Quello che non funzionerebbe è se si prova ad usare Guidedog per reindirizzare una porta appartenente alla macchina B. I pacchetti che passano attraverso Guidedog verranno reindirizzati, ma i pacchetti provenienti da A no perché quest ultimo comunica direttamente con B senza attraversare il Gateway.

Ciò significa che Guidedog deve essere in esecuzione su una macchina che funge anche da Gateway per la rete locale.



TCP/IP Port Forwarding

- L'applicazione Client si connette al Client SSH che cifra tutti i dati prima della trasmissione.
- Il Client SSH reindirizza i dati cifrati verso il Server SSH, che li decifra e li reindirizza all'applicazione Server.
- I dati inviati dall'applicazione Server sono cifrati allo stesso modo dal Server SSH e reindirizzati indietro al Client.

Interfacce ascoltate: 127.0.0.1 (Questo assicura che solo le connessioni provenienti dalla macchina Client locale, o connessioni loopback, siano accettate per il forwarding)
Host Destinazione: 127.0.0.1 (L'indirizzo target è relativo al server, non al client, quindi 127.0.0.1 funzionerà bene se l'applicazione server obiettivo sta ascoltando su tutte le interfacce 0.0.0.0. Questo è l'indirizzo al quale il Server SSH si connetterà quando occorrerà reindirizzare una connessione)

Connection Protocol
Considerazioni sulla sicurezza

- Si assume che questo protocollo giri al di sopra di un protocollo di trasporto sicuro che abbia già
  - ***Autenticato la macchina server***
  - ***Stabilito un canale di comunicazione cifrato***
  - ***Computato un identificatore di sessione***
  - ***Autenticato l'utente***
- Migliorata la sicurezza per connessioni al server X11
- Il ***port forwarding*** può potenzialmente permettere ad un intruso di scavalcare sistemi di sicurezza come i firewall; tuttavia questo poteva già essere fatto in altri modi
- In quanto cifrato, il traffico non può essere esaminato da un firewall

Considerazioni sulla sicurezza e flessibilità

- Algoritmi di crittografia e autenticazione ben conosciuti e testati
- Chiavi di taglia sufficiente a garantire protezione a lungo termine
- Algoritmi negoziati: in caso di rottura di uno di essi, è possibile utilizzarne altri senza modifiche al protocollo

OpenSSH

- Suite di protocolli gratuita
- Caratteristiche principali
  - Progetto Open Source
  - Licenza Gratuita
  - Crittografia forte (3DES, Blowfish)
  - X11 Forwarding (cifratura del traffico X11)
  - Inoltro di porte (canali criptati per altri protocolli)
  - Autenticazione Forte (Chiave Pubblica, One-Time Password e Kerberos)
  - Interoperabilità (Conformità con SSH 1.3, 1.5, e gli Standard del protocollo 2.0)
  - Supporto per client e server SFTP nei protocolli SSH1 e SSH2
  - Compressione Dati

File Interessati

- **$HOME/.ssh/known_hosts**
  Registrano le chiavi pubbliche degli host in cui l'utente ha effettuato il login
- **$HOME/.ssh/identity, $HOME/.ssh/id_dsa**
  Contengono le chiavi private RSA SSH1 e DSA dell'utente rispettivamente
- **$HOME/.ssh/identity.pub, $HOME/.ssh/id_dsa.pub**
  Contengono le chiavi pubbliche RSA SSH1e DSA dell'utente rispettivamente
- **$HOME/.ssh/config**
  File di configurazione per l'utente usato dal client SSH
- **$HOME/.ssh/authorized_keys**
  Lista le chiavi pubbliche RSA SSH1 degli utenti autorizzati ad effettuare il login

Comandi Principali

- **ssh [<opzioni>] <host>[<comando>]** '**ssh**' è in grado di instaurare una connessione per l'accesso presso un servente in cui sia in funzione il demone **sshd**'

  Alcune opzioni

  -l <utente> specifica il nominativo-utente remoto

  -i <file-di-identificazione> specifica il file con la chiave privata diverso da quello di default

- **scp [<opzioni>][<utente>@]<host>:]<origine file>...[[<utente>@]<host>:]<destinazione file>**
  '**scp**' usa '**ssh**' per la copia di file tra elaboratori

  Alcune opzioni:

  -p Fa in modo che gli attributi originali dei file vengano rispettati il più possibile nella copia

  -r Copia ricorsiva delle directory

Esempi

- $ ssh -l tizio linux.brot.dg ls -l /tmp

User Name | Host Name | Comando

- $ ssh -l tizio linux.brot.dg tar czf-/home/tizio>backup.tar.gz

Comando:
Copia della directory personale dell'utente'tizio' nell'elaboratore remoto. L'operazione genera il file 'backup.tar.gz' nella directory corrente dell'elaboratore locale

- $ scp tizio@linux.brot.dg:/etc/profile

User Name, Host e file remoto da copiare

Directory destinazione locale