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

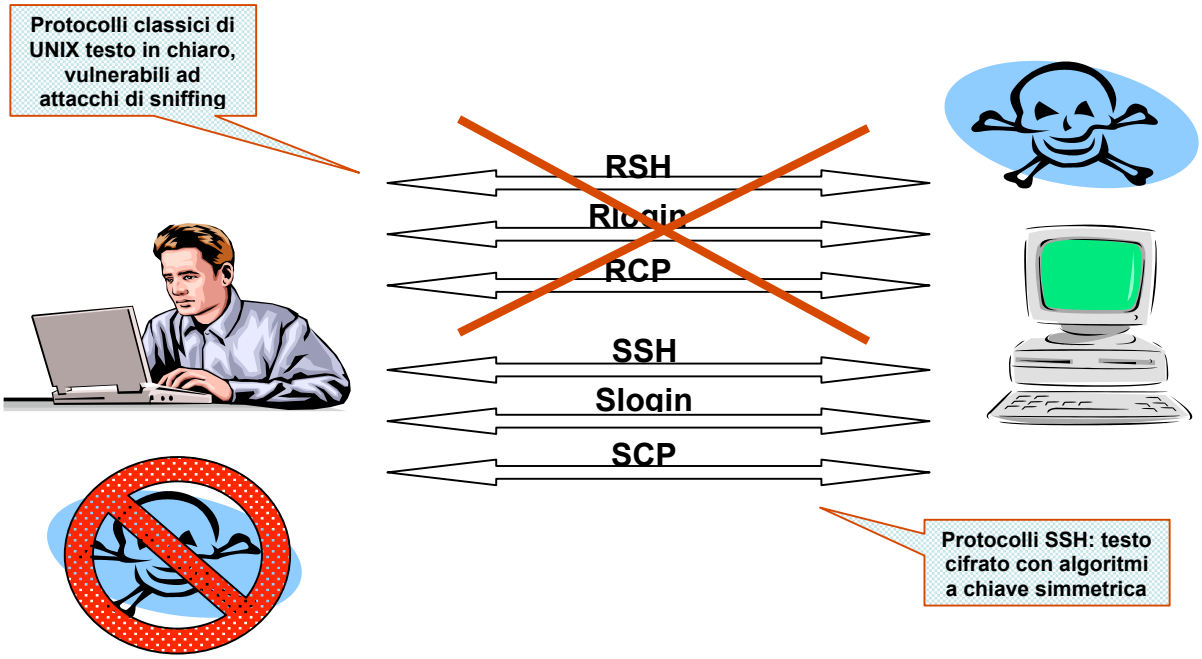
Cesario Daniele
Cangemi Carlo
Ruggeri Chiara

Cos'è l'SSH?

E' un protocollo che permette di stabilire una sessione remota interagendo con un'altra macchina e permettendo l'esecuzione di comandi, attraverso una connessione criptata.
SSH significa "Secure SHell" ed è una versione sicura dell' RSH, "Remote SHell". Nato, infatti, per rimpiazzare i comandi Berkeley r* (**rsh, rlogin, rcp**) con le rispettive versioni sicure (**ssh, slogan, scp**), fornendo in esse una serie di servizi quali:

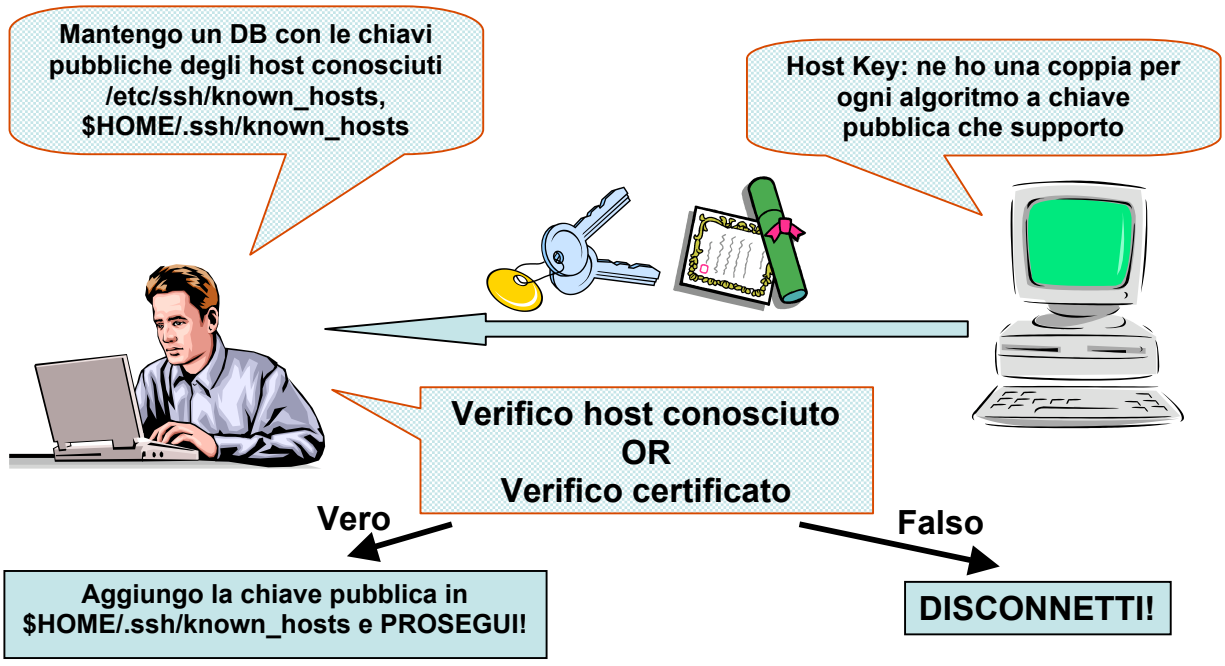
- Una infrastruttura per connessioni crittografate.
- Un'autenticazione forte tra host e host e tra utente e host.
- La possibilità di creare un canale di comunicazione sicuro attraverso il quale veicolare qualsiasi connessione TCP/IP.
- La chiusura di alcuni noti problemi di sicurezza dei protocolli TCP/IP come:
  - l'*IP spoofing* (falsificazione dell'indirizzo IP del mittente)
  - il *DNS spoofing* (falsificazione delle informazioni contenute nel DNS)
  - il *Routine spoofing* (falsificazione delle rotte intraprese dai pacchetti e instradamento su percorsi diversi)

Connessioni crittografate



Autenticazione forte

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



Ogni Host ha una coppia di chiavi (*host key*):

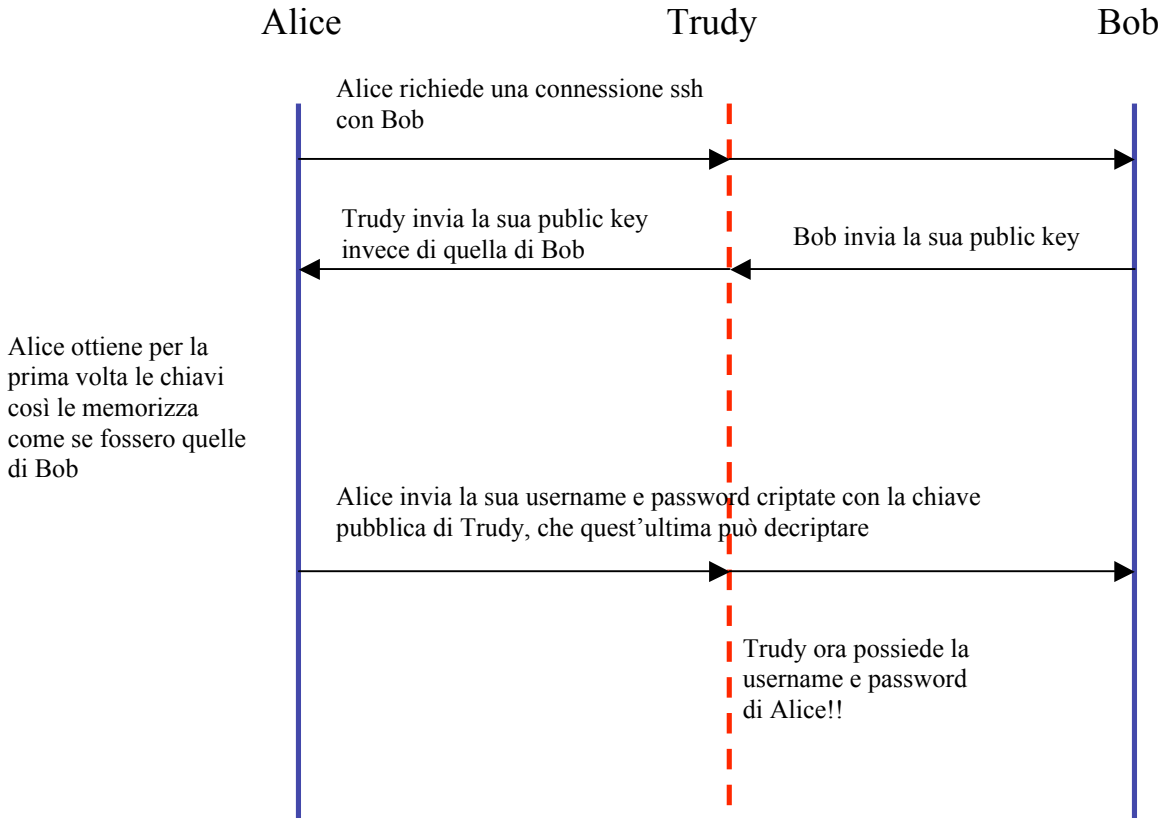
- */usr/local/etc/ssh_host_key* contiene la chiave privata
  - creato all'installazione
  - owned dalla root
  - readable and writable solo dalla root
- */usr/local/etc/ssh_host_key.pub* contiene la chiave pubblica
  - creato all'installazione
  - owned dalla root
  - writable solo dalla root, readable da tutti

Le due chiavi sono legate matematicamente tra di loro e non è in alcun modo possibile, in una scala di tempi ragionevolmente utile, derivare l'una dall'altra. La peculiarità di questi algoritmi a chiave asimmetrica è che un messaggio codificato con una delle chiavi può essere decodificato solo ed esclusivamente utilizzando l'altra chiave.
Il server possiede una coppia di chiavi, ed eventualmente di certificati, per ogni algoritmo a chiave pubblica da esso supportato.

L'host remoto (client) verificherà la corrispondenza tra un server e la sua chiave pubblica

- Il client confronta la server host public key con quella memorizzata in un database locale nei file */etc/ssh/known_hosts* (file di sistema) e *$HOME/.ssh/known_hosts* (file utente)
- Normalmente, ma non necessariamente, un client accetta host public key sconosciuti e le memorizza nel database per utilizzi futuri
- Il protocollo può essere soggetto al man-in-the-middle

The Man in the middle Attack



Autenticazione Client → Server

• PROBLEMA: Metodo classico di autenticazione *rhost* di UNIX vulnerabile ad attacchi di spoofing.

RhostsAuthentication
Questo metodo si basa sulla normale autenticazione dei comandi r*: se il nome dell'host client è inserito all'interno del file */etc/hosts.equiv* sul server allora ad un utente è concesso il login senza password a patto che abbia uno username sul server uguale a quello sul client; analogamente è concesso il login senza password se la coppia "client username/client host" è inserita all'interno del file *$HOME/.rhosts* sul server (dove *$HOME* è la home directory dell'utente sul server).

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)**
   - **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

DNS SPOOFING

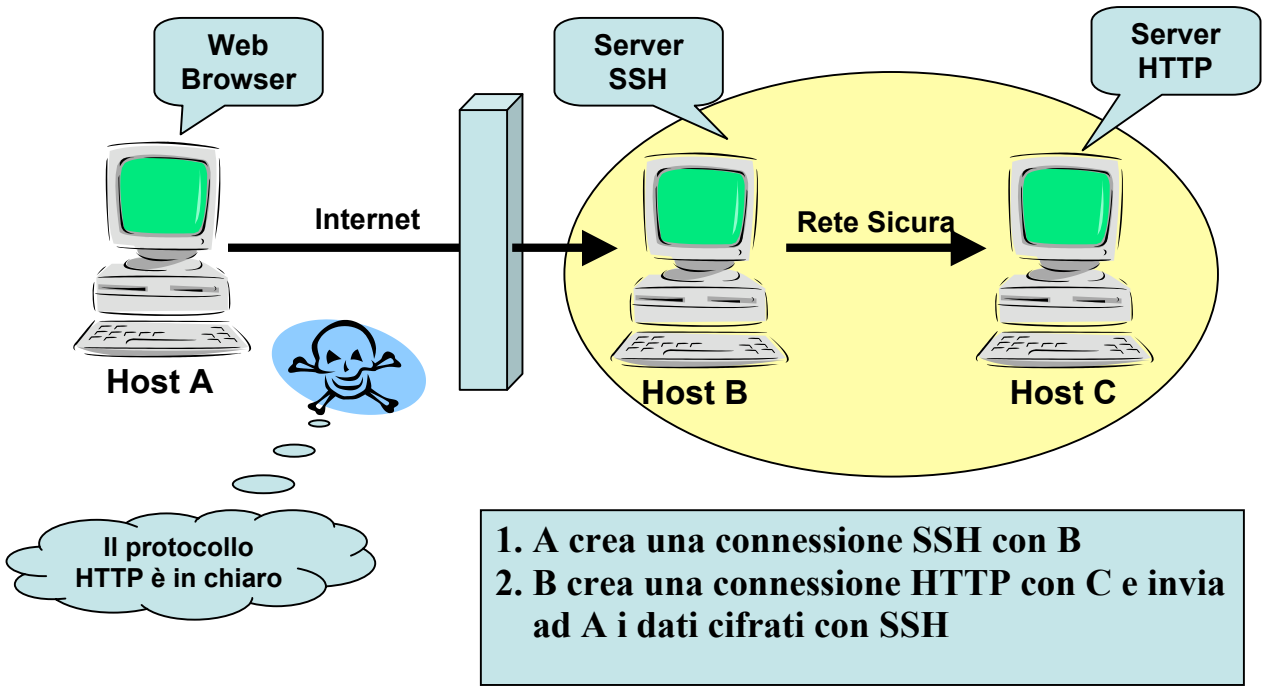
Azioni per un tipico attacco DNS SPOOFING da parte di un hacker (H):
1. **Scegliere l'host target (B)**
2. **Scoprire il nome host di un host fidato(A)**
3. **Falsificare il DNS di B facendo in modo che il nome host di A venga risolto con l'indirizzo IP di H**
4. **Stabilire una connessione con B, fingendosi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



Caratteristica interessante di ssh è la possibilità di effettuare una codifica di qualsiasi tipo di connessione, creando così un canale di comunicazione crittografato tra client e server sul quale veicolare la connessione vera e propria. Questa caratteristica è chiamata **port forwarding**, di cui ne parleremo in seguito.

Credit e Versioni

Esistono due versioni del protocollo SSH, incompatibili fra loro:

- La versione 1.x (1.3 e 1.5) la quale possiede un protocollo "integrato"
- La versione 2, la quale ridefinisce il precedente in tre "strati"rendendo il protocollo molto più sicuro:
  1. SSH Transport Layer Protocol (SSH-TRANS)
  2. SSH Authentication Protocol (SSH-AUTH)
  3. SSH Connection Protocol (SSH-CONN)

La tabella seguente riassume le principali differenze tra le due versioni:

SSH1	SSH2
Design integrato	Separazione in strati diversi delle funzioni di autenticazione, connessione e trasporto
Integrità via CRC32 (non sicuro)	Integrità via HMAC (hash encryption)
Un canale per sessione	Molteplici canali per sessione
Algoritmo simmetrico per la negoziazione, stesso identificativo di sessione da entrambi le parti	Più metodi e dettagli di negoziazione (chiave simmetrica, public key, compressione…); chiave di sessione separata; compressione ed integrità da entrambi le parti
Solo RSA per l'algoritmo a chiave pubblica	RSA e DSA per algoritmo a chiave pubblica
Chiave di sessione trasmessa dal lato client	Chiave di sessione negoziata attraverso il protocollo Diffie-Hellman
Stessa chiave di sessione per l'intera durata della sessione stessa	Chiave di sessione rinnovabile

Il protocollo SSH è gestito da due società finlandesi:

- **SSH Communications Security** (http://www.ssh.org) che ha sviluppato i protocolli SSH1 e SSH2 e mantiene le distribuzioni principali di SSH
- **Data Fellows** (http://www.datafellows.com) che ha acquistato dalla prima la licenza di vendere e supportare SSH.
  In particolare la data 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 invece può essere utilizzato liberamente solo per uso privato o in ambito educational.

Tuttavia i protocolli ssh1 e ssh2 sono pubblicati come Internet Drafts (con scadenza semestrale) e ci sono alcuni progetti che si occupano di scriverne una implementazione free:

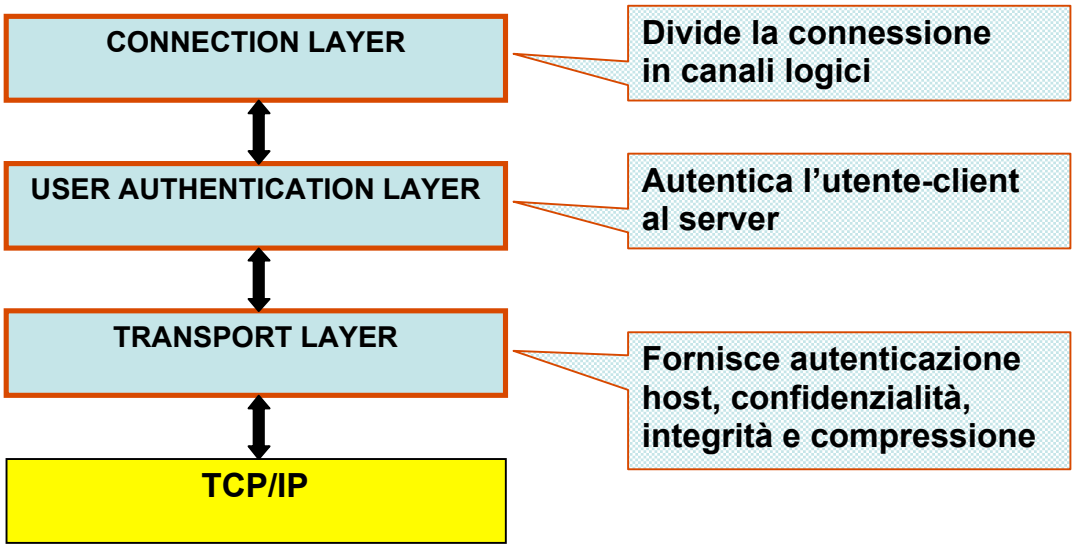
- OpenSSH (http://www.openssh.com), nato per far parte del sistema operativo OpenBSD, implementa il protocollo ssh1 utilizzando solo algoritmi di crittografia senza restrizioni di utilizzo
- OSSH (ftp://ftp.pdc.kth.se/pub/krypto/ossh/), che implementa il protocollo ssh1
- lsh (http://www.net.lut.ac.uk/psst/), che implementa il protocollo ssh2.

Caratteristiche
Politiche Locali

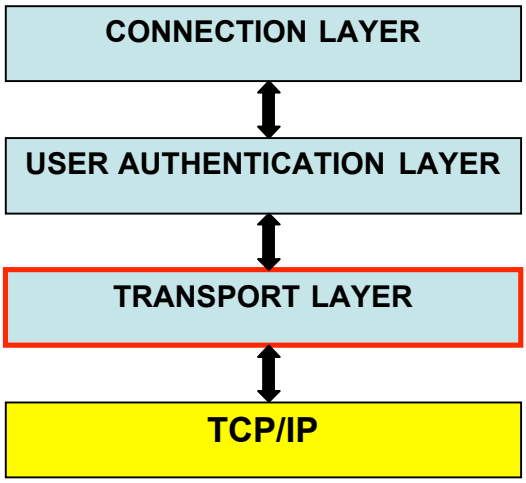
- Piena negoziazione degli algoritmi di crittografia, integrità, scambio delle chiavi e compressione
- Per ogni categoria le politiche locali specificano un algoritmo preferito
- A seconda dell'utente e della sua locazione possono essere richiesti differenti modalità di autenticazione o addirittura autenticazioni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

Architettura

Tre componenti primari:



Transport Layer Protocol



Il protocollo di trasporto fornisce la cifratura basandosi su algoritmi noti: per lo scambio della chiavi usa un algoritmo a chiave pubblica (DSA o RSA da 768 fino a 3072 bit), la cifratura in sé è a chiave simmetrica (DES normale e triplo, Blowfish/Twofish, CAST 128, Arcfour da 56-128-168-256 bit). A questo livello sono contratti anche gli algoritmi *hash* e i Message Authentication Code (HMAC-SHA1, HMAC-MD5, HMAC-RIPEMD 160, con le versioni aggiornate del 96).

Questo strato si occupa delle operazione di:

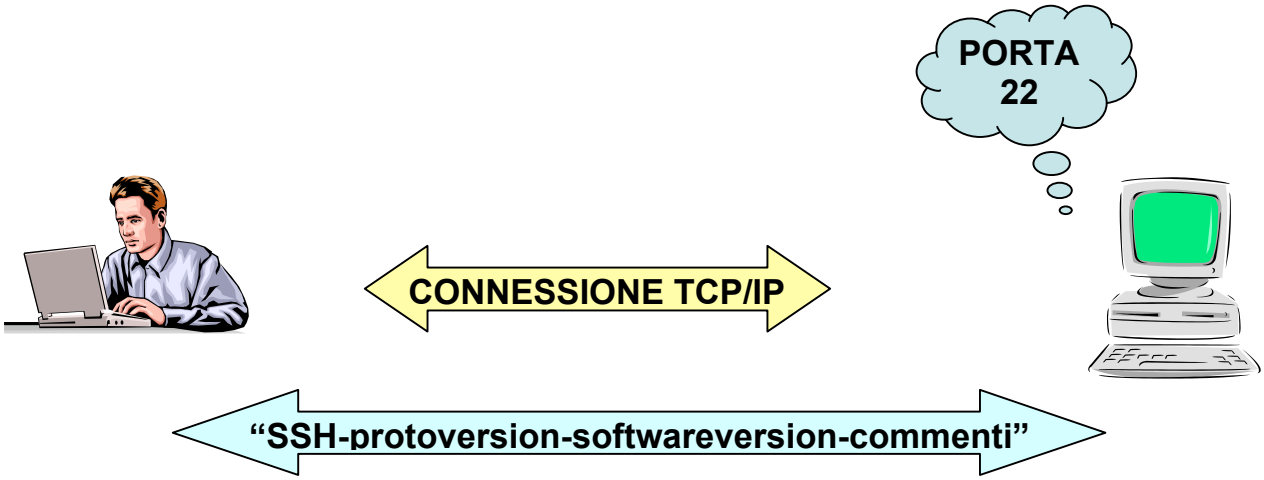
- **Autenticazione dell'host**
- **Scambio di chiavi**
- **Cifratura**
- **Protezione dell'integrità**
- **Calcolo di un unico *session_id*, utilizzato dai protocolli di livello superiore**

Esso è progettato per essere utilizzato su un livello di trasporto affidabile quale il TCP/IP, o sua qualunque altro flusso continuo ed affidabile di dati

E' bene notare che l'autenticazione dell'utente non è implementata a questo livello, bensì dal protocollo al livello più alto, quello, appunto, di autenticazione
Esso, inoltre, è progettato per essere semplice e flessibile: sono necessari in media solo 2 (max 3) round trip per una operazione di connessione.

Connessioni crittografate

All'inizio di una connessione, avviene lo scambio di una stringa trai 2 host che vogliono comunicare utilizzando il protocollo SSH:



Scambio di stringa della versione di SSH

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

Essa viene utilizzata per indicare le capacità dell'implementazione di ssh delle rispettive parti.
Sono presenti al suo interno le versioni major e minor dell'implementazione stessa.
Va ricordato che la versione attuale del protocollo è 2.0
Da questo momento in poi tutti i pacchetti a seguire, inviati tra i due host, adottano la convezione del Binary Packet Protocol che verrà esplicitata successivamente.

Negoziazione algoritmi

A questo punto avviene lo scambio di una nuova stringa: SSH_MSG_KEXINIT

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

Entrambi il server ed il client, sono ora consci delle rispettive capacità. La lista degli algoritmi del client viene confrontata con quella del server per verificare se esiste un punto di incontro tra le due macchine.
Questo confronto avviene in maniera sequenziale, dove l'ordine degli algoritmi è dettato da un criterio di preferenza. Se l'algoritmo preferito dal client per una categoria coincide con quello del server, la negoziazione può proseguire per le altre categorie. In caso contrario si continua a sfogliare la lista di algoritmi conosciuti finchè non viene trovata una coincidenza.
Se si verifica il caso nel quale questo punto di incontro risulti impossibile, la connessione non può avvenire. Il server, a questo punto, invia un messaggio di disconnessione. La connessione non è stata stabilita.
Una volta trovato un algoritmo comune per ogni categoria la connessione prosegue.
E' bene notare che entrambe le parti possono adottare algoritmi diversi per ognuna delle categorie, una volta che è noto che tale algoritmo risulta comprensibile dalla parte con la quale comunica.

Scambio di Chiavi

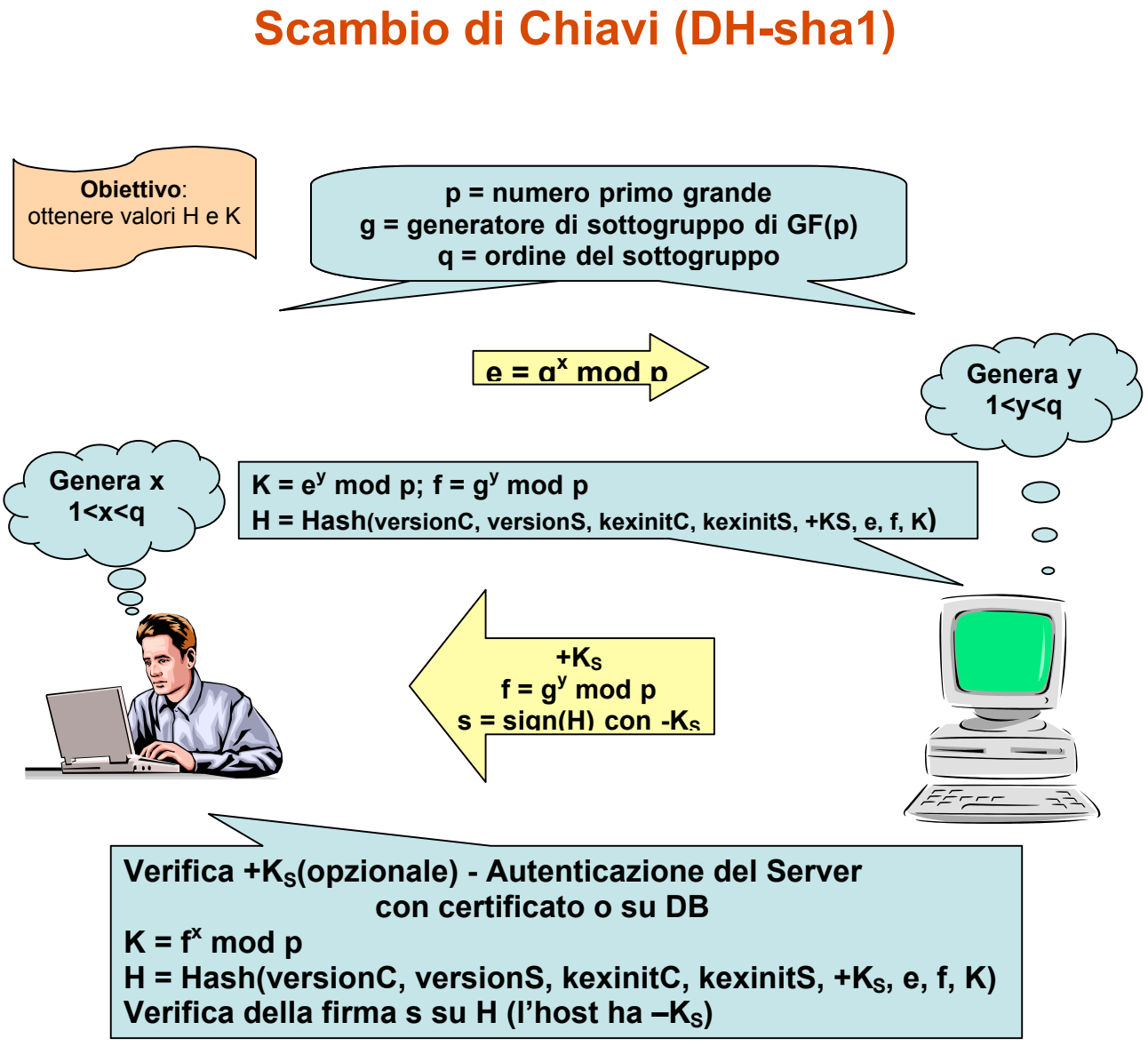
L' obiettivo di questa fase è quello di ottenere dei valori H e K, che verranno utilizzati per il calcolo delle chiavi di sessione utilizzate dal protocollo SSH.

Il valore H verrà inoltre usato come identificatore di sessione univoca, presente per tutta la durata della connessione.

Una volta computato l'identificatore di sessione (*session_id*) non cambia, anche se le chiavi sono rinegoziate in seguito.

Per questa fase l'unico algoritmo richiesto risulta essere il **DH-sha1**.

E' qui che avviene la verifica dell'identità del server, tramite la sua chiave pubblica ed eventuale certificato.



1.

- il client genera un numero casuale x e calcola e = g^x mod p

- il client invia “e” al server

2.

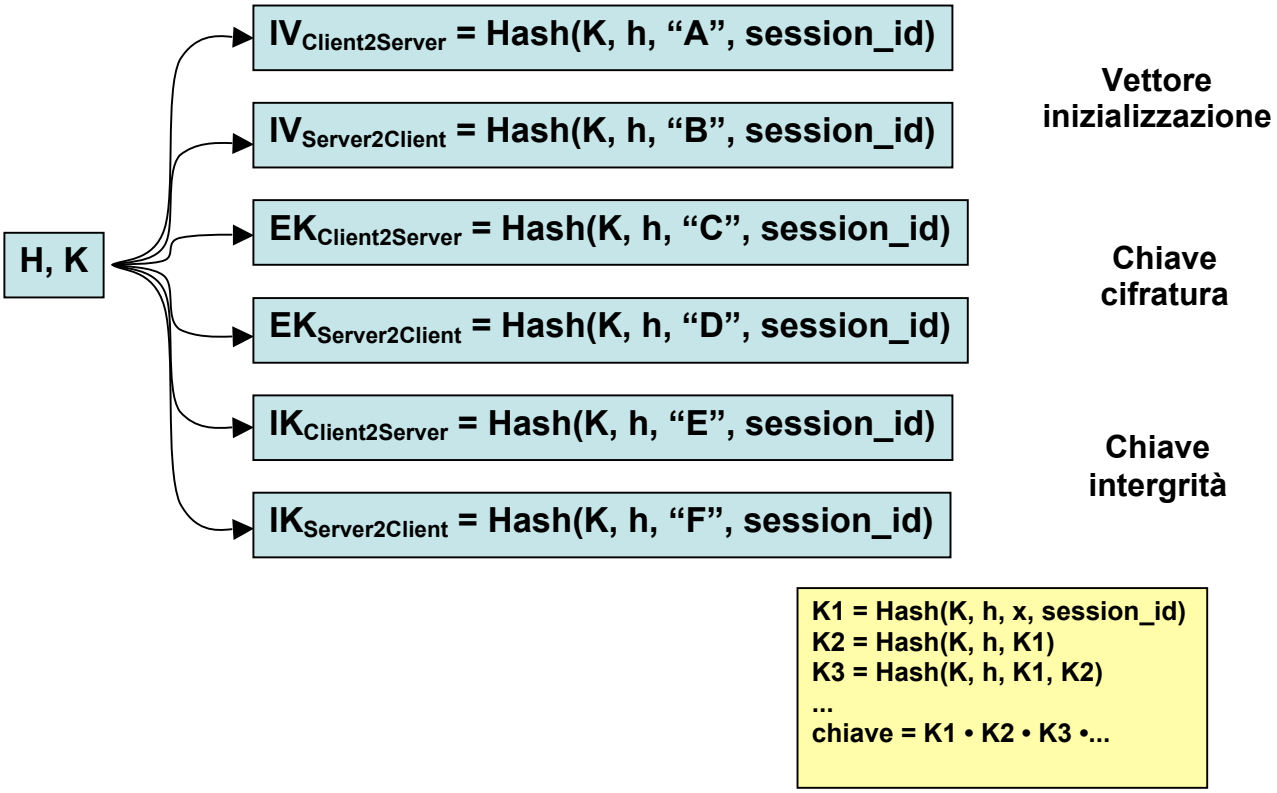
- il server genera un numero casuale y e calcola f = g^y mod p
- il server riceve “e” dal client
- esso calcola K = e^y mod p = g^xy mod p e H = HASH(stringa di versione del client | stringa di versione del server | messaggio kexinit del client | messaggio kexinit del server | chiave del server host | e | f | K)
- esso genera una firma s su H utilizzando la parte privata della chiave del server (questa fase potrebbe risultare in un’ulteriore computazione dell’hash su H)
- esso invia (K+ | f | s) al client

3.

- il client verifica che K+ è veramente la chiave pubblica del server
- il client calcola K = f^x mod p = g^xy mod p e l’hash di scambio H
- il client verifica la firma s su H

Scambio di Chiavi

Con i valori finora ottenuti è possibile calcolare le chiavi che verranno utilizzate durante la sessione



I vettori di inizializzazione risulteranno utili negli algoritmi di cifratura.
La chiave di cifratura stessa viene calcolato come risultato dei valori H e K.
Infine la stessa chiave di integrità viene calcolato con lo stesso meccanismo.

Le chiavi effettive, infatti, risulteranno essere composte dai primi n byte degli hash mostrati sopra. N, infatti, è la lunghezza della chiave dettata dall’algoritmo che la utilizzerà.
Se dovesse accadere che l’hash non risulti lungo abbastanza per poter formare una chiave viene utilizzato un meccanismo ricorsivo, effettuando operazioni di hashing.

K1 = Hash(K, h, x, session_id)

K2 = Hash(K, h, K1)

K3 = Hash(K, h, K1, K2)

...

chiave = K1 • K2 • K3 •...

Rinegoziazione delle chiavi

In qualsiasi momento, entrambe le parti posso inizializzare una nuova negoziazione di chiavi.

Viene inviato un pacchetto SSH_MSG_KEXINIT.

A questo punto il nuovo processo di scambio di chiavi risulterà identico a quello iniziale.

L'unico valore che non cambierà è la session_id che rimarrà la stessa.

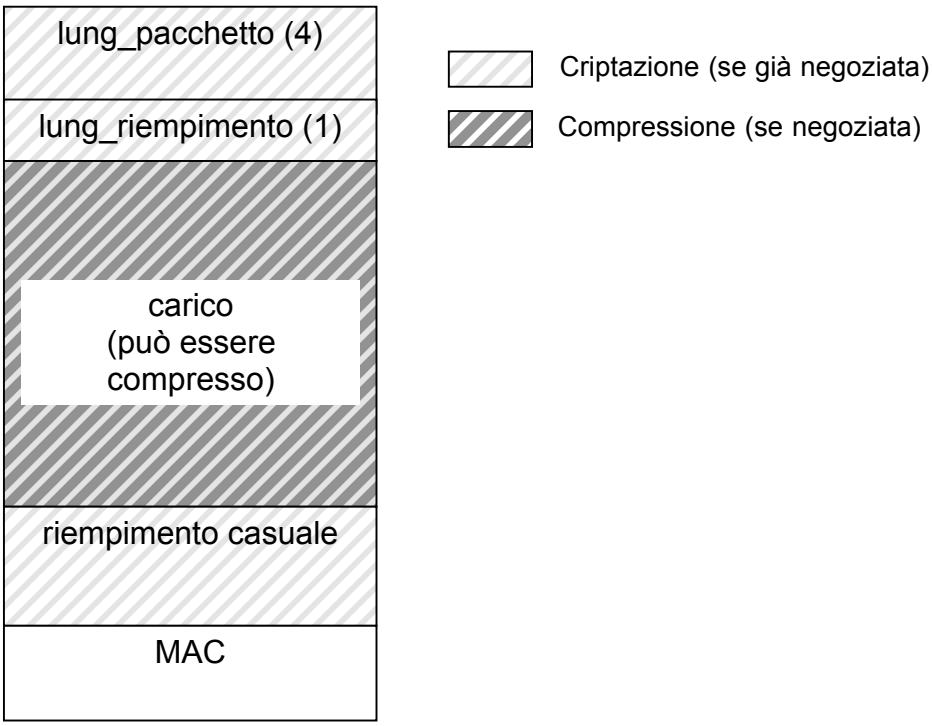
In questa fase possono essere cambiati gli algoritmi da entrambe le parti, le chiavi ed i vettori vengono ricalcolati ed i contesti di cifratura vengono riavviati.

E' raccomandabile effettuare uno scambio di chiavi dopo ogni gigabyte trasmesso o dopo un'ora di connessione, per ovvi motivi di sicurezza.

In questo modo, anche se la comunicazione fosse intercettata, un utente malintenzionato non avrebbe i mezzi necessari per decriptarla in tempo utile.

Formato dei pacchetti

Viene ora illustrato il formato dei pacchetti utilizzato durante le comunicazione in una sessione SSH:



- *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

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**

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**

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
- Sono correntemente supportati i seguenti cifrari:

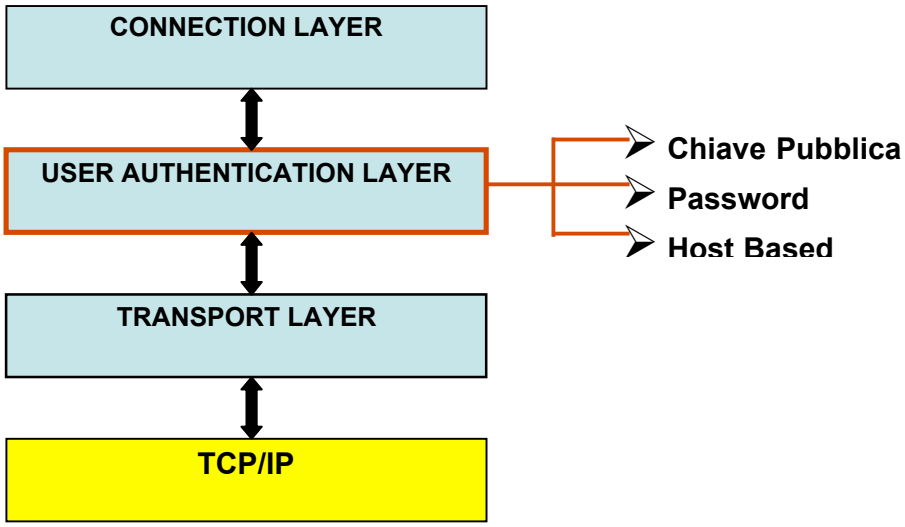
3des-cbc	**RICHIESTO**	**three-key 3DES in modalità CBC**
blowfish-cbc	**RACCOMANDATO**	**Blowfish in modalità CBC**
twofish256-cbc	**OPZIONALE**	**Twofish in modalità CBC, con chiave a 256-bit**
twofish-cbc	**OPZIONALE**	**alias per "twofish256-cbc"**
twofish192-cbc	**OPZIONALE**	**Twofish con chiave 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**

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



L'Authentication Protocol ha il compito di autenticare l'utente-client al server.
Si assume che il protocollo di trasporto sicuro, il Transport Layer, abbia già

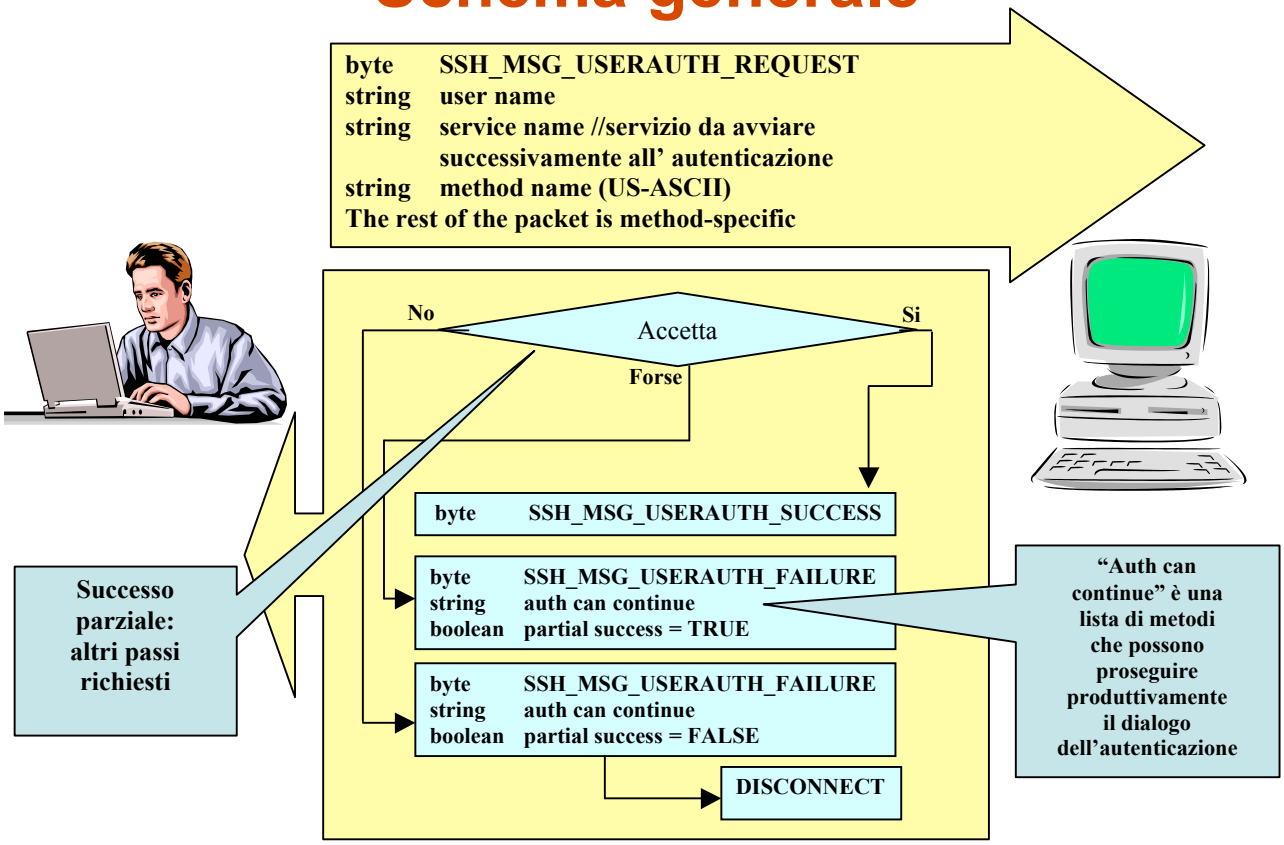
- Autenticato la macchina server
- Stabilito un canale di comunicazione cifrato
- Computato un identificatore di sessione

All'avvio del protocollo lo strato sottostante, il Transport Layer, gli fornisce l'identificatore di sessione (il valore H del primo pacchetto di scambio di chiavi) e assicura ad esso protezione sull'integrità e confidenzialità dei dati scambiati.
Se il livello di trasporto non utilizza il MAC, non deve essere possibile cambiare la password per evitare attacchi del tipo *denial of service*.

Il server guida l'autenticazione:

- Invia al client i metodi di autenticazione consentiti:
  - Chiave Pubblica
  - Host Based
  - Password
- Il client sceglierà quello più conveniente tra quelli da lui supportati

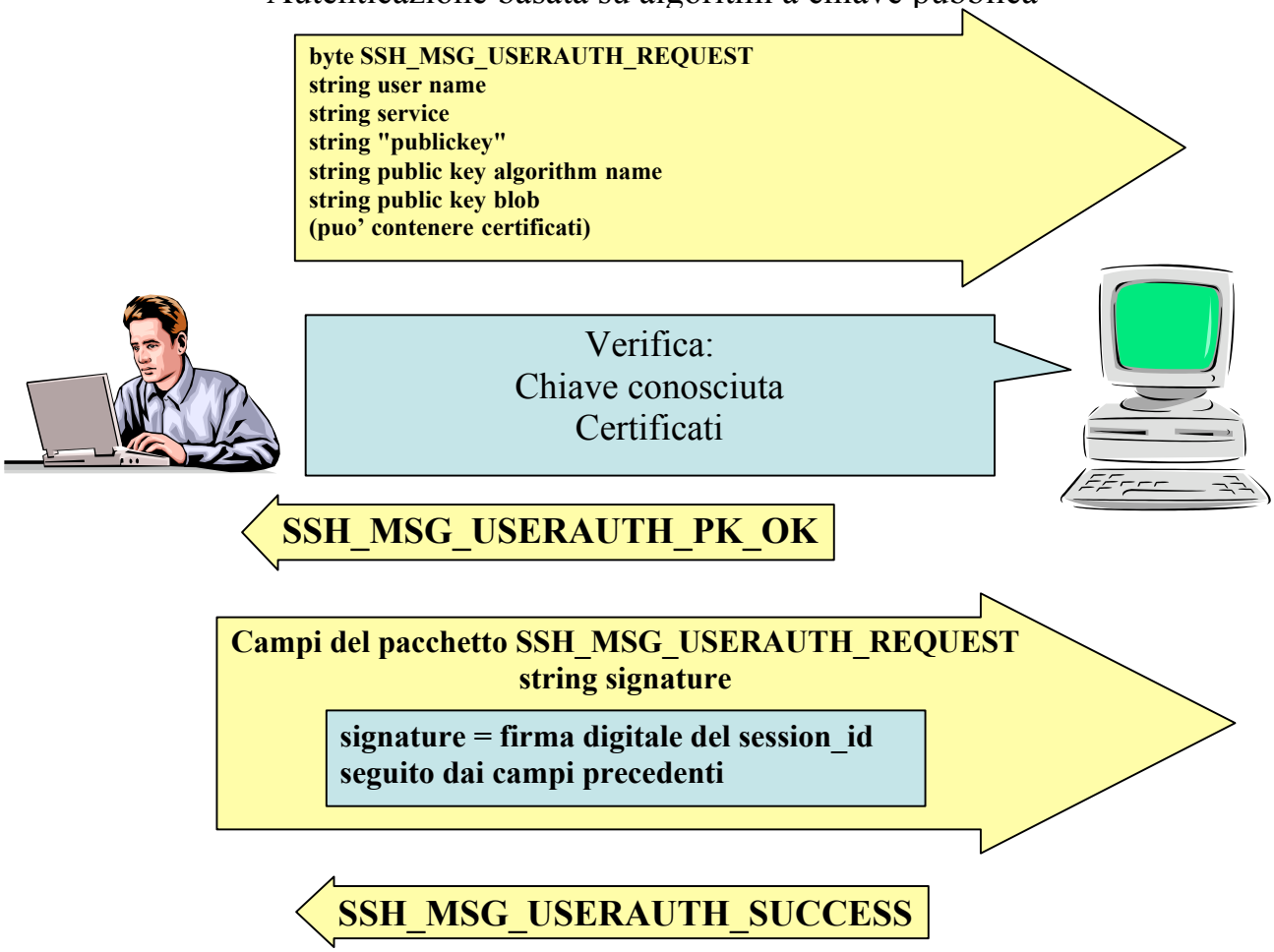
Authentication Protocol
Schema generale



La politica locale decide quali metodi accettare per ogni utente o se sono necessarie ulteriori verifiche. Il server stabilirà un *timeout* o un numero di tentativi non andati a buon fine, dopo i quali andare in una fase di *sleep*.
Infine i messaggi di debug possono essere disabilitati per evitare che forniscano troppe informazioni sull'host.

Authentication Protocol
Autenticazione a Chiave Pubblica

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



RSAAuthentication

Ogni utente possiede una coppia di chiavi pubblica e privata:

- $HOME/.ssh/identity (-rw-------)
- $HOME/.ssh/identità.pub (-rw-rw-r--)

Il server conosce le chiavi pubbliche dei clienti autorizzati a collegarsi

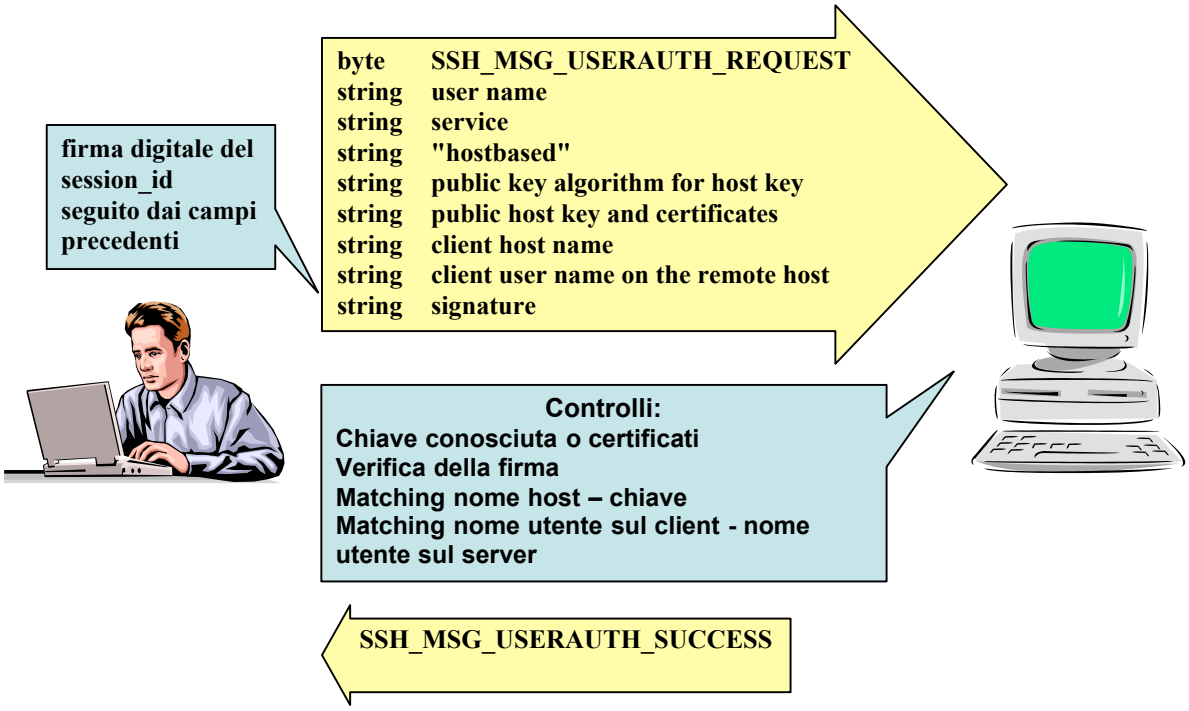
- $HOME/.ssh/authorized_key (-rw-r--r--) contiene le chiavi degli utenti autorizzati

Questo tipo di autenticazione si basa sulla sola crittografia a chiave asimmetrica.
Se la chiave pubblica dell'utente è inserita nel file *$HOME/.ssh/authorized_keys* sul server, il server stesso genera una sequenza casuale di bit, la codifica con la chiave pubblica dell'utente e spedisce il tutto al client; il client decodifica con la chiave privata dell'utente la sequenza di bit e la rispedisce al server codificandola con le chiavi pubbliche del server. Cosi' facendo il client dimostra di conoscere la chiave

privata, autenticandosi col server. Con questo tipo di autenticazione è richiesto all'utente di inserire la password che sblocca la sua chiave privata.

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**

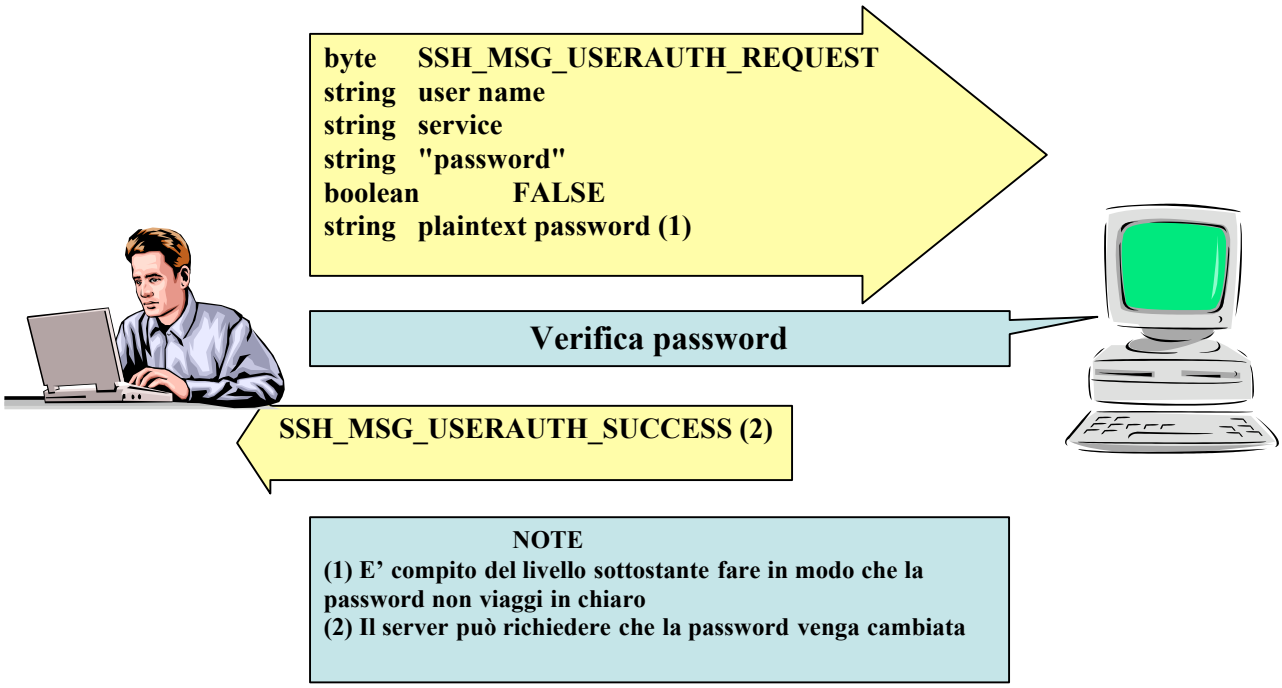


RhostsRSAAuthentication

Questo è il metodo principale di autenticazione. Si basa fondamentalmente sulla RhostsAuthentication, ma l'identità dell'host client viene accertata attraverso la sua chiave pubblica RSA: il server possiede una copia della chiave pubblica degli host autorizzati alla connessione all'interno del file */etc/ssh_known_hosts* ed inoltre ogni utente può crearsi una propria lista personalizzata di chiavi pubbliche utilizzando il file *$HOME/.ssh/known_hosts*, nella sua home directory sul server. Una volta accertata l'identità dell'host client la connessione viene autorizzata e il login viene effettuato con le modalità della RhostsAuthentication. Alternativamente ai file */etc/hosts.equiv* e *$HOME/.rhosts* sul server si possono utilizzare i file */etc/shosts.equiv* e *$HOME/.shosts*, di formato identico; in questo modo ci si svincola completamente da qualsiasi legame con i vecchi comandi r* e non si incorre nei loro problemi di security.

Authentication Protocol
Autenticazione con Password

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



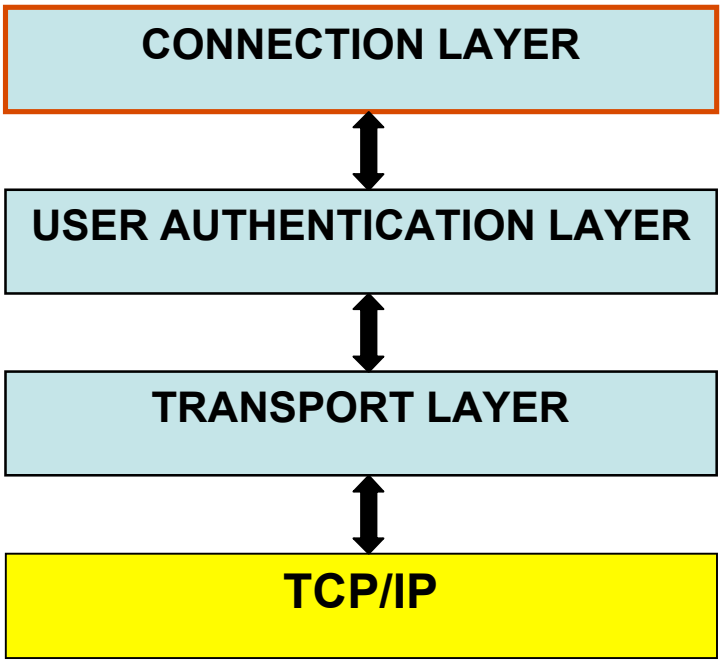
Quando tutti gli altri metodi falliscono, all'utente viene chiesto di inserire la password del suo account sul server. Da notare che la comunicazione sta comunque avvenendo in maniera crittografata.

La tabella di seguito riportata mette in evidenza l'ordine in cui i vari metodi di autenticazione sono applicati

Metodo	Abilitato	Ordine
RhostsAuthentication	NO	-
RhostsRSAAuthentication	Sì	1
RSAAuthentication	Sì	2
TISAuthentication	NO	-

PasswordAuthentication	Sì	3

Connection Protocol



Fornisce:

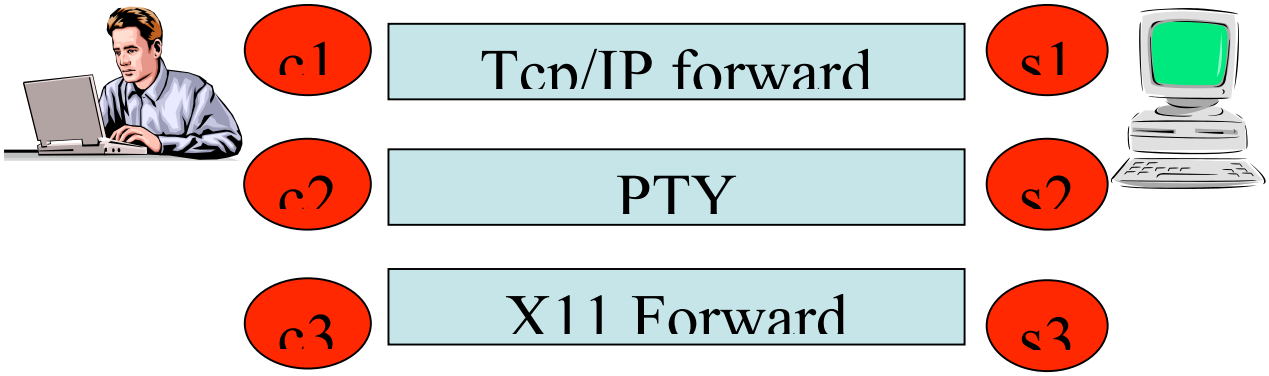
- Sessioni interattive di login
- Esecuzione remota di comandi
- Inoltro di connessioni TCP/IP
- Inoltro di connessioni X11

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

Meccanismo dei canali

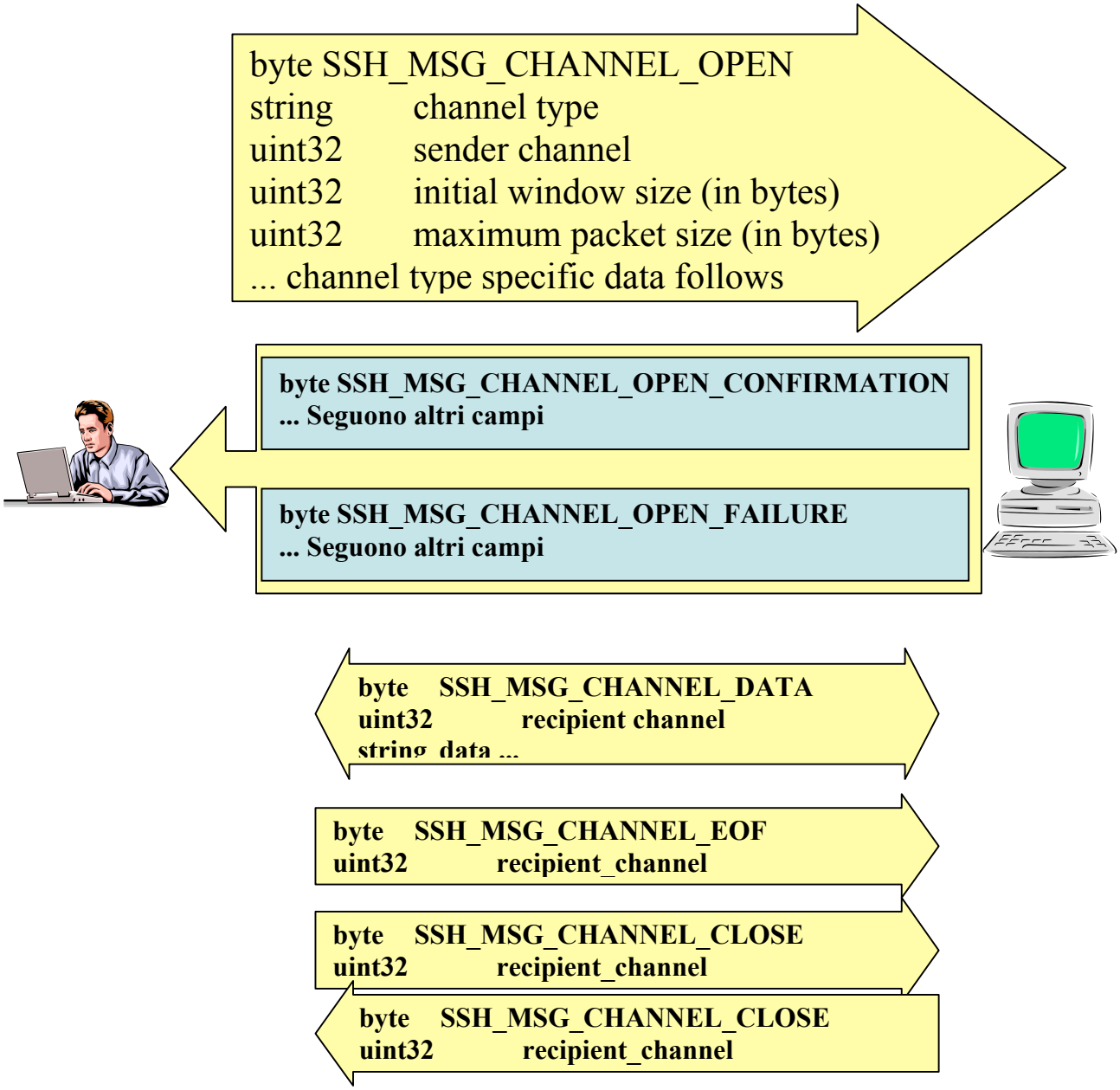
Tutte gli pseudo-terminali, connessioni inoltrate, ecc. sono canali identificati da numeri (ad entrambe le parti), che possono essere differenti tra client e server.
Sono inoltre tipizzati: “session”, “x11”, “forwarded-tcpip”,“direct-tcpip”...
I canali hanno un meccanismo di controllo del flusso attuato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

Connessione cifrata



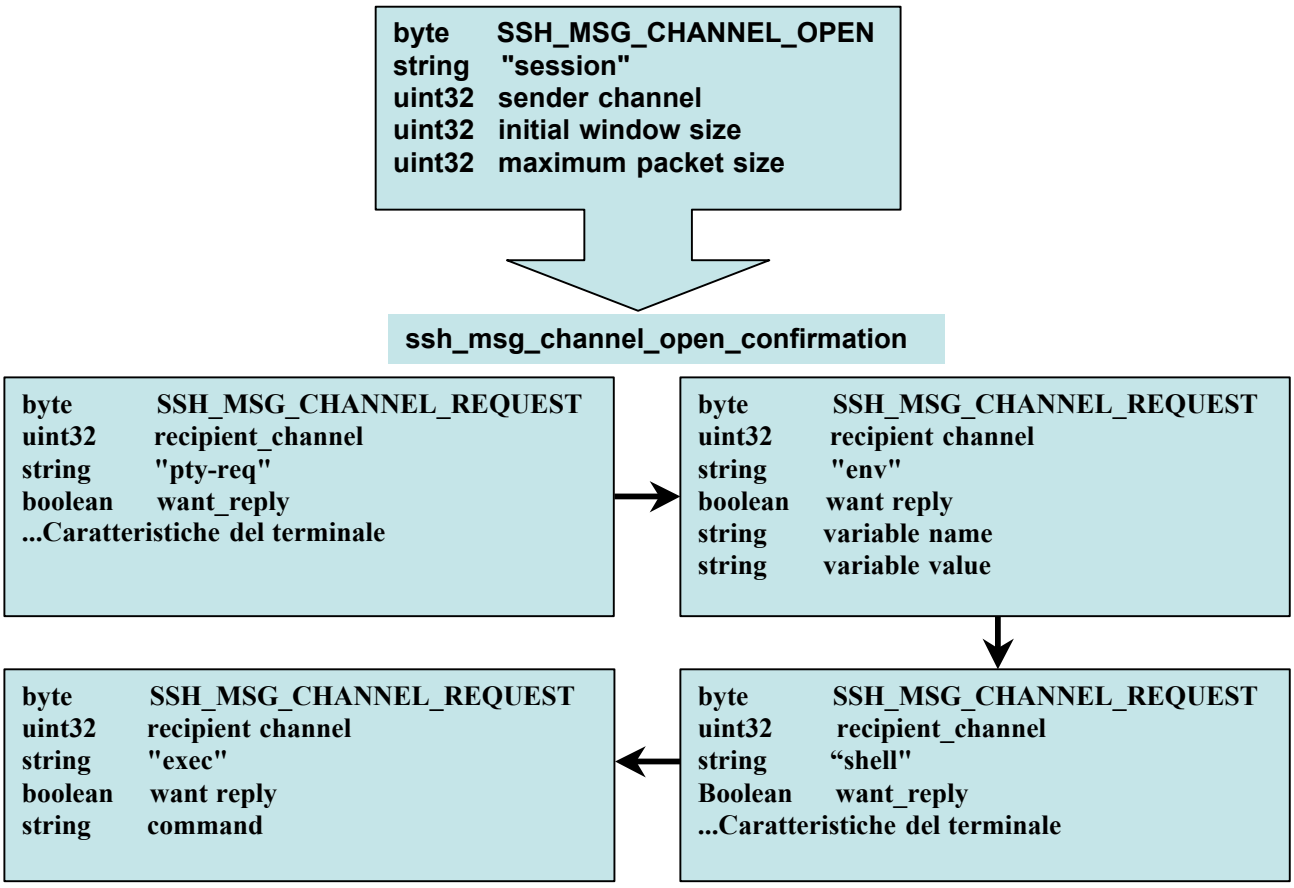
Per poter utilizzare un canale occorre prima effettuare una apertura dello stesso tramite una richiesta globale, nella quale si specifica il tipo di canale che si vuole aprire (session, x11, ecc…), il mittente, la dimensione iniziale della finestra per il controllo di flusso ed la massima dimensione dei pacchetti.
In risposta a questa richiesta si più ottenere una conferma od un fallimento.
In caso affermativo, si è in grado di scambiare dati tra il client ed il server. Quando per esempio il client invia un EOF (End of File), la connessione rimane ancora attiva. Per chiuderla effettivamente occorre che entrambe le parti inviino un messaggio di Channel_close.
Segue lo schema generale appena descritto per stabilire una connessione.

Meccanismo dei canali:
Schema generale



Channel type: Session,X11,Forwarded-tcpip,direct-tcpip
Sender Channel : Identificatore locale scelto dal richiedente del canale
Initial Window size: Specifica quanti Bytes possono essere mandati
Maximum packet size: Max dimensione del pacchetto accettata dal canale

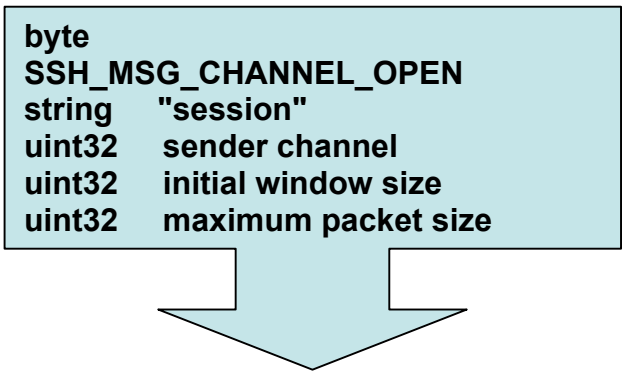
Sessioni Interattive & avvio shell



pty-req: Per richiedere un terminale (canale)
env: Per settare variabili d'ambiente
shell: Per avviare una "finestra prompt" sulla macchina remota
exec: Per lanciare comandi sulla macchina remota

In questo caso si è fatta richiesta di un canale di tipo "Sessione" e ricevendo una risposta affermativa si è passati a richiedere un canale di tipo "pty-req" per poter comunicare con una finestra prompr. Un passo facoltativo è quello di richiedere un canale di tipo "env" per poter settare variabili d'ambiente.
Successivamente si richiede una "shell" e con successive richieste di comandi "exec" da lanciare attraverso la shell appena aperta sulla macchina remota.

Sessioni Interattive: X11




```
byte    SSH_MSG_CHANNEL_REQUEST
uint32          recipient channel
string  "x11-req"
boolean         want reply
boolean         single connection
string  x11 authentication protocol
string  x11 authentication cookie
uint32          x11 screen number
```

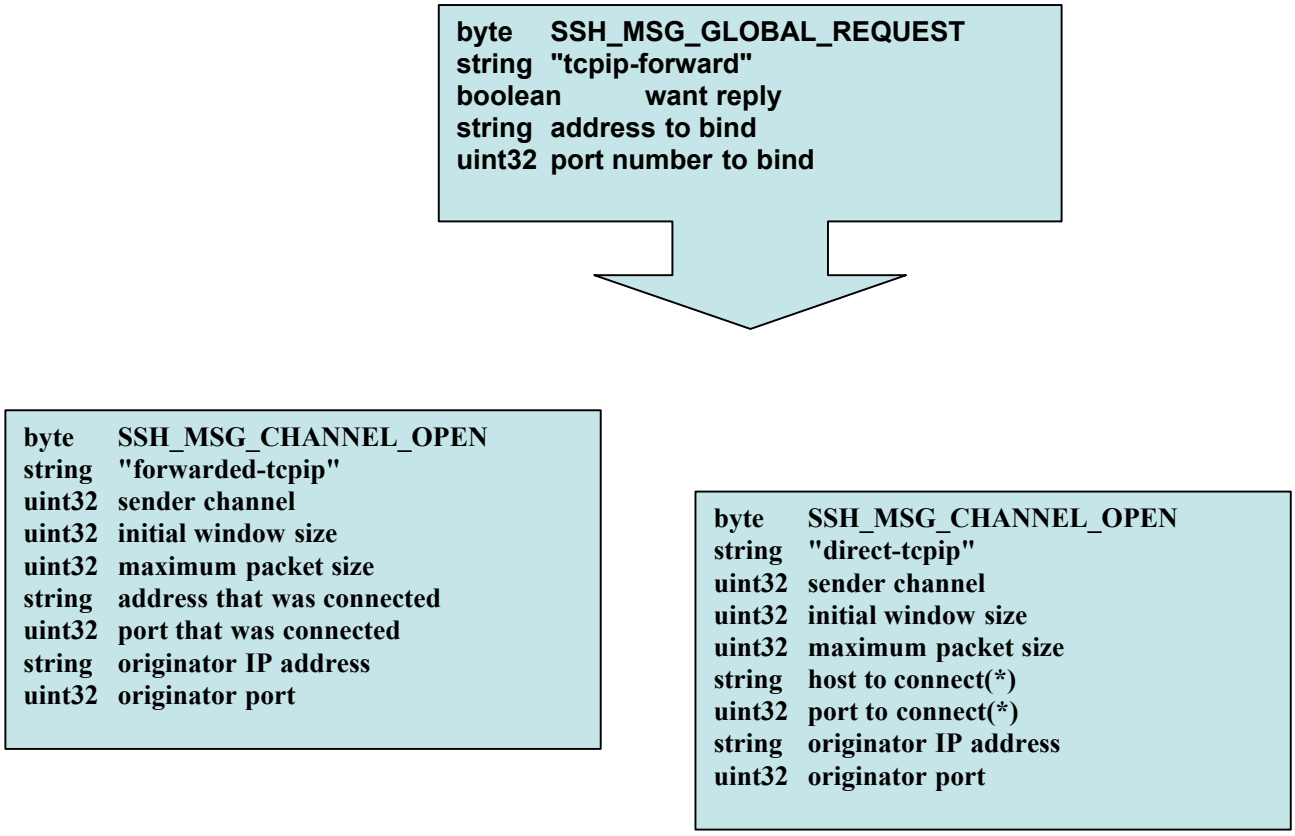
```
byte    SSH_MSG_CHANNEL_OPEN
string  "x11"
uint32          sender channel
uint32          initial window size
uint32          maximum packet size
string  originator address
uint32          originator port
```

Analogamente al caso precedente qui si fa dapprima una richiesta globale per l'apertura di un canale di tipo "sessione" con successiva richiesta di canale "x11" nella quale si specifica il protocollo di autenticazione.
Per ultimo si apre il canale vero e proprio di tipo X11 in cui si è specificato le dimensioni della finestra di flusso e dei pacchetti oltre che l'indirizzo e la porta del mittente della richiesta.

Se dal lato server, tramite un'applicazione, è richiesto l'accesso al terminale del client (l'X11-server in esecuzione sul client), viene aperto un nuovo canale di tipo X11, con l'indirizzo IP originatore e il numero di porta come parametri addizionali

TCP/IP Port Forwarding

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



forwarded-tcpip:

Stabilita una connessione ad una porta per la quale è stato richiesto il "forwarding", viene aperto un nuovo canale di tipo "direct-tcpip" attraverso il quale si ha la comunicazione diretta, ma criptata, dei dati tra client e server.

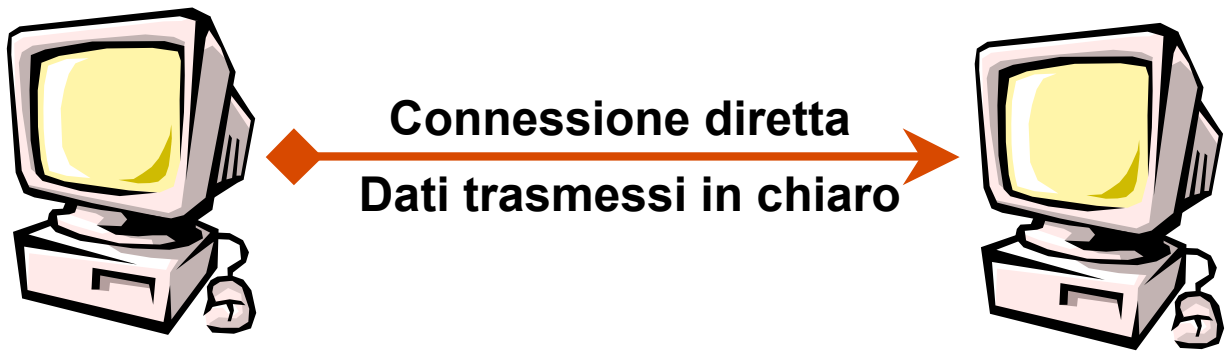
direct-tcpip:

Quando una connessione giunge alla porta del client, che è localmente settato per il "forwarding", viene richiesto un nuovo canale di tipo "direct-tcpip"

Port Forwarding, tunneling

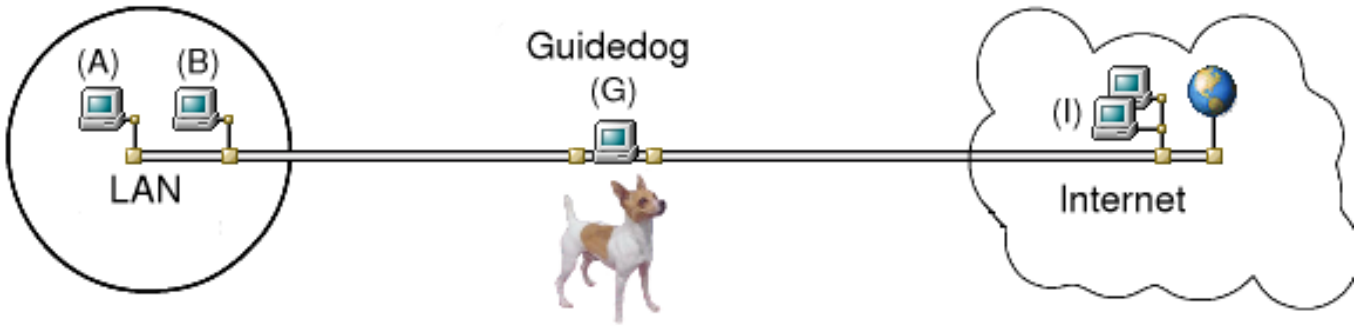
- **Port Forwarding è la tecnica utilizzata per intercettare i pacchetti destinati ad una specifica porta e macchina TCP o UDP , e reindirizzarli (Forward) ad una differente porta e/o macchina. Questo viene fatto in modo trasparente, ovvero i clients in rete non possono vedere che è stato fatto un port 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

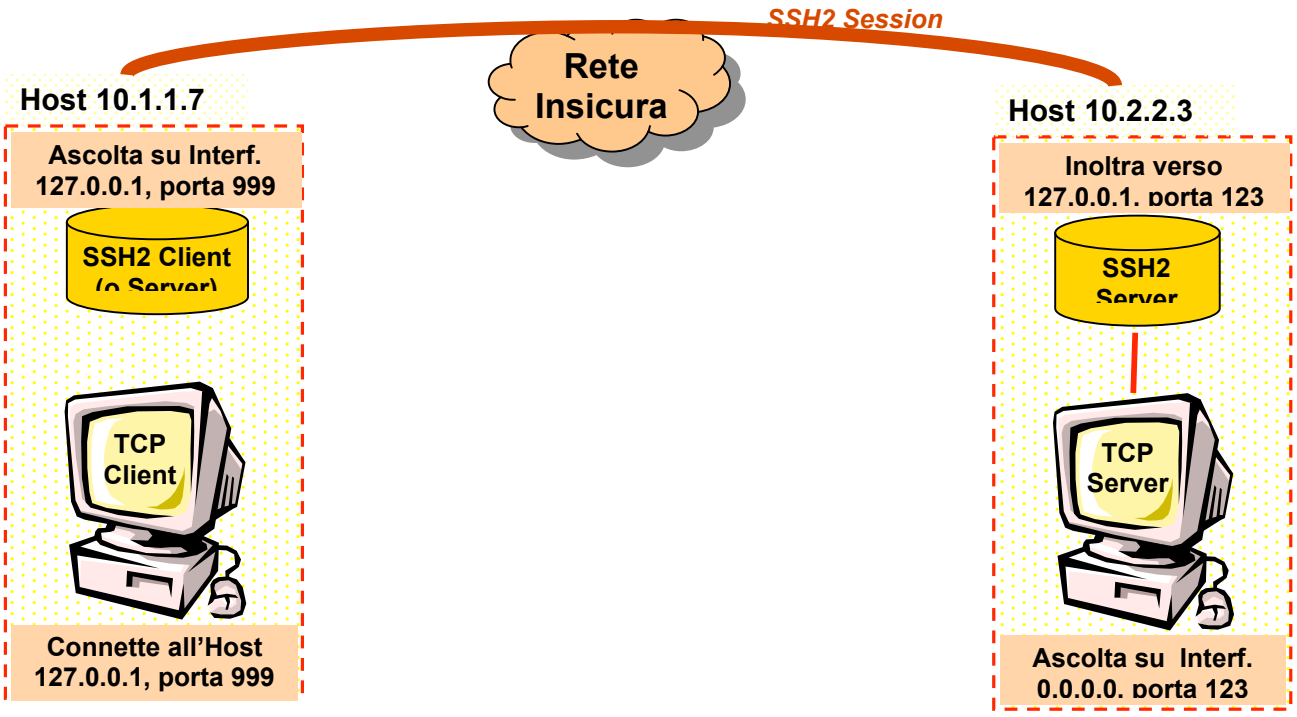


Port Forwarding, tunneling (Esempio)

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.



Port Forwarding, tunneling



- **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 e flessibilità

- 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
- 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

Open SSH

È una Suite di protocolli gratuita che presenta le seguenti caratteristiche:

– 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

Esempio di File Interessati dal protocollo:

- $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