# SECURE SHELL
# (SSH)

**Giulia Carboni**

## Breve introduzione.

Nell'epoca moderna l'accesso ad un sistema remoto è una cosa normalissima. Con l'avvento di Internet nascono i protocolli della suite TCP/IP, ad esempio POP, IMAP,SMTP e Telnet, dettati per lo più dall'esigenza dello scambio di dati, considerata primaria rispetto a quella della loro sicurezza. Per questo motivo Internet è un mezzo di comunicazione intrinsecamente insicuro.
Infatti caratteristica di questi protocolli è che le informazioni e i dati attraversano la rete in chiaro, cioè senza alcuna protezione, quindi un malintenzionato che avesse il controllo di una macchina situata in un punto qualsiasi del percorso di comunicazione che connette due host può spiare e alterare il contenuto del flusso di dati e prendere possesso di informazioni riservate senza che il legittimo destinatario se ne accorga.

## Perché SSH?

SSH è un protocollo di comunicazione che dà la possibilità di instaurare una comunicazione "sicura" su una rete insicura come Internet.

La prima versione di SSH, chiamata SSH1, nasce ad opera del ricercatore finlandese Tatu Ylonen nel 1995 per rimpiazzare i comandi **rsh (*remote shell***, esegue comandi su un host remoto**), rlogin** (***remote login***, inizia una sessione su un host remoto ), **rcp** (***remote copy***, copia files tra due differenti macchine collegate in rete) e i protocolli **telnet** (egli stesso aveva subito attacchi di tipo sniffing usando questo protocollo) e **ftp** in cui tutte le comunicazioni avvengono in chiaro e senza alcuna autenticazione dell'utente, con le rispettive versioni sicure **ssh**, **slogin**, **scp**, **sftp.**
Nel 1996 venne rilasciata la versione del protocollo *SSH2*, che non è considerata compatibile con il protocollo SSH1 iniziale. Nel 1999 alcuni sviluppatori di software diedero vita alla prima implementazione libera di *SSH*. Björn Grönvall creò *OSSH* che da lì a poco, venne rielaborata dagli sviluppatori di *OpenBSD*6 dando vita a *OpenSSH* inclusa nella versione 2.6 di *OpenBSD*.
OpenSSH è una suite di protocolli gratuita che fornisce cifratura per servizi di rete, come il login remoto o il trasferimento di file remoto.
OpenSSH cripta tutto il traffico (password compresa) per eliminare a tutti gli effetti l'ascolto passivo e altri attacchi a livello di rete. Fornisce inoltre una miriade di possibilità di tunneling sicuro, così come vari metodi di autenticazione

## Versioni a confronto

La differenza sostanziale tra queste due implementazioni, oltre alla totale incompatibilità, risiede nell'architettura adottata.
La prima versione di Secure Shell, adotta una struttura "integrata" senza stratificazioni nelle varie funzioni di autenticazione, connessione e trasporto. La versione SSH2 del protocollo ridefinisce l'architettura "integrata" adottata nella prima versione, dividendo in strati le varie funzioni, offrendo così maggiore sicurezza; vengono introdotti gli strati di SSH Transport Layer Protocol, SSH Authentication Protocol e SSH Connection Protocol. Per il controllo dell'integrità, la versione SSH1 adotta CRC32 (non sicuro) mentre la versione SSH2 adotta il metodo di crittazione con hash

HMAC. HMAC non è propriamente una funzione di hash, poiché per eseguirlo è necessario non solo il testo in chiaro (preso come unico parametro di input dalle funzioni di hash più comuni), ma anche una chiave. Un'altra caratteristica che contraddistingue le due versioni è la gestione dei canali di comunicazione tra host e host: la versione SSH1 può mantenere un solo canale per sessione, a differenza della versione di SSH2 che per ogni sessione adotta molteplici canali di comunicazione.
Per quanto riguarda l'utilizzo dei metodi di cifratura, e la gestione delle chiavi,la versione SSH2 adotta vari metodi di negoziazione rispetto alla precedente versione ed inoltre utilizza sia l'RSA che il DSA come algoritmo a chiave pubblica. La negoziazione della chiave di sessione per il protocollo SSH2 viene gestita tramite l'algoritmo Diffie-Hellman7, ed inoltre viene rinnovata più volte nell'arco di una stessa sessione. Parlando invece della versione SSH1, le cose sono molto più semplici a livello implementativo, ma indubbiamente meno sicure: la chiave di sessione viene trasmessa dal lato client, e questa rimane invariata per tutta la sessione.

*Tabella comparativa delle versioni di SSH*

| | SSH1 | SSH2 |
|---|---|---|
| Architettura | Struttura "integrata" | Divisione della struttura negli strati di autenticazione, connessione e trasporto |
| Integrità | CRC-32 | HMAC (hash encryption) |
| Sessione | Utilizzo di un canale per sessione. | Più canali per una singola sessione. |
| Algoritmi | RSA a chiave pubblica | RSA e DSA a chiave pubblica |
| Chiave di sessione | Trasmessa dal lato client. Stessa chiave per la sessione | Negoziata attraverso l'algoritmo Diffie-Hellman. Chiave di sessione rinnovabile. |

SSH viene utilizzato per loggarsi in modo sicuro su un altro computer in una rete, eseguire comandi sul sistema remoto, copiare file da una macchina all'altra, richiedere una shell remota (la shell è il programma che permette agli utenti di comunicare con il sistema e di avviare i programmi. È una delle componenti principali di un sistema operativo) .

SSH permette:

- Connessioni crittografate;
- Verifica d'integrità;
- Autenticazione forte;
- TCP port forwarding.

Proteggendoci così da alcuni frequenti attacchi di tipo *spoofing*, dove con questo termine si intende la falsificazione delle informazioni inerenti ad un servizio:

- **IP SPOOFING**: tecnica tramite la quale viene falsificato l'indirizzo IP del mittente di un pacchetto modificando il suo campo SOURCE ADDRESS, ovviamente questo tipo di attacco ha tanto più successo quanto maggiori sono i rapporti di fiducia tra le due macchine in comunicazione.
  Un hacker che voglia intraprendere un attacco di questo tipo deve:
  1. scegliere l'host target (B);
  2. scoprire l'indirizzo IP di un host fidato (A) che si trova nel file *etc/hosts.equiv* di B;
  3. disabilitare A per evitare che si accorga dell'attacco, infatti i successivi pacchetti inviati da B potrebbero pervenire ad A e destare qualche sospetto. Per evitare questo, l'attaccante potrebbe fare in modo che A scarti tutti i pacchetti di sincronizzazione che riceve da B mediante la tecnica del **TCP SYN floading**: l'hacker invia dei messaggi di sincronizzazione alla porta di A che vuole disabilitare fino ad eccedere il backlog per le connessioni incomplete di quella porta. Così facendo TCP scarta tutti i pacchetti di sincronizzazione finché le connessioni pendenti non sono completate;
  4. fingersi l'host fidato per ottenere una shell su B cercando di indovinare i Sequence number. Notare che a questo punto l'attaccante potrebbe catturare i pacchetti in uscita da B falsificando la rotta di ritorno (**ROUTING SPOOFING**);
  5. aggiungere il proprio indirizzo in *etc/hosts.equiv* di B;
- **DNS SPOOFING**: tecnica che consiste nella falsificazione delle informazioni contenute nel DNS che, quindi, accetta e usa informazioni non corrette ma fornite da un host che non ne ha l'autorità.
  Un hacker che voglia intraprendere un attacco di questo tipo deve:
  1. scegliere l'host target (B);
  2. scoprire il nome host di un host fidato (A) che si trova nel file *etc/hosts.equiv* di B;
  3. falsificare il DNS di B in modo che esso risolva il nome host di A con l'indirizzo IP dell'attaccante;
  4. stabilire una connessione con B fingendosi A.

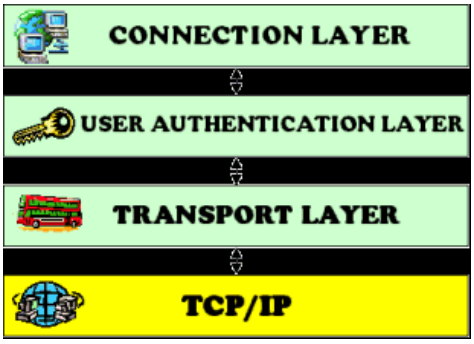
Nonostante questo il sistema non potrà dirsi assolutamente sicuro, infatti SSH non ci tutela da:

- configurazione o utilizzo scorretto;
- un account root compromesso: se vi loggate da un host ad un server e l'aggressore ha i privilegi di root dall'altra parte può vedere la vostra sessione poiché SSH cifra la rete ma comunica in chiaro con i dispositivi di terminale;
- directory home insicure : se, ad esempio, un aggressore può modificare i files nella vostra home directory tramite NFS può riuscire ad aggirare SSH.

Inoltre l'SSH port forwarding cifrando il traffico non permette che questo venga esaminato dai firewall, quindi questo meccanismo può potenzialmente permettere ad un intruso di scavalcare questo tipo di sistemi di sicurezza.

## Architettura

L'architettura SSH e la sua divisione in protocolli si può schematizzare nel modo seguente:




Ogni livello si appoggia ai servizi del livello sottostante e fornisce servizi al livello superiore.

## Transport Layer Protocol

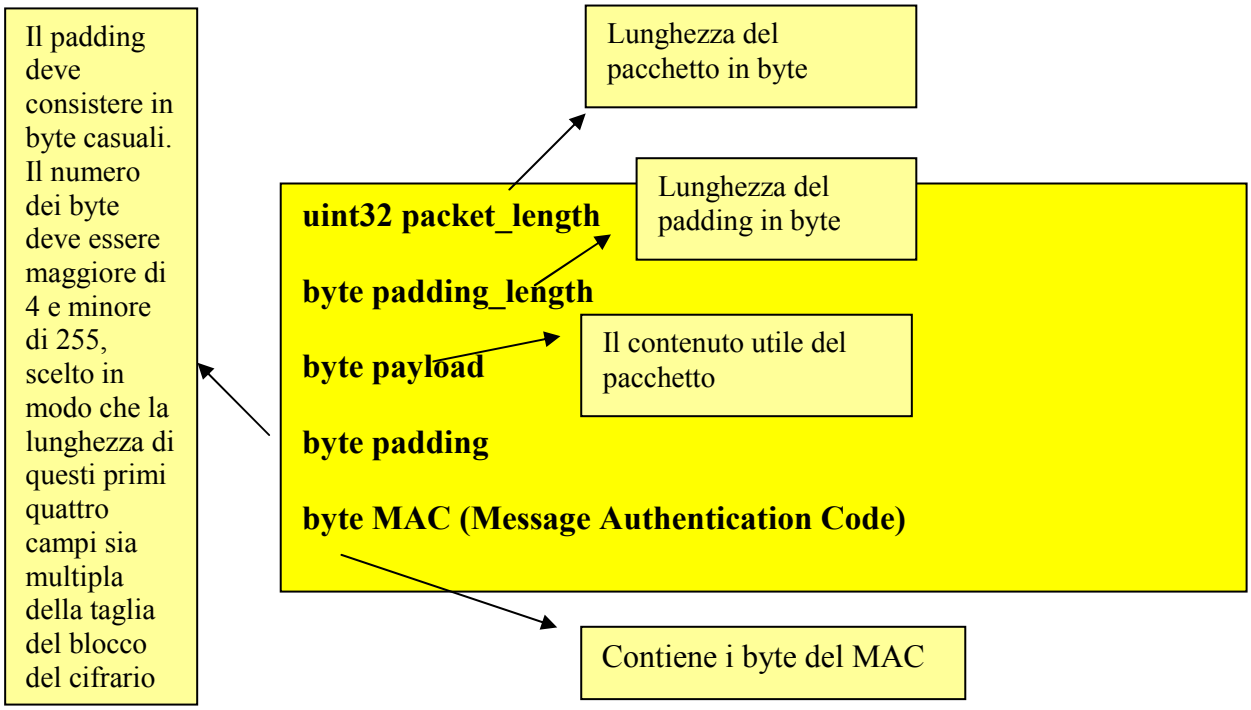
Il Transport Layer Protocol fornisce un canale criptato sicuro su di una rete insicura. Effettua autenticazione del server, scambio di chiavi, cifratura e protezione d'integrità.
E' progettato per essere usato su uno strato di trasporto affidabile e in modo tale che se avvengono errori di trasmissione o manipolazione dei messaggi la connessione venga chiusa.
Quando SSH gira su TCP/IP il server si lega alla porta 22.

Analizziamo, nello specifico, le fasi del Transport Layer Protocol.


***Fase 1)*** *Scambio delle versioni.*

Inizialmente entrambe le parti devono mandare una stringa identificativa in formato US-ASCII della forma *"SSH-protoversion-softwareversion-comments"* seguita da un *carriage return* ed un carattere di *newline*. Il tutto deve essere lungo al massimo 255 caratteri.
Serve per garantire la compatibilità e l'inserimento dei commenti può essere utile per risolvere problemi dell'utente.

Dopo questa stringa i pacchetti devono avere il seguente formato:




Tutte le implementazioni devono supportare pacchetti di taglia massima (almeno) 3500 byte.

***Fase 2)*** *Negoziazione degli algoritmi per cifratura, integrità e compressione*.

Entrambe le parti si inviano la lista degli algoritmi supportati per ogni categoria (algoritmo a chiave pubblica, cifratura, MAC, compressione) separati da una virgola e citati in ordine di precedenza (il primo algoritmo è quello preferito) mediante l'invio del pacchetto SSH_MSG_KEXINIT .
Entrambi fanno la scelta per un determinato algoritmo:

- se questa è la stessa viene usato questo algoritmo ;
- altrimenti l'algoritmo va scelto iterando sulla lista del client e scegliendo il primo che è supportato anche dal server.

Oppure ogni lato può indovinare quale algoritmo sta usando l'altro e mandare un pacchetto di scambio di chiavi relativo a quell'algoritmo: se la scelta è giusta il pacchetto mandato viene considerato come il primo pacchetto per lo scambio di chiavi altrimenti viene ignorato e la parte appropriata dovrà rinviare un nuovo pacchetto.
Ogni server dovrebbe avere una Host Key per ogni algoritmo a chiave pubblica supportato.

***Fase 3)*** *Scambio delle chiavi.*

La fase dello scambio delle chiavi comincia con la generazione di due valori: un segreto condiviso K ed un valore hash di scambio H che verranno utilizzati per il calcolo delle chiavi di cifratura e d'integrità utilizzate dal protocollo SSH.
Il modo in cui essi vengono generati dipende dall'algoritmo usato.
Ogni metodo di scambio di chiavi specifica una funzione hash che è usata nello scambio.
Oggigiorno l'unico metodo supportato da tutte le implementazioni è **Diffie-Hellman**.

## Come funziona?

Il criptosistema Diffie-Hellman, come tutti quelli a chiave pubblica, nasce per evitare un preventivo scambio di chiave tra gli utenti (necessario nella crittografia simmetrica) facendo in modo che una parte della chiave (relativa alla codifica) sia pubblica mentre il resto della chiave (relativa alla decodifica) sia rigorosamente tenuta segreta.
Quindi la decodifica sarà enormemente più difficile (in termini computazionali) della codifica se non si conosce la parte privata della chiave.
Per soddisfare queste esigenze nasce la nozione di *funzioni a senso unico*, ovvero funzioni di cifratura facilmente computabili in modo che chiunque possa codificare rapidamente un messaggio, invertibili (per consentire la decodifica) ma in modo tale che il calcolo dell'inversa sia proibitivo in assenza di "ulteriori informazioni" che costituiscono la chiave privata dell'utente.
Questa nozione di funzione a senso unico non ha un significato matematico rigoroso, poiché la "difficoltà" nel calcolo dell'inversa potrebbe anche essere solo un problema temporaneo.

Nel criptosistema di Diffie-Hellman la funzione a senso unico è il **logaritmo discreto**.
Sia *N* un numero primo molto grande, *a* un numero primo con *N* , cioè MCD(*N,a*)=1 e diverso da *1modN* e l'insieme G={1*,a,...,a^N-1*}.
Il punto chiave è il seguente:

- Esistono metodi rapidi di calcolo di *a^x mod N* per ogni intero *x*
- Ma, ad oggi, non sono noti metodi ragionevoli per recuperare da una data potenza *y* di *a* un esponente naturale *x<N* per cui *y=a^x*.

Forte di ciò, lo scambio di chiavi Diffie-Hellman tra il client C e il server S e quindi la generazione dei valori K ed H avviene nel modo seguente.
Sia p un numero primo grande, g un generatore per un sottogruppo di GF(p) e q l'ordine del sottogruppo. Sia, inoltre, Vs la stringa di versione di S, Vc la stringa di versione di C , Ks la host key pubblica di S, Ic il messaggio KEXINIT di C (contenente il cookie generato dal client) e Is il messaggio KEXINIT di S che è stato scambiato nella fase di negoziazione degli algoritmi.
A questo punto:

1. C genera un numero casuale x (1<x<q) e calcola e=g^xmodp. C manda e ad S.
2. S genera un numero casuale y (1<y<q) e calcola f=g^ymodp. S riceve e.
   S calcola
   $$K=e^{y} \bmod p \quad \text{e} \quad H=\text{hash}(Vc \cdot Vs \cdot Ic \cdot Is \cdot Ks \cdot e \cdot f \cdot K)$$
   E firma H con la sua chiave privata ottenendo s.
   S manda (Ks·f·s) a C.
3. C verifica che Ks sia effettivamente la host key di S.
   Ciò si può fare in due diversi modi:

   - Il client ha un database locale nel file */etc/ssh/known_hosts* in cui memorizza le chiavi pubbliche dei server SSH conosciuti. Inoltre ogni utente ha il suo database personale nel file *$HOME/.ssh/known_hosts*. I database vengono aggiornati nel modo seguente: è aggiunta una nuova chiave pubblica del server, a cui il client si connette per la prima volta, nel database del client se sono validi i certificati che accompagnano la chiave, altrimenti il client chiede all'utente la fiducia nella chiave del server e la memorizza in caso affermativo.
     In questo modo non è necessaria un'infrastruttura centralizzata ma la gestione del database può diventare gravosa.

- ❖ L'associazione nome-chiavi è garantita da un'autorità fidata di certificazione. Quindi il client deve conoscere esclusivamente la chiave dell'autorità centrale che certificherà o meno tutte le altre.

4. Infine C calcola

   K=f^xmodp e H=hash(Vc·Vs·Ic·Is·Ks·e·f·K)

   E verifica la firma su H.

Quindi, anche se in due modi diversi, il client e il server sono venuti a calcolare gli stessi valori K e H. Il valore H è chiamato *hash di scambio* ed è utilizzato per autenticare lo scambio delle chiavi e come identificatore di sessione. E' unico per la connessione e non cambia anche se le chiavi sono ricambiate in seguito.
Una volta ottenuti i valori K e H, essi vengono concatenati mediante una funzione HASH.

**Nota)** Entrambe le parti non devono accettare o mandare valori di f non compresi nel range [1,p-1]. Se viene violata questa condizione lo scambio di chiavi fallisce.
L'algoritmo a chiave pubblica per firmare è negoziato con il messaggio KEXINIT contenuto nel pacchetto tramite cui si negoziano gli algoritmi.
Gli algoritmi a chiave pubblica usati per la firma sono RSA e DSA.

RSA utilizza come *funzione a senso unico* la fattorizzazione di numeri interi molto grandi, è infatti considerato sicuro perché, allo stato attuale, non sono noti algoritmi con basso costo computazionale per la fattorizzazione di interi molto grandi.
DSA è un algoritmo che viene utilizzato prettamente per firmare messaggi e non può essere utilizzato per criptare. La sicurezza di questo algoritmo dipende dalla difficoltà di calcolare logaritmi discreti, cioè la sua *funzione a senso unico* è il calcolo delle potenze in un campo finito.

COMPRESSIONE

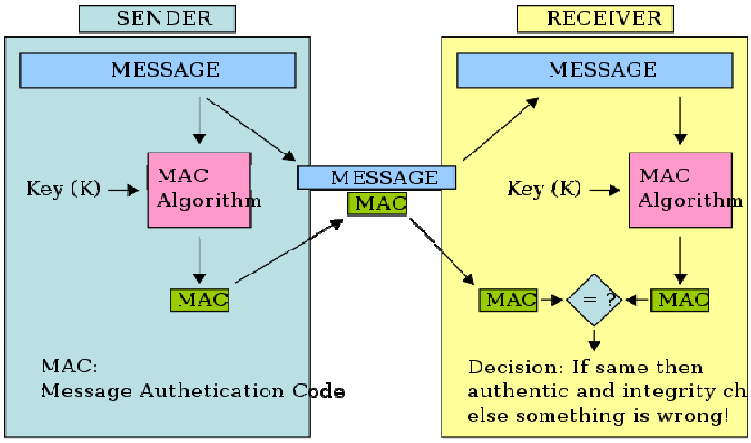
Se è stata negoziata la compressione sarà compresso SOLO il campo payload utilizzando l'algoritmo negoziato. La cifratura sarà successiva alla compressione.

INTEGRITA' E CONFIDENZIALITA' DEI DATI

L'integrità dei dati è protetta includendo in ogni pacchetto un MAC (Message Authentication Code) calcolato prima della cifratura concatenando un segreto condiviso, un numero di sequenza del pacchetto, cioè un intero a 32 bit privo di segno inizializzato a 0 per il primo pacchetto ed incrementato dopo ogni pacchetto che non è incluso nella parte dati, e il contenuto del pacchetto mediante un algoritmo e una chiave negoziati nella fase dello scambio delle chiavi, cioè

mac=MAC(key, sequence_number, unencrypted_packet)

Il campo mac deve essere trasmesso alla fine del pacchetto senza essere cifrato.




*La figura descrive, in generale, il funzionamento dei codici MAC.*

Il mittente (sender) deve inviare un messaggio (message) ad un destinatario (receiver), i due già condividono una chiave segreta (key). Stabilita la chiave, utilizza una funzione di MAC (MAC Algorithm) per calcolare il MAC del messaggio ed infine invia entrambi al destinatario. Questi calcola il MAC del messaggio ricevuto e poi lo confronta con il MAC allegato al messaggio stesso: se coincidono, il messaggio è integro, altrimenti qualcosa è andato storto.

Ovviamente nel nostro caso il "MESSAGE" non verrà inviato in chiaro, ovvero i campi padding_length, payload_length, payload e padding vengono cifrati mediante l'algoritmo stabilito, altrimenti garantiremmo l'integrità del messaggio ma perderemmo la confidenzialità per lo stesso.

Quindi, nel caso in cui venga utilizzato l'algoritmo HAMC, il mittente calcola il valore hash per i dati originali  e invia i dati originali criptati e il codice HMAC in un solo messaggio. Il destinatario ricalcola il valore hash sul messaggio ricevuto dopo averlo decodificato e verifica che il valore hash calcolato corrisponda a quello trasmesso.
Il codice HMAC può essere utilizzato con qualsiasi funzione hash di crittografia iterativa, come MD5 o SHA-1, insieme a una chiave segreta condivisa. Il livello di crittografia del codice HMAC dipende dalle proprietà della funzione hash sottostante.
Qualsiasi modifica ai dati o al valore hash causa un'errata corrispondenza, perché è necessario conoscere la chiave segreta per modificare il messaggio e riprodurre il valore hash corretto. Pertanto, se i valori hash originale e calcolato corrispondono, il messaggio viene autenticato.

Questo confronto porta alla verifica dell'integrità del messaggio.

- Ogni algoritmo di cifratura dovrebbe usare chiavi di lunghezza pari o superiore a 128 bit.
  Gli algoritmi di cifratura devono girare indipendentemente l'uno dall'altro.
  I più utilizzati sono 3des, blowfish, idea.
  La crittografia simmetrica si basa su un algoritmo che modifica i dati in base a una chiave (di solito una stringa di qualche tipo) che permette il ripristino dei dati originali soltanto conoscendo la stessa chiave usata per la cifratura. Per utilizzare una cifratura simmetrica, due persone si devono accordare sull'algoritmo da utilizzare e sulla chiave. La forza o la debolezza di questo sistema, si basa sulla difficoltà o meno che ci può essere nell'indovinare la chiave, tenendo conto anche della possibilità elaborative di cui può disporre chi intende spiare la comunicazione.

  Il 3DES è un cifrario a blocco che consiste nella tripla applicazione del DES (Data Encryption Standard), ciò lo rende più resistente agli attacchi rispetto al DES il quale utilizza chiavi lunghe 56 bits, lunghezza già da tempo non più sufficiente per garantire un adeguato livello di sicurezza.

Blowfish opera su blocchi di 64 bits con chiavi di lunghezza variabile fino a 448 bits.
È significativamente più veloce del DES e si ritiene che sia, forse, l'algoritmo più sicuro attualmente disponibile.

- Al momento gli algoritmi a chiave pubblica sono usati solo per la firma ma, in futuro potrebbero essere utilizzati anche per cifrare. Tener presente che alcuni tipi di chiavi non supportano sia la firma che la cifratura.

## Authentication Protocol

Gira al di sopra dell'SSH Transport Layer Protocol, quindi assume che il protocollo sottostante abbia già autenticato il server, stabilito un canale di comunicazione criptato e computato un identificatore che identifica univocamente la sessione.
Se il livello trasporto non fornisce la cifratura, i metodo di autenticazione su dati segreti dovrebbero venir disabilitati.
Se il livello trasporto non fornisce integrità, le richieste di cambiare i dati di autenticazione (password) devono essere disabilitati per evitare che vengano cambiati da un hacker rendendo impossibile un accesso futuro (attacco *Denial of Service*).
Il server comunica al client i metodi supportati per l'autenticazione, in tal modo il server ha il completo controllo sul processo di autenticazione e il client può decidere di usare il metodo più conveniente.

In base alla sua politica il server può

- richiedere a degli utenti connessi da un determinato host un'autenticazione multipla;
- avere un timeout per l'autenticazione e quindi disconnettere se l'autenticazione non è stata completata nell'intervallo di tempo stabilito;
- limitare il numero di tentativi di autenticazione falliti da parte del client.

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

**RhostsAuthentication:**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).

Con SSH si possono avere tre diversi metodi per autenticarsi.

- ***public key*** (deve essere supportato da tutte le implementazioni): è basato sulla chiave pubblica **dell'utente**.
  Il client invia al server un pacchetto contenente l'identificatore di sessione firmato con la chiave privata dell'utente e il server autorizza l'autenticazione se la firma è valida, cioè se applicando la chiave pubblica del client il risultato è di nuovo l'identificatore di sessione che era già stato calcolato nella fase di scambio delle chiavi.
  Ovviamente il server deve supportare l'algoritmo con cui avviene la firma altrimenti deve respingere la richiesta.

  In che modo si può generare la coppia di chiavi per l'utente?

  A questo scopo SSH fornisce il programma *ssh-keygen*.

Ad esempio digitando

**giovanni@sole:~> ssh-keygen**

**Generating RSA keys:**

si aspetta un po' ti tempo prima che la coppia di chiavi venga generata.
Viene ora richiesto il file per l'archiviazione delle chiavi.

**Enter file in which to save the key: /home/giovanni/.ssh/identity**

Poi bisogna salvare l'impostazione con una passphrase (testo lungo dai 10 ai 30 caratteri, preferibilmente non parole o frasi semplici). In particolare va usata una passpharase se la chiave privata rimane in un sistema non amministrato direttamente dal suo proprietario oppure se si condivide la directory in cui è salvata via NFS.

**Enter passphrase (empty for no passphrase):**

Ne viene richiesta la ripetizione per conferma.

Viene mostrato l'output del deposito delle chiavi, pubblica e privata, nel nostro caso vengono salvate nel file identity e identity.pub.

**Enter same passphrase again:**

**Your identification has been saved in /home/giovanni/.ssh/identity.**

**Your public key has been saved in /home/giovanni/.ssh/identity.pub.**

A questo punto va copiata la parte pubblica della chiave sul computer remoto e lasciarvela come *~/.ssh/authorized_keys*. Al prossimo accesso verrà chiesta la passphrase come convalida.

- ***Host-based*** (forma di autenticazione opzionale) è basato sulla chiave pubblica **dell'host client.**
  Il client firma un pacchetto contenente l'identificatore di sessione con la chiave privata dell'host ed il server verifica, con la chiave pubblica dell'host stesso, che la firma sia valida.
  Una volta stabilità l'identità l'autorizzazione è effettuata basandosi sugli username dell'utente sul client e sul server e sul nome host del client.
  Inoltre il client firma il pacchetto che rappresenta la richiesta di autenticazione con la sua chiave privata, il server dovrà quindi controllare se la host key appartiene al client, se la firma è valida e se l'utente è autorizzato a fare il login.

- ***Con password*** (deve essere supportato da tutte le implementazioni)
  Il server deve verificare la password all'interno del proprio database e risponde con un messaggio di successo o di fallimento, oppure potrebbe richiedere il cambio della password.
  NB) Anche se la password è trasmessa in chiaro all'interno del pacchetto, l'intero pacchetto è criptato a livello trasporto.

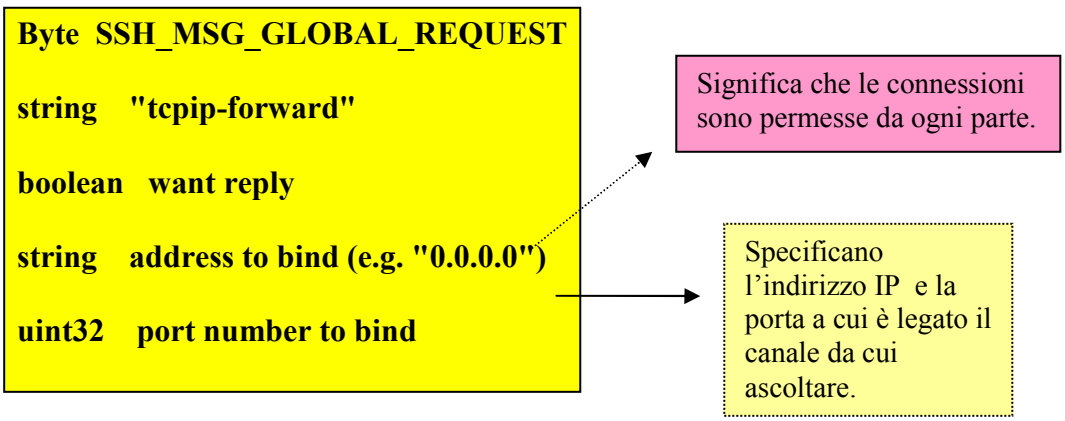
## Connection protocol

Gira sopra all'Authentication Protocol  e il suo obiettivo è quello di permettere sessioni interattive di login, esecuzione remota di comandi e inoltro di connessioni TCP/IP.
La connessione è divisa in canali logici multiplexati in un singolo tunnel cifrato e identificati da numeri che possono essere diversi da lato client a lato server e che dovranno essere specificati in una richiesta di apertura di canale.
I canali hanno un meccanismo di controllo del flusso , cioè non possono essere inviati dati su un canale se prima non si sono ricevute informazioni circa la disponibilità della "window space".
Il protocollo permette l'esecuzione di comandi su macchine remote e al server l'esecuzione di comandi sul client. Quest'ultima possibilità dovrebbe essere evitata dalle implementazione per non permettere ad un server corrotto di attaccare tutti i client che si connettono.

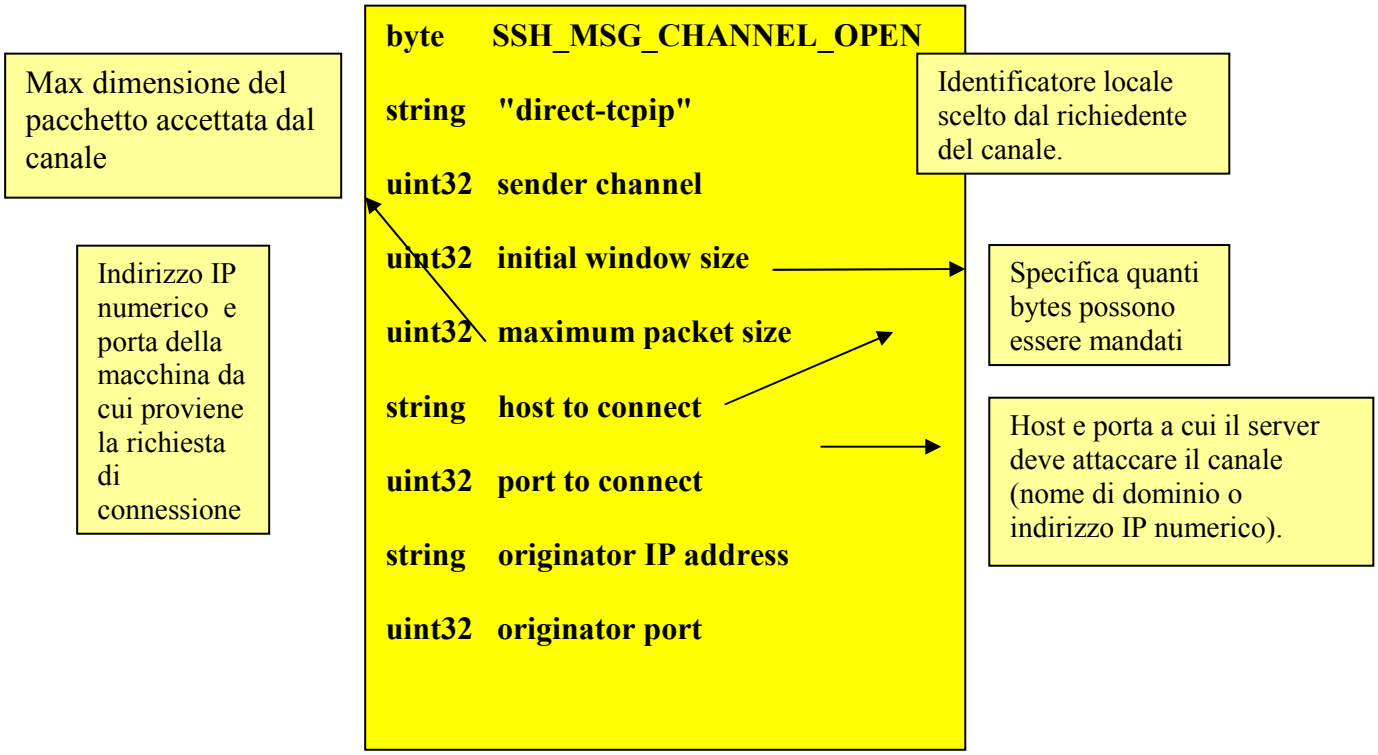
## TCP/IP Port Forwarding

E' un meccanismo che permette di creare un canale di comunicazione sicuro attraverso il quale veicolare qualsiasi tipo di connessione TCP.
In pratica viene creato un canale di comunicazione cifrato tra la porta all'indirizzo remoto a cui ci si vuole collegare e una porta locale libera.
In questo modo le applicazioni punteranno il collegamento alla porta locale e la connessione verrà inoltrata automaticamente all'host remoto tramite un canale sicuro.

La richiesta di un port forwarding è di tipo globale (coinvolge globalmente lo stato della porta remota ed è indipendente dai singoli canali) ed avviene tramite l'invio del seguente pacchetto:




Solo nel momento in cui una connessione arriva alla porta locale per la quale era stato richiesto un forwarding viene effettivamente aperto il canale che inoltra la porta verso l'altro lato.
A questo scopo viene inviato il seguente pacchetto al server:




**Perché fare TCP/IP Port Forwarding?**

Il Tunneling di SSH permette con semplicità di rendere sicure applicazioni che, altrimenti, invierebbero informazioni non protette su reti pubbliche.
I messaggi dell'applicazione trasmessi da una parte all'altra del canale SSH sono infatti protetti da sistemi crittografici scelti per quella connessione.
Inoltre poiché più applicazioni possono essere commutate su una singola connessione SSH , i filtri effettuati dai firewall e router possono essere ristretti ad una sola porta: la porta SSH 22.

**Problematiche**

1. Consideriamo un tele-lavoratore che usa Internet per accedere al proprio servizio di posta elettronica. Quando il client del tele-lavoratore spedisce la posta, i messaggi vengono trasmessi ad un server SMTP; invece quando il client legge la posta, le intestazioni e il corpo dei messaggi vengono scaricati da un server POP3 o IMAP. Chiunque, e in qualsiasi punto del percorso tra il proprio computer e il server Internet, mediante l'utilizzo di uno "***sniffer***" può intercettare non solo il testo in chiaro del messaggio, ma anche indirizzi e-mail, nomi utenti e password.
   Quindi, con un attacco del tipo *"man in the middle"* un intruso può leggere, intercettare o distruggere i messaggi.
   Sono state prese alcune misure per la sicurezza della posta elettronica (ad esempio PGP) ma queste
   - cifrano ed applicano la firma digitale solo al testo dei messaggi lasciando in chiaro le intestazioni;
   - non proteggono il mail server da attacchi, quindi un hacker munito di un software di scansione delle porte può riuscire ad individuare mail server in ascolto sulle porte SMTP, POP, IMAP ben note e quindi sfruttare le vulnerabilità conosciute nel sistema o nel software di posta elettronica , trasmettere posta indesiderata (spam) o prendere il controllo del server con attacchi del tipo "*Denial of Service*".

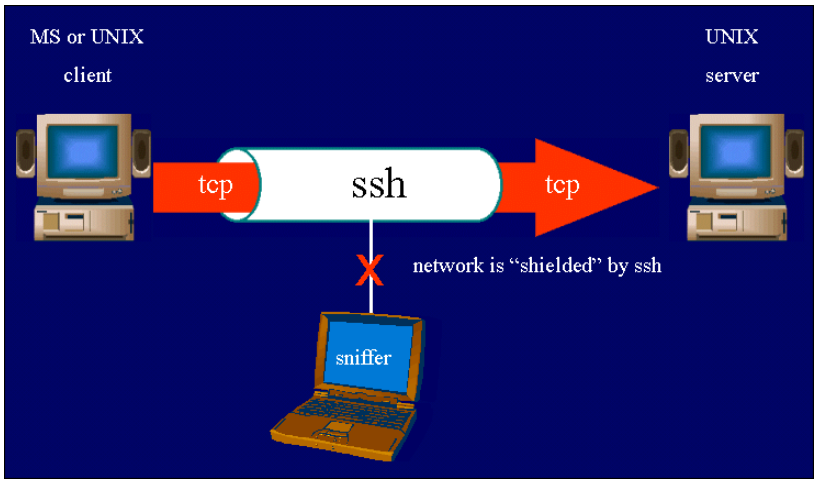
   Il tunneling con Secure Shell può aiutare ad eliminare le porte accessibili, bloccando gli utenti non autorizzati e assicurando la privacy e l'integrità di tutto il traffico SMTP, POP, IMAP scambiato tra client di posta e server.

2. All'interno di una Intranet le applicazioni client/server che scambiano dati sensibili devono essere protette da abusi da parte di malintenzionati e non è sufficiente limitarsi al controllo dell'accesso e all'autenticazione.
Ad esempio Ethernet è una particolare LAN (Local Area Network) che permette il broadcast, quindi qualsiasi PC sulla LAN può bloccare passivamente il traffico senza essere rilevato. Un hacker può, quindi, realizzare attacchi del tipo "*man in the middle*" modificando ed inserendo pacchetti sul traffico LAN.

3. Oggi molte compagnie condividono risorse in rete, ad esempio tramite il protocollo NFS. Ciascuna risorsa condivisa è un bene economico che va protetta da attacchi di tipo "*Denial of Service*" , modifiche e accessi non autorizzati. Il controllo degli accessi non basta, va infatti anche garantita l'integrità dei dati.
Il tunneling con SSH fornisce un meccanismo di autenticazione forte, un controllo dell'accesso e privacy per i files condivisi ed effettua il multiplexing dei vari flussi di dati non sicuri su una singola porta sotto il completo controllo di del server SSH che ne è l'unico amministratore.

**Funzionamento**

Una volta inoltrata una porta, il Client Secure Shell (SecureCRT) rimane in ascolto sulla specifica porta sull'host locale. Il server Secure Shell (Vshell) apre una connessione con l'host remoto dove il server delle applicazioni è in esecuzione.
Indicando con localhost l'host dell'applicazione client e con localport la porta su cui il client invia i dati e su cui la SecureCRT è in ascolto, con remotehost l'host dell'applicazione server e con remoteport la porta su cui la Vshell invia i dati e su cui l'applicazione server è in ascolto, la trasmissione avviene nel modo seguente.
I pacchetti inviati dal client sul localhost:localport vengono intercettati da SecureCRT, criptati e viene quindi effettuato il tunneling attraverso la connessione Secure Shell verso la Vshell che decodifica tali pacchetti rinviandoli **come testo in chiaro** attraverso la connessione TCP al server in ascolto sulla remotehost:remoteport.




Notare che mentre il traffico che transita tra la Secure CRT e la Vshell è protetto mediante la crittografia, il traffico tra tra la Vshell e il remote host non lo è.

Di norma Vshell è situato all'interno del perimetro della rete locale protetto da un firewall che viene configurato in modo da consentire connessioni di tipo Secure Shell ma non connessioni da parte di altre applicazioni di cui viene effettuato il tunneling.
In questo modo il firewall protegge lo scambio delle informazioni in chiaro verso il server.
Ovviamente non siamo certi della completa sicurezza del sistema: se un hacker penetra in un firewall mal configurato e, ad esempio, sfrutta la debole password dell'amministratore per loggarsi nel Server Secure Shell, il Secure Tunneling non può evitare che dei dati cadano nelle mani sbagliate.
Lasciando aperta la porta della Secure Shell sul firewall, viene delegato il controllo delle applicazioni in tunneling alla Vshell. Abbiamo così un **unico** punto di controllo per l'autenticazione remota degli utenti.
Inoltre possono anche essere creati filtri Vshell in modo da permettere o negare la connessione Secure Shell da specifici indirizzi IP, host, subnet o interi domini.

## Conclusioni

Come osservato, nonostante l'utilizzo di SSH il sistema non potrà dirsi completamente sicuro ma impiegando Vshell e SecureCRT si può creare un'ampia piattaforma di tunneling che può essere usata per implementare un'ampia varietà di politiche di sicurezza garantendo privatezza, integrità e autenticità di molti tipi di applicazioni.

## Bibliografia

Le informazioni sono state prese dal sito segnalato dal professore www.dia.unisa.it.
Inoltre ho consultato due tesine reperibili on line svolte da studenti del corso di laurea in Ingegneria Informatica dell'Università degli Studi "Roma Tre".