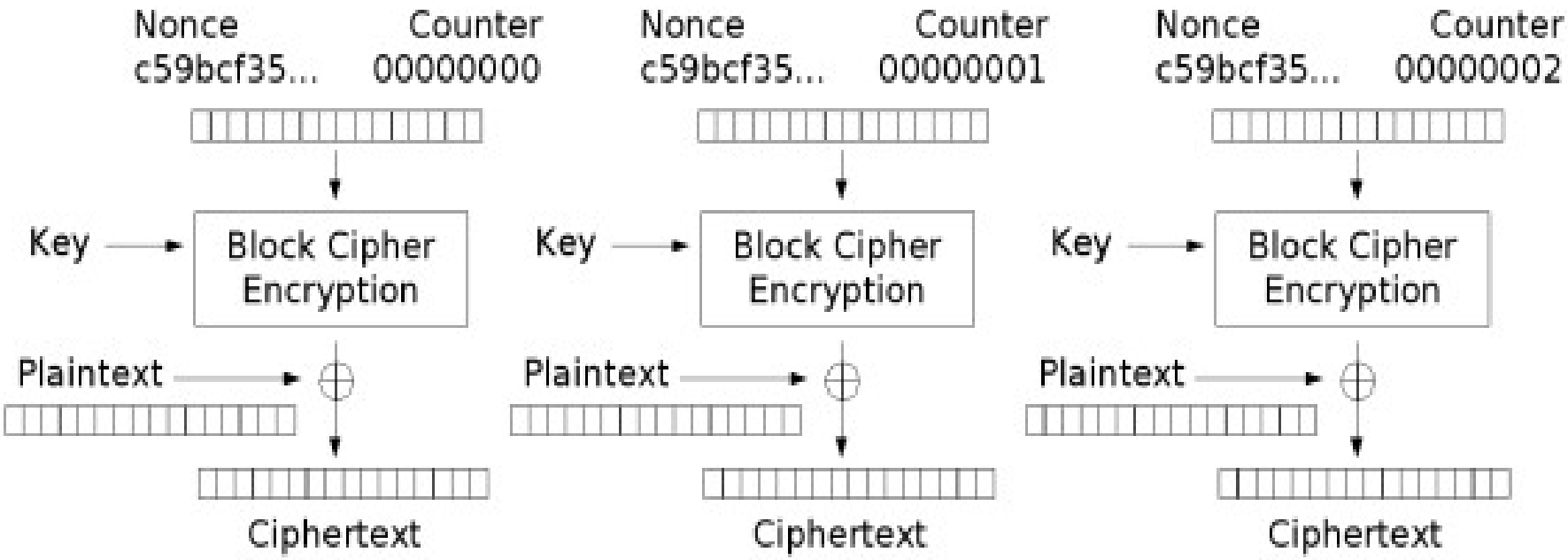
Crittografia avanzata
Lezione dell'11 Maggio 2011

CTR: counter mode



Counter (CTR) mode encryption

Harware fault attack

- Vedi paper: ”Specifications overview for counter mode of operation. security aspects in case of faults”

Secure Shell/SSH

Storia

- Sviluppato nel 1995 da Tatu Ylönen a Helsinki per sostituire telnet
  - Telnet era la prassi per la gestione remota, insieme alle r-utility
  - Presentato allo Usenix del 1996... c'ero :)
  - In quel periodo sono state sviluppate altre proposte (es. STEL di Unimi), il tema era caldo
  - SSH era un sostituto trasparente per le r-utility, e questo ne ha determinato il successo

Versioni

- 1.x Vulnerabile a diversi attacchi, non deve essere più utilizzata
- 2.x Nel 2008 è stata trovata una vulnerabilità di SSH2 (tutte le versioni) legata all'uso di CBC
  - “soluzione” simile a quella per il padding di SSL, non del tutto efficace
    - Abbastanza per gli usi pratici, e CBC non è più la modalità di default
- Altre vulnerabilità alla fine

Funzionamento generale

- Tre “strati”
  - Trasporto: stabilisce una connessione con il server, autentica il server e crea un canale cifrato
    - Di nuovo, la prima attenzione è autenticare il server
  - Autenticazione: autentica (in modo flessibile) il client
  - Connessione: gestisce più canali all'interno dello stesso canale cifrato

Transport layer

Transport layer

- Fornisce:
  - Integrità
  - Compressione
  - PFS
- Si appoggia a TCP per la garanzia di consegna ordinata

Il protocollo

- Scambio di messaggi sulla versione
  - Messaggi in chiaro, non autenticati, non c'è enforcement dell'ordine, neppure sullo scambio successivo
  - Possono essere preceduti da stringhe di testo che non iniziano per SSH-
    - I client devono supportare questa funzionalità

SSH-protoversion-softwareversion SP comments CR LF

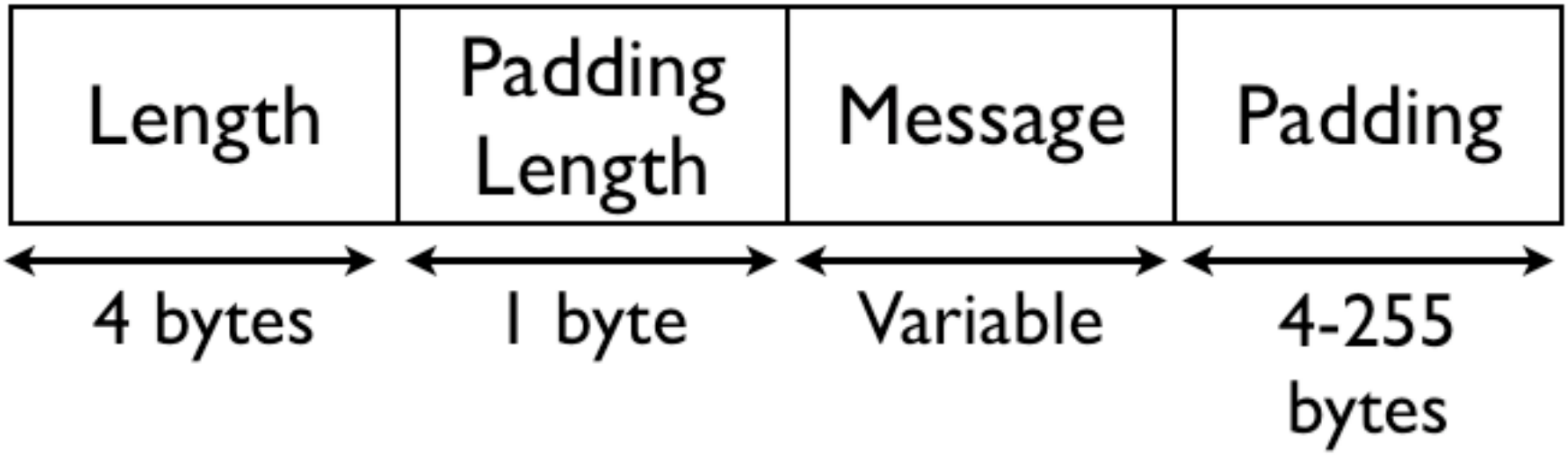
es.

SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu6

Compatibilità

- Server che supportino la 2.0, se sono in modalità “compatibile” con le 1.x, identificano la 2.0 con 1.99
  - I client 2.0 lo devono capire
- Client che supportino il 2.0, se ricevono dal server un'indicazione 1.x diversa da 1.99 devono eventualmente chiudere la connessione in caso di incompatibilità o corruzione del protocollo

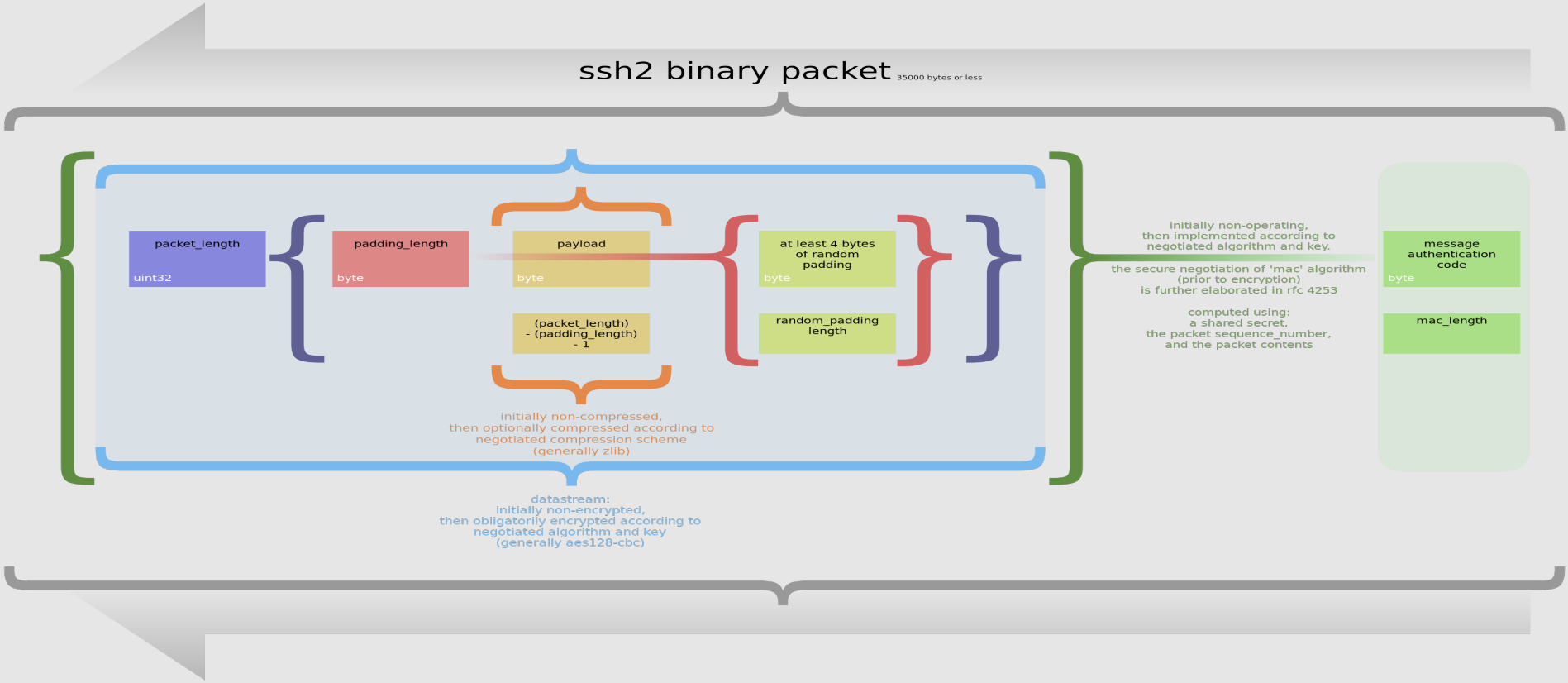
Pacchetto binario



Message type

- Il primo byte del messaggio indica il tipo
  - SSH_MSG_DISCONNECT 1
  - SSH_MSG_IGNORE 2
  - SSH_MSG_UNIMPLEMENTED 3
  - SSH_MSG_DEBUG 4
  - SSH_MSG_SERVICE_REQUEST 5
  - SSH_MSG_SERVICE_ACCEPT 6
  - SSH_MSG_KEXINIT 20
  - SSH_MSG_NEWKEYS 21

Pacchetto binario



sequence_number

...is an implicit packet sequence number represented as uint32. The packet sequence_number itself is not included in the packet sent over the wire. The sequence_number is initialized to zero for the first packet, and is incremented after every packet (regargless of whether encryption or MAC is in use). It is never reset (even if keys/algorithms are renegotiated later!). It wraps around to zero after every 2^32 packets.

Padding

- Obbligatorio di almeno 4 byte
  - Anche per stream cipher
- Dovrebbero essere bit casuali

SSH_MSG_KEXINIT: Key Exchange Init

- Contrattazione degli algoritmi
  - Client a server, server a client, di nuovo senza ordine
    - Per ogni lista c'è la lunghezza
  - Ogni lista deve avere almeno un elemento
  - Il primo di ogni lista è quello preferito (guessed)
  - Linguaggio: è possibile indicare un linguaggio, se non è necessario la lista deve essere vuota
  - È possibile cercare di “indovinare” anche prima di vedere la lista
    - Se si sbaglia, il pacchetto errato deve essere ignorato
  - Il pacchetto contiene un Cookie casuale (un nonce)
  - first_kex_packet_follows: true se segue un pacchetto guessed

Pacchetti ammessi

- Dopo SSH_MSG_KEXINIT, fino alla SSH_MSG_NEWKEYS, i pacchetti possono essere solo:
  - Transport layer generic messages (1 to 19) (but SSH_MSG_SERVICE_REQUEST and SSH_MSG_SERVICE_ACCEPT MUST NOT be sent);
  - Algorithm negotiation messages (20 to 29) (but further SSH_MSG_KEXINIT messages MUST NOT be sent);
  - Specific key exchange method messages (30 to 49).

Scelta dell'algoritmo di scambio chiavi

- Se le parti preferiscono lo stesso algoritmo, si usa quello. Altrimenti, si scorre la lista del client scegliendo il primo che:
  - the server also supports the algorithm,
  - if the algorithm requires an encryption-capable host key, there is an encryption-capable algorithm on the server's server_host_key_algorithms that is also supported by the client, and
  - if the algorithm requires a signature-capable host key, there is a signature-capable algorithm on the server's server_host_key_algorithms that is also supported by the client.
- Se non si trova un algoritmo, si devono disconnettere

Scelta degli altri algoritmi

- Guidata dall'algoritmo di scambio chiavi
- Per il resto, si sceglie il primo della lista del client che è supportato anche dal server
- Esiste l'algoritmo “none”, ma va usato solo per test

Key Exchange

- Segue la fase di key exchange effettiva
  - dipende dall'algoritmo
  - Comprende una funzione di hash
- Output:
  - Una chiave condivisa K
  - Un hash H di alcuni campi degli scambi precedenti, comprendenti i cookies, il payload del SSH_MSG_KEXINIT, la chiave pubblica del server, ecc.
  - H è anche il session identifier, e non viene ricontrattato

Calcolo delle chiavi

- Initial IV client to server: HASH(K || H || "A" || session_id) (Here K is encoded as mpint and "A" as byte and session_id as raw data. "A" means the single character A, ASCII 65).
- Initial IV server to client: HASH(K || H || "B" || session_id)
- Encryption key client to server: HASH(K || H || "C" || session_id)
- Encryption key server to client: HASH(K || H || "D" || session_id)
- Integrity key client to server: HASH(K || H || "E" || session_id)
- Integrity key server to client: HASH(K || H || "F" || session_id)
- La chiave sono i primi bit ottenuti dall'HASH; se i bit non bastano, si calcola:
  - K1 = HASH(K || H || X || session_id) (X is e.g., "A")
  - K2 = HASH(K || H || K1)
  - K3 = HASH(K || H || K1 || K2)
  - ...
  - key = K1 || K2 || K3 || ...

SSH_MSG_NEWKEYS

- Corrisponde allo “switch” al nuovo contesto
- Si invia con le vecchie chiavi (se ci sono, o in chiaro)
- Di qui in poi si usano le nuove chiavi
- In caso di problemi, si risponde con un SSH_MSG_DISCONNECT
- Di qui, un nuovo SSH_MSG_KEXINIT avvia una nuova ricontrattazione
  - I ruoli non possono essere scambiati
  - È suggerita una ricontrattazione dopo 1Gbyte di dati scambiati

MAC

- mac = MAC(key, sequence_number || unencrypted_packet)
- Il sequence number non viene trasmesso, è nello stato locale
- Non viene mai reinizializzato
  - Cicla su 32 bit

SSH-DISCONNECT

- SSH_DISCONNECT_HOST_NOT_ALLOWED_TO_CONNECT 1
- SSH_DISCONNECT_PROTOCOL_ERROR 2
- SSH_DISCONNECT_KEY_EXCHANGE_FAILED 3
- SSH_DISCONNECT_RESERVED 4
- SSH_DISCONNECT_MAC_ERROR 5
- SSH_DISCONNECT_COMPRESSION_ERROR 6
- SSH_DISCONNECT_SERVICE_NOT_AVAILABLE 7
- SSH_DISCONNECT_PROTOCOL_VERSION_NOT_SUPPORTED 8
- SSH_DISCONNECT_HOST_KEY_NOT_VERIFIABLE 9
- SSH_DISCONNECT_CONNECTION_LOST 10
- SSH_DISCONNECT_BY_APPLICATION 11
- SSH_DISCONNECT_TOO_MANY_CONNECTIONS 12
- SSH_DISCONNECT_AUTH_CANCELLED_BY_USER 13
- SSH_DISCONNECT_NO_MORE_AUTH_METHODS_AVAILABLE 14
- SSH_DISCONNECT_ILLEGAL_USER_NAME 15

Altri pacchetti

- Ignored Data Message (usato contro attacchi a CBC)
  - byte SSH_MSG_IGNORE
  - string data
- Debug Message
  - byte SSH_MSG_DEBUG
  - boolean always_display
  - string message in ISO-10646 UTF-8 encoding
  - string language tag
- Non implementato (risposta)
  - byte SSH_MSG_UNIMPLEMENTED
  - uint32 packet sequence number of rejected message

Autenticazione del server

- Una volta calcolato H, il server lo firma con la propria chiave privata, in modo che il client lo possa verificare (H è un nonce)
- Il client come ottiene la chiave pubblica del server?
  - È possibile certificare la chiave (il certificato è fornito insieme alla chiave)
  - Scambio “off line”: sistemista raccoglie un hash all'installazione e poi lo verifica
  - Scambio nel protocollo: al primo accesso viene accettata la chiave, eventuali modifiche devono essere autorizzate (rischio di MitM)

Service requests

- Dopo (in generale) una contrattazione, possono essere inviate delle service request (protocol layers);
  - ssh-userauth
  - Ssh-connect

byte SSH_MSG_SERVICE_REQUEST

string service name

Protocol downgrade

- Il primo scambio, che stabilisce la versione del protocollo, è incluso in H
  - Se viene modificato, il protocollo 2.0 se ne accorge
  - Ma se il downgrade è tale da scendere a una versione vulnerabile del protocollo che non abbia questa protezione, allora il meccanismo non è efficace
  - Il problema però è il fatto che client e server accettino una versione debole
    - Necessario per alcuni client/server obsoleti..?

DH definito nell'RFC?

- L'RFC 4253 descrive DH nella sezione 8
- Non è meglio avere un'unica definizione:
  - Degli algoritmi
  - Dei loro codici
  - …
- Perché non usa direttamente HMAC?
- Perché riscrivere tante cose?
- Esempio: introduzione di SHA-256
  - Ancora non è completa...

User Authentication

Richiesta

- Messaggio:
  - byte SSH_MSG_USERAUTH_REQUEST
  - string user name in ISO-10646 UTF-8 encoding
  - string service name in US-ASCII
  - string method name in US-ASCII
  - .... method specific fields
- Sono suggeriti dei timeout (10 min. per l'autenticazione, 20 tentativi...)

Metodi

- I metodi disponibili vengono elencati dal server nella risposta
  - Il client li può tentare nell'ordine che preferisce
- Definiti:
  - "publickey" REQUIRED
  - "password" OPTIONAL
  - "hostbased" OPTIONAL
  - "none" NOT RECOMMENDED

Autenticazione

- Possono essere mandate più richieste senza aspettare la risposta alle precedenti
  - Questa “asincronicità” compare in diverse parti del protocollo, e non è comune
  - Se una richiesta richiede più scambi di messaggi, viene abortita da una richiesta successiva prima dei messaggi

Risposte

- Fallimento o “continua”
  - byte SSH_MSG_USERAUTH_FAILURE
  - name-list authentications that can continue
  - boolean partial success
- Elenco di metodi ancora validi
- Partial success è per autenticazione multi-step
- Successo:
  - byte SSH_MSG_USERAUTH_SUCCESS
  - Messaggi di autenticazione successivi sono ignorati

Banner

- Necessario in alcuni casi/giurisdizioni/politiche
  - byte SSH_MSG_USERAUTH_BANNER
  - string message in ISO-10646 UTF-8 encoding [RFC3629]
  - string language tag [RFC3066]

Publickey

- Messaggio:
  - byte SSH_MSG_USERAUTH_REQUEST
  - string user name in ISO-10646 UTF-8 enc.
  - string service name in US-ASCII
  - string "publickey"
  - boolean FALSE
  - string public key algorithm name
  - string public key blob (può contenere cert.)

Publickey

- Risposta del server:
  - byte SSH_MSG_USERAUTH_PK_OK
  - string public key algorithm name from the request
  - string public key blob from the request
- Client:
  - byte SSH_MSG_USERAUTH_REQUEST
  - string user name
  - string service name
  - string "publickey"
  - boolean TRUE
  - string public key algorithm name
  - string public key to be used for authentication
  - string signature

Firma

- Dati firmati:
  - string session identifier
  - byte SSH_MSG_USERAUTH_REQUEST
  - string user name
  - string service name
  - string "publickey"
  - boolean TRUE
  - string public key algorithm name
  - string public key to be used for authentication
  -

Autenticazione host based

- Messaggio:
  - byte SSH_MSG_USERAUTH_REQUEST
  - string user name
  - string service name
  - string "hostbased"
  - string public key algorithm for host key
  - string public host key and certificates for client host
  - string client host name expressed as the FQDN in US-ASCII
  - string user name on the client host
  - string signature
- Si usa la chiave del client host, verificata dal server

Connection layer

Funzionalità “di rete”

- Fornisce:
  - Login interattivo
  - Esecuzione di comandi
  - Forwarding di sessioni X
  - Forwarding di connessioni TCP
  - Tutto in multiplexing sullo stesso canale
- Di interesse limitato per questo corso

Apertura di un canale

- Apertura di un canale:
  - byte SSH_MSG_CHANNEL_OPEN
  - string channel type in US-ASCII only
  - uint32 sender channel
  - uint32 initial window size
  - uint32 maximum packet size
  - .... channel type specific data follows
- I ogni messaggio il canale è identificato da un sender number e un receiver number , per il multiplexing

Tipi di canale

- *shell* for terminal shells, SFTP and exec requests (including SCP transfers)
  - Nota SFTP non è ftp su SSH, è un protocollo a parte, più simile a un filesystem remoto – draft
- *direct-tcpip* for client-to-server forwarded connections
- *forwarded-tcpip* for server-to-client forwarded connections
-

Altre note sulla sicurezza

- Possibili perché ci si riferisce a un protocollo “stagionato”
- Il sequence number cicla ogni 2^{32} pacchetti, ma il rekeying ogni Giga avviene al più ogni 2^{28}
- Il session ID può essere reso noto (es. alle applicazioni) senza perdita di sicurezza
- Il protocollo offre spazio per covert channels (es. nel padding)
- La PFS si ha anche se poi le chiavi private vengono rivelate,se si usa DH come indicato
- La traffic analysis può rivelare la lunghezza della password dell'utente

Ancora note sulla sicurezza

- L'autenticazione del client si appoggia alla protezione del transport layer
  - Alcuni meccanismi sarebbero utilizzabili anche con un livello di trasporto più debile
- Criticità dei messaggi di debug
  - es. per analizzare la configurazione del server
- Configurazione del server
  - es. sulle shell interattive

RFC di riferimento

- The Secure Shell (SSH) Protocol Assigned Numbers (RFC 4250) (44010 bytes)
- Generic Message Exchange Authentication For The Secure Shell Protocol (SSH) (RFC 4256) (24728 bytes)
- Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints (RFC 4255) (18399 bytes)
- The Secure Shell (SSH) Connection Protocol (RFC 4254) (50338 bytes)
- The Secure Shell (SSH) Transport Layer Protocol (RFC 4253) (68263 bytes)
- The Secure Shell (SSH) Authentication Protocol (RFC 4252) (34268 bytes)
- The Secure Shell (SSH) Protocol Architecture (RFC 4251) (71750 bytes)
- The Secure Shell (SSH) Transport Layer Encryption Modes (RFC 4344) (27521 bytes)
- Secure Shell (SSH) Session Channel Break Extension (RFC 4335) (11370 bytes)
- Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol (RFC 4419) (18356 bytes)
- Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell Protocol (RFC 4462) (65280 bytes)
- The Secure Shell (SSH) Public Key File Format (RFC 4716) (18395 bytes)
- Secure Shell Public-Key Subsystem (RFC 4819) (32794 bytes)

Usi di SSH

- Come si vede, l'uso per “inviare comandi” non è centrale
  - Anche se esplicitamente supportato nel connection layer
- Si tratta essenzialmente di un meccanismo di tunneling cifrato
  - Diversamente da SSL, l'autenticazione del client non è parte del setup del canale, ma è fatta dentro al canale