Università degli studi di Roma Tre

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Protocollo SSH

Tesina di Elementi di Crittografia
Prof.ssa Maria Gabriella Murciano

Autori: *Cappiello Irene - Di Fazio Alessio - El Saidi Amir*

Anno accademico 2005/2006

Sommario

In questa tesina presentiamo una panoramica generale sul protocollo SSH *Secure SHell*, che facilita i collegamenti sicuri tra due sistemi, usando un'architettura di tipo client/server e permettendo agli utenti di registrarsi in sistemi host server, in remoto.
Dopo un'introduzione storica del protocollo SSH e l'analisi delle sue versioni utilizzate,vengono presentati i principali algoritmi di crittografia e le funzioni hash utilizzate da SSH, per una maggiore comprensione dei suoi meccanismi. Viene infine analizzata l'architettura del protocollo e il suo funzionamento.

Indice

Introduzione **4**
Sicurezza informatica 4
Secure Shell 5
Un po' di storia 6
Versioni a confronto 6

1 Algoritmi di crittografia supportati **8**
Algoritmo Diffie-Hellman 8
1.1 Algoritmi a chiave pubblica 9
1.1.1 Algoritmo RSA 9
1.1.2 Algoritmo DSA 11
1.2 Algoritmi a chiave simmetrica 12
1.3 Funzioni hash 13
1.3.1 Algoritmi di cifratura 13

2 Struttura e Funzionamento **15**
2.1 Architettura del protocollo 15
2.1.1 Host Keys 16
2.2 Transport Layer Protocol 17
2.2.1 Scambio delle versioni 17
2.2.2 Negoziazione degli algoritmi 17
2.2.3 Scambio di chiavi 18
2.2.4 Compressione 19
2.2.5 Integrità dei dati 19
2.2.6 Cifratura 19
2.3 Authentication Protocol 20
2.3.1 Autenticazione: schema generale 20
2.3.2 Metodo di autenticazione a chiave pubblica. 21
2.3.3 Metodo di autenticazione con password 21
2.3.4 Autenticazione Host-based 21

2.4 Connection Protocol 22
2.4.1 Meccanismo dei canali: Schema generale 22
2.4.2 TCP/IP Port Forwarding 23
2.5 Estensibilità di SSH 23
2.5.1 Nomi di algoritmi 23
2.6 Politiche Locali 24
2.7 Aspetti di Sicurezza 24

3 OpenSSH **26**

4 Una sessione sicura **28**

Conclusioni **29**

Introduzione

La necessità di nascondere messaggi strategici da occhi nemici è molto antica: si pensi che la prima traccia storica di uso della crittografia risale a *Gaio Giulio Cesare*. A lui si attribuisce l'uso (se non addirittura l'invenzione) del *cifrario di Cesare*, un sistema crittografico che diede vita ad un concetto totalmente nuovo e ottimo per comprendere le idee basilari della crittografia e i primi attacchi della sua "avversari": *la crittanalisi*.

A partire dal *cifrario di Cesare*, i sistemi via via escogitati sono stati vari e man mano più evoluti: da questo elementare cifrario si è passati al più generalizzato *cifrario a scorrimento*, successivamente al *cifrario monoalfabetico*, che a sua volta ha gettato le basi per i *cifrari polialfabetici*, tra cui l'imbattibile *cifrario di Vigénère*[1], come almeno era considerato quando Vigénère lo ideò (unendo molte idee che erano già state introdotte da tempo), visto che nel 1863 il colonnello prussiano Friedrich Kasiski pubblicò un metodo per "forzarlo" (chiamato *metodo Kasiski - Prima guerra mondiale, 1863*).

Sicurezza informatica

La Sicurezza informatica è quella branca dell'informatica che si occupa della salvaguardia dei sistemi informatici da potenziali rischi e/o violazioni dei dati. I principali aspetti di protezione del dato sono la confidenzialità, l'integrità e la disponibilità.

La protezione dagli attacchi informatici è quindi la chiave della sicurezza dei sistemi telematici e si ottiene agendo su più livelli: a *livello fisico/materiale* (anche se in questa fase intendiamo la sicurezza reale, non si tratta naturalmente di "sicurezza informatica") agendo direttamente sulla protezione dei sistemi più importanti, quali Server, Mainframe di database, ecc... Il livello logico che antepone uno strato di *autenticazione* e quindi una *autorizzazione* da parte dell'entità che amministra il sistema protetto.

Sulla base di quanto detto, nella "sicurezza informatica" si usa spesso dividere il concetto di sicurezza, in due sottoclassi: la *sicurezza passiva* e la *sicurezza attiva*.

Per *sicurezza passiva* si intendono un insieme di strumenti e tecniche che vengono adottate a scopo puramente *difensivo*, ossia quel complesso di soluzioni il cui obiettivo è quello di impedire che utenti non autorizzati possano accedere a risorse, sistemi, impianti, informazioni e dati di natura *riservata*. È chiaro che il concetto di *sicurezza passiva* può essere esteso a molte altre situazioni al di fuori

[1] Il *cifrario di Vigénère* è il più semplice dei cifrari polialfabetici; fu pubblicato nel 1586 in un trattato di *Blaise de Vigénère*. Ritenuto per secoli un cifrario inattaccabile, ha goduto di una fama dovuta soprattutto alla sua semplicità e in buona parte immeritata essendo molto più debole di altri codici polialfabetici precedenti quali il *disco dell'Alberti* o *le cifre del Bellaso*.

dell'ambiente informatico, basti pensare alle soluzioni che si adottano per proteggere luoghi importanti come caveau di banche o locali che devono rimanere protetti ad eventuali azioni di intrusione non autorizzate. In questi casi si adottano strumenti protettivi quali: porte di accesso blindate, personale di sicurezza, ecc... Questi sono indubbiamente considerati componenti di *sicurezza passiva*.

Per *sicurezza attiva* si intendono, invece, le tecniche e gli strumenti mediante i quali le informazioni ed i dati di natura riservata sono resi intrinsecamente sicuri, proteggendo gli stessi sia dalla possibilità che un utente - *hacker* - non autorizzato possa accedervi (*proprietà di confidenzialità*[2]), sia dalla possibilità che un individuo non autorizzato possa modificarli (*proprietà di integrità*[3]).

Secure Shell

Partendo dal concetto di *sicurezza passiva*, uno strumento fondamentale che si è affermato il questi ultimi anni, è il protocollo *SSH* (***S**ecure **SH**ell*). Tale sistema permette di avere una sessione remota con un'altra macchina, dove si possono eseguire operazioni in maniera protetta, attraverso una connessione dati criptata.

Analogamente ad altri protocolli come il famosissimo *Telnet*, SSH si presenta con un'interfaccia a riga di comando (solitamente una Shell[4]), dove tutto lo scambio delle informazioni avviene in maniera cifrata. Secure Shell fornisce una serie di servizi che lo rendono uno degli strumenti più potenti per l'amministrazione di remota di sistemi "Unix Like" [5] e di dispositivi di rete quali:

- un'infrastruttura per connessioni crittografate;
- un'autenticazione forte tra i due host e tra utente ed host;
- la possibilità di creare un canale di comunicazione sicuro attraverso il quale veicolare qualsiasi connessione TCP/IP;
- la soluzione ad alcuni problemi noti in TCP/IP come l'*IP spoofing*, il *DNS spoofing* e il *Routine spoofing*; dove con il termine *spoofing* si intende la falsificazione delle informazioni inerente ad un servizio.

[2]Con il termine confidenzialità si intende la protezione dei dati e delle informazioni scambiate tra un mittente e uno o più destinatari nei confronti di terze parti.

[3]Con il termine integrità si intende, nell'ambito della sicurezza informatica, la protezione dei dati e delle informazioni nei confronti delle modifiche del contenuto effettuate da una terza parte; viene compreso anche il caso limite della generazione ex novo di dati ed informazioni non lecite.

[4]In un sistema operativo, 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, insieme al kernel.

[5]per sistemi "Unix Like" si intende quei sistemi operativi derivati da *Unix* come Gnu/Linux, BSD, ecc...

Un po' di storia

Il protocollo *Secure Shell* venne sviluppato per la prima volta nel 1995 dal ricercatore finlandese *Tatu Ylönen* presso l'università finlandese *Helsinki*, dopo essere stato vittima di vari attacchi durante alcune sessioni tramite *Telnet*. In particolare gli attacchi erano stati compiuti tramite *sniffing* per cercare di recuperare la password. Per definizione, la parola *sniffing* indica quell'attività che permette un'intercettazione passiva dei dati che transitano in una rete telematica. Tale attività può essere svolta sia per scopi legittimi (ad esempio l'individuazione di problemi di comunicazione o di tentativi di intrusione) sia per scopi illeciti come l'intercettazione fraudolenta di password o altre informazioni riservate.

Con il passare del tempo la versione di *SSH* - denominata versione *SSH1* - sviluppata da Tatu venne arricchita di funzionalità nuove tramite l'inserimento di modifiche (anche pesanti) nella struttura del codice. Divenne chiaro che il codice andava totalmente riscritto e riorganizzato. 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*, partendo dalla release 1.2.12 che era l'ultima rilasciata sotto autorizzazione libera. 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*. Successivamente vennero creati vari port ad altri sistemi operativi cosicché nel 2005 erano ormai moltissimi i sistemi che adottavano *OpenSSH*.

Versioni a confronto

Come introdotto, le versioni di *SSH* esistenti sono due: la versione *SSH1* e la versione *SSH2*. 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

[6]OpenBSD è un sistema operativo basato sulla variante BSD di Unix, considerato particolarmente interessante per la sua sicurezza e per possedere un sistema di crittografia integrato.

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-Hellman*[7], 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 1: Tabella comparativa delle versioni di SSH1 vs SSH2.

	Release SSH1	**Release 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.	Molteplici canali per una singola sessione.
Negoziazione	Algoritmo simmetrico per la negoziazione; stesso identificativo di sessione per entrambe le parti.	Algoritmo simmetrico a chiave pubblica, e compressione; chiave di sessione indipendente tra le due parti.
Algoritmi	Algoritmo RSA a chiave pubblica.	Utilizzo di RSA e DSA a chiave pubblica.
Chiave di sessione	Trasmessa dal lato client. Stessa chiave per la sessione.	Negoziata attraverso l'algoritmo Diffie-Helman. Chiave si sessione rinnovabile.

[7] Si veda la sezione 1

1 Algoritmi di crittografia supportati

I dati scambiati con SSH sono cifrati con algoritmi a chiave simmetrica negoziati tra le parti. Le chiavi per gli algoritmi di cifratura sono scambiate in precedenza tramite l'algoritmo di *Diffie-Hellman*. La cifratura previene, tra le altre cose, gli attacchi di tipo sniffing (cattura del traffico passante per una interfaccia di rete). Poiché i server SSH e i client possono essere configurati per autorizzare diversi tipi di autenticazione, questo metodo offre un ottimo controllo a entrambe le parti. Il server stabilisce i metodi di cifratura supportati e il client può scegliere l'ordine dei metodi di autenticazione da utilizzare. SSH supporta algoritmi di crittografia e sfrutta sia sistemi a chiave pubblica (RSA e DSA) che quelli a chiave simmetrica quali IDEA, DES, 3DES, Arcfour e Blowfish.

Algoritmo Diffie-Hellman

L'algoritmo *Diffie-Hellman* risale al 1976 ed è quindi uno dei più antichi algoritmi a chiave pubblica; gli autori furono anche i primi a proporre l'idea dei cifrari a chiave pubblica. Tale sistema è particolarmente adatto alla generazione di una chiave segreta tra due corrispondenti che comunicano attraverso un canale non sicuro. La sicurezza dell'algoritmo in esame si basa sulla complessità computazionale del calcolo del *logaritmo discreto*.

Passi dell'algoritmo Supponiamo che due utenti Alice e Bob vogliono scambiarsi una chiave segreta in modo sicuro. Alice genera e comunica pubblicamente a Bob un numero primo N molto elevato (ad esempio un numero costituito da 1024bit, circa 300 cifre decimali) e un generatore g (che esiste sicuramente essendo N primo); si potrebbero anche usare due numeri N e g pubblicati da un qualche ente. Arrivati a questo punto, l'algoritmo si composta nel seguente modo:

1. Alice genera un numero casuale $a < N$ e calcola $A = g^a mod N$. Il numero A viene comunicato pubblicamente a Bob;
2. In modo del tutto analogo Bob genera due numeri b e $B = g^b mod N$ e invia B ad Alice;
3. Alice calcola il numero $k = B^a mod N$;
4. Bob calcola $k = A^b mod N$.

È semplice intuire che le due chiavi k sono uguali! Infatti entrambe vale che $g^{a*b} mod N$. A questo punto Alice e Bob possono usare la chiave k per comunicare con un cifrario simmetrico ad esempio il *DES*.

Sicurezza dell'algoritmo - L'attacco dell'Hacker La cosa importante per la sicurezza è che un terzo che intercettasse i quattro numeri N, g, A, B non sarebbe in grado di ottenere $k = g^{ab}$ non conoscendo né a né b. In effetti $a = \log_g(A)$ e $b = \log_g(B)$ ma essendo il calcolo del logaritmo discreto computazionalmente proibitivo come una fattorizzazione è pressoché impossibile calcolare questi logaritmi per numeri di 1024 bit.

Come altri sistemi a chiave pubblica anche *DH* è però esposto all'attacco del terzo uomo interposto. Nel nostro esempio immaginiamo che Trudy intercetti il numero A che Alice invia a Bob e, fingendo di essere Bob, generi un suo numero B e lo invii ad Alice. A questo punto Alice e Trudy generano la chiave segreta k e comunicano via *DES*; e Trudy può tranquillamente leggere tutti i messaggi che Alice crede di inviare a Bob! Peggio ancora: Trudy può ripetere il gioco anche con Bob fingendosi Alice e di qui in avanti, intercettare e leggere tutta la corrispondenza tra i mal capitati Alice e Bob. Per prevenire simili attacchi la soluzione è quella di usare una *Certification Authority*[8] che garantisca l'identità dei corrispondenti. Alice e Bob potranno ad esempio identificarsi con un meccanismo di *firma digitale*, utilizzando le chiavi pubbliche fornite dall'ente certificatore.

1.1 Algoritmi a chiave pubblica

La crittografia a chiave pubblica o asimmetrica, conosciuta anche come crittografia a coppia di chiavi o a chiave pubblica/privata, è fondata sull'algoritmo RSA. Ogni attore coinvolto possiede una coppia di chiavi:

- la chiave privata, personale e segreta, viene utilizzata per decodificare un documento criptato;
- la chiave pubblica, che deve essere distribuita, serve a criptare un documento destinato alla persona che possiede la relativa chiave privata.

La sicurezza del sistema dipende dalla segretezza della chiave di codifica utilizzata, quindi è necessario almeno un canale sicuro attraverso il quale trasmettere la chiave.

1.1.1 Algoritmo RSA

Il nome dell'algoritmo RSA deriva dalla prima lettera dei cognomi di coloro che lo inventarono nell'Aprile del 1977: Ronald L. Rivest, Adi Shamir e Leonard M. Adleman.

[8] In crittografia, una *Certificate Authority* o *Certification Authority* (in breve *CA*) è una entità che rilascia certificati digitali verso terze parti.

Analisi dell'algoritmo L'algoritmo potrebbe essere facilmente suddiviso in due parti: la generazione della coppia di chiavi e il loro utilizzo.

La prima parte, la generazione della coppia di chiavi, viene solitamente effettuata in questo modo:

- vengono scelti due numeri primi *p*, *q* molto grandi;
- viene calcolato $n = pq$, e la funzione di Eulero $\phi(n) = (p-1)(q-1)$ dopo di che i due primi p, q vengono eliminati;
- si sceglie un intero e minore di $\phi(n)$ e primo con esso;
- utilizzando la versione estesa dell'algoritmo di Euclide viene calcolato l'intero d così da avere $e \,\mathrm{x}\, d = 1 \, mod \, \phi(n)$;
- vengono resi pubblici i valori e, n che costituiscono la chiave pubblica e mantenuto segreto d che, utilizzato con n rappresenta la chiave privata.

Una volta generata la coppia di chiavi, le operazioni effettuabili da due utenti, l'Utente A e l'Utente B, entrambi con la propria coppia di chiavi, pubblica e privata, più la chiave pubblica della controparte, sono:

- L'Utente A invia del materiale cifrato all'Utente B, utilizzando la chiave pubblica dell'Utente B.
  L'operazione di firma prevede l'utilizzo della chiave privata dell'Utente A
- Ricevuti i dati cifrati, l'Utente B utilizza la sua chiave privata per eseguire l'operazione opposta di decifratura.
  Per verificare i dati ricevuti, l'Utente B utilizzerà la chiave pubblica dell'utente A.

Il testo in chiaro viene visto come una stringa di bit e viene diviso in blocchi costituiti da kbit, dove k è il più grande intero che soddisfa la disequazione $2^k < n$.

A questo punto per ogni blocco M si procede con la cifratura calcolando $M_c = M^e \; mod \; n$. In ricezione, invece, per decifrare M_c si calcola $M_c^d \; mod \; n$. Come si può capire, da quanto appena detto, per la cifratura si devono conoscere e ed n, cioè la chiave pubblica, mentre per decifrare è necessario conoscere d ed n, ovvero la chiave segreta.

Sicurezza dell'RSA Il codice RSA viene considerato sicuro perché non è ancora stato trovato il modo per fattorizzare numeri primi molto grandi, che significa riuscire a trovare p e q conoscendo n. Nel corso degli anni l'algoritmo RSA ha più volte dimostrato la sua robustezza: in un esperimento del 1994, coordinato da Arjen Lenstra dei laboratori Bellcore, per rompere una chiave RSA di 129 cifre, svelando il meccanismo con cui quella chiave generava messaggi crittografati,

sono stati necessari 8 mesi di lavoro coordinato effettuato da 600 gruppi di ricerca sparsi in 25 paesi, che hanno messo a disposizione 1600 macchine da calcolo, facendole lavorare in parallelo collegate tra loro attraverso Internet.

Data la mole delle risorse necessarie per rompere la barriera di sicurezza dell'algoritmo RSA, è chiaro come un attacco alla privacy di un sistema a doppia chiave non sia praticamente realizzabile. Inoltre, nell'esperimento era stata utilizzata una chiave di 129 cifre mentre i programmi di crittografia attualmente a disposizione prevedono chiavi private con una robustezza che raggiunge e supera i 2048 bit, risultando quindi praticamente inattaccabili, visto anche che l'ordine di grandezza dei tempi necessari alla rottura di chiavi di questo tipo è esponenziale.

1.1.2 Algoritmo DSA

DSA (*Digital Signature Algorith*m) 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.
Nel 1994 il NIST, l'ente federale USA, nel quadro degli standard DSS (Digital Signature Standard) ha approvato l'algoritmo DSA come standard federale per la firma digitale (DSA non è utilizzabile per la crittazione). Usa un algoritmo inventato da David Kravitz e derivato da quello di Schnorr.
La configurazione dell'algoritmo DSA richiede di fissare alcuni parametri e quindi di generare le chiavi segreta e pubblica:

- Si fissa un numero primo m da usare nelle operazioni di resto modulo, con $2^{511} < m < 2^{512}$.
- Si fissa un numero Q, divisore primo di $m - 1$, con $2^{159} < Q < 2^{160}$.
- Si fissa un numero G, con $0 < G < m - 1$.
- Si sceglie una chiave segreta S_A, con $0 < S_A < Q$.
- La chiave pubblica si calcola come $P_A = G^{S_A}$.

Indicando esplicitamente il modulo impiegato come pedice della coppia di parentesi quadrate, per la firma del messaggio M si procede così:

- Si sceglie un numero h, con $0 < h < Q$.
- Si calcola $u = [[G^h]_m]_Q$.
- Si determina il valore di v risolvendo la relazione $D(M) = [S_A u + hv]_Q$, dove D() è la funzione digest scelta.
- La firma digitale di M è la coppia di numeri (u, v).

Per la verifica della firma digitale:

- Si determina w tale che $wv = [1]_Q$;
- Si calcola $i = [D(M)w]_Q$;
- Si calcola $l = [uw]_Q$;
- Si calcola $t = [[G^i P_A]_m]_Q$;
- Si verifica che sia $t = u$.

DSA è stato criticato dalla comunità crittografica per varie ragioni: la verifica della firma, una operazione senzaltro più comune della sua generazione, è particolarmente inefficiente ed è più lenta dalla generazione della firma stessa; l'iter di approvazione di questo standard da parte del NIST è stato piuttosto arbitrario e con un ampio contributo da parte di NSA; esistono alcuni particolari numeri primi che, se usati dal programma per la generazione della firma, possono indebolire l'algoritmo e, sebbene alcuni di questi numeri sono stati individuati, è plausibile che ve ne siano altri ancora da scoprire; il range dei parametri ammessi sembra piuttosto arbitrario.

1.2 Algoritmi a chiave simmetrica

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.

3DES Il 3DES è un cifrario a blocco che consiste nella tripla applicazione del DES (*Data Encryption Standard*) usando due chiavi, ciò irrobustisce molto il sistema rispetto al DES (anche se sembra che il 3DES non sia così sicuro quanto ci aspetteremmo). Il DES fu sviluppato da IBM nei primi anni '70 con la collaborazione dell'NSA ed usa chiavi a 64 bits, dei quali solo 56 sono effettivamente utilizzati, gli altri 8 servono per il controllo di parità. Il DES è stato forzato generando tutte le chiavi possibili (ricerca esaustiva); per il 3DES sono stati proposti vari tipi di attacco, ma la richiesta di informazioni necessarie per attuarli li rendono impraticabili. Il 3DES è compatibile all'indietro col DES: tutto ciò che è stato cifrato col DES può essere decifrato col 3DES.

BLOWFISH Il Blowfish, sviluppato da B. Schneier, 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.

1.3 Funzioni hash

Esistono dei meccanismi di cifratura che permettono soltanto di mascherare un'informazione senza prevedere una chiave per poter successivamente decodificare il messaggio. Tali metodi sono detti *funzioni hash* o cifratura a senso unico (*one-way hash function*) e sono utilizzati per la cifratura delle password o per la generazione di impronte digitali o message digest.
Le password, infatti, non sono memorizzate in chiaro sul sistema, ma viene memorizzata soltanto la loro cifratura, dalla quale non è possibile ritornare alla password in chiaro. Quando un utente vuole accedere al sistema, il meccanismo di autenticazione richiede la password, la cifra con l'algoritmo di hashing considerato e confronta la password cifrata con quella memorizzata sul sistema per l'utente in questione. Se le due password cifrate coincidono, all'utente è permesso accedere al sistema.
Il *message digest* invece è un valore caratteristico di un messaggio o documento, calcolato applicando al messaggio stesso la funzione hash, che varia sensibilmente al variare del contenuto del messaggio.
L'algoritmo di hashing deve essere in grado di generare un messaggio cifrato quanto più scorrelato possible dall'informazione originale e tale che sia estremamante improbabile che due messaggi diversi diano luogo allo stesso messaggio cifrato.

1.3.1 Algoritmi di cifratura

Di seguito sono riportati alcuni tra gli algoritmi più utilizzati per la cifratura con funzioni hash.

- MD5 (*Message Digest 5*) creato da R. Rivest nel 1991, è nato per sostituire MD4. Produce valori hash di 128 bit, accettando messaggi di input con lunghezza massima di 264 bit e scomponendoli in blocchi di 512 byte (viene eventualmente effettuato un padding dei bit mancanti nell'ultimo blocco - il padding comprende comunque un valore di 64 bit che indica la lunghezza del messaggio originale). Una descrizione dell'algoritmo è riportata nella RFC 1321.

- SHA (*Secure Hash Algorithm*) è un algoritmo sviluppato nel 1994 dal NIST (*National Istitute of Standard and Tecnology*), FIPS 180, e dalla NSA (*National Security Agency*). Fornisce valori hash di 160 bit. La cifratura viene effettuata su messaggi con lunghezza massima di 264 bit, scomponendoli in blocchi di 264 bit. Risulta un po' più lento di MD5, ma produce

un message digest più grosso che lo rende più sicuro agli attacchi per forza bruta.

- SHA-1 è una revisione del 1995 dell'algoritmo SHA, che ne corregge alcune falle (v. FIPS 180-1). Una sua descrizione è contenuta in nello standard ANSI X9.30.
- SHA-2 è una nuova versione dell'algoritmo SHA, pubblicata nel 2002, che fornisce valori hash di 256 bit

2 Struttura e Funzionamento

2.1 Architettura del protocollo

Il funzionamento di SSH è molto simile a quello di SSL (infatti non è una coincidenza che le funzioni crittografiche usate da OpenSSH siano fornite da OpenSSL, una versione open source del Secure Socket Layer sviluppato da Netscape). Entrambi possono instaurare una comunicazione cifrata autenticandosi usando host key ed eventualmente certificati che possono essere verificati tramite una autorità fidata.

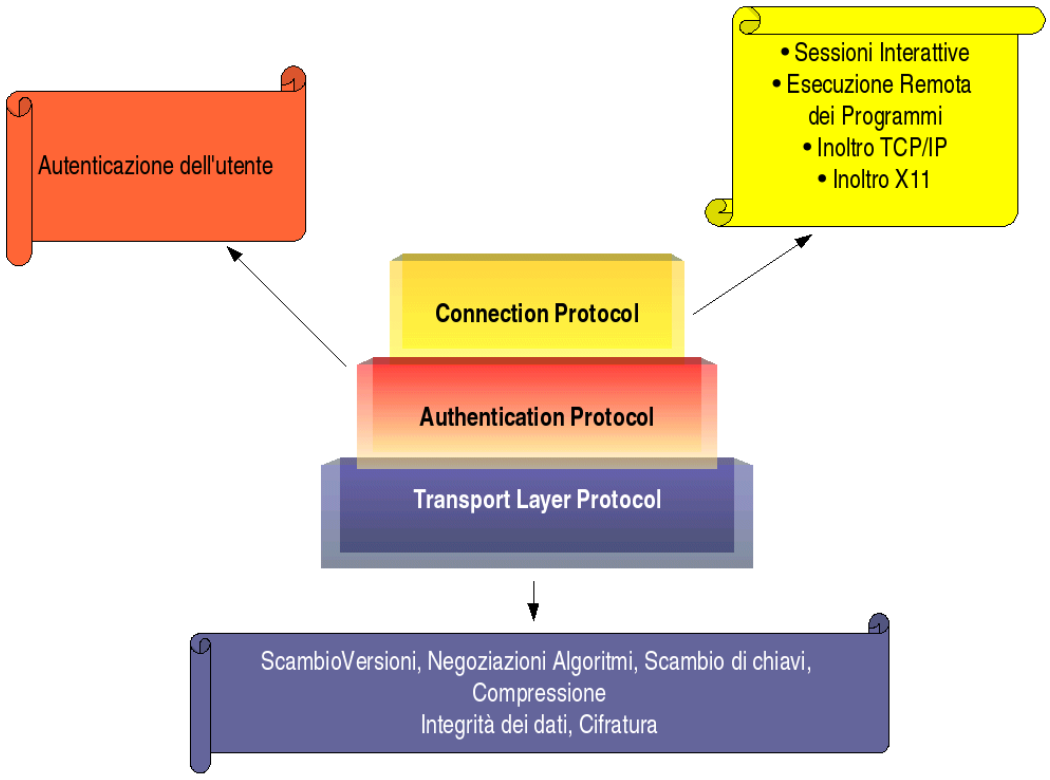

Figura 1: Architettura del protocollo SSH

Ecco come si instaura una connessione:
Prima il client ed il server si scambiano le loro chiavi pubbliche. Se la macchina del client riceve per la prima volta una data chiave pubblica, SSH chiede all'utente se accettare o meno la chiave. Successivamente client e server negoziano una chiave di sessione che sarà usata per cifrare tutti i dati seguenti attraverso un cifrario a blocchi come il 3DES o Blowfish. Il *Transport Layer Protocol* si occupa di questa prima parte, cioè di fornire autenticazione host, confidenzialità, integrità e opzionalmente compressione. Tale protocollo gira tipicamente su una

connessione TCP, ma potrebbe essere usato anche su un qualsiasi altro flusso di dati affidabile.

Lo *User Authentication Protocol* autentica l'utente del client sul server. Sono attualmente supportati tre tipi di autenticazione:

- Con **chiave pubblica**: il client invia un pacchetto firmato con la propria chiave privata. La firma è verificata dal server tramite la chiave pubblica del client, che il server può possedere da eventuali connessioni precedenti. Nel caso in cui il server non possieda la chiave pubblica del client, questo metodo fallisce.
- Con **password**: all'utente sul client viene presentato il solito *prompt* per l'inserimento della password. Generalmente questo metodo viene utilizzato solo alla prima connessione ad un server.
- **Host based**: è simile al metodo *rhosts* di UNIX, ma è più sicuro, in quanto aggiunge ad esso la verifica dell'identità dell'host tramite la Host key.

Il server guida l'autenticazione, inviando al client la lista contenente i metodi di autenticazione supportati. Il client sceglie quello che considera più conveniente. Infine, dopo una autenticazione con successo, tramite il *Connection Protocol* comincia la vera sessione: il client può richiedere una shell remota, l'esecuzione di un comando , un trasferimento di file sicuro, ecc. Il protocollo divide la connessione in canali logici; tutti questi canali sono multiplexati in una singola connessione. In questo modo è possibile accedere a più servizi con un singolo *tunnel cifrato*.

2.1.1 Host Keys

Ogni server dovrebbe avere una Host Key per ogni algoritmo a chiave pubblica supportato. La Host Key del server è utilizzata durante lo scambio di chiavi per verificare che il client stia realmente comunicando con il server corretto. Due differenti modelli di fiducia possono essere utilizzati:

- Il client ha un database locale che associa ciascun nome di host con la corrispondente chiave pubblica. Questo metodo non richiede infrastrutture centralizzate. Lo svantaggio è che la gestione del database di associazione nome-chiavi può diventare gravosa.
- L' associazione nome-chiavi è certificata da una autorità fidata di certificazione. Il client deve conoscere solo la chiave dell'autorità centrale e può verificare la validità di tutte le chiavi in base alla certificazione dell'autorità centrale. Il protocollo prevede anche l'opzione che l'associazione nome-chiave non sia controllata alla prima connessione ad un dato host. Questo consente la comunicazione senza che ci sia stata prima comunicazione di chiavi o certificazioni.

2.2 Transport Layer Protocol

Il Transport Layer Protocol è un protocollo di trasporto sicuro a basso livello. Fornisce crittografia forte, autenticazione crittografica degli host e protezione dell'integrità delle informazioni. Questo protocollo non si occupa dell'autenticazione in quanto questo compito è demandato ai protocolli di livello superiore. Il protocollo è stato progettato per essere semplice, flessibile, per permettere la negoziazione dei parametri e per minimizzare il numero di round-trips [9]. Tutti gli algoritmi (scambio di chiavi, chiave pubblica, cifratura simmetrica ed hash)vengono negoziati. Ci si aspetta che in molti ambienti siano necessari solo 2 round-trips per completare lo scambio delle chiavi, l'autenticazione del server, la richiesta dei servizi e la notifica dell'accettazione della richiesta dei servizi. Il caso peggiore è 3 round-trips. Una volta completata la connessione TCP/IP, si procede, in primis, allo scambio delle versioni di protocollo e software, che le due parti si inviano vicendevolmente. Questo scambio serve a garantire la compatibilità. Durante la fase di negoziazione degli algoritmi, poi, vengono stabiliti gli algoritmi per la cifratura, il controllo d'integrità e la compressione. Infine viene effettuato lo scambio delle chiavi. In questa fase avviene anche la verifica dell'identità del server da parte del client e viene inoltre generato un identificatore di sessione, utilizzato in molte fasi del protocollo e che verrà passato ai protocolli allo strato superiore.

2.2.1 Scambio delle versioni

Entrambe le parti devono mandare una stringa identificativa della forma **SSH-protoversion-softwareversion-comments** seguita da un *carriage return* ed un carattere di *newline*. La lunghezza massima della stringa è di 255 caratteri, compresi il carriage return ed il newline. La stringa di versione deve essere in formato US-ASCII, priva di spazi bianchi o segni '-'. La stringa di versione è utilizzata per abilitare estensioni di compatibilità e per indicare le capacità di una implementazione. La stringa di commento dovrebbe contenere informazioni addizionali utili per risolvere problemi dell'utente.

2.2.2 Negoziazione degli algoritmi

All'inizio della fase di negoziazione degli algoritmi, entrambe le parti inviano la lista degli algoritmi supportati. Ogni parte coinvolta ha un algoritmo preferito per ogni categoria, e si assume che molte implementazioni ad un dato tempo possano avere lo stesso algoritmo preferito. Ogni lato può indovinare quale algoritmo sta usando l'altro, e può mandare un pacchetto iniziale di scambio chiavi

[9] Tempo che trascorre tra l'invio di un pacchetto dati e la ricezione del corrispondente pacchetto di riscontro

appropriato per quei metodi. Se la scelta è giusta il pacchetto mandato viene considerato come il primo pacchetto di scambio di chiavi, altrimenti il pacchetto viene ignorato e la controparte deve rinviare correttamente il pacchetto iniziale. Ciascuna delle stringhe degli algoritmi deve essere una lista di nomi separati da virgola. Ogni algoritmo supportato deve essere citato in ordine di preferenza. Il primo algoritmo deve essere quello preferito. Ogni stringa deve contenere almeno un nome. Il cookie deve essere un valore casuale generato dal mittente. Il suo scopo è di rendere impossibile per entrambe le parti di determinare pienamente le chiavi e l'identificatore di sessione. Se entrambe le parti fanno la stessa scelta per un determinato algoritmo, quest'ultimo deve essere usato. Altrimenti l'algoritmo deve essere scelto iterando sulla lista del client e scegliendo il primo supportato anche dal server.

2.2.3 Scambio di chiavi

Lo scambio di chiavi parte dalla generazione di due valori: un segreto condiviso K, ed un valore hash di scambio H. La modalità di generazione di tali valori dipende dall'algoritmo utilizzato. Al momento l'unico metodo che tutte le implementazioni devono supportare è quello di Diffie-Hellman. Le chiavi per gli algoritmi di cifratura ed autenticazione sono derivate da questi. Il valore di scambio H del primo scambio di chiavi è anche utilizzato come identificatore di sessione, che è unico per la connessione ed una volta calcolato non cambia anche se le chiavi sono riscambiate in seguito. Ogni metodo di scambio di chiavi specifica una funzione hash che è usata nello scambio. Lo stesso algoritmo hash deve essere utilizzato per il calcolo delle chiavi.
Una volta ottenuti H e K, chiavi e vettori di inizializzazione (IV) sono calcolati come funzioni HASH con i seguenti parametri:

- IV iniziale da client a server: $HASH(K||H||''A''||session_id)$
- IV iniziale da server a client: $HASH(K||H||''B''||session_id)$
- Chiave di cifratura da client a server: $HASH(K||H||''C''||session_id)$
- Chiave di cifratura da server a client: $HASH(K||H||''D''||session_id)$
- Chiave di integrità da client a server: $HASH(K||H||''E''||session_id)$
- Chiave di integrità da server a client: $HASH(K||H||''F''||session_id)$

I dati della chiave vanno scelti dall'inizio dell'output dell'hash. Per gli algoritmi a chiave di lunghezza variabile dovrebbero essere usati 128 bit. Lo scambio delle chiavi termina con entrambe le parti che si scambiano il messaggio SSH_MSG_NEWKEYS. Altri messaggi validi dopo lo scambio delle chiavi sono: SSH_MSG_DEBUG, SSH_MSG_DISCONNECT e SSH_MSG_IGNORE

2.2.4 Compressione

La fase di compressione viene inizializzata dopo lo scambio delle chiavi. Se è stata negoziata la compressione, il campo *payloa*d (e solo quello) sarà compresso utilizzando l'algoritmo negoziato.
I campi *lunghezza* e *MAC* saranno computati dal campo payload compresso. La cifratura sarà eseguita dopo la compressione. Le implementazioni devono consentire di scegliere l'algoritmo indipendentemente per ogni direzione.
Sono attualmente definiti i seguenti metodi di compressione:

none	REQUIRED	no compression
zlib	OPTIONAL	GNU ZLIB (LZ77) compression

2.2.5 Integrità dei dati

L'integrità dei dati è protetta includendo in ogni pacchetto un MAC che è computato da un segreto condiviso, un numero di sequenza del pacchetto e il contenuto del pacchetto.
Gli algoritmi e le chiavi di autenticazione sono negoziate durante lo scambio delle chiavi.
Dopo lo scambio delle chiavi, il MAC selezionato sarà computato prima della cifratura dalla concatenazione dei seguenti dati:
mac = MAC(key, sequence_number || unencrypted_packet)
dove unencrypted_packet è l'intero pacchetto senza MAC (campi lunghezza, payload e padding), e sequence_number è un numero di sequenza del pacchetto rappresentato come un intero a 32 bit privo di segno. Il sequence_number è inizializzato a zero per il primo pacchetto, ed è incrementato dopo ogni pacchetto. Esso non è mai incluso nei dati del pacchetto.

Gli algoritmi di MAC devono girare in maniera indipendente l'uno dall'altro, quindi le implementazioni devono permettere di scegliere gli algoritmi indipendentemente per ogni direzione. Il campo **mac** deve essere trasmesso alla fine del pacchetto senza essere cifrato. Il numero di byte di questo campo dipendono dall'algoritmo scelto.

2.2.6 Cifratura

L'algoritmo di cifratura viene negoziato durante lo scambio delle chiavi. I campi *lunghezza*, *padding_length*, *payload* e *padding* devono essere cifrati con l'algoritmo stabilito. Tutti gli algoritmi di cifratura dovrebbero usare chiavi di lunghezza effettiva pari o superiore a 128 bit. Gli algoritmi di cifratura devono girare in maniera indipendente l'uno dall'altro, quindi le implementazioni devono permettere

3des-cbc	REQUIRED	three-key 3DES in CBC mode
blowfish-cbc	RECOMMENDED	Blowfish in CBC mode
twofish256-cbc	OPTIONAL	Twofish in CBC mode, with 256-bit key
twofish-cbc	OPTIONAL	alias for twofish256-cbc
twofish192-cbc	OPTIONAL	twofish with 192-bit key
twofish128-cbc	RECOMMENDED	Twofish with 128-bit key
aes256-cbc	OPTIONAL	AES (Rijndael) in CBC mode, with 256-bit key
aes192-cbc	OPTIONAL	AES with 192-bit key
aes128-cbc	RECOMMENDED	AES with 128-bit key
serpent256-cbc	OPTIONAL	Serpent in CBC mode, with 256-bit key
serpent192-cbc	OPTIONAL	Serpent with 192-bit key
serpent128-cbc	OPTIONAL	Serpent with 128-bit key
arcfour	OPTIONAL	the ARCFOUR stream cipher
idea-cbc	OPTIONAL	IDEA in CBC mode
cast128-cbc	OPTIONAL	CAST-128 in CBC mode
none	OPTIONAL	no encryption; NOT RECOMMENDED

di scegliere gli algoritmi indipendentemente per ogni direzione. Sono attualmente definiti i seguenti algoritmi di cifratura:

2.3 Authentication Protocol

L'*SSH Authentication Protocol* è un protocollo di autenticazione dell'utente *general-purpose*. Gira al di sopra dell'*SSH Transport Layer Protocol*.
L'*Authentication Protocol* assume che il protocollo sottostante fornisca protezione dell'integrità dei dati e confidenzialità. Quando questo protocollo parte, riceve dal protocollo al livello inferiore un identificatore (questo è il valore di scambio H dal primo scambio di chiavi) che indica univocamente la sessione.

2.3.1 Autenticazione: schema generale

Il server guida l'autenticazione comunicando al client quali metodi di autenticazione possono essere utilizzati ad ogni istante. Il client ha la libertà di provare i metodi elencati dal server in qualsiasi ordine. Questo dà al server il completo controllo sul processo di autenticazione ed al client la libertà di scegliere i metodi più convenienti quando il server offre più metodi. I metodi sono identificati con dei nomi:

- Nel metodo *publickey*, il client invia al server un pacchetto firmato con la sua chiave privata, il server autorizza l'autenticazione se la firma è valida.
- Il metodo *"password"* è quello tradizionale: Il client invia la sua password di login ed il server la verifica.
- Il metodo *host based* è simile agli stili di autenticazione rhosts e *hosts.equiv* di UNIX, ma l'identità dell'host del client è controllata più rigorosamente tramite una firma generata con la chiave dell'host del client.
- Il metodo "none" (nessuna autenticazione) è riservato è non deve essere incluso nella lista dei metodi supportati. Il server deve comunque respingere un tale richiesta, a meno che il client non sia effettivamente autorizzato ad entrare nel sistema senza autenticarsi.

Da notare che, in base alla sua politica, il server può richiedere autenticazioni multiple per un dato utente o gruppo di utenti o per client connessi da un determinato host; inoltre dovrebbe avere un timeout per l'autenticazione e disconnettere se l'autenticazione non è stata completata entro quel periodo. Il periodo raccomandato è 10 minuti. Addizionalmente, le implementazioni dovrebbero limitare il numero di tentativi di autenticazione falliti da parte del client.

2.3.2 Metodo di autenticazione a chiave pubblica.

L'unico metodo di autenticazione richiesto è quello a chiave pubblica. Tutte le implementazioni devono supportare tale metodo che consente di autenticarsi grazie al possesso di una chiave privata. Il metodo funziona inviando una firma creata con la chiave privata dell'utente. Il server deve controllare sia che la chiave è un autenticatore valido per l'utente, sia che la firma è valida. Se entrambe queste cose sono verificate, il server accetta la richiesta, altrimenti la rifiuta.

2.3.3 Metodo di autenticazione con password

È compito del server verificare la password all'interno del proprio database. Anche se la password è trasmessa in chiaro all'interno del pacchetto, l'intero pacchetto è criptato al livello di trasporto. Se non vengono utilizzati confidenzialità e controllo dell'integrità, il cambio della password dovrebbe essere disabilitato. Normalmente il server risponde con un messaggio di successo o di fallimento, ma potrebbe anche chiedere il cambio della password.

2.3.4 Autenticazione Host-based

Questa forma di autenticazione è opzionale. È basata sul nome dell'host su cui gira il client e dell'utente che lo sta utilizzando. Non è consigliabile per siti ad alta sicurezza, ma può risultare molto utile in alcuni ambienti.

Quando viene utilizzata, si dovrebbe porre molta attenzione a che l'utente non ottenga la chiave privata dell'host.
Questo metodo funziona tramite un algoritmo di firma: il client crea una firma con la chiave privata dell'host che il server verifica tramite la chiave pubblica dell'host stesso. Una volta stabilita l'identità, l'autorizzazione è effettuata basandosi sugli username dell'utente sul client e sul server, e sul nome host del client.
Il client include la firma, generata usando la sua chiave privata, nel messaggio seguente. Il server deve verificare che la host key appartenga effettivamente al client, che l'utente specificato abbia il permesso di fare il login, e che la firma sia valida.

2.4 Connection Protocol

Il Connection Protocol è stato progettato per girare al di sopra dello User Authentication Protocol. Il suo scopo è quello di permettere sessioni interattive di login, esecuzione remota di comandi, inoltro di connessioni TCP/IP ed inoltro di connessioni X11.
Il *service name* per questo protocollo è **ssh-connection**. La connessione è divisa in canali logici; tutti questi canali sono multiplexati in un singolo tunnel cifrato. Tutti gli pseudo-terminali, connessioni inoltrate, ecc. sono considerati canali, essi possono essere aperti sia dal client che dal server e sono identificati da numeri(da entrambe le parti), che possono essere differenti tra lato client e lato server. Le richieste di apertura di un canale conterranno il numero di canale del mittente ed ogni altro messaggio relativo ad un canale conterrà il numero del canale del ricevente. I canali hanno un meccanismo di controllo del flusso, questo significa che non possono essere inviati dati su di un canale se prima non si è ricevuto messaggio circa la disponibilità della **"window space"**. Si possono avere richieste che coinvolgono globalmente lo stato della parte remota e che sono indipendenti dai canali. Un esempio è una richiesta di iniziare un *TCP/IP forwarding* per una porta specifica.

2.4.1 Meccanismo dei canali: Schema generale

Quando una delle parti desidera aprire un nuovo canale per prima cosa alloca localmente un numero per quel canale. Fatto ciò invia un messaggio alla controparte contenente il numero di canale allocato, il tipo di canale desiderato (pty, x11 forwarding ecc.), la dimensione iniziale per la "window size" e la dimensione massima per i pacchetti in transito su quel canale (ad esempio si potrebbero volere pacchetti più piccoli per sessioni interattive come terminali per migliori prestazioni su link lenti).
La parte remota quindi decide se può aprire il canale. Quando una delle parti vuole chiudere una canale spedisce un messaggio SSH_MSG_CHANNEL_CLOSE.

Un canale è considerato chiuso quando è stato sia inviato che ricevuto un messaggio di chiusura.

2.4.2 TCP/IP Port Forwarding

Questo è un meccanismo che permette di creare un canale di comunicazione sicuro attraverso il quale veicolare qualsiasi tipo connessione TCP. Quello che avviene nella pratica è che viene creato una canale di comunicazione cifrato tra la porta all'indirizzo remoto a cui ci si vuole collegare e una porta locale (libera). Fatto ciò le applicazioni punteranno il collegamento alla porta locale e la connessione verrà inoltrata automaticamente all' host remoto attraverso un canale sicuro.

2.5 Estensibilità di SSH

Per come è strutturato il protocollo SSH potrà essere oggetto di evoluzioni continue nel tempo, ed alcune organizzazioni potrebbero voler utilizzare i propri algoritmi di cifratura, autenticazione e scambio delle chiavi. Una registrazione centrale di tutte le estensioni è gravosa ma senza essa ci sarebbero conflitti negli identificatori dei metodi, rendendo difficile l'interoperabilità. Si è perciò scelto di identificare algoritmi, metodi, formati e protocolli di estensione con nomi testuali di un formato specifico: vengono utilizzati i nomi DNS per creare spazi di nomi locali, di modo che estensioni sperimentali o estensioni classificate possano essere utilizzati senza timore di conflitti con altre implementazioni. Lo scopo è quello di mantenere il protocollo quanto più semplice possibile, con un insieme di algoritmi fondamentali quanto più piccolo possibile. Tutte le implementazioni devono supportare tale insieme per assicurare l'interoperabilità.

2.5.1 Nomi di algoritmi

Il protocollo identifica gli algoritmi e i protocolli con stringhe in formato US-ASCII non più lunghe di 64 caratteri. Tali nomi devono essere case-sensitive.
Ci sono degli algoritmi standard che tutte le implementazioni devono supportare. Anche altri algoritmi sono stati definiti nell specifica del protocollo, ma sono opzionali. Ci sono due formati per i nomi degli algoritmi. I nomi che non contengono il segno '@' sono riservati e non possono essere utilizzati senza il consenso dell'IETF (es. 3des-cbc, sha-1, hmac-sha1 e zlib). Chiunque invece può definire algoritmi addizionali usando nomi nel formato name@domainname. La parte che segue il carattere '@' deve essere un autentico nome di dominio internet.

2.6 Politiche Locali

Il protocollo consente la piena negoziazione di algoritmi e formati di cifratura, integrità, scambio di chiavi e compressione. Le politiche locali dovrebbero guidare la configurazione dei seguenti passi:

- Specificare l'algoritmo preferito di cifratura, integrità e compressione (separatamente per ogni direzione).
- Stabilire i metodi a chiave pubblica e di scambio delle chiavi da usare per l'autenticazione dell'host.
- Stabilire i metodi di autenticazione richiesti per ogni utente. Le politiche locali possono richiedere autenticazioni multiple per alcuni o per tutti gli utenti.
- Le operazioni che l'utente può effettuare tramite il Connection Protocol. Per esempio le politiche non dovrebbero permettere al server di iniziare sessioni o impartire comandi al client. Le politiche dovrebbero inoltre stabilire quali porte possono essere inoltrate e da chi, dal momento che la tecnica di port forwarding può oltrepassare le restrizioni di sicurezza stabilite dai firewall .

2.7 Aspetti di Sicurezza

Il Transport Layer Protocol fornisce un canale criptato sicuro su di una rete insicura. Effettua autenticazione del server, scambio di chiavi, cifratura e protezione dell'integrità. Ottiene anche un unico identificatore di sessione che può essere usato dai protocolli allo strato superiore. Questo protocollo è progettato per essere usato su uno strato di trasporto affidabile. Se avvengono errori di trasmissione o manipolazione dei messaggi, la connessione è chiusa. Se ciò accade, la connessione dovrebbe essere ristabilita.
Attacchi del tipo **denial of service** sono quasi impossibili da evitare. Il protocollo SSH non è stato progettato per eliminare canali coperti. Per esempio il padding, messaggi SSH_MSG_IGNORE, ed altri punti del protocollo possono essere usati per passare informazioni coperte, ed il destinatario non ha alcun mezzo affidabile per verificare che informazioni del genere siano state spedite. Lo scopo dell'Authentication Protocol è di effettuare l'autenticazione dell'utente. Si assume che questo protocollo giri su un protocollo a livello di trasporto sicuro, che ha già autenticato il server, stabilito un canale di comunicazione criptato e computato un identificatore di sessione. È il protocollo di trasporto a fornire segretezza per l'autenticazione con password. Il server può andare in sleep dopo numerosi tentativi di autenticazione falliti per evitare attacchi di ricerca della chiave. Se il livello di trasporto non fornisce cifratura, i metodi di autenticazione che si basano su

dati segreti dovrebbero essere disabilitati. Se non fornisce integrità, le richieste di cambiare i dati di autenticazione (password) dovrebbero essere disabilitati per evitare che vengano cambiati da un hacker in un attacco di tipo denial of service, rendendo impossibile un accesso futuro. Il server può decidere, in base alle politiche locali di sicurezza, quali e quanti metodi di autenticazione utilizzare per ciascun utente. Particolare attenzione va posta nel progettare i messaggi di debug. Essi potrebbero rivelare più informazioni sull'host di quante questi non ne voglia rivelare. I messaggi di debug possono comunque essere disabilitati.
Si assume che il *Connection Protocol* giri al di sopra di un protocollo di trasporto sicuro che ha già autenticato la macchina server, stabilito un canale di comunicazione cifrato, computato un identificatore di sessione, autenticato l'utente.
Come si è visto questo protocollo permette l'esecuzione di comandi su macchine remote e permette così anche al server di eseguire comandi sul client. Normalmente le implementazioni devono evitare questa possibilità dato che un server corrotto può essere usato per attaccare tutti i client che si connettono. Migliorata la sicurezza per connessioni al server X11, alla chiusura del canale non restano informazioni riguardo all'autenticazione sul server. 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 ogni caso si rende necessaria una buona politica di controllo poiché, in quanto cifrato, il traffico non può essere esaminato dal firewall. Quindi le implementazioni di SSH dovrebbero disabilitare tutte le politiche potenzialmente dannose (ad esempio X11 forwarding e TCP/IP forwarding) nel momento dello scambio delle chiavi.

3 OpenSSH

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 è principalmente sviluppato dal Progetto OpenBSD, e la sua prima inclusione all'interno del sistema operativo avvenne in OpenBSD 2.6. Il software viene sviluppato al di fuori degli USA, usando codice proveniente da circa 10 paesi diversi, ed esso è utilizzabile e riutilizzabile da chiunque sotto la licenza BSD. OpenSSH cripta tutto il traffico (password compresa) per eliminare a tutti gli effetti l'ascolto passivo, l'hijacking della connessione e altri attacchi a livello di rete. Fornisce inoltre una miriade di possibilità di tunneling sicuro, così come vari metodi di autenticazione.
Ecco alcune caratteristiche:

- Progetto Open Source;
- Licenza Gratuita;
- Crittografia forte (3DES, Blowfish);
- X11 Forwarding (cifratura automatica del traffico X11);
- Port Forwarding (canali criptati per altri protocolli);
- Autenticazione Forte (Rhosts, Chiave Pubblica, Password);
- 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;
- Kerberos e AFS Ticket Passing;
- Compressione Dati.

Progetto Open Source Il codice di OpenSSH è disponibile gratuitamente per chiunque via Internet. Questo incoraggia il riutilizzo e la verifica del codice.

Licenza Gratuita OpenSSH non è coperto da nessuna licenza restrittiva. Può essere usato per qualsiasi scopo, incluso quello commerciale. Tutte le componenti soggette a brevetto, sono state rimosse dal codice sorgente. Tutte le componenti soggette a restrizioni sulla licenza o brevetto, sono state scelte da librerie esterne (ad esempio OpenSSL).

Crittografia forte (3DES, Blowfish) OpenSSH usa 3DES e Blowfish come algoritmi di cifratura. Nessuno di essi è soggetto a brevetto. 3DES è un cifrario ben conosciuto e a prova di tempo che fornisce crittografia forte. Blowfish è un cifrario a blocchi veloce, inventato da Bruce Schneier, che risponde bene all'esigenza di cifratura veloce. La cifratura inizia prima dell'autenticazione e non sono trasmesse password o altre informazioni in chiaro.

X11 Forwarding L'X11 [10] forwarding consente la cifratura del traffico remoto di X11, cosicché nessuno possa curiosare in xterm remoti di altri o inserire comandi maliziosi. Il programma setta automaticamente la variabile di ambiente DISPLAY sulla macchina server e inoltra ogni connessione X11 su di un canale sicuro.

Port Forwarding Il port forwarding consente l'inoltro di connessioni TCP/IP ad una macchina remota verso un canale cifrato. Servizi standard di Internet, come i server POP, possono essere resi sicuri con esso.

Autenticazione Forte L'autenticazione forte protegge contro alcuni problemi di sicurezza, come IP spoofing, rotte false e DNS spoofing. I metodi di autenticazione sono: rhosts con autenticazione host basata su RSA, autenticazione RSA pura e autenticazione con password.

Interoperabilità Le versioni di OpenSSH anteriori alla 2.0 supportano i protocolli SSH 1.3 e SSH 1.5, permettendo la comunicazione con molte versioni di UNIX, Windows ed altre implementazioni commerciali di SSH.
OpenSSH supporta anche il protocollo SSH 2.0, che consente di scegliere tra RSA e DSA come algoritmi di firma.
OpenSSH include un supporto completo per SFTP, usando il comando sftp(1) come client. Il deamon sftp-server(8) funziona automaticamente sia in SSH1 che in SSH2.
La suite di OpenSSH include il programma ssh che sostituisce l'rlogin e il telnet, scp che sostituisce l'rcp, e l'sftp che sostituisce l'ftp. Inoltre è incluso sshd che è il lato server del pacchetto e altre utility di base come ssh-add, ssh-agent, ssh-keysign, ssh-keyscan, ssh-keygen and sftp-server.

[10] X Window System, noto anche come X11 o ancor più semplicemente X, è il sistema grafico standard per tutti i sistemi Unix

4 Una sessione sicura

Il protocollo SSH permette di creare dei tunnel che permettono ad altri programmi di raggiungere una destinazione specifica attraverso appunto questi tunnel. Il transito dei pacchetti in tal modo è criptato anche se i programmi ed i protocolli usati non hanno la capacità di farlo autonomamente. Il tuneling SSH viene messo in pratica da un meccanismo chiamato "*port forwarding*". Configuriamo il nostro client SSH in modo che accetti connessione su certe porte sulla macchina locale. Tutti i dati mandati a questa porte sono inviati e tornano attraverso il tunnel. Dall'altra parte dei tunnel, il server SSH passa i dati e li riceve alla macchina che avete indicato. Questo meccanismo è conosciuto come "port forwarding".
Con SSH potete rendere sicuri protocolli TCP/IP altrimenti insicuri mediante il port forwarding. Con questa tecnica, il server SSH diventa un passaggio cifrato al client SSH.
Il *port forwarding* consiste nel mappare una porta locale sul client verso una porta remota sul server. SSH vi permette di mappare qualsiasi porta dal server verso qualsiasi porta sul client. I numeri delle porte non devono essere uguali per funzionare. Il port forwarding può risultare particolarmente utile per ricevere informazioni in modo sicuro tramite i firewall di rete. Se firewall è configurato per consentire il traffico SSH tramite la porta standard, ma blocca l'accesso tramite le altre porte, una connessione tra due host che usano porte bloccate è comunque possibile se si reindirizza la comunicazione tramite una connessione SSH.

Conclusioni

In definitiva lo scopo primario del protocollo SSH è migliorare la sicurezza su Internet. Tutti gli algoritmi di cifratura, integrità e a chiave pubblica sono ben conosciuti e ben stabiliti.Inoltre vengono tutti negoziati. Nel caso in cui uno di essi venga rotto è facile passare ad un altro senza modificare il protocollo di base. Vengono infine utilizzate chiavi sufficientemente lunghe da stabilire una forte protezione contro gli attacchi crittoanalitici per decadi.

Riferimenti bibliografici

[1] SSH -
http://www.dia.unisa.it/ãds/corso-security/www
/CORSO-0001/SSH/protocollo.htm
http://it.wikipedia.org/wiki/Crittografia_asimmetrica

[2] RSA -
http://www.amagri.it/Crittologia/Crittografia/
Algoritmi_crittografici/RSA/algoritmo_rsa.htm
http://sicurezza.html.it/guide/lezione.asp?IdGuida=6&idlezione=93

[3] DSA - *http://www.firmasicura.it/faq_a.asp*
http://www.icosaedro.it/crittografia/chiavi-asimmetriche.html

[4] Algoritmi a chiave simmetrica - *http://a2.pluto.it/a2320.htm*
http://telemat.die.unifi.it/book/1999/crittografia/capitolo11.html

[5] Funzioni Hash -
http://www.techtown.it/public/html/InfoLinux/MyLinuxsu209.html

[6] OpenSSH - *http://www.dia.unisa.it/ãds/corso-security/www/CORSO-0001/SSH/introduzione.htm*

[7] Tunneling SSH - *http://www.freshnet.org/docs/pdf/ssh_tunneling.pdf*

[8] Protocollo SSH -*http://www.dia.unisa.it/ ads/corso-security/www/CORSO-0001/SSH/protocollo.htm*

[9] Secure Shell - *http://it.wikipedia.org/wiki/Secure_shell*

[10] Algoritmo Diffie-Hellman-
http://www.liceofoscarini.it/studenti/crittografia/critto/rsa/dh.html

[11] Storia della Crittografia -
http://it.wikipedia.org/wiki/Crittografia#Approcci_storici_alla_crittografia

[12] Sicurezza Informatica - *http://it.wikipedia.org/wiki/Sicurezza_Informatica*