Università degli Studi di Verona
Dipartimento di Informatica
A.A. 2019-2020

RIASSUNTO DEL CORSO DI "PROGRAMMAZIONE E SICUREZZA DELLE RETI"

Creato da: *Davide Zampieri*

Indice degli argomenti

SICUREZZA DELLE RETI

❖ Introduzione

Quali risorse (o asset) vogliamo proteggere?

Proteggere vuol dire garantire le proprietà di confidenzialità, integrità, disponibilità e, in aggiunta, anche le proprietà di autenticità e tracciabilità. Tra le *risorse* che vogliamo proteggere troviamo: hardware (ad es. componenti e dischi), software (ad es. sistemi operativi e applicativi), dati (ad es. file e database), rete (ad es. collegamenti e apparati). Vediamo ora in dettaglio cosa implica garantire le proprietà appena citate:

- *Confidenzialità* – nessun utente deve poter ottenere o dedurre dal sistema informazioni che non è autorizzato a conoscere, ovvero le informazioni confidenziali (*riservatezza dei dati*) non devono essere rivelate o essere rilevabili da utenti non autorizzati (*privacy*).
- *Integrità* – bisogna impedire l'alterazione diretta o indiretta delle informazioni, sia da parte di utenti e processi non autorizzati sia a seguito di eventi accidentali e quindi, se i dati vengono alterati, è necessario fornire strumenti per poterlo verificare facilmente; inoltre, si può fare distinzione tra *integrità dei dati* (le informazioni e i programmi possono essere modificati solo se si è autorizzati) e *integrità del sistema* (il sistema funziona e non è compromesso).
- *Disponibilità* – bisogna rendere disponibili a ciascun utente abilitato le informazioni alle quali ha diritto di accedere nei tempi e nei modi previsti (in determinate condizioni, in un preciso istante, in un intervallo di tempo); inoltre, nei sistemi informatici, i requisiti di disponibilità includono *prestazioni e robustezza*.
- *Autenticità* – ciascun utente deve poter verificare l'autenticità delle informazioni (messaggi, mittenti, destinatari) e se un'informazione è stata manipolata (anche per informazioni non riservate).
- *Tracciabilità* – le azioni di un'entità devono essere tracciate in modo univoco in maniera tale da supportare la non-ripudiabilità e l'isolamento delle responsabilità (ad es. nessun utente deve poter ripudiare o negare in tempi successivi messaggi da lui spediti o firmati).

In che modo le risorse sono minacciate?

Le minacce compromettono le proprietà di confidenzialità, integrità e disponibilità. Quindi una *minaccia* è una possibile violazione della sicurezza, mentre la violazione effettiva è chiamata *attacco*. Gli attacchi possono essere di diverse *tipologie*:

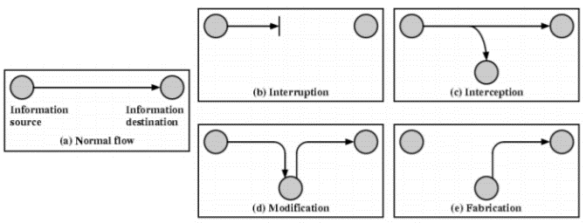
- *Attivi*, cioè tentativi di alterare le risorse o modificare il funzionamento dei sistemi.
- *Passivi*, cioè tentativi di carpire informazioni e utilizzarle senza intaccare le risorse.
- *Interni*, cioè iniziati da un'entità interna al sistema.
- *Esterni*, cioè iniziati da un'entità esterna (tipicamente attraverso la rete).

Inoltre, esistono le seguenti *classi* di attacchi:

- *Disclosure*, cioè l'accesso non autorizzato alle informazioni.
- *Deception*, cioè l'accettazione di dati falsi.
- *Disruption*, cioè l'interruzione (o prevenzione) di operazioni corrette.
- *Usurpation*, cioè il controllo non autorizzato di alcune parti del sistema.

	Confidenzialità	Integrità	Disponibilità
HW			Calcolatore rubato
SW	Copia non autorizzata	Eseguibile modificato	Eseguibili cancellati
Dati	Lettura non autorizzata	File modificati	File cancellati
Rete	Lettura messaggi inviati	Messaggi modificati / ritardati / duplicati	Messaggi distrutti Rete fuori uso

Effetti delle minacce sulle proprietà



Esempi di attacco

Cosa bisogna fare per contrastare le minacce?
La *complessità della sicurezza* è dovuta alla necessità di dover rispondere a questa domanda, in quanto non esiste una risposta unica e le risposte stesse cambiano nel tempo. Inoltre, le risorse da proteggere sono *sistemi complessi* composti da sottosistemi. Bisognerà quindi garantire non solo la sicurezza di un sistema ma anche la sicurezza dei suoi componenti. Infine, la *teoria* (condizioni ideali e prevedibili) si scontra inevitabilmente con la *pratica* (condizioni reali e imprevedibili).

Sfide poste dalla sicurezza.

- *Attacchi potenziali* (da considerare nella progettazione dei sistemi).
- *Soluzioni contro-intuitive* nello sviluppo dei meccanismi di sicurezza (dovute alla complessità del sistema e alle possibili minacce).
- *Dove usare i meccanismi* di sicurezza (sia a livello fisico che logico).
- *Dipendenza* da algoritmi e protocolli ma anche *dagli utenti* (creazione, distribuzione e protezione delle password).
- Continua *battaglia tra amministratori e attaccanti* (per l'attaccante è sufficiente sfruttare una singola vulnerabilità, mentre gli amministratori devono prevederle ed eliminarle tutte).
- *Non-percezione del beneficio* della sicurezza (fino a quando non avviene un incidente).
- Richiesta di un *continuo controllo delle risorse*.
- Meccanismi di sicurezza visti come *elementi aggiuntivi* (invece che parte integrante della progettazione).
- Visione della sicurezza come un *impedimento* (rallentamento) al normale funzionamento dei sistemi.

Principi fondamentali di progettazione della sicurezza.
Nonostante anni di ricerca, è difficile progettare sistemi che prevengano completamente le falle nella sicurezza. Tuttavia, è stato codificato un insieme di *pratiche e regole* (analogamente a quanto succede per l'ingegneria del software). Tra di esse troviamo:

- *Aspetti economici* dei meccanismi – la progettazione delle misure di sicurezza deve essere il più semplice possibile (da implementare e verificare).
- *Fail-safe default* – comportamenti non specificati devono prevedere un default sicuro (ad es. permessi di accesso).
- *Progettazione aperta* – preferibile rispetto a codice segreto.
- *Tracciabilità delle operazioni* – qualsiasi operazione può essere ricostruita e il sistema ripristinato.
- *Separazione dei privilegi* – differenziazione degli accessi alle risorse create da ciascun utente (file) e alle risorse critiche.
- *Separazione delle funzionalità* – distinzione dei ruoli nei diversi punti del sistema fisico e logico.
- *Isolamento dei sottosistemi* – un sistema compromesso non dovrebbe compromettere gli altri.
- *Modularità* – i meccanismi di sicurezza devono essere indipendenti, sostituibili e riusabili.

Politiche di sicurezza.
Una *politica di sicurezza* è un'indicazione di cosa è permesso e di cosa non è permesso. Le regole possono riguardare: protezione dei *dati*; controllo sulle *operazioni possibili*; controllo degli *utenti singoli* e dei *profili*.

Politiche e meccanismi.
Un *meccanismo di sicurezza* è un metodo (strumento/procedura) per garantire una politica di sicurezza. Data una politica, che distingue le azioni "sicure" da quelle "non sicure", i meccanismi di sicurezza devono *prevenire*, *scoprire* o *recuperare* un attacco. Nella *prevenzione*, il meccanismo deve rendere impossibile l'attacco. Spesso però i meccanismi di prevenzione sono pesanti ed interferiscono con il sistema al punto da renderlo scomodo da usare (ad es. richiesta di password come modo di autenticazione). Nella *scoperta*, il

meccanismo è in grado di scoprire se un attacco è in corso. Infatti, si usa solitamente un monitoraggio delle risorse del sistema alla ricerca di eventuali tracce di attacchi. Il meccanismo di scoperta è utile quando non è possibile prevenire l'attacco, ma può servire anche a valutare le misure preventive. Il *recupero* da un attacco si può fare in due modi: fermare l'attacco e recuperare/ricostruire la situazione pre-attacco (attraverso copie di backup); oppure continuare a far funzionare il sistema correttamente durante l'attacco (fault-tolerant).

Meccanismi e livelli.
In quale livello del computer conviene inserire un determinato meccanismo? Nei *livelli bassi* possiamo inserire meccanismi generali, semplici e grossolani, ma di cui è possibile dimostrare la correttezza. Nei *livelli alti*, invece, possiamo inserire meccanismi sofisticati e ad hoc per gli utenti, ma di cui è difficile dimostrare la correttezza. Tra i *meccanismi specifici* (legati ad uno specifico livello OSI) troviamo:

- *Crittografia*, ovvero la trasformazione dei dati in un formato non intellegibile.
- *Firma digitale*, usata per provare la sorgente dei dati.
- *Integrità*, usata per provare che i dati non sono stati manipolati.
- *Autenticazione e autorizzazione*, ovvero la gestione dei diritti degli utenti rispetto alle risorse.

Infine, tra i *meccanismi generali* troviamo: rilevamento degli eventi; gestione degli audit; recovery.

Come ottenere un sistema sicuro.
Le *fasi* per ottenere un sistema sicuro sono:

- *Specifica*, ovvero la descrizione del funzionamento del sistema desiderato.
- *Progetto*, ovvero la traduzione delle specifiche in componenti che le implementeranno.
- *Implementazione*, ovvero la creazione del sistema che soddisfa le specifiche.

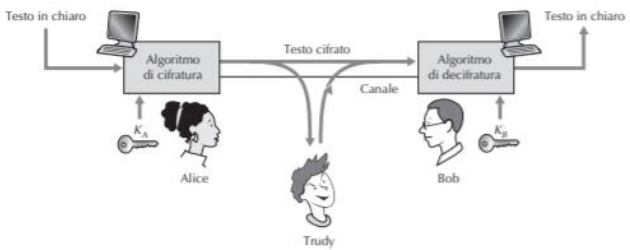
È indispensabile verificare continuamente la correttezza dell'implementazione, ma bisogna anche tenere conto di alcune *considerazioni implementative*, tra cui:

- *Analisi costi-benefici* della sicurezza.
- *Analisi dei rischi*, ovvero valutare la probabilità di subire attacchi e i danni che essi possono causare.
- *Aspetti legali e morali*.
- *Problemi organizzativi*, in quanto la sicurezza non produce nuova ricchezza ma riduce solo le perdite.
- *Aspetti comportamentali* delle persone coinvolte.

❖ Cenni di crittografia

Crittografia e crittoanalisi.
La *crittografia* è la scienza che si occupa di proteggere l'informazione rendendola sicura, in modo che un utente non autorizzato che ne entri in possesso non sia in grado di comprenderla. La *crittoanalisi* è invece la scienza che cerca di aggirare o superare le protezioni crittografiche, per accedere alle informazioni protette. L'insieme di crittografia e crittoanalisi è detto *crittologia*.

Elementi del processo crittografico.
Un *algoritmo crittografico* è una funzione che, prendendo in ingresso un messaggio in chiaro (plaintext o cleartext) e un parametro detto chiave, produce in uscita un messaggio cifrato (ciphertext).



L'algoritmo crittografico può essere:

- *Simmetrico*, se le chiavi segrete di cifratura e decifratura sono uguali.
- *Asimmetrico*, se le chiavi sono diverse (una è pubblica e l'altra è segreta e privata).

Robustezza crittografica.

Non deve essere possibile (facilmente) che:

- Dato un testo cifrato, si possa ottenere il corrispondente testo in chiaro senza conoscere la chiave di decifratura.
- Dato un testo cifrato e il corrispondente testo in chiaro, si possa ottenere la chiave di decifratura.

In generale, *nessun algoritmo crittografico è assolutamente sicuro*. Si dice quindi che un algoritmo crittografico è computazionalmente sicuro se:

- Il costo necessario a violarlo è superiore al valore dell'informazione cifrata.
- Il tempo necessario a violarlo è superiore al tempo di vita utile dell'informazione cifrata.

Crittoanalisi.

La *crittoanalisi* tenta di ricostruire il testo in chiaro senza conoscere la chiave di decifratura. L'attacco più banale è quello "*a forza bruta*", il quale:

- Tenta di decifrare il messaggio provando tutte le chiavi possibili.
- È applicabile a qualunque algoritmo, ma la sua praticabilità dipende dal numero di chiavi possibili.
- Necessita comunque di avere informazioni sul formato del testo in chiaro, per riconoscerlo quando si trova la chiave giusta.

Infine, secondo il *principio di Kerckhoffs*, nel valutare la sicurezza di un algoritmo crittografico si assume che il crittoanalista conosca tutti i dettagli dell'algoritmo, perciò la segretezza deve risiedere nella chiave e non nell'algoritmo.

Crittografia a chiave simmetrica.

La *crittografia simmetrica*, detta anche crittografia a chiave segreta, utilizza una chiave comune e il medesimo algoritmo crittografico per la codifica e la decodifica dei messaggi. Quindi, due utenti che desiderano comunicare devono accordarsi sull'uso di un algoritmo e di una chiave comuni. Inoltre, la chiave deve essere scambiata su un *canale sicuro*.

Esempi classici di cifrari a chiave simmetrica.

Un primo esempio è il *cifrario di Cesare*, che sostituisce ogni lettera del testo in chiaro con quella che si trova K posizioni più avanti nell'alfabeto. K è quindi la chiave. Usando l'alfabeto italiano, le chiavi possibili sono solamente 20. Un altro esempio è la *cifratura monoalfabetica*, in cui ogni carattere viene sostituito da un altro (permutazione) secondo un certo alfabeto che costituisce la chiave. Le chiavi possibili sono quindi pari al numero di permutazioni possibili (21!).

Esempio: K = 3

- In chiaro: A B C D E F G H I L M N O P Q R S T U V Z
- Cifrate: D E F G H I L M N O P Q R S T U V Z A B C
- Esempio di messaggio in chiaro / cifrato: CIAO / FNDR

Cifrario di Cesare

Esempio:

- In chiaro: A B C D E F G H I L M N O P Q R S T U V Z
- Cifrate: M Z N C B V L A H S G D F Q P E O R I T U
- Esempio di messaggio in chiaro / cifrato: CIAO / NHMF

Cifratura monoalfabetica

Analisi delle frequenze.

Come abbiamo visto, lo spazio delle chiavi di un algoritmo monoalfabetico è molto grande. Tuttavia, la crittoanalisi può essere molto semplice se si usa l'*analisi delle frequenze*. Infatti, in un testo scritto in una determinata lingua ogni lettera dell'alfabeto si presenta secondo una certa frequenza (ad es. in italiano E ed A sono molto comuni mentre Q e Z sono poco comuni). Inoltre, è possibile anche considerare gruppi di 2 o 3

lettere (ad es. in italiano CH, CHE, QU). Quindi in definitiva, contando il numero di occorrenze di ogni lettera nel testo cifrato è possibile ipotizzare con buona probabilità quale sia la lettera corrispondente.

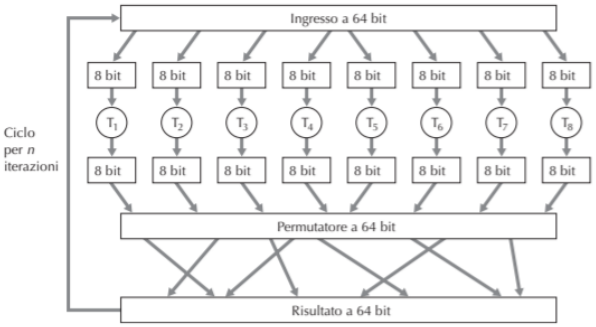
Cifrari a blocchi.

I *cifrari a blocchi* (chiamati anche cifrari a flusso) sono tecniche di cifratura simmetrica e sono usati in molti protocolli sicuri di Internet, tra cui: PGP (posta elettronica), SSL (connessione TCP) e IPSec (trasmissione a livello di rete). Il *funzionamento* di un cifrario a blocchi prevede che, dati k bit, i possibili 2^k ingressi vengano permutati. Inoltre, le permutazioni possono essere combinate tra loro per creare schemi più complessi.

Esempio: k = 3

Ingresso	Uscita	Ingresso	Uscita
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

Esempio di funzionamento



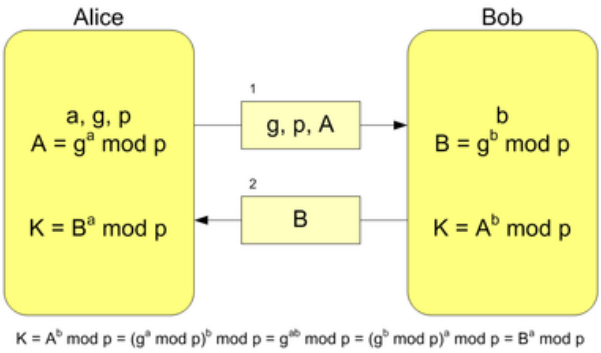
Schema complesso

Esempi di cifrari a blocchi.

Il *DES* (Data Encryption Standard) è il più noto algoritmo crittografico simmetrico moderno. Nato negli anni '70 a seguito di un progetto di IBM, è stato adottato ufficialmente nel 1977 come standard dal governo americano. Il DES utilizza chiavi di *56 bit*, ma è da considerarsi ormai obsoleto. Infatti, è meglio utilizzare il *Triplo-DES*, ovvero applicare tre volte il DES con chiavi diverse per aumentarne la sicurezza (ne esistono varianti con chiave a *112 o 168 bit*). Tuttavia, nel 1997 è cominciata la ricerca per trovare il successore del DES come algoritmo standard. Nel 2000 è stato scelto come vincitore l'algoritmo di Rijndael, chiamato anche *AES* (Advanced Encryption Standard). AES è un algoritmo simmetrico che sta gradualmente soppiantando il Triplo-DES, in quanto può utilizzare chiavi di *128, 192 o 256 bit*.

Distribuzione delle chiavi.

Negli algoritmi simmetrici la chiave è la stessa sia in cifratura che in decifratura e dunque deve essere segreta. Esiste quindi il problema della *distribuzione delle chiavi*, in quanto c'è bisogno di un canale di comunicazione sicuro per poterle trasmettere. Nel 1976 *Diffie e Hellman* propongono uno schema che non condivide chiavi per superare questa limitazione.



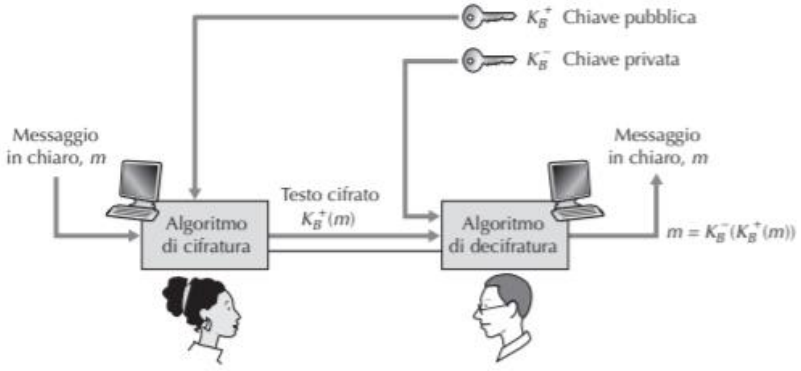
$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

Crittografia asimmetrica.

Nella crittografia asimmetrica ogni utente ha una coppia di chiavi, costituita da una *chiave pubblica* e da una *chiave privata*. La chiave pubblica viene resa nota mentre quella privata deve rimanere segreta. Il dato verrà quindi cifrato con la chiave pubblica del destinatario, il quale potrà poi decifrarlo con la propria chiave privata.

I *vantaggi* della crittografia asimmetrica sono che non è più necessario incontrarsi per scambiare le chiavi e che la stessa chiave (quella pubblica) può essere usata da più utenti. Tuttavia, ci sono anche dei *requisiti* da garantire:

- La *generazione* di una coppia di chiavi pubblica/privata deve essere *semplice*.
- L'operazione di *cifratura/decifratura* (conoscendo la relativa chiave) deve essere *semplice*.
- *Ricavare la chiave privata* da quella pubblica deve essere *computazionalmente impraticabile*.
- *Ricavare il testo in chiaro* avendo il testo cifrato e la chiave pubblica deve essere *computazionalmente impraticabile*.



Algoritmo RSA.

RSA (1977), così chiamato dalle iniziali dei suoi inventori (Rivest, Shamir, Adleman), è sicuramente il più noto algoritmo crittografico asimmetrico. RSA si basa sulla difficoltà di *scomporre un numero in fattori primi*. Solitamente, la chiave di RSA ha dimensioni di almeno 2^{10} bit (oltre 300 cifre decimali). Quindi, un attacco a forza bruta contro RSA non consiste nel provare tutte le chiavi possibili, ma nel fattorizzare il prodotto di due numeri primi.

Esempio di cifratura/decifratura con RSA.

Per capire come avviene la cifratura e la decifratura con RSA ci si deve avvalere della *matematica a modulo*. Per la *generazione delle chiavi* si procede in questo modo:

1. Si scelgono due numeri primi *p* e *q*
2. Si calcola *n = p*q*
3. Si calcola *z = (p-1)*(q-1)*
4. Si sceglie un numero *e* relativamente primo a *z* tale che *1 < e < n*
5. Si sceglie un numero *d* tale che *(e*d-1)* sia un multiplo di *z*
6. La chiave pubblica sarà *(n,e)* mentre la chiave privata sarà *(n,d)*

A questo punto:

- Per cifrare un messaggio *m* si calcola $c = m^e \bmod n$
- Per decifrare il messaggio *c* si calcola $m = c^d \bmod n$

- ❑ p = 5, q = 7
- ❑ Segue:
  - n = p*q = 35
  - z = (p-1)*(q-1) = 24
  - e = 5 (relativamente primo a 24; andavano bene anche 7, 9, 11, …)
  - d = 29 (infatti 5*29 - 1 = 144 → multiplo di 24)
- ❑ Messaggio da inviare: la parola "love"
  - Supponiamo di rappresentare le lettere con numeri da 1 a 26 (incluse le lettere x,y,w, …)

Generazione delle chiavi

Lettere in chiaro	m: rappresentazione numerica	m^e	Testo cifrato $c = m^e \bmod n$
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

Testo cifrato c	c^d	$m = c^d \bmod n$	Lettere in chiaro
17	481968572106750915091411825223071697	12	l
15	12783403948858939111232575685359375	15	o
22	851643319086537701956194499721106030592	22	v
10	100000000000000000000000000000	5	e

Cifratura e decifratura

Considerazioni sugli algoritmi asimmetrici.
Gli algoritmi asimmetrici richiedono molte risorse computazionali (sono 100-1000 volte più lenti degli algoritmi simmetrici). Per questo motivo vengono solitamente utilizzati per scambiarsi una *chiave di sessione* che verrà poi usata con un algoritmo simmetrico sicuro e computazionalmente più efficiente. Inoltre, con RSA ciò che viene cifrato con la chiave pubblica si può decifrare con la chiave privata, ma vale anche l'inverso ovvero ciò che viene cifrato con la chiave privata si può decifrare con la chiave pubblica. Quest'ultimo fatto ci fornisce un mezzo per garantire l'*autenticazione*.

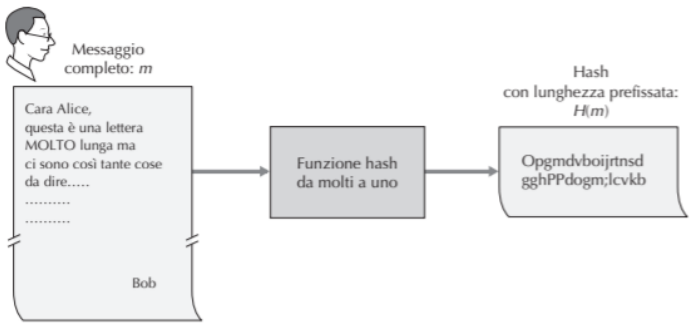
❖ Integrità, autenticazione, autorizzazione

Considerazioni sulla crittografia.
L'obiettivo storico della crittografia è quello di garantire la *privacy*, ossia garantire che i dati non possano essere letti da utenti non autorizzati. Quindi, dato un messaggio che sembra provenire da un utente, per essere sicuro che esso arrivi effettivamente da tale utente devo verificarne l'*autenticità* e l'*integrità*. Questo problema è intrinsecamente diverso da quello di proteggere i contenuti, perciò per affrontarlo ci servono altri strumenti, come le funzioni hash.

Funzioni hash.
Una *funzione hash* trasforma un messaggio di lunghezza arbitraria in un output di lunghezza fissa chiamato hash o digest del messaggio originale. Per soddisfare le condizioni di sicurezza stabilite per le funzioni hash, gli *algoritmi* devono essere:

- *Coerenti*, ossia input uguali devono corrispondere ad output uguali.
- *Casuali*, per impedire l'interpretazione accidentale del messaggio originale.
- *Univoci*, ossia la probabilità che due messaggi generino lo stesso hash deve essere virtualmente nulla.
- *Non invertibili*, ossia risalire al messaggio originale dall'output deve essere impossibile.

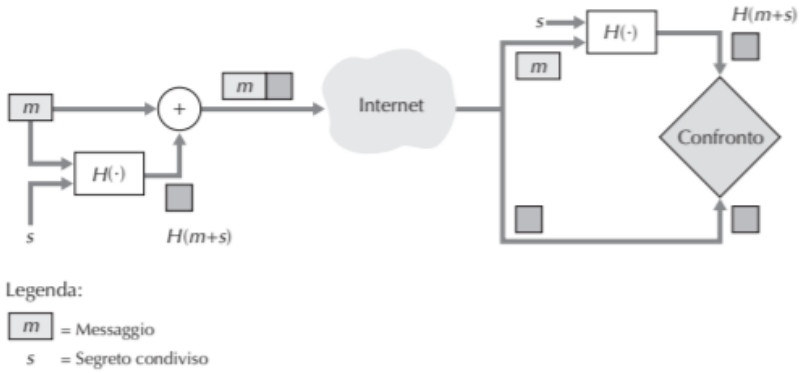
Le funzioni hash non invertibili vengono normalmente utilizzate per assegnare un'*impronta digitale* a un messaggio o a un file. Infatti, come le impronte dei polpastrelli, un'impronta hash è univoca e costituisce una prova dell'integrità e dell'autenticità del messaggio. Quindi, se A e B vogliono accertarsi che nessuno sia intervenuto sul contenuto del messaggio in fase di transizione devono utilizzare una funzione hash non invertibile.



Esempio di comunicazione che sfrutta le funzioni hash.
L'approccio è simile al checksum usato negli header dei protocolli (ma con l'aggiunta della crittografia):

1. A scrive un messaggio e ne utilizza il testo come input di una funzione hash non invertibile.
2. Il risultato della funzione hash viene accodato al messaggio e ne costituisce l'impronta digitale.
3. Il tutto viene cifrato e inviato a B.
4. B decifra ciò che ha ricevuto, separa il messaggio dall'impronta e utilizza il testo del messaggio come input della medesima funzione hash utilizzata da A.
5. Se i due hash corrispondono, B è certo che nessun altro sia intervenuto sul messaggio.

Altri esempi.
In alcune situazioni, A e B non sono interessati ad occultare (cifrare) il contenuto del messaggio, ma vogliono comunque preservare l'integrità. Serve quindi disporre di un codice *segreto condiviso*, ossia una chiave di autenticazione o *MAC* (Message Authentication Code).



Considerazioni sulle funzioni hash.
Le funzioni hash più comuni sono: l'algoritmo *MD5* (Message Digest 5) e l'algoritmo *SHA* (Secure Hash Algorithm). In aggiunta agli usi mostrati negli esempi precedenti, le funzioni hash possono essere utilizzate anche per gestire le *password* (ad es. UNIX). Infatti, piuttosto che memorizzare la password si memorizza la sua hash e, quando un utente fa il login, il sistema fa l'hash della password digitata e lo confronta con il valore in memoria.

Autenticità dei messaggi.
Finora abbiamo considerato solo l'integrità dei messaggi, ossia dato un messaggio riesco ad essere sicuro che non sia stato manipolato. Con il *MAC*, invece, c'è anche garanzia di *autenticità*, in quanto la chiave di autenticazione è segreta ed associata ad un utente. Tuttavia, tale chiave deve essere preventivamente distribuita tra chi comunica attraverso un canale sicuro. Per riuscire a garantire l'autenticità senza distribuire chiavi segrete, si usa la *firma digitale* (attuabile con la *crittografia asimmetrica*).

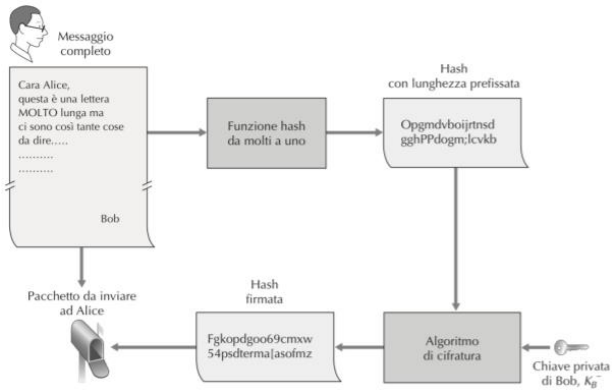
Firma digitale.
Una *firma digitale* è l'equivalente informatico di una firma convenzionale. Inoltre, una firma digitale è (in generale) non ripudiabile. Per la firma digitale si sfrutta *RSA* in modo inverso a quanto visto per la cifratura, ovvero l'algoritmo di cifratura diventa l'algoritmo di verifica mentre l'algoritmo di decifratura diventa l'algoritmo di firma. Tuttavia, firmare l'*intero documento* (ossia cifrarlo con la chiave privata) è molto oneroso. Infatti, si ottiene un approccio più efficiente se RSA viene combinato con le funzioni hash, permettendo quindi di firmare solamente l'*hash del documento*.

Firma di un documento e controllo dell'integrità.
La *firma di un documento* (con l'approccio RSA+hash) si attua nel seguente modo:

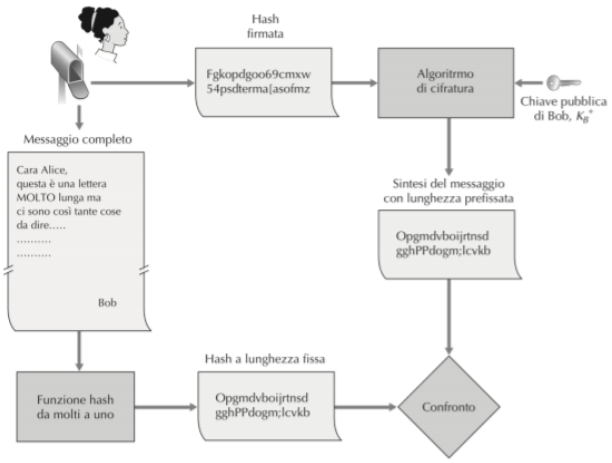
1. Viene calcolato l'hash del messaggio.
2. Viene firmato l'hash del messaggio con la chiave privata del mittente.
3. Si manda al destinatario l'hash del messaggio firmato e il messaggio stesso.

Per la verifica dell'*autenticità della firma digitale*, il destinatario procede così:

1. Si decifra l'hash firmato ricevuto usando la chiave pubblica del mittente.
2. Si calcola l'hash del messaggio ricevuto.
3. Si comparano l'hash calcolato con quello ottenuto decifrando l'hash firmato e, se combaciano, vuol dire che la firma è autentica.



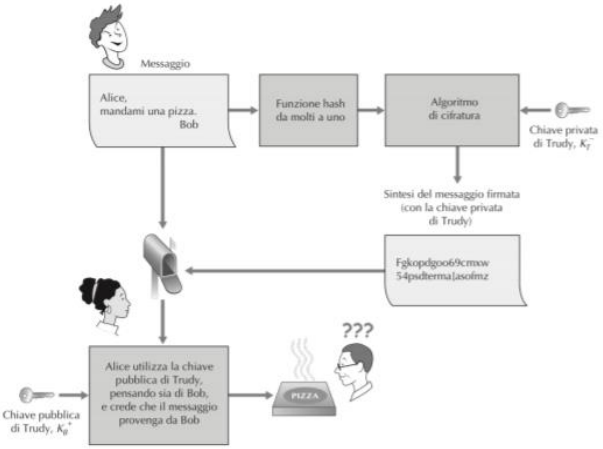
Firma di un documento



Controllo dell'integrità del messaggio firmato

Problemi della firma digitale.

Per quanto due utenti A e B possano scambiare un segreto, *non è garantita l'autenticità*. Infatti, non è possibile verificare che la chiave che si sta usando per decifrare il messaggio ricevuto sia proprio quella del mittente e quindi A potrebbe scambiare la chiave con un perfetto *sconosciuto* che si spaccia per B.



Autorità di certificazione.

Si rende quindi necessaria la *certificazione della chiave pubblica*. L'autorità di certificazione *CA* (acronimo di Certification Authority) convalida l'identità ed emette un certificato. ITU (International Telecommunication Union) e IETF (Internet Engineering Task Force) hanno sviluppato uno standard per i certificati rilasciati dalle autorità di certificazione chiamato *X.509*. Inoltre, per gestire i certificati c'è anche bisogno di un'infrastruttura gerarchica detta *PKI* (Public Key Infrastructure).

Certificazione della chiave pubblica

Nome campo	Descrizione
Versione	Numero di versione della specifica X.509
Numero seriale	Identificatore unico del certificato fornito dalla CA
Firma	Specifica l'algoritmo utilizzato dalla CA per firmare il certificato
Nome dell'emittente	Identificativo della CA che rilascia il certificato, in formato DN [RFC 4514]
Periodo di validità	Inizio e fine del periodo di validità del certificato
Nome del soggetto	Identificativo dell'entità la cui chiave pubblica è associata al certificato (in formato DN)
Chiave pubblica del soggetto	Chiave pubblica del soggetto e indicazioni dell'algoritmo da utilizzare

Campi essenziali di un certificato X.509

Autenticazione.
L'autenticazione è il servizio di sicurezza che permette di *garantire l'identità* degli interlocutori. Gli *interlocutori*, che possono essere utenti o computer, comunicano tra loro secondo queste combinazioni:

- *Computer-computer* (ad es. per la stampa in rete)
- *Utente-utente* (ad es. per i protocolli di sicurezza)
- *Computer-utente* (ad es. per autenticare un server web)
- *Utente-computer* (ad es. per accedere a un sistema)

La proprietà primaria dell'autenticazione è la richiesta di un *corretto controllo d'accesso*.

Tipologie di autenticazione.
Si possono distinguere quattro categorie di sistemi di autenticazione:

- *Locale* – l'utente accede in locale al servizio, che effettua l'autenticazione (ad es. accesso allo smartphone).
- *Diretta* – l'utente accede da remoto al servizio, che effettua direttamente l'autenticazione (ad es. accesso ai servizi Google o alla mail).
- *Indiretta* – l'utente accede da remoto a diversi servizi, che si appoggiano su un servizio di autenticazione separato (ad es. RADIUS, Kerberos, SSO di UniVR).
- *"Off-line"* – i servizi possono prendere decisioni autonome anche senza dover contattare ogni volta l'autorità di autenticazione (ad es. PKI).

Fattori di autenticazione.
I fattori di autenticazione sono basati su qualcosa che l'utente:

- *Conosce* (ad es. segreti come password o PIN).
- *Possiede* (ad es. cose fisiche o elettroniche come chiavi convenzionali o smart card).
- *È* (ad es. caratteristiche biometriche quali le impronte digitali, l'iride o il tono di voce).

Inoltre, è possibile combinare diversi fattori per ottenere un'autenticazione più forte (detta a *fattori multipli*). Per esempio, possiamo combinare:

- Qualcosa che si *possiede* e si *conosce*, come il *bancomat* che richiede un *PIN*.
- Qualcosa che si *possiede* e si *è*, come l'*impronta digitale* memorizzata nel *passaporto elettronico*.

Qualcosa che si conosce.
Il metodo di autenticazione più semplice è quello basato su *username e password*. L'utente inserisce un nome che lo identifica (lo username), solitamente non segreto, e una parola segreta (la password). I *vantaggi* sono che: è semplice per l'utente; è economico; non richiede di immagazzinare un segreto lato client. Gli *svantaggi*, invece, sono che: spesso gli utenti scelgono password deboli; gli stessi metodi di autenticazione basati su password sono deboli. Infatti, tra i possibili *attacchi alle password* troviamo:

- *Intercettazione*, se la password passa in chiaro.
- *Guessing/cracking*, ossia un attacco a pura forza bruta o, più spesso, un "attacco a dizionario" realizzato provando parole di senso compiuto e loro minime variazioni.
- Altri attacchi come *social engineering* e *trojan horse*.

Una password dovrebbe quindi essere abbastanza lunga, non essere una parola di senso compiuto ed essere cambiata di frequente. Tuttavia, questo si scontra con la comodità e la pigrizia degli utenti.

Qualcosa che si possiede.
Esistono sistemi in cui viene generata una *nuova password ad ogni accesso* da parte dell'utente, per risolvere il problema dell'intercettazione. Tali sistemi sono chiamati *OTP* (One-Time Password) e consistono in password monouso che vengono generate sulla base di un contatore (esiste quindi una sequenza di password in successione) o, più spesso, sulla base dell'istante temporale. I sistemi OTP si appoggiano su *token*

(dispositivi hardware che forniscono all'utente la password da inserire) oppure su *SMS* inviati sul cellulare dell'utente. Inoltre, la one-time password viene spesso utilizzata congiuntamente ad un PIN per *combinare possesso e conoscenza*. I token (oppure anche le smart card) forniscono prova dell'identità. Quindi, se i token vengono persi, rubati o clonati, l'autenticazione dimostrerà solo l'identità del token e non più quella dell'utente.

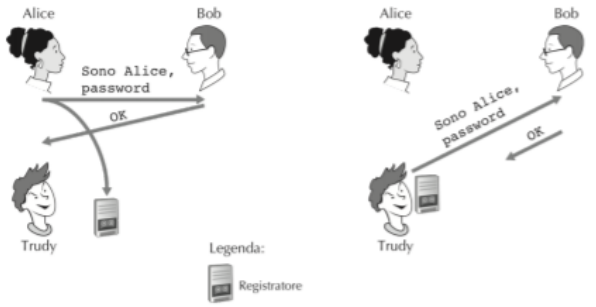
Qualcosa che si è.
Esistono sistemi per fornire prova dell'identità basati sul possesso di univoche *caratteristiche*:

- *Fisiche* (ad es. impronte digitali, forma della mano, impronta della retina, forma del viso).
- *Comportamentali* (ad es. firma, timbro di voce, scrittura sulla tastiera o "keystroke dynamic").

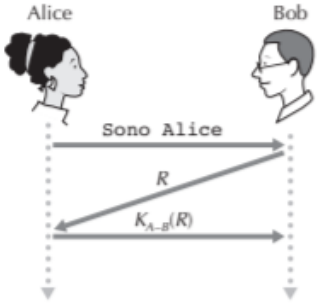
Si tratta di tecniche recenti (in termini di fattibilità) ma promettenti. Tuttavia, esistono comunque dei *punti deboli*:

- L'autenticazione si basa sul confronto con un template e quindi esiste la possibilità di *falsi positivi e falsi negativi* (dovuti ad esempio ad una misura imprecisa durante la creazione del template).
- Le caratteristiche fisiche e comportamentali *non sono sostituibili* se vengono compromesse (ad esempio tramite la falsificazione).

Autenticazione diretta.
Le tipologie di autenticazione viste finora si possono usare senza modifiche per l'*autenticazione locale*. Ma se l'autenticazione avviene da remoto (*autenticazione diretta*) sorgono altri problemi. Un intruso, infatti, potrebbe registrare e replicare le informazioni. Alcune soluzioni possono prevedere che: l'autenticazione avvenga su un canale sicuro; si aggiunga una "sfida" a cui l'utente che si autentica deve rispondere per dare prova della sua identità.



Problemi dell'autenticazione diretta



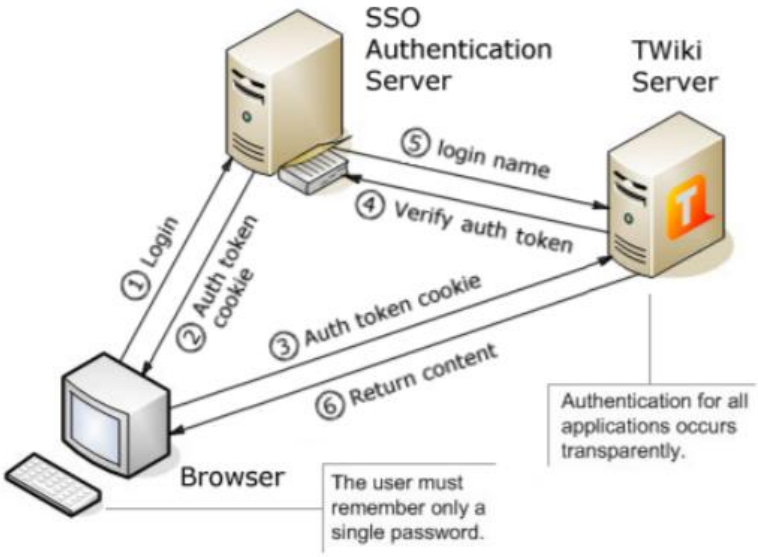
Autenticazione diretta con aggiunta della "sfida"

Autenticazione indiretta.
Quando molti sistemi/applicazioni condividono gli stessi utenti, si ricorre spesso all'autenticazione indiretta. L'*autenticazione indiretta* si basa su un sistema di autenticazione esterno a cui si appoggiano tutti gli altri sistemi e dove vengono centralizzate le informazioni sugli utenti. Due esempi di sistemi di autenticazione indiretta sono:

- *RADIUS* (Remote Authentication Dial In User Service), nato per l'accesso remoto dial-up.
- *Kerberos*, usato come sistema Single Sign-On (SSO) per garantire con un solo login l'accesso a tutti i servizi di un certo dominio amministrativo.

L'autenticazione basata su *SSO* funziona nel seguente modo:

1. L'utente effettua il login sul server SSO.
2. L'utente riceve dal server SSO un token di autenticazione.
3. Il token viene mandato dall'utente al server dove risiede il servizio che necessita di autenticazione.
4. Il server del servizio confronta il token ricevuto confrontandolo con il server SSO.
5. Il server SSO conferma il token.
6. Il server del servizio manda il contenuto necessario al client che ha a questo punto eseguito il login.



Autenticazione "off-line".
È basata sui certificati emessi da un'*autorità di certificazione* (CA) e distribuiti attraverso un'*infrastruttura a chiave pubblica* (PKI). Un *certificato reale* (cartaceo) è emesso da un'autorità riconosciuta e associa l'identità di una persona al suo aspetto fisico. Analogamente, un *certificato digitale* (elettronico) è emesso da una CA riconosciuta (è quindi firmato con la chiave privata della CA) e associa l'identità di una persona ad una chiave pubblica.

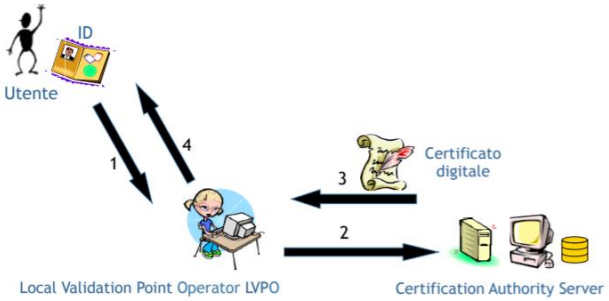
I 10 compiti di una CA.
1. Identificare con certezza la persona che fa richiesta della certificazione della chiave pubblica.
2. Rilasciare e rendere pubblico il certificato.
3. Garantire l'accesso telematico al registro delle chiavi pubbliche.
4. Informare i richiedenti sulla procedura di certificazione e sulle tecniche per accedervi.
5. Dichiarare la propria politica di sicurezza.
6. Attenersi alle norme sul trattamento di dati personali.
7. Non rendersi depositario delle chiavi private.
8. Procedere alla revoca/sospensione dei certificati in caso di richiesta dell'interessato o venendo a conoscenza di abusi/falsificazioni.
9. Rendere pubblica la revoca o la sospensione delle chiavi.
10. Assicurare la corretta manutenzione del sistema di certificazione.

Ottenere un certificato digitale.
Per ottenere un certificato digitale, l'utente deve *generare* sul proprio PC *una coppia di chiavi*, memorizzando la chiave privata localmente in un file nascosto. Oppure, l'utente può generare la coppia di chiavi tramite smart card collegata al PC per garantire maggiore sicurezza, in quanto la chiave privata non uscirà mai dalla smart card (che è protetta da un PIN). Poi, l'utente deve *inviare alla CA la richiesta di certificato* e la chiave pubblica generata (a meno che non sia la CA a generare la coppia di chiavi per l'utente). A questo punto la CA, per autenticare il richiedente, chiede all'utente di *recarsi di persona ad uno sportello di LVP* (Local Validation Point) collegato ad essa. Verificata l'identità, *la CA emette il certificato* (ovvero inserisce la chiave certificata nel registro delle chiavi pubbliche) e lo invia al richiedente tramite posta elettronica.

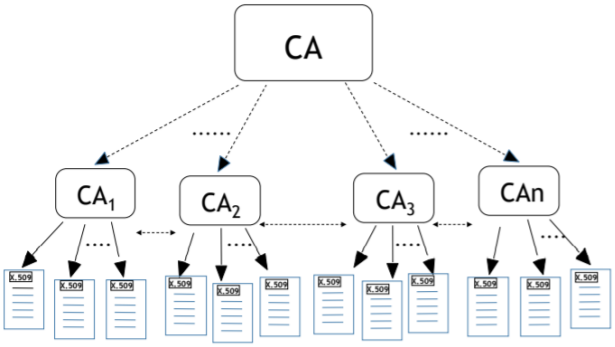
PKI (Public Key Infrastructure).
L'intera procedura per ottenere il certificato digitale accade nell'ambito di una *PKI*. La struttura *minima* di una PKI prevede una *CA* con uno o più sportelli *LVP* associati. In realtà, la struttura di una PKI è *gerarchica*,

con alcune CA che ne certificano altre per ottenere una "catena di fiducia". La PKI a struttura gerarchica può essere quindi vista come un *albero*:

- La CA root certifica le CA di primo livello.
- Le CA di primo livello certificano le CA di secondo livello e così via.
- Le CA di ultimo livello certificano i singoli utenti.



Ottenere un certificato digitale



PKI a struttura gerarchica

Problemi nell'utilizzo dei certificati.

Bob *invia* ad Alice il proprio certificato (firmato dalla CA). Poi, Alice *verifica la firma della CA* sul certificato di Bob e, se è corretta, *estrae la chiave pubblica* di Bob dal certificato (Alice deve già avere il certificato auto-firmato dalla CA per poterne verificare la firma). A questo punto, Alice ha ottenuto la chiave pubblica di Bob, la cui identità è garantita dalla CA. Tuttavia, come si è già visto in altri schemi, sorgono i seguenti *problemi*:

- *Distribuzione delle chiavi pubbliche* (anche se su scala molto più ridotta), in quanto è comunque necessario ottenere in modo sicuro il certificato della CA.
- *Revoca dei certificati* (ad es. se il proprietario si accorge del furto della chiave privata corrispondente), in quanto la verifica della firma della CA su un certificato revocato va comunque a buon fine; la soluzione è quella di accertarsi della validità del certificato controllando la lista dei certificati revocati che viene pubblicata e firmata dalla CA.
- *Fiducia implicita nella CA* da parta del sistema, senza avere alcuna garanzia.
- *Dipendenza dall'utente*, in quanto egli deve generare, custodire (in modo che nessuno se ne impossessi) e trasportare (in modo da poterla utilizzare in qualunque luogo) la sua chiave privata.

Autorizzazione.

Il servizio di controllo dell'accesso (detto anche *autorizzazione*) garantisce che l'accesso alle risorse sia limitato ai soli utenti che ne hanno diritto. Infatti, soggetti diversi possono avere diritto a diverse modalità di interazione con le risorse. Si identificano quindi tre tipi di attori:

- *Soggetti* (utenti, applicazioni, altri sistemi).
- *Privilegi* (lettura, scrittura, esecuzione, possesso).
- *Oggetti* (file, funzioni, applicazioni, altri sistemi).

I *soggetti* sono in possesso di *privilegi* sugli *oggetti* in accordo con le politiche definite sul sistema. Le *politiche* di controllo dell'accesso, infatti, definiscono l'attribuzione dei privilegi di accesso dei soggetti sugli oggetti. I *meccanismi* di controllo dell'accesso, invece, specificano come le relazioni tra i soggetti e gli oggetti (ovvero i privilegi) sono rappresentate. Per assegnare in maniera più affidabile possibile i privilegi, ci si affida a due principi:

- *Privilegio minimo*, ovvero ad un soggetto dovrebbero essere concessi solo i privilegi minimi necessari a compiere l'azione che deve compiere.
- *Separazione dei compiti*, ovvero nessun soggetto dovrebbe avere abbastanza potere per sovvertire il sistema.

Matrice di controllo dell'accesso.
È un meccanismo di controllo dell'accesso in cui le righe contengono i soggetti, le colonne contengono gli oggetti e nelle caselle vengono rappresentati i permessi. Siccome soggetti e oggetti possono essere molto numerosi, la matrice può diventare sparsa e quindi sorgono *problemi di scalabilità*. Alcune soluzioni prevedono di memorizzare la matrice per righe/colonne oppure di effettuare raggruppamenti per gestire i privilegi in modo omogeneo (gruppi/ruoli).

	File 1	File 2	Progr.1	Progr.2
Alice	rwx	rwx,own	x	rwx
Bob	rwx,own	r	x	rwx
Progr.1	rw	rw	-	x

Access Control List (ACL).
Prevede di memorizzare la matrice di controllo dell'accesso per *colonne*. Ciascuna risorsa viene quindi memorizzata con i relativi permessi e con la lista dei soggetti che possono interagire con essa (ad es. nel filesystem di Unix). Le ACL sono adatte a contesti in cui la protezione è *orientata ai dati* perché è semplice gestire i permessi associati ad un oggetto; non sono invece adatte se si vogliono *gestire centralmente* i permessi di ciascun soggetto e/o se si vogliono introdurre meccanismi di delega temporanea.

Capabilities.
Prevede di memorizzare la matrice di controllo dell'accesso per *righe*. A ciascun soggetto è associato l'insieme degli oggetti con cui può interagire, ovvero si memorizza una lista di oggetti con i relativi permessi. Le capabilities permettono di gestire in modo efficiente i permessi associati ad un *singolo utente* e rendono quindi semplice anche un meccanismo di delega temporanea. Per determinare tutti i soggetti che hanno diritto di interagire con un oggetto si deve semplicemente *scorrere la lista*.

Politiche di controllo dell'accesso.
Si distinguono due diversi approcci: *DAC* (Discretionary Access Control) e *MAC* (Mandatory Access Control). Tali approcci possono essere combinati con una suddivisione degli utenti in gruppi e ruoli. Un *gruppo* è una lista di soggetti, mentre un *ruolo* è un insieme prefissato di permessi di accesso che uno o più soggetti possono acquisire per un certo periodo di tempo (spesso corrisponde ad una funzione all'interno di un'organizzazione).

DAC (Discretionary Access Control).
In un modello DAC i singoli utenti possono, *a loro discrezione*, concedere e revocare permessi su oggetti che sono sotto il loro controllo. In genere ci si basa sul concetto di proprietà (ownership): *ogni oggetto ha un proprietario*, cioè il soggetto che ne definisce i diritti di accesso. Eventualmente, il proprietario può assegnare la proprietà ad un altro soggetto. DAC è *flessibile* ed è utilizzato in molti ambiti (ad es. nei sistemi operativi Unix e Windows), ma non permette di controllare la diffusione dell'informazione (chi ha permessi in lettura su un file potrebbe infatti inviarlo a chi non ha permessi).

MAC (Mandatory Access Control).
In un modello MAC la politica di controllo dell'accesso è determinata *centralmente* dal sistema, non dai singoli utenti. È utilizzato per esempio in ambito militare e si basa su una *classificazione* degli oggetti e dei soggetti (ad es. top-secret, secret, confidential, classified). Il modello MAC è meno flessibile ma più *robusto* del modello DAC. Infatti, nel modello di Bell-LaPadula il sistema forza il rispetto delle seguenti regole:

- *No read up*, cioè non è possibile leggere informazioni classificate a livelli più alti del proprio.
- *No write down*, cioè non è possibile scrivere informazioni classificate a livelli più bassi del proprio.

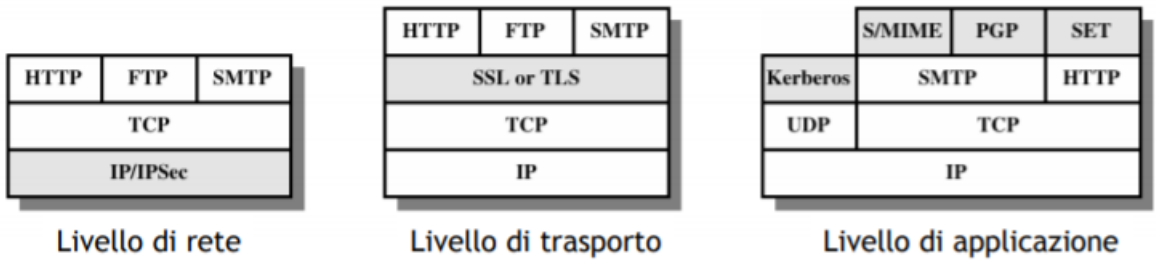
Gruppi e ruoli.
I *gruppi* permettono di gestire insiemi di soggetti/oggetti in modo *omogeneo*. Ciò semplifica l'associazione dei permessi agli utenti in quanto:

- L'accesso alle risorse è basato sui permessi dei gruppi.
- Diversi gruppi possono avere proprietà sovrapposte.
- Modificare i diritti di un gruppo permette di cambiare direttamente quelli di tutte le entità appartenenti.

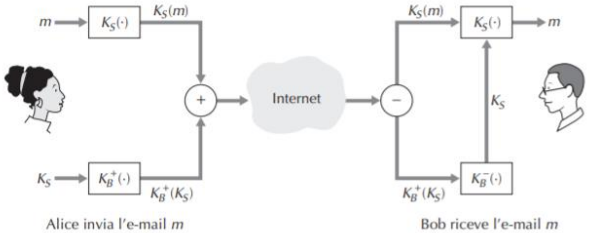
I *ruoli* definiscono insiemi di *proprietà e responsabilità* solitamente associate alla struttura organizzativa a cui fa capo il sistema (ad es. RBAC – Role Based Access Control).

❖ Sicurezza delle e-mail, del livello di trasporto e delle wireless LAN

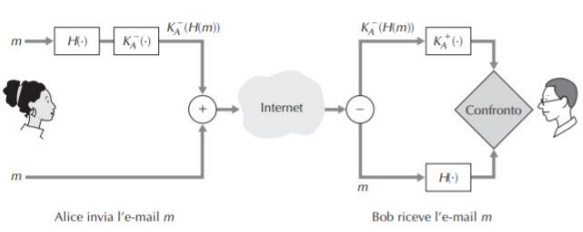
Sicurezza nello stack protocollare TCP/IP.
Ogni *applicazione* implementa già all'interno della sua logica la sicurezza e quindi il TCP non riuscirebbe comunque a capire il contenuto dei dati che riceve in quanto essi sono già cifrati. Chi sviluppa l'applicazione si prende in carico di sviluppare anche la parte legata alla cifratura che può essere tuttavia inutile ed onerosa. A *livello di trasporto*, invece, il messaggio viene cifrato con SSL/TLS e quindi il TCP continua a non vederne il contenuto. Il vantaggio è che questo strato intermedio che rende più sicura la connessione può essere utilizzato da più di un'applicazione e quindi il protocollo a livello applicativo non deve essere modificato. Nella cifratura a livello di trasporto, però, si può riuscire ad intercettare parte del payload del pacchetto IP e quindi anche parte del pacchetto TCP. Per risolvere questo problema si può utilizzare la cifratura a *livello di rete*, ovvero quella che avviene direttamente sul pacchetto IP. In questo modo, infatti, al massimo si può riuscire ad intercettare l'header IP (necessario per gli indirizzi sorgente e destinazione) in quanto subito dopo c'è un secondo header, appartenente al protocollo IPSec, che contiene le informazioni utilizzate per la cifratura. Infine, esiste anche la sicurezza a *livello di collegamento*, in cui viene cifrato il pacchetto usato nelle wireless LAN che attraversa il tratto condiviso tra l'host e l'access point.



Rendere sicura la posta elettronica.
Per la *sicurezza delle e-mail* è necessario garantire le seguenti caratteristiche: riservatezza, autenticazione del mittente, integrità, autenticazione del destinatario. Per ottenere la *riservatezza* basta cifrare il messaggio con una chiave simmetrica (DES o AES), mentre per ottenere l'*integrità* basta usare le funzioni di hash. Per ottenere anche l'*autenticazione* bisogna utilizzare la crittografia a chiave pubblica (tramite RSA).



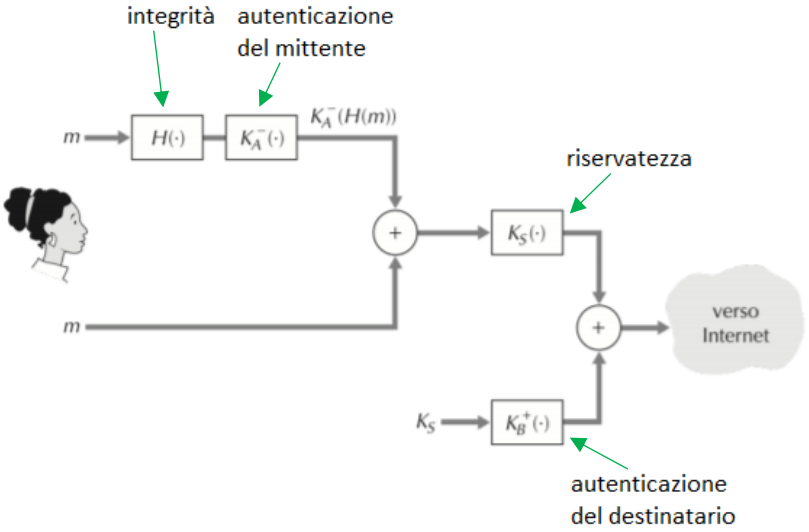
Alice invia e-mail a Bob con la chiave simmetrica



Utilizzo di funzioni hash per l'integrità delle e-mail

Esempio di schema.

In uno *schema completo*, Alice utilizza la crittografia a chiave simmetrica, quella a chiave pubblica, una funzione di hash e la firma digitale per ottenere riservatezza, autenticazione e integrità del messaggio.

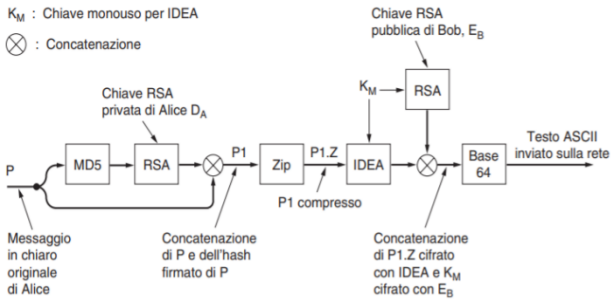


PGP (Pretty Good Privacy).

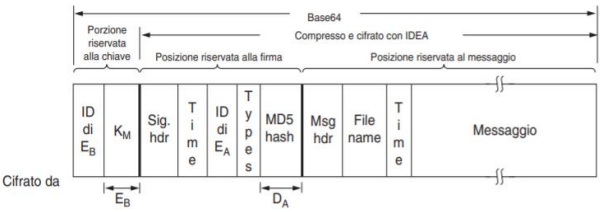
È un programma per lo *scambio sicuro di messaggi* testuali che garantisce confidenzialità e autenticazione. È stato sviluppato da P. Zimmerman e integra *algoritmi di crittografia* consolidati. È indipendente dal sistema operativo e dall'architettura in quanto sorgenti, librerie e documentazione sono disponibili gratuitamente su Internet.

Servizi offerti da PGP.

PGP garantisce l'*autenticazione* tramite l'utilizzo di SHA-1 e RSA. Inoltre, PGP garantisce la *confidenzialità* tramite l'utilizzo di una chiave di sessione one-time che viene usata per cifrare il messaggio con CAST-128, IDEA o Triplo-DES. I messaggi vengono poi sottoposti a *compressione* con successiva *codifica* in Base 64 (ovvero gruppi di 6 bit vengono tradotti in caratteri) per poter essere correttamente inviati sulla rete.



Invio di un messaggio con PGP



Formato dei messaggi PGP

Uso della fiducia.

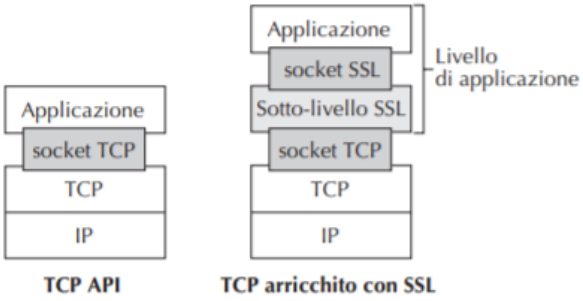
È compito dell'utente assegnare un livello di fiducia ad ogni conoscente ed intermediario. Per fare ciò esistono i campi:

- *Owner trust*, che esprime il grado di fiducia nel proprietario come certificatore ed è assegnato dall'utente (unknown, untrusted, marginally trusted, completely trusted).
- *Signature trust*, che esprime il grado di fiducia nel firmatario come certificatore ed è uguale a owner trust se il firmatario è tra i conoscenti, altrimenti vale unknown.
- *Key legitimacy*, che viene calcolato da PGP in base al valore dei campi signature trust.

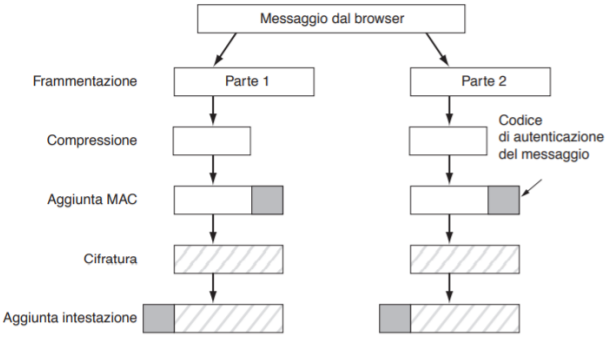
PGP assegna quindi il livello di fiducia all'abbinamento chiave pubblica-utente.

Rendere sicure le connessioni TCP.

SSL (Secure Sockets Layer) è una versione di TCP arricchita con servizi di sicurezza quali riservatezza, integrità dei dati e autenticazione del client e del server. Ci si accorge che viene usato SSL quando nel browser l'URL comincia con HTTPS invece che con HTTP. Il protocollo SSL è stato progettato inizialmente da Netscape con lo scopo specifico di *proteggere le transazioni web*. È divenuto poi uno standard IETF, con il nome *TLS* (Transport Layer Security), quando ha cominciato a focalizzarsi principalmente sulle proprietà di *confidenzialità* e *integrità* del traffico di rete.



Differenza tra TCP e TCP con SSL

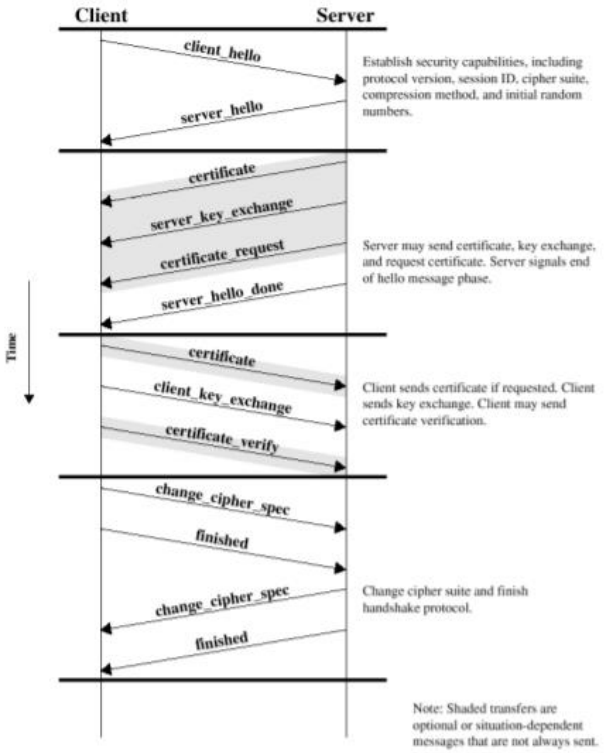


Trasmissione dei dati con SSL

SSL/TLS handshake.

La fase di handshake provvede a:

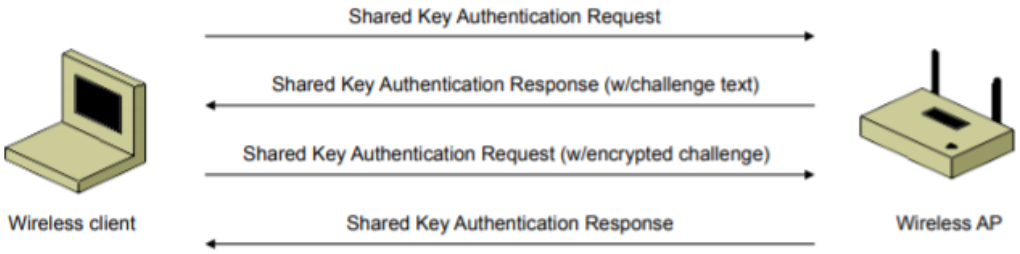
1. *Autenticare le parti* (opzionale), tramite autenticazione del solo server o mutua autenticazione.
2. Accordare le due parti sugli *algoritmi crittografici* da usare.
3. *Generare le chiavi di sessione* per la cifratura dei dati.



Rendere sicuro il transito dei dati su reti wireless.

WEP (Wired Equivalent Privacy) è un protocollo 802.11 pensato per assicurare un livello di sicurezza simile a quello delle reti cablate (anche se nella pratica è poco sicuro). WEP lavora al *livello data link* e fornisce *autenticazione* e *crittografia* (codifica dei dati) tra terminale e access point wireless con un approccio a chiave simmetrica condivisa. Richiede quindi che la stessa chiave segreta venga usata da tutti i sistemi in comunicazione (host e AP).

802.11 shared key authentication.
L'AP invia un *testo di challenge in chiaro* ad ogni device che cerca di comunicare. Il device che richiede l'autenticazione *cripta il testo di challenge* e lo invia all'AP. A questo punto, se il testo di challenge è criptato correttamente, l'AP ritiene autenticato il device. C'è però un problema fondamentale, ovvero testo in chiaro e testo criptato sono entrambi disponibili agli attaccanti. Quindi è possibile effettuare attacchi di "brute force" per individuare la chiave segreta condivisa.



Vulnerabilità di WEP.
Sebbene usi una *chiave segreta*, un *checksum cifrato* (con shared key) per garantire l'integrità dei dati e l'algoritmo di cifratura *RC4*, WEP è un protocollo vulnerabile e con scarso controllo di accesso. Infatti, la chiave è *condivisa* e, siccome ogni utente ha la stessa chiave, se intercetto la chiave catturo il traffico di tutti gli utenti compromettendo la confidenzialità e l'integrità dei dati.

Wi-Fi Protected Access.
Sono protocolli che si basano sull'emendamento *802.11i* creato appositamente per la sicurezza. Operano a *livello MAC* (Media Access Control) e forniscono le seguenti caratteristiche crittografiche:

- Crittazione dei dati.
- Integrità dei dati.
- Protezione da attacchi di tipo "replay".

Caratteristiche di sicurezza di WPA.
Il protocollo *WPA* contiene un sottoinsieme delle feature di sicurezza che sono nello standard *802.11i* consentendogli di risolvere molte delle debolezze di WEP. In particolare, garantisce *autenticazione* tramite il protocollo EAP e *cifratura e integrità dei dati* tramite i protocolli TKIP, MIC e AES.

Caratteristiche di sicurezza di WPA2.
Il protocollo *WPA2* è attualmente il più sicuro in quanto rispetta lo standard *802.11i* usando anche diversi meccanismi come una pre-shared key per l'autenticazione (PSK), una cache per permettere il fast roaming (PMK) e una cifratura obbligatoria di 128 bit di dati alla volta con una chiave a 128 bit tramite AES.

❖ Firewall e Intrusion Detection System

Firewall.
I *firewall di rete* sono apparecchiature o sistemi che controllano il flusso del traffico tra due reti con differenti livelli di sicurezza. I *compiti* di un firewall sono:

- Prevenire accessi non autorizzati alla rete privata.
- Prevenire l'esportazione di dati dall'interno verso l'esterno.
- Schermare alcune reti interne e nasconderle agli altri.
- Bloccare alcuni accessi a servizi o ad utenti.

I *problemi* dei firewall, invece, sono che:

- Assumono che gli attacchi avvengano sempre dall'esterno (ma possono iniziare anche dall'interno).
- Non difendono contro nuovi bachi non ancora documentati nei protocolli.
- I loro filtri sono difficili da settare e da mantenere perché esiste un difficile compromesso tra libertà e sicurezza.
- Possono degradare le performance della rete.

Due filosofie per i firewall.

La prima è *default deny*, ovvero tutto quello che non è espressamente ammesso è proibito, in cui:

- I servizi sono abilitati caso per caso dopo una attenta analisi.
- Gli utenti hanno molte restrizioni e non possono facilmente rompere la policy di sicurezza.

La seconda è *default permit*, ovvero tutto quello che non è espressamente proibito è ammesso, in cui:

- Il system administrator deve reagire prontamente ogni volta che viene scoperto un nuovo baco su un protocollo.
- I servizi vengono rimossi/ridotti quando si scopre che sono pericolosi.
- Gli utenti hanno meno restrizioni.

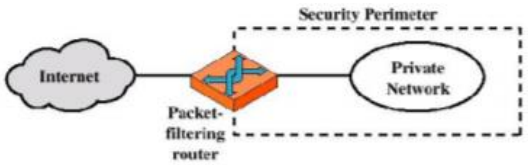
Firewall di 1ª generazione – filtri a livello 3.

Agiscono sul *singolo pacchetto* e controllano: indirizzi IP sorgente e destinazione, tipo di traffico (ad es. IP, ICMP, protocolli di livello 2) e, possibilmente, alcune caratteristiche del livello 4 come porta sorgente e destinazione. Talvolta, controllano anche informazioni interne al router, ossia informazioni circa le interfacce sorgente e di destinazione del pacchetto (utile per i router con più interfacce). Tra i *vantaggi* di questi firewall troviamo:

- Disponibilità in molti router.
- Costo d'acquisto contenuto.
- Trasparenza, in quanto non lavorano a livello applicativo e quindi non ostacolano il normale utilizzo della rete.
- Velocità, in quanto effettuano meno controlli.

Tra le *limitazioni*, invece, troviamo che:

- Le regole sono difficili da configurare.
- Possono avere bug.



Firewall di 2ª generazione – stateful packet filtering.

Il loro funzionamento prevede che quando viene stabilita una connessione, se le regole di filtraggio non la bloccano, allora le informazioni relative ad essa diventano entry di una *tabella di stato*. Quindi i successivi pacchetti in ingresso saranno valutati in base all'appartenenza ad una delle connessioni consentite presenti nella tabella. Infine, quando la connessione è conclusa, la entry nella tabella sarà cancellata, per evitare che questa si riempia completamente. Tra i *vantaggi* di questi firewall troviamo:

- Tutti i vantaggi della generazione precedente.
- Buon rapporto prestazioni/sicurezza, in quanto effettuano meno controlli sulla connessione.
- Protezione da IP spoofing, in quanto il controllo non si limita al singolo IP o alla porta.

Tra le *limitazioni*, invece, troviamo che:

- Mancano servizi aggiuntivi, in quanto non possono agire a livello di applicazione.
- Il testing è complesso, in quanto verificare la loro corretta configurazione non è facile.

Inoltre, tra le *informazioni riguardanti la connessione* che vengono memorizzate troviamo:

- Identificatore univoco della connessione.
- Stato della connessione (handshaking, established, closing).
- Informazioni sulla sequenzialità dei pacchetti.
- Indirizzi IP sorgente e destinazione.
- Interfacce di rete utilizzate.

Source address	Source port	Dest. Address	Dest. Port	Connection state
192.168.0.199	1051	192.168.1.10	80	Handshaking
192.168.0.212	1109	192.168.1.23	25	Closing
192.168.3.105	1212	192.168.0.111	80	Established

Firewall di 3ª generazione – filtri a livello 7.

In questi firewall il routing tra due interfacce viene effettuato a livello applicazione dal software del firewall (in caso di malfunzionamento del software il routing è disabilitato). C'è quindi la possibilità di *gestire le autenticazioni* e di configurare *filtri su specifici comandi* (ad es. permettere GET ma non PUT). Tra i *vantaggi* di questi firewall troviamo:

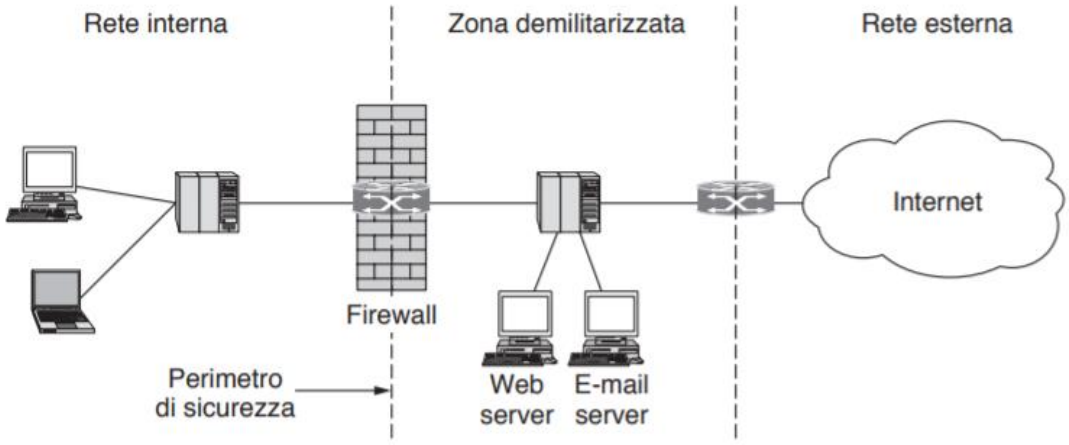
- Maggiore sicurezza rispetto alle generazioni precedenti.
- Controllo solo su un numero limitato di applicazioni (HTTP, FTP, posta).
- Facilità di controllo del traffico.

Tra gli *svantaggi*, invece, troviamo che:

- C'è un processing overhead su ogni connessione.
- Possono controllare solo un numero limitato di applicazioni (HTTP, FTP, posta).

Proxy server dedicati e personal firewall.

I *proxy server dedicati* sono specifici per ogni applicazione e aiutano il proxy gateway principale nel lavoro di ispezione dei contenuti. Usi tipici di questi server sono ad esempio: antivirus, web-cache proxy, e-mail proxy. I *personal firewall*, invece, proteggono solo la macchina dove sono installati e sono necessari, ad esempio, agli utenti del mondo mobile.



Sistema di rilevamento delle intrusioni.

Un *IDS* (Intrusion Detection System) è uno strumento, software o hardware, che automatizza il processo di monitoraggio impiegato per individuare eventi che rappresentano intrusioni non autorizzate ai computer o alle reti locali. Tale monitoraggio si può quindi fare a livello di host (HIDS) o a livello di rete (NIDS). Un IDS può quindi essere considerato come un *antifurto* che cerca di rilevare eventuali intrusioni. Un firewall, invece, può essere considerato come una *porta blindata* che serve a bloccare le eventuali intrusioni.

IDS perfetto vs reale.
Un IDS perfetto dovrebbe riuscire ad individuare tutte le reali intrusioni. Ma, nella realtà, si possono verificare due situazioni problematiche:

- *Falsi positivi*, ovvero l'IDS rileva un'anomalia quando invece non è successo niente.
- *Falsi negativi*, ovvero l'IDS non rileva un'intrusione effettivamente avvenuta.

In sostanza, i requisiti minimi per un IDS reale sono:

- Scoprire un'ampia gamma di intrusioni (sia già note che non note).
- Scoprirle velocemente (anche non necessariamente in tempo reale).
- Presentare i report delle analisi in formato semplice e facilmente comprensibile.
- Essere accurato.

Modelli di IDS.
Il *principio di base* è quello di distinguere situazioni normali da quelle anomale. L'utente, in condizioni normali, si comporta in modo più o meno prevedibile: non compie azioni atte a violare la sicurezza e i suoi processi compiono solo azioni permesse. I *modelli* (statici o adattivi) su cui gli IDS vengono implementati possono essere basati su:

- *Detection di anomalie*, quando è possibile che sequenze di azioni non usuali siano intrusioni.
- *Detection di uso malevolo*, quando si conosce quali sequenze di azioni possono essere intrusioni.
- *Detection in base a specifiche*, quando si conoscono le situazioni derivanti da intrusioni.

Detection di anomalie.
Si analizzano insiemi di caratteristiche del sistema confrontando i valori con quelli attesi e segnalando quando le statistiche non sono paragonabili a quelle attese. Tali situazioni sono verificate tramite: *metriche a soglia*, *momenti statistici* o *modelli di Markov*.

Metriche a soglia.
Si conta il numero di volte in cui un evento si presenta. Se ci si aspetta tra m e n occorrenze e il numero cade al di fuori, allora c'è un'anomalia. Ad esempio, lo svolgimento dell'azione di *login* troppe volte desta sospetto. Il problema principale di questo metodo consiste nella difficoltà a trovare l'*intervallo* corretto, in quanto talvolta si possono creare situazioni in cui esso dovrebbe essere più grande (ad es. utenti francesi che usano una tastiera americana).

Momenti statistici.
L'analizzatore calcola la deviazione standard (*primi due momenti*) o altre misure di correlazione (momenti di ordine superiore). Se i valori misurati di un certo momento cadono al di fuori di un certo *intervallo*, allora vi è un'anomalia. Il problema principale di questo metodo è che i profili possono evolvere nel tempo. Si può cercare di risolverlo pesando opportunamente i dati o alterando le regole di detection.

Modelli di Markov.
L'ipotesi di questo metodo è che la storia passata influenzi la prossima transizione di stato. Le anomalie sono quindi riconosciute da sequenze di eventi e non dalle occorrenze dei singoli eventi. Il sistema deve però essere addestrato per riconoscere le sequenze valide. Tale *addestramento* viene svolto con utenti non anomali e produce migliori risultati con una quantità maggiore di dati, i quali dovrebbero coprire tutte le sequenze normali del sistema.

Detection di uso malevolo.
Si controlla se una *sequenza* di istruzioni da eseguire è *già nota* per essere potenzialmente dannosa per la sicurezza del sistema. La conoscenza è rappresentata mediante regole e il sistema controlla se la sequenza soddisfa una di queste regole. Non si possono quindi scoprire intrusioni non note precedentemente.

Detection in base a specifiche.
Si determina se una sequenza di azioni viola una specifica di come un programma o un sistema dovrebbe funzionare.

Architettura di un IDS.
Un IDS è essenzialmente un *sistema di auditing* sofisticato composto da tre attori principali: agente (logger), direttore (analizzatore) e notificatore (esecutore). L'*agente* ottiene le informazioni e le invia al direttore, ma può anche manipolare le informazioni (ad es. estrarne parti rilevanti o cancellare quelle non necessarie). Il *direttore* colleziona le informazioni inviate dagli agenti e può anche eliminare i record ridondanti o non necessari. Inoltre, analizza le informazioni per determinare se si è sotto attacco (con le tecniche viste in precedenza) e, per non influenzare le performance dei sistemi monitorati, gira di norma su un sistema separato. Infine, il *notificatore* ottiene i risultati elaborati dal direttore e prende le decisioni appropriate, ossia: notificare messaggi agli amministratori, riconfigurare gli agenti, rispondere all'attacco.

Combinazione DIDS.
I monitoraggi di host e di network non sono generalmente sufficienti per scoprire alcuni tipi di attacchi:

- Un attaccante prova a fare telnet con vari login (gli IDS di rete lo possono scoprire, gli IDS di host no).
- L'attaccante prova ad entrare senza la password (gli IDS di host lo rilevano, gli IDS di rete no).

Per questo *DIDS* usa gli agenti host ed un monitor di rete per controllare.

Risposte alle intrusioni.
L'attacco dovrebbe essere scoperto prima del suo completamento (*prevenzione*). Una tecnica alternativa è rappresentata dal *jailing*, ovvero far credere all'attaccante che l'intrusione è andata a buon fine ma confinare le sue azioni in un dominio che imita il sistema reale in cui non può fare danni o al limite causarne pochi (ad es. scaricare file falsi o corrotti).

Falsi positivi e falsi negativi.
In caso di rilevamento di una possibile intrusione è bene svolgere più volte lo stesso test (oppure svolgere un test diverso anche meno accurato) per minimizzare la possibilità di riscontrare un falso positivo (parvenza di intrusione) o un falso negativo (intrusione che non viene segnalata).

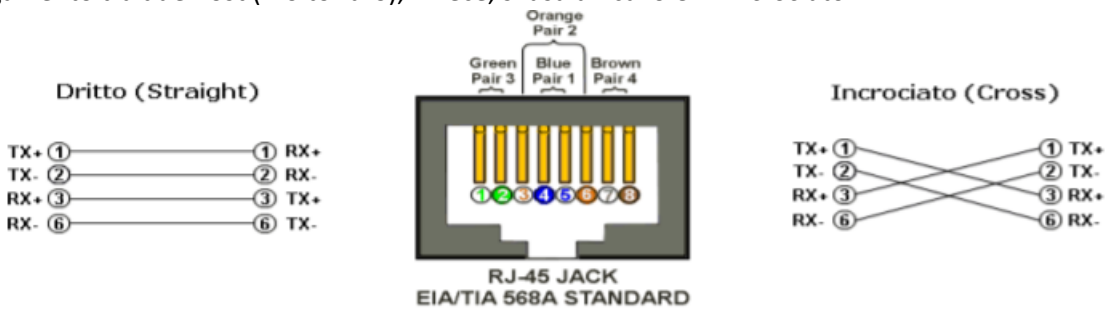
CONFIGURAZIONE DI RETE

❖ Introduzione alla gestione dei sistemi di rete

Supporto agli apparati di rete.
Tra le *tipologie di cavi* si possono trovare:

- Doppino UDP (Un-shielded Twisted Pair) non schermato.
- Doppino STP (Shielded Twisted Pair) schermato.
- Connettore RJ 45 (4 coppie) per il cavo ethernet.
- Connettore RJ 11 (2 coppie) per il cavo telefonico.
- Fibra ottica multi-modale (LED) o mono-modale (laser) per trasmissioni ad altissima capacità in ambienti con elevato rumore elettromagnetico.

Uso di UTP dritti e incrociati.
Per il collegamento tra host e switch o tra due switch (tramite porta di uplink) si usa un cavo *UTP dritto*. Per il collegamento tra due host (molto raro), invece, si usa un cavo *UTP incrociato*.



PoE (Power over Ethernet).
Con i cavi RJ 45 è possibile utilizzare la tecnologia PoE che permette di semplificare il cablaggio portando alimentazione sullo stesso doppino UTP usato per i collegamenti Ethernet. Ci sono *due possibilità*: potenza e dati vengono veicolati sulle stesse coppie; oppure, le due coppie non utilizzate per i dati sono usate per portare l'alimentazione.

Switch.
Gli switch spezzano il dominio di collisione ma non quello di broadcast. Possono collegare reti 802 diverse tra loro o uguali ma con velocità diverse e permettono la commutazione tra porte e l'accodamento in memoria dei frame (store & forward). Inoltre, possono eliminare frame errati e frammenti di collisione.

Selective flooding.
Se non si conosce l'associazione MAC/porta, i frame che arrivano da una certa porta vengono poi trasmessi su tutte le altre (non avrebbe senso trasmetterli anche sulla porta dalla quale provengono).

Backward learning.
È un *algoritmo* usato dallo switch per imparare gli indirizzi MAC degli host attaccati sulle sue porte. Il suo *funzionamento* prevede di guardare il campo dell'indirizzo MAC sorgente dei frame che arrivano su ciascuna porta. L'associazione MAC/porta è molti-a-uno perché a quella porta può essere attaccato un intero sottoalbero della rete. Inoltre, le associazioni imparate si aggiornano dinamicamente nel tempo e quindi non è richiesto alcun intervento umano.

Sicurezza degli switch.
La sicurezza degli switch è *debole*, in quanto basta un analizzatore come Wireshark per:

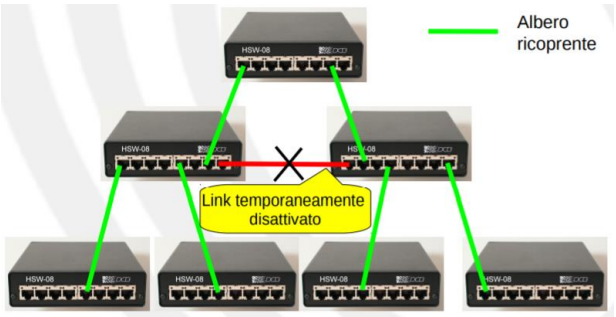
- Re-innescare la modalità di flooding generando PDU ethernet con MAC diverso che sporcano la tabella MAC/porta (*poisoning*).
- Fare uno spoofing di un indirizzo MAC (*furto di identità*).
- Generare PDU ethernet con il MAC di un altro PC per falsare il backward learning dello switch che quindi inoltrerà sull'interfaccia desiderata tutto il traffico diretto a quel PC (*furto di informazioni*).

Affidabilità degli switch.
Tra le maggiori *cause di malfunzionamento* troviamo:

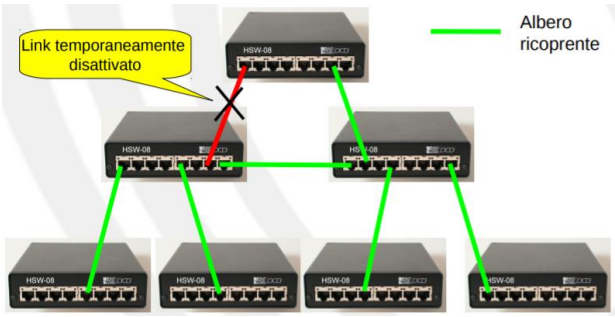
- Crash di uno switch.
- Guasto di porte.
- Interruzione di link tra switch.
- Errore nel cablaggio a livello di armadio (cablaggio strutturato).

Per risolvere questi problemi si deve seguire lo *standard 802.1D*, il quale prevede l'introduzione e la gestione di switch e link ridondanti.

Algoritmo spanning tree.
Uno svantaggio dello standard 802.1D è la *duplicazione dei frame*. Per risolverlo, si deve usare l'algoritmo *spanning tree* che prevede uno scambio periodico di PDU ethernet con indirizzo MAC destinazione di tipo multicast in modo che gli switch possano comunicare tra di loro lo stato dei propri link. La rete viene quindi coperta mediante un albero privo di cicli ottenuto disattivando temporaneamente i link ridondanti. In caso di caduta di un link, i link disattivati possono essere riattivati per generare un nuovo albero.

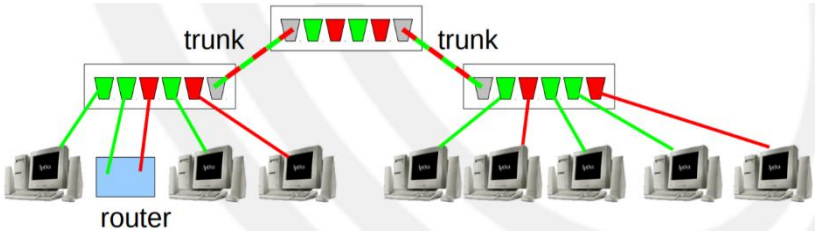


Albero ricoprente (link ridondanti disattivati)



Nuovo albero ricoprente dopo la caduta di un link

Virtual LAN.
Gli switch separano i domini di collisione ma non quelli di broadcast, per cui il protocollo ARP e i malfunzionamenti generano *traffico broadcast* che occupa banda inutilmente. A questo si aggiungono anche i *problemi di sicurezza* (furto di informazioni e di identità). Una soluzione può quindi essere quella di partizionare una LAN in tante *VLAN* da collegare tramite router IP (creando corrispondenti sottoreti IP). La separazione degli host in VLAN diverse può avvenire anche se essi sono collegati allo stesso switch. Infatti, l'amministratore può cambiare in qualsiasi momento l'assegnazione delle porte tramite un software di network management senza bisogno di spostare cavi. Invece, per distribuire VLAN su più switch, occorre scrivere l'id della VLAN in un nuovo campo della trama ethernet (secondo lo standard *802.1Q*). Le porte che collegano tra loro due switch verranno chiamate *porte trunk*.



Sottoreti IP.
Un insieme di switch e/o access point Wi-Fi crea una rete di livello 2. Dal punto di vista IP una rete di livello 2 è considerata una *sottorete IP*, in quanto il prefisso IP e la maschera sono comuni a tutte le interfacce. Le sottoreti IP sono collegate tra loro tramite *router*, i quali comunicano tra loro per compilare le tabelle di routing e realizzare il routing su Internet, che non è altro che un insieme di sottoreti IP.

❖ Cisco IOS

Tipi di memoria in un router.
In un router sono presenti diversi tipi di memorie:

- *RAM*, che è la memoria di lavoro e contiene le informazioni di configurazione dinamica (tabelle di routing, cache ARP).
- *NVRAM*, che è la RAM non volatile e contiene una copia di backup della configurazione.
- *Flash*, che contiene una copia del Cisco IOS (Internetwork Operating System).
- *ROM*, che contiene il programma di inizializzazione e bootstrap.

Come accedere all'IOS.
IOS è un sistema operativo per i router derivato da BSD UNIX e progettato da Cisco per diverse piattaforme. È statico, proprietario e ha un'interazione basata su CLI (esiste un'interfaccia grafica ma è poco usata). L'accesso all'IOS può avvenire:

- Tramite la *porta console* collegando un PC.
- Tramite la *porta AUX*.
- Da remoto con *Telnet*, ma solo se il dispositivo possiede già un indirizzo IP.

Modalità di accesso.
Il primo livello di accesso che si presenta quando si effettua il login sul router è detto *User EXEC Mode*. Tale modalità permette di dare una serie di comandi non distruttivi per esaminare performance e visualizzare informazioni di sistema. Il secondo livello di accesso è detto *Privileged EXEC Mode* e permette, oltre a tutti i comandi precedenti, anche comandi di configurazione e di debug.

Comandi principali.

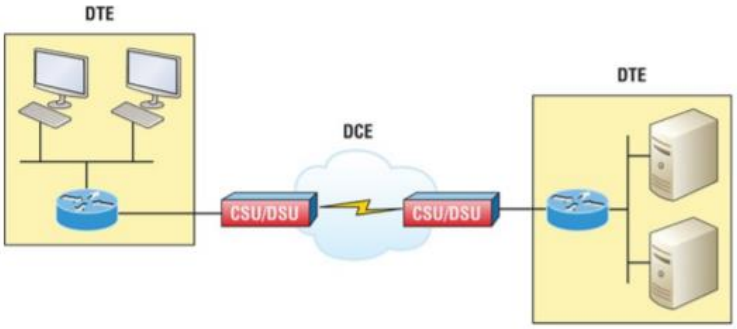
- *enable*: permette di entrare in Privileged EXEC Mode.
- *?*: permette di ottenere un help.
- *no*: permette di negare un comando.
- *configure terminal*: permette di entrare in Configuration Mode per configurare un router.
- *show interface*: per mostrare le interfacce del router.
- *end*: permette di uscire dalla modalità di configurazione/accesso.

Interfacce seriali.
Per il collegamento di due router attraverso le interfacce seriali occorre prestare attenzione. L'interfaccia seriale del router viene denominata *DTE* (Data Terminal Equipment) e deve essere collegata ad un dispositivo *DCE* (Data Communications Equipment) tramite un cavo di tipo DTE/DCE. Il dispositivo DCE riceve il segnale digitale trasmesso dal DTE e lo trasforma in un segnale (analogico o digitale) da trasmettere sulla linea di comunicazione fornita dall'ISP.

Dispositivi DCE.
Il DCE può essere:

- Un *modem analogico*, nel caso in cui la connettività fornita dall'ISP si basi su linee che utilizzano un segnale di tipo analogico.
- Un *dispositivo CSU/CDU* (ossia un terminal adapter digitale), nel caso di linee di comunicazione basate su un segnale di tipo digitale.

In assenza di modem analogico o CSU/DSU, si può usare un apposito cavo (*DTE/DCE crossover cable*) avente due estremità in cui una fa da DTE e una fa da DCE. L'interfaccia del router a cui è collegata l'estremità di tipo DCE dovrà anche fornire il clock.



❖ Configurazione di una rete con NetSimK

Switching di base.
Per collegare un PC ad uno switch (modello 2950 24+2) bisogna usare un cavo di rete Ethernet *Straight through*. Per collegare tra loro due switch, invece, bisogna controllare che essi siano dotati della funzionalità di autosensing che permette di adattare la modalità di trasmissione. In tal caso, per il collegamento non serve il cavo incrociato. Negli switch senza questa funzionalità, come nel caso del modello 2950 24+2, i collegamenti tra switch dovranno essere fatti con cavi Ethernet *Crossover*.

Configurazione di uno switch.
Per configurare uno switch, è necessario collegarsi tramite PC alla sua porta seriale (COM) con un cavo *Console*. Fatto questo sul PC collegato allo switch basterà aprire HyperTerminal-PE per iniziare a dare i comandi.

Comandi importanti.
Tra i comandi principali troviamo:

- *show version*, per ricevere informazioni sul modello dell'apparecchio, la versione del firmware, ecc.
- *show interface*, per vedere lo stato delle interfacce dello switch.
- *enable*, per entrare in modalità privilegiata.

I comandi più importanti della modalità privilegiata sono:

- *show running-config*, per vedere la configurazione che sta girando sullo switch.
- *copy running-config*, per salvare la configurazione corretta nella memoria flash dello switch.
- *configure terminal*, per entrare in modalità configurazione terminale.

I comandi più importanti della modalità configurazione terminale sono:

- *hostname*, per cambiare il nome dell'apparecchio.
- *interface*, per entrare in modalità configurazione interfaccia.

I comandi più importanti della modalità configurazione interfaccia sono:

- *ip address*, per assegnare l'indirizzo IP e la maschera all'interfaccia.
- *no shutdown*, per abilitare l'interfaccia.

Assegnare un indirizzo IP ad uno switch.

Un *indirizzo IP* assegnato ad uno switch serve solo per management. Ad esempio, infatti, possiamo usarlo per collegarci allo switch tramite Telnet con lo scopo di modificare la sua configurazione. Ovviamente, non è necessario configurare il *gateway* in quanto non è necessario per comunicare nella stessa VLAN.

```
enable
configure terminal
interface vlan1
ip address 192.168.0.254 255.255.255.0
no shutdown
```

VLAN base.

Un'infrastruttura di rete può essere segmentata facendo uso di Virtual LAN. Ad ogni VLAN dovrà quindi corrispondere una sottorete IP diversa. Una volta che è stato assegnato un nome alla VLAN, è necessario assegnare alla stessa le interfacce dello switch a cui sono collegati i PC.

```
enable
configure terminal
hostname SW1
vlan 10
name studenti
exit
interface F0/1
switchport access vlan 10
exit
exit
show vlan
```

Porte di trunk.

Se si prova a fare un ping tra PC appartenenti a due segmenti di rete separati si può notare che i PC di un segmento pingano solo i PC del proprio segmento. Infatti, siccome i due segmenti di rete sono su due switch diversi, non possono comunicare tra di loro. Per risolvere il problema si usano le *porte di trunk*, che servono tipicamente a collegare fra loro due switch in modo che una o più VLAN possano estendersi su di essi.

```
enable
configure terminal
interface F0/3
switchport mode trunk
```

Intra-VLAN routing.

Per poter far comunicare due o più VLAN tra di loro, è necessario aggiungere un *router* (modello 2620) che metta in comunicazione i segmenti di VLAN. Per il collegamento tra il router e lo switch bisogna usare un cavo *Straight through*. Prima di programmare il router in modo che faccia routing tra le due VLAN, è necessario configurare la porta dello switch che si collega al router come *porta di trunk*. Per il *routing* si deve: creare due sotto-interfacce dell'interfaccia a cui è collegato il trunk proveniente dallo switch; specificare quale protocollo di incapsulamento utilizzare e l'id della VLAN associata; impostare gli IP delle sotto-interfacce in modo tale che facciano da default gateway per le due VLAN.

```
enable
configure terminal
interface FastEthernet0/0.1
encapsulation dot1q 10
ipaddress 192.168.0.1 255.255.255.0
exit
interface FastEthernet0/0.2
encapsulation dot1q 20
ipaddress 192.168.1.1 255.255.255.0
exit
interface FastEthernet0/0
no shutdown
```

DHCP.

Un *server DHCP* serve per l'assegnazione automatica degli indirizzi IP. Un router può essere configurato in modo che funga da server DHCP. Siccome le reti VLAN hanno uno spazio di indirizzamento separato è necessario creare diversi *pool* da cui ogni rete riceverà poi gli indirizzi IP. L'identificazione della VLAN è trasparente perché ogni sotto-interfaccia del router ha come indirizzo IP un indirizzo dello specifico pool per quella VLAN. Una volta che tutti i pool sono stati creati possiamo togliere l'indirizzamento statico ai PC e abilitare l'indirizzamento dinamico tramite DHCP. Per configurare la durata di assegnamento degli indirizzi IP usare il comando *lease*, il quale accetta fino a 3 numeri (giorni, ore, minuti).

```
enable
configure terminal
ip dhcp pool studenti
network 192.168.0.0 255.255.255.0
default-router 192.168.0.1
lease 7
```

Routing.

Se si vogliono gestire collegamenti tra più reti bisogna metterle in comunicazione aggiungendo un router per ogni nuova rete creata e collegando i router tra di loro con dei cavi WAN. Siccome tali cavi sono del tipo DTE/DCE è necessario scoprire in quale lato del cavo si trova il *DCE* per poter impostare il clock sulla corrispondente interfaccia del router. Per garantire il funzionamento del routing si deve anche abilitare il *protocollo RIP* e aggiungere alla tabella di routing la lista delle reti collegate al router stesso.

```
enable
configure terminal
interface S0/0
ip address 10.0.0.1 255.255.255.0
no shutdown
exit
exit
show controllers S0/0

configure terminal
interface S0/0
clock rate 64000
exit
exit

configure terminal
router rip
network ...
...
exit
exit
show ip route
```

NAT.

Il *NAT* (Network Address Translation) consente di mappare tutti i PC di una sottorete in modo che i loro pacchetti risultino tutti provenienti dallo stesso indirizzo IP pubblico. Storicamente il NAT si è affermato come mezzo per ovviare alla *scarsità di indirizzi IP* pubblici disponibili. Come prima cosa occorre configurare l'interfaccia del router che è rivolta all'interno del NAT e, successivamente, si dovrà configurare l'interfaccia del router che funge da interfaccia pubblica.

```
enable
configure terminal
interface F0/0
ipaddress 192.168.6.1 255.255.255.0
ip nat inside
no shutdown
exit
```

```
interface F0/1
ip address 10.10.10.1 255.255.255.0
ip nat outside
no shutdown
exit

ip nat pool servizi 10.10.10.1
10.10.10.1 prefix-length 24
access-list 1 permit 192.168.6.0
0.0.0.255
ip nat inside source 1 pool servizi
overload

router rip
network 10.10.10.0
```

Port forwarding.

Il *port forwarding* è l'operazione che permette il trasferimento dei dati da un computer ad un altro tramite una specifica porta di comunicazione. Questa tecnica può essere usata per permettere ad un utente esterno di raggiungere un host con indirizzo IP privato (all'interno di una LAN) mediante una porta dell'IP pubblico (NAT) dello stesso. Per vedere su quale porta è stato "nattato" un host basta dare il comando *show ip nat translations*.

PROGRAMMAZIONE DI RETE

❖ Strumenti di analisi della rete

Analizzatori di rete.
Esistono *strumenti software* che consentono di analizzare i pacchetti che arrivano alla propria interfaccia di rete (ad es. Tcpdump, Wireshark). Questi tool di analisi si basano tutti sulla libreria C chiamata *libpcap*, le cui funzioni principali sono la possibilità di cercare e trovare interfacce di rete, gestire potenti filtri di cattura, analizzare ciascun pacchetto, gestire errori e statistiche di cattura. Per poter utilizzare le funzionalità di cattura di questi tool è meglio essere autenticati come utente root (amministratore).

Sniffing in reti non-switched.
Nelle reti ethernet *non-switched* (ad es. Wi-Fi) il mezzo trasmissivo è condiviso e quindi tutte le schede di rete dei computer nella rete locale ricevono tutti i pacchetti, anche quelli destinati ad altri, selezionando i propri a seconda dell'indirizzo MAC (indirizzo hardware specifico della scheda di rete). Lo *sniffing* in questo caso consiste nell'impostare sull'interfaccia di rete la cosiddetta modalità promiscua, che disattiva il "filtro hardware" basato sul MAC permettendo al sistema l'ascolto di tutto il traffico passante sul cavo.

Sniffing in reti ethernet switched.
In questo caso l'apparato centrale della rete, definito switch, si preoccupa di inoltrare su ciascuna porta solo il traffico destinato ai dispositivi collegati a quella porta. Ciascuna interfaccia di rete riceve quindi solo i pacchetti destinati al proprio indirizzo, i pacchetti multicast e quelli broadcast. L'impostazione della modalità promiscua è quindi insufficiente per poter intercettare tutto il traffico in una rete gestita da switch. Un metodo per poter ricevere tutto il traffico sullo switch da una porta qualunque è il *MAC flooding*. Tale tecnica consiste nell'inviare ad uno switch pacchetti appositamente costruiti per riempire la tabella MAC/porta dello switch di indirizzi MAC fittizi. Questo attacco costringe lo switch ad entrare in una condizione detta di fail open che lo fa comportare come un hub, inviando così gli stessi dati a tutti gli apparati ad esso collegati.

Caratteristiche di Wireshark.
- I dati possono essere acquisiti direttamente dall'*interfaccia di rete* oppure possono essere letti da un *file di cattura* precedente.
- I dati possono essere catturati dal vivo da reti Ethernet, Wi-Fi, ADSL, ecc.
- I dati di rete catturati possono essere esplorati nelle loro parti tramite un'*interfaccia grafica*.
- I *filtri di cattura* possono essere usati per catturare solo determinati pacchetti quando la quantità di pacchetti che passano sul tratto di rete osservato è molto alta.
- I *filtri di visualizzazione* possono essere usati per colorare o visualizzare selettivamente le informazioni sommarie sui pacchetti.
- I protocolli di comunicazione possono essere scomposti, in quanto Wireshark riesce a "comprendere" la struttura dei diversi protocolli di rete e quindi è in grado di visualizzare *incapsulamenti* e *campi singoli* (in versione binaria) interpretandone il significato.
- È possibile studiare le statistiche di una *connessione TCP* ed estrarne il contenuto.

Visualizzazione del livello datalink.
Mentre il driver della scheda ethernet fornisce a Wireshark l'esatto header della PDU di livello datalink, altre interfacce di rete non ethernet (ad es. Wi-Fi) potrebbero non farlo a meno che non si utilizzino plugin specifici per Wireshark o per la scheda di rete. Nel caso in cui tale header non venga fornito, Wireshark visualizza un header ethernet "autocostruito" con la dicitura *linux cooked capture*.

Comando ping.

Il comando *ping* è un semplice strumento, molto usato nell'amministrazione delle reti, che verifica la raggiungibilità di un computer connesso alla rete e il relativo *RTT* (Round Trip Time), ossia il tempo che passa dall'invio del pacchetto verso il computer bersaglio al ritorno della risposta. Il comando ping utilizza il protocollo *ICMP* (Internet Control Message Protocol), ovvero un protocollo di servizio del protocollo IP che trasmette: informazioni riguardanti malfunzionamenti (ad es. TimeExceeded, Destination Unreachable); informazioni di controllo (ad es. Echo Request, Echo Reply); messaggi tra i vari componenti di una rete.

Funzionamento e problemi di ping.

Il funzionamento di ping prevede che un pacchetto ICMP venga inviato, con il messaggio di *Echo Request*, verso l'indirizzo IP o il nome DNS della destinazione. Se la destinazione è attiva, il software ICMP sull'host o sul router risponde con un messaggio di *Echo Reply* che trasporta gli stessi dati del messaggio di richiesta. La natura di ping, secondo cui il destinatario risponde automaticamente all'Echo Request con il messaggio di Echo Reply, lo ha reso uno strumento molto utilizzato per *attacchi informatici* di tipo DoS (Denial-of-Service) e DDoS (Distributed Denial-of-Service). Per esempio, se si dispone di una grande quantità di banda, è possibile effettuare un attacco chiamato *ping flood* che consiste nell'invio continuato di pacchetti che può portare alla congestione della rete del destinatario, rendendolo quindi inaccessibile dagli altri utenti. Per questo motivo e per evitare di mostrare al pubblico un'interfaccia di rete, molti amministratori di sistema bloccano i messaggi Echo Request in entrata a livello di firewall oppure ne disabilitano la gestione a livello di singola interfaccia. In questi casi il comando ping non funzionerà.

Comando tracert.

Il commando *tracert* è un semplice strumento per scoprire il percorso che un pacchetto segue dalla sorgente alla destinazione. Il comando mostra un elenco di tutte le interfacce dei router che il pacchetto attraversa finché raggiunge la destinazione. Per capire il *funzionamento* di questo strumento occorre ricordare che ogni pacchetto IP contiene un campo TTL (Time To Live) con un valore intero che viene decrementato ogni volta che incontra un router. Quando un router trova TTL=1 scarta il pacchetto e invia all'IP sorgente il messaggio ICMP TIME-TO-LIVE_EXCEEDED mettendo come IP sorgente sé stesso e quindi rivelando la propria identità.

Pacchetti sonda.

Il comando tracert invia quindi una sequenza di *pacchetti sonda* verso la destinazione con TTL con valori crescenti a partire da 1, facendo generare via via messaggi ICMP TIME-TO-LIVE_EXCEEDED alle interfacce dei router attraversati e in questo modo viene a conoscenza del percorso del pacchetto. Quando un pacchetto sonda raggiungerà la destinazione finale, questa invierà indietro un pacchetto ICMP diverso da TIME-TO-LIVE_EXCEEDED e il processo terminerà. Tuttavia, certe interfacce di router non spediscono indietro il messaggio ICMP TIME-TO-LIVE_EXCEEDED come scelta dei loro amministratori per evitare di svelare la topologia di rete ai malware. In tal caso tracert non potrà mostrare tali passi del percorso e perciò li indicherà con degli asterischi.

```
bob@ubuntu-comp:/home$ traceroute www.google.com
traceroute to www.google.com (173.194.116.145), 30 hops max, 60 byte packets
 1  speedtouch.lan (192.168.1.1)  12.669 ms  11.902 ms  11.053 ms
 2  78.134.144.1-dsl.net.metronet.hr (78.134.144.1)  13.987 ms  15.648 ms  17.37
1 ms
 3  10.50.0.73 (10.50.0.73)  22.473 ms  23.213 ms  26.523 ms
 4  10.50.0.74 (10.50.0.74)  29.124 ms  30.102 ms  34.318 ms
 5  213.147.96.110 (213.147.96.110)  35.266 ms  38.136 ms  39.979 ms
 6  212.162.29.1 (212.162.29.1)  41.924 ms  36.542 ms  38.233 ms
 7  ae-2-3.bar1.Ljubljana1.Level3.net (4.69.151.233)  36.919 ms  13.695 ms  15.1
55 ms
 8  * * *
 9  * * *
10  ae-3-80.edge3.Frankfurt1.Level3.net (4.69.154.135)  81.414 ms ae-4-90.edge3.
Frankfurt1.Level3.net (4.69.154.199)  39.745 ms ae-1-60.edge3.Frankfurt1.Level3.
net (4.69.154.7)  42.889 ms
11  4.68.70.186 (4.68.70.186)  45.590 ms  48.088 ms  50.605 ms
12  209.85.240.64 (209.85.240.64)  65.602 ms  55.049 ms  57.458 ms
13  66.249.94.69 (66.249.94.69)  60.435 ms  23.370 ms  23.618 ms
14  173.194.116.145 (173.194.116.145)  24.013 ms  23.452 ms  23.585 ms
```

Comando nslookup.
È uno strumento presente in tutti i sistemi operativi che utilizzano il protocollo TCP/IP. Il comando *nslookup* consente di effettuare interrogazioni ai server DNS per poter ottenere da un hostname il relativo indirizzo IP o viceversa. Si può utilizzare in due modi:

- *Interattivo*, che permette di effettuare più query e visualizza i singoli risultati (viene abilitato in maniera automatica quando il comando non è seguito da alcun argomento).
- *Non interattivo*, che permette di effettuare una sola query e ovviamente visualizza il risultato della singola query (viene abilitato quando si specifica l'host-to-find).

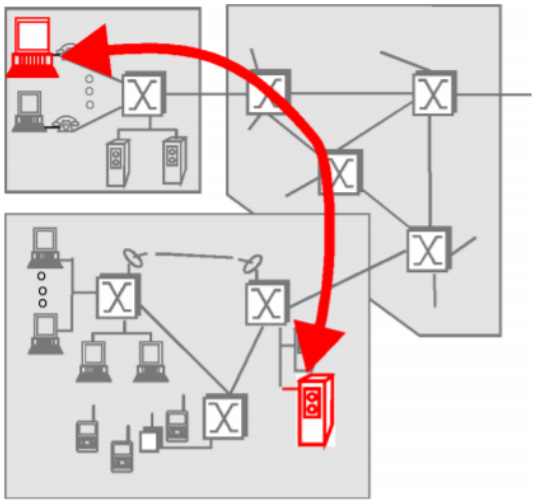
Si ricorda che *DNS* (Domain Name System) è un sistema di server, organizzato gerarchicamente, che serve per la gestione dei namespace. Il compito principale di questo servizio è quello di rispondere alle richieste di risoluzione del nome di un dominio, ovvero la conversione dei nomi di dominio in indirizzi IP.

Comandi ipconfig e route.
Il comando *ipconfig* è utilizzato per configurare e controllare un'interfaccia di rete TCP/IP. L'esecuzione di tale comando con l'opzione */all* mostra a video le informazioni di tutte le interfacce di rete. Tali interfacce possono essere *ethernet*, *wireless* o di *loopback* (ovvero un'interfaccia di rete speciale che il sistema usa per comunicare con sé stesso). Il comando *route*, invece, è utilizzato per vedere e modificare le tabelle di routing. L'esecuzione di tale comando con l'opzione *PRINT* permette di visualizzare la tabella di routing di un host.

Comando whois.
Whois è un protocollo di rete che consente, mediante l'interrogazione di appositi database server da parte di un client, di stabilire il nome del privato, azienda o ente al quale è intestato un determinato indirizzo IP o uno specifico dominio DNS. Nel whois vengono solitamente mostrate anche informazioni riguardanti l'intestatario, la data di registrazione e la data di scadenza. Whois si può consultare tradizionalmente da *riga di comando*, anche se ora esistono numerosi *siti web* che permettono di consultare gli archivi dove sono contenute tali informazioni.

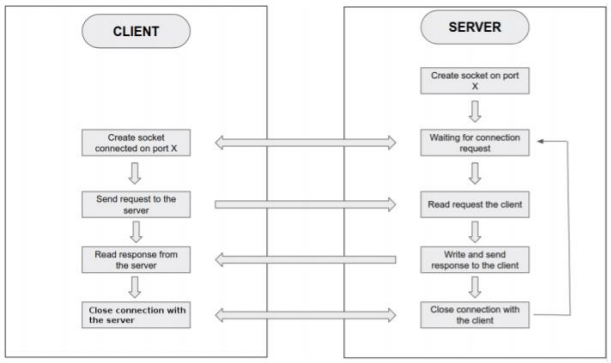
❖ Socket in Java

Introduzione.
I *socket* sono il meccanismo software che permette la gestione di un flusso di dati tra due host posizionati in qualsiasi punto di Internet. I socket possono essere visti come un tubo bidirezionale alle cui estremità è associato un indirizzo IP e un numero di porta di livello trasporto. Quindi un socket è univocamente identificato dalla *5-upla*: (IP_A, IP_B, Porta_A, Porta_B, TCP/UDP). Java mette a disposizione una serie di semplici classi che consentono la scrittura di applicazioni di rete basate sull'utilizzo dei socket.

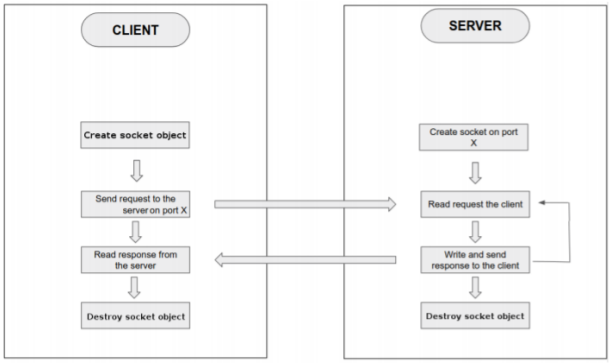


Implementazione del paradigma client/server.

Le applicazioni che utilizzano i socket sono quasi sempre strutturate secondo il paradigma client/server. Quando dobbiamo creare una applicazione client/server bisogna definire quali sono i compiti lato client e lato server. Il *server* è sempre il primo a partire ed attende richieste dal client, le soddisfa e torna ad attendere. Il *client* manda richieste al server e per farlo deve conoscere il nome o l'indirizzo IP dell'interfaccia di rete su cui il server è in esecuzione e il numero della porta TCP o UDP sulla quale il server è in ascolto. I *protocolli* coinvolti nell'implementazione dei socket sono quelli del livello di trasporto, ovvero UDP (User Datagram Protocol) e TCP (Transmission Control Protocol).



Paradigma orientato alla connessione (TCP)



Paradigma non orientato alla connessione (UDP)

Package di Java.

In Java le classi per usare i socket sono contenute nel package *java.net*, il quale fornisce interfacce e classi per l'implementazione di applicazioni di rete. Questo package definisce fondamentalmente:

- Le classi *Socket* e *ServerSocket* per le connessioni TCP.
- La classe *DatagramSocket* per le connessioni UDP.
- La classe *URL* per le connessioni HTTP.
- La classe *InetAddress* per rappresentare gli indirizzi Internet.
- La classe *URLConnection* per rappresentare le connessioni ad un URL.

Classe DatagramSocket.

La classe *DatagramSocket* implementa trasmissioni orientate ai pacchetti senza connessione mediante UDP. Ogni pacchetto gira attraverso la rete guidato dall'indirizzo IP di destinazione. Diversi pacchetti possono prendere percorsi differenti all'interno della rete e potrebbero arrivare in un ordine differente da quello con cui sono stati spediti. Inoltre, il raggiungimento della destinazione da parte del pacchetto non è nemmeno garantito. È compito dell'applicazione che utilizza UDP (e non del protocollo) gestire questi problemi se necessario. Tuttavia, con l'utilizzo di UDP si ha una corrispondenza uno-a-uno tra:

- *Messaggi spediti* con l'opportuna funzione software.
- *Messaggi ricevuti* chiamando l'opportuna funzione software.
- *PDU in circolazione* nella rete.

Nonostante queste particolarità sembrino degli svantaggi, ci sono in realtà numerosi vantaggi come il fatto che le trasmissioni che usano *DatagramSocket* sono più veloci delle trasmissioni con connessione (data la minor quantità di pacchetti che è necessario inviare).

Classi Socket e ServerSocket.

La classe *Socket* implementa, tramite il protocollo TCP, trasmissioni: affidabili, orientate alle connessioni e basate sul flusso. La classe Socket racchiude la logica del *client*. La classe *ServerSocket*, invece, rappresenta la logica del *server* in ascolto che soddisfa le richieste di connessione su una porta specifica da parte dei client. Quando una connessione viene richiesta, viene creato un oggetto di tipo Socket per gestire la comunicazione.

Classe InetAddress.
La classe *InetAddress* racchiude un indirizzo IP (Internet Protocol). Gli oggetti *InetAddress* sono usati dalle classi che specificano gli indirizzi di destinazione dei pacchetti in uscita nella rete, come *DatagramSocket* e *Socket*. La classe *InetAddress* non fornisce alcun costruttore pubblico e perciò bisognerà utilizzare alcuni metodi statici per poter creare oggetti di questo tipo.

Classe URL.
La classe *URL* rappresenta uno Uniform Resource Locator, ovvero un puntatore verso una risorsa nel web. Questa risorsa può essere un semplice file o cartella, oppure può essere un riferimento ad un oggetto più complicato come una query su un database o su un motore di ricerca. La classe *URL* fornisce metodi per creare un URL e accedere alla risorsa specificata. Un URL assoluto è formato tipicamente da: un protocollo, un hostname, un numero di porta e un filename.

Classe URLConnection.
La classe *URLConnection* è una classe astratta che è superclasse di tutte le classi che rappresentano una comunicazione tra un'applicazione e un URL. I metodi di questa classe possono essere usati sia per leggere che per scrivere sulla risorsa a cui fa riferimento l'URL. In generale, creare una connessione ad un URL richiede i seguenti passaggi:

- L'oggetto "connessione" è generato invocando il metodo *openConnection()* su un URL.
- La connessione effettiva all'oggetto remoto è fatta usando il metodo *connection()* sull'oggetto ritornato al punto precedente.
- L'oggetto remoto diventa disponibile e i suoi campi diventano accessibili.

Simulazione della perdita di pacchetti.
Nei sistemi operativi Linux esiste un servizio di firewall, chiamato *IPTABLES*, che serve per bloccare alcuni pacchetti simulando un guasto di rete. Questo firewall consente di specificare delle *regole* sul traffico di rete:

- Blocco della ricezione su una porta specifica (sudo iptables -A INPUT -p … --dport … -j DROP).
- Sblocco della ricezione su una porta specifica (sudo iptables -D INPUT -p … --dport … -j DROP).

Differenze tra programmi UDP e TCP.
UDP e TCP sono protocolli di livello trasporto dell'architettura di rete Internet. La differenza tra UDP e TCP è che UDP è un protocollo di tipo *connection-less* (ovvero senza connessione) e quindi non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi; il TCP, invece, è un protocollo di tipo *connection-oriented* che si occupa del controllo di trasmissione rendendo affidabile la comunicazione (che è inaffidabile a livello IP) e fornendo un servizio di ritrasmissione automatica di pacchetti persi o corrotti, oltre ad un servizio di ricezione basato su acknowledge (ACK) per segnalare quando una sequenza di dati è stata ricevuta. Sebbene il protocollo TCP permetta di recuperare automaticamente la trasmissione dopo un blocco, questa funzionalità impiega uno scambio di pacchetti molto maggiore rispetto a UDP. Il protocollo UDP, infatti, è da preferire al protocollo TCP quando non è essenziale avere un sistema di ritrasmissione oppure quando l'acknowledge del pacchetto inviato è implicito nella risposta ottenuta a livello di applicazione. Inoltre, la complessità di scrittura di programmi con protocollo UDP è ridotta nel numero di oggetti da istanziare per la comunicazione.

I/O stream tramite socket TCP.
Siccome il socket TCP permette di instaurare una connessione affidabile e byte-oriented, in molti linguaggi di programmazione per scambiare dati mediante esso viene utilizzato il modello a stream come per i file. In particolare, in Java si utilizzano le classi *InputStream* e *OutputStream*. Ciò che avviene è che l'output stream associato ad un socket nel processo A invia tramite la rete dei dati che verranno ricevuti dall'input stream dello stesso socket visto dal processo B. Quindi nel caso di TCP, a differenza di quanto accade con UDP, i

gruppi di byte scritti con l'opportuna funzione software, i gruppi di byte letti con l'opportuna funzione software e le PDU che circolano in rete non sono correlati tra loro. Infatti, ad esempio, posso decidere di scrivere 500 byte con un'unica chiamata di scrittura, leggerli un byte alla volta con 500 chiamate di lettura e magari in rete tale sequenza viene spezzata in due PDU da 200 e 300 byte rispettivamente.

Thread in Java.

Java, al contrario di molti altri linguaggi, fornisce un supporto incorporato per la programmazione multithread. Un programma multithread contiene parti che possono essere eseguite contemporaneamente. Ogni parte di un programma di questo tipo viene definita *thread* e ogni thread definisce un percorso separato di esecuzione. Questo significa che un singolo programma può eseguire contemporaneamente due o più attività (parallele) a differenza di un programma puramente ad esecuzione sequenziale. Esistono due modalità per implementare le thread in Java. La prima modalità consiste nell'estendere la classe *Thread* per:

- Ridefinire il suo metodo *run()* specificando le istruzioni che vogliamo eseguire parallelamente.
- Creare un'istanza di questa sottoclasse.
- Eseguire la thread andando a chiamare il metodo *start()* sull'istanza appena creata, il quale andrà implicitamente ad eseguire il metodo *run()* definito precedentemente (non si deve avviare una thread chiamando esplicitamente il metodo *run()*).

```
public class MyClass extends Thread {

     public MyClass () {
         // costruttore ...
     }
     public void run () {
         // esecuzione parallela ...
     }
}

public class TestMain {
     public static void main (String[] args) {
         new MyClass().start();
     }
}
```

La seconda modalità consiste nell'implementare l'interfaccia *Runnable* per:

- Ridefinire il suo metodo *run()* specificando le istruzioni che vogliamo eseguire parallelamente.
- Creare un'istanza di questa classe.
- Creare un'istanza della classe *Thread* e passargli come parametro l'istanza della classe creata prima.
- Invocare il metodo *start()* sull'oggetto *Thread* appena creato.

```
public class MyClass implements Runnable {

     public MyClass () {
         // costruttore ...
     }

     public void run () {
         // esecuzione parallela
     }
}

public class TestMain {
     public static void main (String[] args) {
         MyClass c = new MyClass();
         Thread t = new Thread(c);
         t.start();
     }
}
```

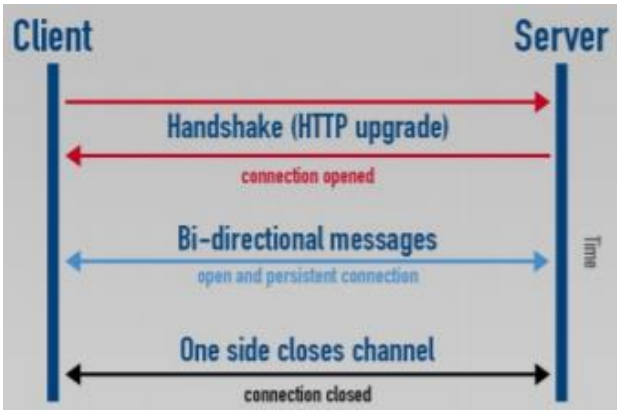
Il primo meccanismo è molto facile da utilizzare, ma è limitato dal fatto che la classe creata debba essere una sottoclasse di *Thread*. Il secondo meccanismo, invece, è più generale e flessibile perché la classe creata può ereditare da una classe diversa da *Thread*.

❖ WebSocket

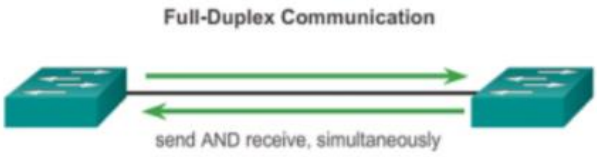
Introduzione.
Il *WebSocket* è un protocollo di comunicazione web che fornisce un canale di comunicazione bidirezionale attraverso una singola connessione TCP inizialmente utilizzata per il protocollo HTTP. Il protocollo WebSocket permette maggiore interazione tra un browser e un server, facilitando la realizzazione di applicazioni web che devono fornire contenuti in tempo reale. Questo è reso possibile poiché i WebSocket, a differenza del tradizionale protocollo HTTP, permettono al server la possibilità di "prendere l'iniziativa" ed effettuare dei PUSH autonomi di dati verso il browser per aggiornarlo.

Comunicazione bidirezionale.
Un sistema bidirezionale (detto anche *full-duplex*) permette la comunicazione in entrambe le direzioni simultaneamente. Una buona analogia per il full-duplex potrebbe essere una strada a due corsie con una corsia per ogni direzione. Una connessione TCP è bidirezionale non solo come direzione dei dati ma anche come libertà di entrambi gli agenti coinvolti di trasmettere per primi. Questa possibilità è poi persa nel protocollo HTTP dove è sempre il client a fare il primo passo. Con i WebSocket, invece, si vuole far ritornare "paritaria" la connessione usata dall'HTTP come se fosse una connessione TCP di tipo base, col vantaggio che è stata instaurata inizialmente per l'HTTP e quindi è compatibile con i firewall presenti su Internet.

Upgrade request.
I WebSocket sono basati sul protocollo TCP e nascono da una connessione HTTP attraverso una *upgrade request* verso il server. Per permettere una compatibilità iniziale tra browser e server, si utilizza l'HTTP upgrade header in cui viene richiesto di passare dal protocollo HTTP a quello WebSocket. Il protocollo HTTP fornisce un meccanismo speciale che permette di stabilire un upgrade del protocollo da usare durante la connessione. Questo meccanismo può essere inizializzato solo dal client, mentre il server, se è in grado di fornire il servizio col protocollo stabilito dal client, decide se accettare o meno questo cambio di protocollo.



Protocollo WebSocket



Sistema di comunicazione bidirezionale

Interazione con gli apparati di rete.
Le connessioni WebSocket utilizzano porte HTTP standard (80 e 443). Pertanto, i WebSocket non richiedono l'apertura di nuove porte sulle reti, e questo li rende compatibili con le impostazioni di sicurezza dei *firewall* e dei *proxy* e con il meccanismo del *NAT*. A differenza del normale traffico HTTP, che utilizza un protocollo di richiesta/risposta, le connessioni WebSocket possono rimanere aperte per un lungo periodo e permettono uno scambio paritetico di dati tra browser e server. Ma i firewall, i proxy e il NAT eventualmente interposti continuano a consentire questo tipo di comunicazioni.

Limitazioni.
I WebSocket non rappresentano la soluzione a tutto. L'HTTP riveste ancora un ruolo chiave nella comunicazione tra browser e server come mezzo per aprire e chiudere connessioni per trasferimenti di dati di tipo one-time, come i caricamenti iniziali. Le richieste HTTP sono in grado di eseguire questo tipo di operazioni in modo più efficiente dei WebSocket, chiudendo le connessioni una volta utilizzate invece di mantenerne lo stato. Inoltre, i WebSocket possono essere utilizzati solo se gli utenti dispongono dei moderni browser con *JavaScript* abilitato, e questo potrebbe non essere vero in caso di sistemi embedded. Dovrebbero poi essere considerati i possibili impatti sull'architettura di rete in quanto WebSocket, essendo una connessione persistente, potrebbe richiedere molte più risorse rispetto ad un server web standard.

Applicazioni client/server.
Quando parliamo del *client* intendiamo l'insieme costituito da browser e JavaScript, mentre quando parliamo del *server* intendiamo un framework per la realizzazione di applicazioni web, come ad esempio la piattaforma Node.js basata anch'essa su JavaScript. La caratteristica principale di *Node.js* risiede nella possibilità di accedere alle risorse del sistema operativo in modalità event-driven, invece che sfruttando il classico modello basato su processi o thread concorrenti utilizzato dai classici web server.

Approccio asincrono.
Il modello *event-driven*, o "programmazione ad eventi", si basa su un concetto piuttosto semplice: si esegue un'azione quando accade qualcosa. Ogni azione risulta quindi *asincrona*, a differenza dello stile di programmazione tradizionale in cui un'azione succede ad un'altra solo dopo che la prima è stata completata. Ciò dovrebbe garantire una certa efficienza delle applicazioni grazie ad un sistema di callback gestito a basso livello dal *runtime engine*. L'efficienza deriva dal considerare che le azioni tipicamente effettuate riguardano il networking, ambito nel quale capita spesso di lanciare richieste e di rimanere in attesa di risposte che arrivano con tempi che, paragonati ai tempi del sistema operativo, sono molto più alti. Grazie al comportamento asincrono, durante le attese di una certa azione il runtime engine può gestire qualcos'altro, come ad esempio la logica applicativa.

Funzionamento del server web.
- Creazione del server web in ascolto, tramite le funzioni di una libreria di Node.js chiamata *Express*.
- Implementazione del protocollo WebSocket, tramite una libreria di Node.js chiamata *Socket.IO* che racchiude numerose funzioni come il broadcasting a tutti i socket collegati, il salvataggio dei dati riguardanti ciascun utente e l'approccio asincrono di I/O.
- Dopo la creazione del socket, il *server* si mette in attesa delle connessioni HTTP da parte dei client e, quando una viene aperta, effettua l'upgrade a WebSocket.
- Successivamente il server si mette in ascolto del prossimo *evento* che richiederà il suo intervento.

❖ Web Service REST

Architetture orientate ai servizi.
L'architettura orientata ai servizi (SOA, Service Oriented Architecture) definisce un nuovo modello logico per sviluppare il software. Tale modello è realizzato dai *Web Service*, ovvero moduli software distribuiti che collaborano attraverso un canale web. Il Web Service è una funzionalità messa a disposizione in modalità server ad altri moduli software implementati come client. I componenti software interagiscono tra loro attraverso protocolli Web Service (ad es. REST e SOAP) trasportati in connessioni HTTP. È compito del protocollo Web Service definire i parametri che devono essere passati e i valori che devono essere restituiti. Quando uno sviluppatore crea un Web Service si deve preoccupare di definire: la *logica di funzionamento* del servizio e il *web container* su cui verrà installato per consentirne l'uso da parte dei client.

Vantaggi tecnologici.

- *Software come servizio*: al contrario del software tradizionale, una collezione di metodi esposta tramite Web Service può essere utilizzata come un servizio accessibile da qualsiasi client.
- *Interoperabilità*: i Web Service consentono l'incapsulamento, ovvero i componenti possono essere isolati in modo tale che solo lo strato relativo al servizio vero e proprio sia esposto all'esterno. La logica applicativa incapsulata all'interno dei Web Service, invece, è completamente decentralizzata ed è accessibile attraverso Internet da piattaforme, dispositivi, sistemi operativi e linguaggi di programmazione differenti (indipendenza dall'implementazione). Tale logica rimane "lontano" da chi la utilizza e non può essere manomessa o rubata (sicurezza del sistema interno).
- *Semplicità di sviluppo e di rilascio*: un'applicazione è costituita da moduli indipendenti che interagiscono tramite la rete per semplificare lo sviluppo. Rilasciare un Web Service, inoltre, significa solo esporlo al web.
- *Semplicità di installazione*: la comunicazione avviene grazie allo scambio di informazioni in forma testuale all'interno del protocollo HTTP usato per il web e utilizzabile praticamente su tutte le piattaforme hardware/software. I messaggi testuali sono quindi compatibili con firewall e NAT. Inoltre, la comunicazione tra due sistemi non deve essere preceduta da noiose configurazioni ed accordi tra le parti.
- *Standard*: i concetti fondamentali che stanno dietro ai Web Service sono regolati da standard approvati dalle più grandi ed importanti società ed enti d'Information Technology al mondo.

Motivazioni.

- *Protezione della proprietà intellettuale*: il cuore dell'applicazione rimane sul server e non viene perciò distribuito come eseguibile agli utenti.
- *Requisiti di potenza di calcolo e memoria di massa*: la richiesta di risorse da parte dell'applicazione può essere soddisfatta dal server senza gravare sulle risorse del client (CPU, consumo energetico, memoria di massa).
- *Comodità di distribuzione agli utenti*: ogni utente usa un software client minimale che si connette al server. In questo modo si possono garantire l'*aggiornamento istantaneo* (la logica applicativa interna al server può cambiare in qualsiasi momento senza dover re-distribuire aggiornamenti agli utenti), lo sviluppo e il mantenimento di *una sola versione del prodotto* (a fronte di molteplici piattaforme utente) e *maggiori ritorni economici* (col nuovo approccio pay-per-use rispetto al tradizionale approccio della distribuzione e installazione dell'applicativo sui PC degli utenti).

REST (REpresentational State Transfer).

È uno stile architetturale che permette ai computer di comunicare tra di loro in modo semplice. In particolar modo possiamo dire che REST:

- *Non è un protocollo*, in quanto non descrive i messaggi esatti (o parti di essi) che i sistemi devono scambiarsi tra di loro ma specifica piuttosto i requisiti (vincoli architetturali) che le parti del sistema devono soddisfare.
- *Non è una specifica*, in quanto a differenza dei Web Service basati sul protocollo SOAP, non esiste uno standard ufficiale per le API Web RESTful; quindi REST non è uno standard in sé, ma le implementazioni REST fanno uso di standard come HTTP, URI, JSON e XML.
- *Non è per forza legato ad HTTP*, in quanto nei vincoli REST non c'è nulla che rende obbligatorio l'uso di HTTP come protocollo di trasferimento, ed è quindi perfettamente possibile utilizzare altri protocolli di trasferimento come SMTP e persino altri non basati sull'architettura TCP/IP.

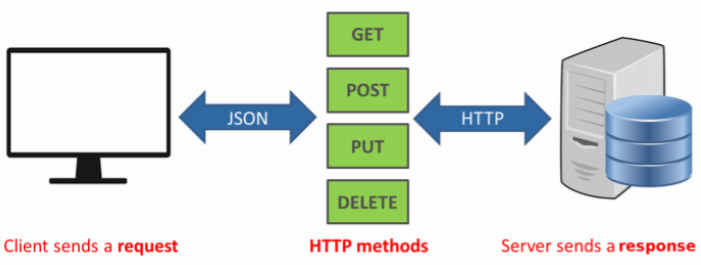
Principi.
Un servizio web che utilizza i metodi HTTP e implementa i principi REST viene chiamato *RESTful*. I principi REST sono: *architettura client/server* e *comunicazione stateless*. Nell'architettura REST chi utilizza il servizio agisce con la modalità del *client* che fa il primo passo richiedendo qualcosa alla controparte *server* che si trova già precedentemente in ascolto e che fornisce una risposta. L'unico requisito è che entrambi gli attori (client e server) sappiano in che formato scambiarsi i messaggi. È importante precisare che il client è un programma scritto ad-hoc per utilizzare un certo servizio e non uno strumento di uso generale come un web browser. In altre parole, client e server di una architettura REST sono due moduli di un unico programma che comunicano attraverso la rete in un canale web. La comunicazione tra client e server deve essere *senza stato* tra richieste/risposte successive. Ciò significa che ogni richiesta da parte di un client dovrebbe contenere tutte le informazioni necessarie al server per comprenderne il significato e fornire la risposta. Inoltre, tutti i dati sullo stato della sessione dovrebbero poi essere restituiti al client nella risposta. In breve, ogni richiesta è come se fosse la prima richiesta e non dovrebbe essere correlata ad una precedente richiesta. Se il server non conserva lo stato, è più leggero e può scalare molto più facilmente sul numero di richieste che può servire. Inoltre, il server può comprendere una richiesta senza aver visto quelle precedenti.

Architettura REST.
Nell'architettura REST il client invia una richiesta per creare, recuperare o modificare una risorsa e il server invia una risposta. Una *richiesta* è generalmente composta da:

- *Metodo di richiesta*, che definisce che tipo di operazione si vuole fare; le tipologie di richieste REST sono mappate sui metodi HTTP GET (recupera una specifica risorsa), POST (crea una nuova risorsa), PUT (aggiorna una specifica risorsa) e DELETE (elimina una specifica risorsa).
- *Percorso della risorsa* a cui è indirizzata la richiesta, sottoforma di URI (Uniform Resource Identifier) il quale contiene il nome Internet del server a cui collegarsi seguito dalla stringa che identifica la specifica funzionalità sul server (sarà poi cura del progettista del server far corrispondere l'URI all'opportuna query sul database remoto).
- *Header*, che permette al client di aggiungere informazioni relative alla richiesta e di specificare il tipo di contenuto che è in grado di ricevere dal server, come ad esempio text/html (per ricevere un file di testo contenente HTML), text/plain (per ricevere del testo) o application/json (per ricevere la risposta in formato JSON).
- *Body* contenente dati, necessario nel caso dei metodi POST e PUT con cui si passano dei dati al server.

Una *risposta*, invece, è generalmente composta da:

- *Content type*, che indica al client il tipo di dati inviati nel corpo del messaggio.
- *Response code*, che indica al client se l'operazione richiesta ha avuto successo o meno (200 OK, 404 NOT FOUND, eccetera).



JSON (JavaScript Object Notation).
È un formato semplice per lo scambio di dati. Questo formato testuale permette di aggregare dati (stringhe, numeri, oggetti complessi) per creare informazioni di più alto livello. Il formato JSON è facile da leggere e scrivere per gli esseri umani ed è facile da generare e "parserizzare" dai calcolatori. Esso si basa su un sottoinsieme del linguaggio di programmazione JavaScript. È inoltre un formato di testo completamente indipendente dal linguaggio, ma che usa convenzioni familiari ai programmatori della famiglia di linguaggi C.

Il punto di forza è la serializzazione che permette di passare in modo semplice da un oggetto ad una struttura dati complessa e viceversa (sono definiti metodi che permettono di serializzare/deserializzare il formato JSON per tutti i linguaggi di programmazione più conosciuti). Questo è possibile grazie alla struttura gerarchica che adotta il formato JSON. Queste proprietà rendono JSON un formato ideale per lo scambio di dati, soprattutto in un contesto client/server.

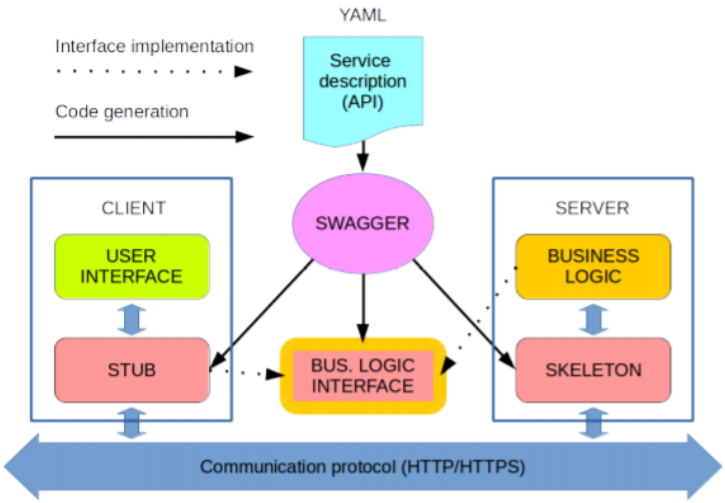
Struttura interna.
La sintassi del linguaggio JSON è costituita dalle seguenti regole:

- L'informazione base è una coppia nome/valore chiamata *dato*.
- Un *oggetto* è descritto all'interno di parentesi graffe e può contenere più dati separati da virgole.
- Un *array* è descritto all'interno di parentesi quadre e può contenere una collezione di oggetti.

Seguendo le regole di questo formato è semplice specificare dati strutturati e facilmente gestibili all'interno del corpo di richieste e risposte HTTP.

Stub e Skeleton.
Si definisce *API RESTful* un insieme di chiamate HTTP che rappresentano l'interfaccia (API, Application Program Interface) ad un insieme di servizi web di tipo REST. Client e server, oltre ad implementare le funzionalità applicative di propria competenza in riferimento all'applicazione concreta, devono anche contenere il codice che implementa chiamate e risposte REST. Il codice lato client viene chiamato *Stub*, mentre il codice lato server viene chiamato *Skeleton*. I codici dipendono della specifica API ma la relazione di dipendenza è formalizzabile rigidamente e quindi è possibile generare automaticamente il codice di Stub e Skeleton evitando noiose e ripetitive operazioni manuali.

Formato YAML.
Il *formato YAML* è utilizzato per le configurazioni dei servizi web e descrive aspetti di serializzazione dei dati simili a quelli di JSON ma con i vantaggi di maggiore leggibilità e supporto ai tipi complessi, ai commenti e ai blocchi di testo. Nel file YAML, le *API RESTful* che devo utilizzare sono descritte in un formato studiato per essere interpretabile correttamente sia dagli esseri umani sia dagli strumenti software di generazione automatica del codice (ad es. Swagger). I vantaggi di questa standardizzazione sono una più obiettiva e condivisibile documentazione delle funzionalità dell'API e la possibilità di generare il codice di Stub e Skeleton sfruttando direttamente le specifiche definite nello schema. Infatti, i metadati presenti nel file forniscono informazioni sufficienti per generare sia lo Stub sia lo Skeleton comprensivo delle URI e della validazione degli input. La generazione dello Stub è vantaggiosa perché rende possibile un adeguamento veloce del codice sorgente del client alle evoluzioni delle API. La generazione dello Skeleton è vantaggiosa in quanto è possibile concentrarsi direttamente sulla programmazione delle funzionalità centrali del servizio erogato (la cosiddetta *business logic*) senza perdersi nei dettagli ripetitivi della gestione delle chiamate REST.



Credits

Basato su slide e dispense fornite dai *proff. Damiano Carra* e *Davide Quaglia*

Vedere anche i seguenti repository GitHub:

- https://github.com/davbianchi/dispense-info-univr/tree/master/triennale
- https://github.com/Xiryl/univr/blob/master/Bachelor/2017-2018/network-security/docs/summary.pdf

Repository GitHub personale: https://github.com/zampierida98/UniVR-informatica

Indirizzo e-mail personale: zampieri.davide@outlook.com