Indice

Capitolo 1

Introduzione

La sicurezza informatica non è mai un fattore a se stante, in quanto per poter parlare di sicurezza bisogna prendere come riferimento anche il sistema informatico che si vuole proteggere. Inoltre non esistono mai delle soluzioni che sono la panacea definitiva, in quanto ogni problema esigerà una risoluzione specifica; inoltre la sicurezza deve ricoprire tutti i livelli, partendo dall'hardware, al sistema operativo, all'utente che utilizza quel dato sistema: infatti la sicurezza di un sistema è determinato dalla tenuta del suo anello più debole, in quanto sarà poi sufficiente concentrarsi su quel punto per ottenere il dominio sull'intero sistema. Si può inoltre notare che questo fenomeno sia anche un fenomeno sociologico e psicologico, ma ci limiteremo all'aspetto informatico.

Restringeremo il nostro campo di studio ai **NIS**, ovvero "Network Information System": è un sistema che non comprende un oggetto unico, ma un insieme di calcolatori collegati all'interno di una rete assieme agli utenti umani che li adoperano (manutentori, configuratori, programmatori, etc.). Il problema è che, oggi giorno, questi sistemi non sono molto affidabili (**trustworthy**). Spesso infatti questi NIS sono cablati con **legacy software** (ovvero software di vecchia data), che per motivi economici viene ancora utilizzato, in quanto sarebbe molto più costoso riscriverne uno nuovo. Questi costituiscono appunto uno dei maggiori punti di debolezza in quanto spesso sono privi di documentazione, o di descrizioni in genere: appaiono dunque delle scatole opache impossibili da decifrare e quindi difficili da manutenere.

In caso di legacy software, l'unico modo per poter effettuare la manutenzione è quindi la creazione di agglomerati software (patch), per rispondere a delle esigenze successive, diventando quindi dei sistemi estremamente complessi da gestire, riducendo così il loro grado di affidabilità.

Altro motivo d'attacco è l'utilizzo di software **off the shelf**, ovvero immediatamente disponibile in commercio: è un software di produzione di massa, e quindi di basso costo. Questo oggetto sarà così diffuso, che sarà molto facile che esistano persone che abbiano scoperto come attaccare questo software: abbassando i costi di acquisizione del software, si abbassa conseguentemente la fiducia che riponiamo nel nostro sistema; caso opposto è rappresentato dal software a produzione "artigianale".

1.1 Tipologie di attacchi

Identity Theft con questo termine ("furto d'identità"), si vuole rubare l'identificazione informatica di un utente, o meglio ci si vuole impossessare dei dati che lo identificano,

in modo da poter effettuare delle operazioni a lui consentite, in quanto appunto utente con particolari privilegi.

Phishing questo termine identifica un particolare furto dell'identità informatica di un individuo che provvede a fornire i dati direttamente al malintenzionato, in quanto quest'ultimo si è camuffato come ente nel quale l'utente ripone fiducia.

Cyberextortion questo termine indica una minaccia perpetrata da un criminale informatico nei confronti di un ente, allo scopo di intimorire l'ultimo per ricevere un beneficio

Cyberwarfare questo termine indica una guerra di tipo tecnologica, che può avvenire quindi tra intelligenze straniere, o colpire direttamente parte informatizzate di alcuni stati.

Hacktivism con questo termine si indica un attacco informatico perpetrato non per scopi economici, ma a fini politici da attivisti.

1.2 Trustworthyness (Affidabilità)

Possiamo dire che il nostro sistema NIS è affidabile se esso continua a resistere agli attacchi che gli vengono quotidianamente inflitti, con attacchi più o meno potenti. Altri disturbi nel quale esso si puo' trovare immerso (questi sono conseguentemente inevitabili) possono essere dovuti al contesto ambientale nel quale è immerso: quest'ultimo ostacolo può essere evitato con la replicazione in altri luoghi del sistema, ma dovremo comunque considerare un limite superiore di tolleranza, entro il quale il servizio non risulterà più fruibile. Deve inoltre funzionare nonostante l'utilizzo del sistema da parte di esseri umani, o della presenza di bug al suo interno.

All'interno di un sistema sono presenti due tipologie differenti di requisiti, quelli **funzionali** (ovvero la stretta aderenza alle specifiche: descrivono quello che ci si aspetta in output dal programma una volta fornito un suo input) e quelli **non funzionali** («descrivono le proprietà del sistema software in relazione a determinati servizi o funzioni e possono anche essere relativi al processo»), tra i quali[1] appare la caratteristica dell'affidabilità.

Dobbiamo per altro considerare perché l'affidabilità di un sistema non sia considerato come un requisito funzionale di un sistema: infatti sarebbe impossibile conoscere tutti i possibili attacchi in input, in quanto non saremmo mai in grado di caratterizzarli completamente, poiché ciò implicherebbe che saremmo stati in grado di risolvere questa problematica già in fase di realizzazione del software. In quanto questi sono imprevedibili, di fatto la nostra soluzione è impraticabile.

1.3 Metafora fisica per realtà metafisica

Possiamo ora effettuare un paragone tra il mondo reale e quello informatico: possiamo vedere che la sicurezza viene (appunto) assicurata in base al valore che si attribuisce ai proprii oggetti (che le minacce tentano di minare) che vengono protetti tramite *lucchetti* (che impediscono l'accesso all'oggetto) apribili solamente mediante tramite certe *chiavi*, ed un sistema di punizione dell'atto criminale coniugato alla certezza della pena. Inoltre non tutti i lucchetti sono uguali, ma avranno una soglia di difficoltà di scassinamento differente

[1] Altro requisito non funzionale che costituisce la QoS ("Quality of Service") è la scalabilità

(**theresold**): saranno lucchetti con chiavi differenti e con diversi gradi di resistenza, usati in base al particolare contesto.

Possiamo inoltre vedere che i lucchetti fungono da **autorizzazione** (specifico chi ha le credenzialità sufficienti per poter accedere), e sono necessari per soddisfare la confidenzialità dell'informazione: questo richiede dei meccanismi di controllo degli accessi. Le chiavi inoltre sono dei meccanismi di **autentificazione** (necessari per aprire i "lucchetti"), ottenendo in questo modo l'autorizzazione del sistema ad accedere alle risorse che gli sono consentite.

Inoltre, anche la totale sicurezza del nostro modello può comportare la totale inusabilità del nostro sistema, in quanto saremo eccessivamente vincolati dalle procedure di sicurezza: inoltre dovremmo garantire un costo per proteggere il nostro oggetto adeguato alla seguente relazione:

$$p(\text{Mancare}) \cdot \text{costo} > \text{CostoSicurezza}$$

solo in questo caso è conveniente appunto assicurare il nostro prodotto tramite dei meccanismi di sicurezza. Tornando ora ai NIS, il sistema di punizione, coincide con il sistema di sicurezza del mondo fisico, mentre il valore degli oggetti è correlabile con il valore dei nostri dati, che vogliamo proteggere concedendoli solamente a chi riteniamo autorizzato.

La nostra informazione inoltre è memorizzata tramite bit, ed assume significato (e quindi valore) solamente se è contestualizzata all'interno di un discorso generale: parallelamente il loro valore dipenderà dal valore che si attribuisce al contesto nel quale esso è immerso.

1. Possiamo effettuare una protezione tramite cifratura: anche se questa si rivelasse affidabile dal punto di vista matematico, in quanto il computer non è programmabile tramite teoremi ma solamente tramite programmi dell'essere umano, anche la crittografia può essere soggetta ad errori.

2. Pretendere sicurezza implica anche l'impostazione di diversi livelli della configurazione, in quanto a differenti livelli possono presentarsi differenti problemi

3. Spesso il problema consta degli utenti stessi, in quanto spesso questi disabilitano la sicurezza prevista nei sistemi: infatti gli utenti di solito preferiscono il prodotto nuovo vetusto, in confronto ad uno aggiornato e sicuro.

Un altro ostacolo alla sicurezza è costituita dai *brevetti commerciali*: questi infatti sono di ostacolo alla resa affidabile dei nostri sistemi, così anche altri problemi possono essere presenti in software artigianale: in quanto non è un programma di vasta fruizione, saremo meno a conoscenza di eventuali problemi che esso presenta. Infatti la sicurezza del nostro sistema non dovrebbe dipendere dalla segretezza dei suoi meccanismi, quanto piuttosto da come esso è costruito e da come esso funziona, in quanto, come abbiamo già potuto osservare, non si ha una forma sufficiente di sicurezza tramite la non divulgazione. Da queste considerazioni segue che è fondamentale il concetto di **open design**: non è tanto importante rendere segreto il meccanismo del "lucchetto", quanto è importante rendere segreta la chiave: in questo modo inoltre possiamo ridurre la quantità delle informazioni da rendere segrete, concentrando la nostra sicurezza in un ambito più ristretto.

1.4 Analisi dei NIS

Come in un qualsiasi altro sistema informatico, possiamo studiare i seguenti aspetti, che possono essere presenti anche nella gestione della sicurezza, ma cambiano di nome (indicati per tanto come secondo identificativo):

- ◇ **Specifica / Policy (Politica**: cosa ci si aspetta che il sistema produca
- ◇ **Implementazione / Mechanism**: messa in atto delle specifiche
- ◇ **Correttezza / Assurance**: verifica della correttezza dell'implementazione

All'interno di un sistema dobbiamo considerare anche:

- ◇ **Vulnerabilità**: è la sensibilità del sistema ai danneggiamenti
- ◇ **Attacco**: è uno specifico metodo per sfruttare una vulnerabilità
- ◇ **Minaccia**: è un ente motivato a lanciare un attacco contro il sistema

Si può inoltre notare che, con il passare del tempo, il tempo tra scoperta della vulnerabilità e l'attacco che sfrutta questo bug, si accorcia sempre di più con il passare del tempo; chiamiamo inoltre **vulnerability window** ("finestra di vulnerabilità") l'intervallo di tempo tra la scoperta della vulnerabilità e la release della patch che cerca di sistemare il problema, e **zero day attack** l'attacco che avviene all'interno di detta finestra di vulnerabilità. Tuttavia si può evidenziare che di solito il tempo medio di attacco dopo la scoperta del bug è nettamente inferiore al tempo medio impiegato per risolvere il problema.

Oggi giorno si assiste anche al fenomeno del **Google Hacking**: mentre un tempo per perpetrare un attacco era necessario possedere delle specifiche competenze in materia, oggi l'accesso facilitato alle informazioni tramite Google o tramite "pacchetti software" specializzati, che forniscono degli attacchi preconfezionati.

1.5 Politiche di sicurezza

Le politiche di sicurezza riguardano i seguenti ambiti:

Segretezza detta anche confidenzialità, impone il controllo di chi ha il permesso di leggere le risorse; storicamente si è sempre inteso la sicurezza all'interno di questo ambito, anche se solamente questo fattore non implica effettivamente sicurezza.

Integrità impone il controllo sui permessi di modifica delle informazioni, o come esse in genere vengono utilizzate

Disponibilità impone un'immediata disponibilità delle risorse. Potrei infatti ottenere sicurezza attenendomi strettamente ai principi di cui sopra, semplicemente non rendendo disponibile la risorsa che voglio proteggere: tuttavia in questo modo la risorsa risulta inservibile in quanto non accedibile. Vogliamo infatti che la nostra risorsa sia accedibile in un tempo ragionevole ed in modo tempestivo.

Accountability con questo vincolo (traducibile con "tracciabilità") vogliamo imporre di venir a conoscenza chi ha avuto accesso alla risorsa, in modo da poter rintracciare e ricostruire gli accessi e le modifiche effettuate; ciò consente anche di rilevare eventuali accessi da parte di criminali informatici. Questo requisito, spesso chiamato anche *auditing*, assieme ad *autenificazione* e *autorizzazione* fanno parte del **Golden Standard of Security**.

Tutte queste caratteristiche insieme garantiscono l'*assurance*, ovvero ci garantiscono che il sistema è affidabile. Noi infatti vorremo essere in grado di fornire una dimostrazione matematica dell'effettiva sicurezza del nostro sistema: tuttavia questa via è praticabile unicamente per piccoli sistemi , in quanto più esso è complesso, più è imprevedibile considerare tutti i diversi casi di possibili vulnerabilità.

Da ciò segue che, per garantire maggiormente la affidabilità del nostro sistema, possiamo creare all'interno dei nostri progetti delle piccole funzionalità (**Economy of Mechanism**), ovvero con un numero limitato di righe di codice, riducendo conseguentemente la complessità del nostro sistema, in quanto avremo piccole zone di codice sulle quali concentrarci per controllare la correttezza del nostro sistema; conseguentemente, un approccio modulare nel nostro sistema garantisce complessivamente una maggiore affidabilità.

Capitolo 2

Crittografia

L'idea iniziale della crittografia era quella di creare una comunicazione privata all'interno di un ambiente pubblico, dove un terzo individuo potrebbe essere in ascolto di una comunicazione in atto tra le due parti. La crittografia moderna è basata su sofisticati formalismi matematici, che però rendono possibile una sua applicazione in un contesto pratico. Al giorno d'oggi non è richiesto saper progettare nuovi sistemi crittografici, ma saper utilizzare quelli odierni, dei quali è già stata dimostrata la validità. Oggi giorno è quindi utilizzata tra due controparti della comunicazione, per scambiarsi un messaggio in codice (**crittogramma**) all'interno di un canale non ritenuto sufficientemente sicuro (all'interno del canale può esistere un intruso, detto **crittoanalista**, che è in ascolto delle comunicazioni che avvengono all'interno del canale): il mittente codificherà il messaggio in chiaro (*plaintext*) in uno cifrato (*ciphertext*) tramite una funzione di codifica $C : m \to c$: questo verrà quindi intercettato dal destinatario che lo convertirà tramite una funzione di decodifica $D : c \to m$.

In particolare, queste due funzioni in questione necessiteranno di una chiave per effettuare la codifica o la decodifica: possiamo quindi esprimere le due funzioni (che matematicamente sono inverse) come:

$$C_k(m) = c \quad D_k(c) = m \quad D_k(C_k(m)) = m$$

In più, se si ha che $C_k(D_k(m)) = D_k(C_k(m)) = m$, allora si dice che le due funzioni sono *commutative*, in quanto l'ordine con il quale esse vengono applicate con le loro rispettive chiavi è irrilevante. In particolare, in questa situazione, l'intruso non dovrà essere a conoscenza della funzione di decodifica D.

Tuttavia dobbiamo anche possedere un'infrastruttura atta a mantenere il sistema crittografico; possiamo avere sicurezza tramite crittografia in due modi:

Segretezza perfetta questa implica che una parte dell'informazione necessaria alla decodifica non dovrà mai essere divulgata; è per tanto facilmente intuibile che questa è impossibile da ottenere

Segretezza computazionale questa implica una disparità di risorse tra l'avversario ed il suo antagonista; queste sono ovviamente le risorse di calcolo, che dovranno essere supposte limitate superiormente (al di sopra delle quali non si garantisce la confidenzialità).

Sappiamo inoltre che i nostri messaggi saranno di dimensione finita (N) e che essa è trasmessa nella rete tramite bit, che vogliamo trasmettere in un modo confidenziale. Un ascoltatore potrebbe modificare gli N bit per vedere, con una probabilità di $\frac{1}{2^N}$, se riceve una risposta esatta: abbiamo conseguentemente che esso avrà una probabilità molto bassa di ottenere un risultato corretto, anche con un N modesto.

Diciamo quindi che un algoritmo di crittografia è ottimale se un ascoltatore non può fare meglio di $\frac{1}{2^N}$ per poter decifrare il nostro crittogramma, con la quale possiamo perseguire una forma debole di segretezza perfetta. Diamo ora le seguenti definizioni:

Crittografia consiste nella progettazione di cifrari sicuri ed efficienti

Crittoanalisi consta nell'insieme dei metodi, degli strumenti e delle tecniche per attaccare i cifrari, eventualmente per testarne la loro qualità, ma per lo più per intenti malevoli. Questo lavoro è svolto dal **crittoanalista**.

Crittologia è l'insieme di crittografia e crittoanalisi

In particolare il sistema di crittografia si chiama **simmetrico** o a **chiave privata** se l'origine e la destinazione utilizzano una stessa chiave k per le funzioni di coding e decoding, spesso con $C \equiv D$; se invece le due chiavi non coincidono il sistema di crittografia si dice **asimmetrico** o a **chiave pubblica**.

Possiamo inoltre distinguere due differenti famiglie di algoritmi crittografici, in base alla segretezza delle funzioni di codifica e di decodifica:

Cifrari per uso ristretto C e D sono mantenute segrete; questi cifrari possono quindi essere utilizzati solamente quando non si vuole tenere un elevato grado di sicurezza, e quando si è all'interno di un contesto molto semplificato (si ha appunto sicurezza tramite non divulgazione).

Cifrari per uso generale C e D sono note a tutti, e quindi la segretezza della comunicazione si basa sulla segretezza della chiave.

Tuttavia non è solo con la crittografia che possiamo mantenere la confidenzialità dei nostri dati; bisogna mantenere:

Integrità il destinatario deve controllare che il messaggio ricevuto non abbia subito delle modifiche parziali o totali durante il tragitto

Autenticazione il destinatario deve poter accertare che il messaggio provenga esattamente da un destinatario noto, ovvero vogliamo la garanzia che il messaggio sia giunto da un preciso mittente.

Non-repudiation in quanto è garantita l'autenticazione, il mittente non può disconoscere il messaggio da lui inviato: se ciò è dimostrabile, il messaggio può essere conservato come prova dell'effettivo mittente

2.1 Metodi d'attacco

L'intruso, all'interno della comunicazione, può avere sia un ruolo *passivo* (ovvero, ascolta solamente la comunicazione come all'interno delle intercettazioni telefoniche; questa forma risulta naturale all'interno delle reti wireless), oppure *attivo* (non solo è in ascolto della comunicazione, ma interviene all'interno di questa andando ad introdurre propri bit, od andando a modificare direttamente quelli dell'utente; questa forma risulta semplice nella modalità Man-In-The-Middle).

Un intruso può effettuare in questo contesto tre tipologie d'attacco:

Cyphertext attack si salvano tutti i crittogrammi che fluiscono all'interno della rete, sui quali poi si effettuerà la crittoanalisi; da questi poi si tenterà di ottenere il plaintext.

Known plain-text attack partendo dal presupposto che si conosca il plain-text di alcuni crittogrammi, l'intruso prova ad ottenere il plain-text anche degli altri che trova all'interno della rete.

Chosen plain-text attack si presuppone che il crittoanalista abbia a disposizione una serie di messaggi in chiaro, che può criptare ottenendo i corrispondenti crittogrammi.

2.2 Crittografia simmetrica o a chiave privata

D'ora in poi, nelle sezioni seguenti, partiremo dal presupposto di conoscere le funzioni C e D; con questo metodo di crittografia, è come se chiudessimo all'interno di uno scrigno l'informazione che, sia la destinazione sia il mittente sono in grado di carpire, tramite una chiave che entrambi conoscono (e conseguentemente anche il lucchetto per accedere all'informazione sarà sempre riconosciuto da entrambi).

In questo scenario tuttavia lo scambio della chiave deve avvenire **out of band**, ovvero al di fuori del canale di comunicazione che si crede non sicuro, in quanto un potenziale intruso potrebbe carpire la chiave ed utilizzare per decifrare i messaggi che verranno trasportati da una parte all'altra, compromettendo conseguentemente l'utilizzo della chiave stessa nella comunicazione criptata. Possiamo quindi trasportare la chiave all'interno di un'intranet, tramite un mezzo personale o tramite telefono, etc. Comunque il canale secondario per la distribuzione della chiave, deve essere scelto in un modo proporzionale al valore del messaggio trasmesso, che deve costituire un canale meno rintracciabile rispetto a quello dove avverrà la normale comunicazione. Analogamente, la stessa chiave non potrà essere utilizzata all'infinito, in quanto dopo un certo periodo di tempo è da considerarsi compromesso, in quanto sarà aumentata la probabilità che un malintenzionato sia riuscito ad ottenerla; tuttavia bisogna sottolineare che la segretezza della chiave dipenderà dal mantenimento della segretezza da entrambe le contro parti. Inoltre, queste chiavi dovranno avere una lunghezza considerevole, in modo che risulti sconveniente effettuare un attacco brute force per ottenerla, e dovranno essere *confidenziali* in quanto, non potendo distinguere a priori gli amici dai nemici, non sappiamo se quello che utilizzerà la nostra chiave sarà interessato ad intercettare le nostre comunicazioni. Alcuni cifrari simmetrici sono *DES*, *triple DES*, *AES* ed *IDEA*.

Possiamo inoltre notare che, per ogni n utenti, si avrà necessità di possedere n^2 chiavi simmetriche, e saranno necessari $\frac{n(n-1)}{2}$ canali per scambiarsi la comunicazione[1].

2.3 Crittografia asimmetrica o a chiave pubblica

Vogliamo ora capire se è possibile mantenere la confidenzialità senza effettuare lo scambio di una chiave segrete tra le controparti della comunicazione. Questa risoluzione è infatti di interesse economico, in quanto elimina la parte di comunicazione out of band, che è un grosso handicap per il sistema simmetrico.

In questo particolare sistema è il desitnatario che sceglie un "lucchetto" ed un'apposita chiave, fornendo unicamente il primo (aperto) al mittente: questo si preoccuperà di chiudere l'informazione con tale lucchetto, e di spedire lo scrigno così ottenuto a destinazione, la

[1] Si divide per due perché dobbiamo considerare il nostro canale di comunicazione bidirezionale

quale potrà aprire il suo lucchetto tramite la chiave che essa stessa ha generato. In questo modo può avvenire una qualsiasi forma di comunicazione tra A e B, senza che questi si conoscano precedentemente; nota inoltre che, se l'ordine di consegna dei lucchetti e delle chiavi fosse invertito, non sarebbe possibile alcuna forma di comunicazione privata.

Questi algoritmi devono inoltre garantire che sia difficile desumere la funzione di codifica da quella di decodifica, sempre non conoscendo al chiave privata (che si traduce quindi in un attacco di tipo brute force per desumerne il valore); questo concetto si applica per transizione anche alle funzioni di codifica e decodifica, implicando sempre la non conoscenza della chiave privata: si parla infatti di funzione **one-way trap-dor**:

1. Si dice che è "a senso unico" in quanto se la funzione C è semplice da conoscere, deve essere complicato ottenere la funzione inversa D
2. Si dice che è una "botola" in quanto se non conosco la chiave giusta, non devo essere in grado di rompere il meccanismo

Un esempio di applicazione di queste formule è direttamente correlato alla teoria dei numeri primi: dati due numeri primi p e q infatti sufficientemente grandi (ovvero con centinaia di cifre), è facile calcolare $n = pq$, ma è difficile ottenere uno dei due numeri primi dato n (sappiamo che la fattorizzazione degli stessi è infatti un problema arduo, in quanto l'algoritmo che risolverebbe questo problema è appunto superpolinomiale), sempre supposto di non conoscere l'altro. Conseguentemente abbiamo che possiamo ottenere la chiave pubblica dal numero n così ottenuto, che avrà alpiù il doppio di cifre dei numeri primi dati in partenza.

Con questo scenario, sarà sufficiente distribuire a tutti gli utente una sola chiave (quella pubblica), e ogni destinatario potrà generare una propria chiave privata per tutte le comunicazioni future.

2.4 Cifrari storici

2.4.1 Cifrari a sostituzione

Il **cifrario di Cesare** è basato sul principio di *sostituzione* dei caratteri tramite traslazione verso destra o sinistra di k posizioni dell'alfabeto criptato rispetto a quello originale (*shift ciclico*); in questo cifrario avremo inoltre una chiave costituito da un solo numero, che viene utilizzato per ogni lettera:

$$C(m_i) = (pos(m_i) + k) \mod 26 = c_i \quad D(c_i) = (pos(m_i) - k) \mod 26$$

Analogamente, potremo generalizzare questo cifrario imponendo una qualsiasi sostituzione del nostro carattere con un altro; in questo modo passiamo da 26 possibili chiavi a un numero di $26! - 1$ trasformazioni, in quanto ogni trasformazione è una permutazione dell'alfabeto. Tuttavia, entrambi questi cifrari possono essere soggetti all'attacco di tipo statistico: l'ascoltatore occulto infatti potrebbe confrontare il campione statistico di una lingua nota con quelli desunti da un campione di crittogrammi, ed assegnare la lettera in plain-text che in quella lingua occorre con la maggior frequenza con quella che occorre più frequentemente nei crittogrammi; se questa formulazione dovesse rivelarsi errata, si potrà procedere per confronto con tutte le altre lettere. Altri cifrari a sostituzione implicano:

- Si ottiene una chiave di lunghezza $|k|$, che viene replicata per tutta la lunghezza del messaggio: si effettua poi una sostituzione del carattere del messaggio m_i con quella della chiave k_j prendendo il carattere $T[m_i, k_j]$ all'interno di una tabella.

- Si può effettuare la sostituzione tra blocchi di caratteri:

2.4.2 Cifrari a Trasposizione

Si stabilisce una chiave che stabilisce una chiave con numeri unici e distinguibili, che identificano una permutazione arbitraria della disposizione del messaggio. Si dispone poi il messaggio per righe facendolo coincidere con la lunghezza della chiave; in seguito si leggono le colonne ottenute dal messaggio in base all'ordine fornito dalla chiave.

In questi cifrari non andrà più a buon fine l'attacco di tipo statistico, in quanto la frequenza di occorrenza delle lettere sarà identica a quella del messaggio originale, ma sarà cambiata unicamente la disposizione delle lettere.

Si può inoltre ripetere la tecnica dettagliata precedentemente, finchè non otterrò un output ancora più sicuro in quanto in questo modo aumento l'entropia (ad ogni combinazione è sempre meno evidente che il crittogramma possa essere ottenuto tramite riordinamneto, sembrando quindi un messaggio casuale), poiché riordiniamo i caratteri in un modo ancora più casuale del precedente.

Nota: con una scelta inappropriata della chiave si potrebbe rischiare di ottenere esattamente lo stesso messaggio di partenza, anche se effettivamente questo non è lo scopo del nostro algoritmo. Si può inoltre notare che, mentre con una sola trasposizione si riscontrano regolarità nell'andamento delle posizioni del messaggio, via via che le trasposizioni successive aumentano l'ordine di decodifica diventa sempre meno chiaro.

2.4.3 Cifrario One-Time Pad

Questo cifrario è anche detto cifrario perfetto, in quanto è un caso limite che esplica quale elevato grado di codifica possiamo ottenere con le tecniche di crittografia.

Dato un messaggio di lunghezza $|n|$, possiamo ottenere una chiave casuale (per questo il cifrario è detto **one-time**, anche perché viene utilizzata una sola volta per spedire un solo messaggio) della stessa lunghezza detta **pad**: possiamo ottenere il crittogramma effettuando lo xor bit a bit tra messaggio e pad. Possiamo inoltre notare che, dato il crittogramma e la chiave, possiamo di nuovo ottenere il messaggio originale effettuando sempre lo xor bit a bit.

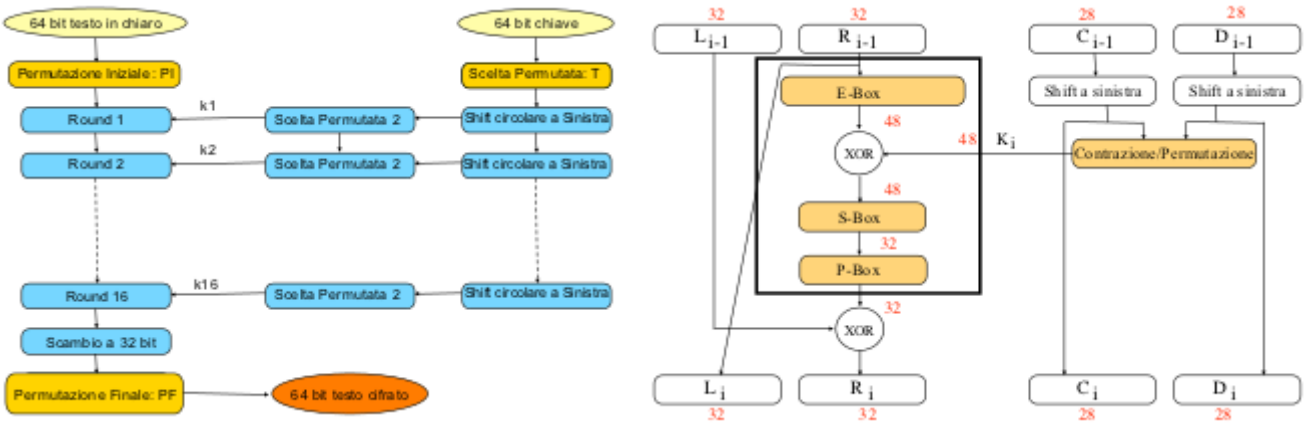
Ora passiamo a trattare la sicurezza dell'algoritmo: abbiamo che la nostra chiave è generata casualmente (ottenuta con probabilità $\frac{1}{2^n}$), ed abbiamo che anche l'intruso avrà tale probabilità di ottenere la chiave: infatti, sapendo che viene utilizzato l'operatore XOR, se l'i-esimo bit della chiave è 0, allora il bit del testo cifrato sarà lo stesso del messaggio, altrimenti questo dovrà essere invertito: per questo si dice che è un "cifrario perfetto".

Tuttavia abbiamo immediatamente che questo algoritmo non è applicabile in pratica in quanto risulta molto costoso: infatti in quanto la chiave è unica per ogni nuovo invio del messaggio, il costo dell'invio della chiave privata tramite un altro canale di comunicazione sicuro non verrà mai ammortizzato; inoltre si ha sempre un costo per lo scambio della chiave proporzionale alla lunghezza del messaggio stesso.

2.4.4 DES

Prima di introdurre questo cifrario, descriviamo alcune tecniche che possono essere utilizzate all'interno degli algoritmi di cifrazione:

Permutazione cambiamento dell'ordine del messaggio



(a) *Trattazione generale dell'algoritmo.* (b) *Analisi del punto k_i di 16.*

Figura 2.1: *DES Overview*

Sostituzione indico a quale blocco ne corrisponderà un altro nel messaggio cifrato

Espansione replicazione all'interno di un messaggio di un bit o di un carattere

Contrazione (detta anche scelta), alcuni caratteri o bit vengono scartati e poi permutati: quando successivamente si effettua una permutazione, allora l'insieme di queste due operazioni viene detta **scelta permutata**.

Come si può vedere dall'immagine 2.1 *(b)*, la chiave (rappresentata da C_k e D_k) è costituita da due blocchi formanti complessivamente 56 bit, contro i due blocchi da 32 bit (complessivamente 64) del messaggio: questo perché i rimanenti 8 bit della chiave sono di parità. Dobbiamo inoltre evidenziare che `E-Box` indica un'espansione, `S-Box` una sostituzione e `P-Box` una permutazione.

2.4.5 RSA

Questo è un algoritmo di crittografia a chiave pubblica, del quale recentemente è scaduto il brevetto. Esso sfrutta la teoria dei numeri e la definizione della funzione di Eulero[2]: dati due numeri primi p e q, si dimostra che $\varphi(pq) = (p-1)(q-1)$. Diciamo inoltre che, i numeri primi che sceglieremo, non dovranno essere contemplati da nessuna fonte facilmente reperibile, in quanto sarebbe immediato decifrare la chiave pubblica per ottenere quella privata.

[2]La funzione di Eulero è quella che associa ad ogni naturale n il numero dei naturali m primi con n tali che $m \leq n$, ed è indicato come $\varphi(n), n \neq 1$. Si può dimostrare che:

- Se n è primo allora $\varphi(n) = n - 1$ (tutti i numeri primi con lui saranno tutti quelli minori di lui, tranne il numero stesso)
- Se n è una potenza di un primo, allora $\varphi(n^r) = n^r - n^{r-1}$, ovvero a tutti i numeri fino ad n^r tolgo tutti quei numeri che sono multipli di n fino ad n^r, che sono appunto n^{r-1}.
- Se a e b sono coprimi, allora $\varphi(ab) = \varphi(a)\varphi(b)$
- Segue che per un qualsiasi numero n scomponibile per la fattorizzazione standard (che si dimostra essere unica) nel modo $n = \prod^i a_i^{e_i}$ con a_i primo, è scrivibile come $\varphi(n) = n \prod^i \left(1 - \frac{1}{a_i}\right)$

In particolare dobbiamo trovare:

$$\exists e.GCD(e, \varphi(n)) = 1 \quad \exists d.d \cdot e \bmod \varphi(n) = 1$$

dove costituiranno le chiavi:

$$\text{pubblica} = (e, n) \quad \text{privata} = (d, n)$$

Da questo si può quindi ricavare la formula per effettuare la codifica del messaggio:

$$C(m) = m^e \bmod n = c$$

In questa funzione, dato il messaggio che si vuole cifrare m, l'utente conosce mediante la sua chiave privata i valori di e ed n. Inoltre è possibile elevare a potenza e un messaggio, considerandolo come costituito da bit, e conseguentemente come un numero. La funzione di decodifica risulta quindi la seguente:

$$D(c) = c^d \bmod n = m$$

Il destinatario è inoltre in grado di effettuare questo calcolo in quanto conosce i valori di d ed n tramite la sua chiave pubblica.

Giunti a questo punto, non abbiamo tuttavia dimostrato nè la correttezza, né la sicurezza e nemmeno l'efficienza del calcolo di questo algoritmo, in quanto i calcoli da svolgere possono essere considerevoli. Ci appresteremo or ora nella loro trattazione.

2.4.5.1 Correttezza

Per dimostrare la correttezza dell'algoritmo, dobbiamo provare che le due funzioni di codifica e decodifica siano inverse, ovvero che $D(C(m)) = m$. Prima di effettuare la dimostrazione, dobbiamo prendere in considerazione i seguenti risultati:

- $\diamond$ $GCD(m, n) = 1 \Rightarrow m^{\Phi(n)} \bmod n = 1$ (Teorema di Eulero)
- $\diamond$ $m^k \bmod p = 1 \wedge m^k \bmod q = 1 \Rightarrow m^k \bmod pq = 1$
- $\diamond$ $x \bmod n = 1 \Rightarrow \forall y.x^y \bmod n = 1$

Possiamo inoltre supporrre che m sia un messaggio tale che $0 < m < n$: tuttavia, non sempre potremo avere la garanzia che tale messaggio sia della dimensione desiderata; in questo caso potremo operare come in DES, ovvero dividendo il messaggio in ingresso in parti .

Dimostrazione. Applicando il valore delle funzioni in $D(C(m))$, otterremo:

$$\begin{aligned} D(C(m)) &= D(m^e \bmod n) \\ &= (m^e \bmod n)^d \bmod n \\ &= m^{ed} \bmod n \end{aligned}$$

In quanto abbiamo per definizione dell'argoritmo e quindi per imposizione che $de \bmod \Phi(n) = 1$, quindi abbiamo che $\exists k > 0.de = k\Phi(n) + 1$. Possiamo quindi riscrivere l'esponente dell'espressione di cui sopra come segue, utilizzando anche il teorema di Eulero:

$$\begin{aligned} D(C(m) &= \dots \\ &= mm^{k\Phi(n)} \bmod n = m \cdot 1 = m \end{aligned}$$

□

2.4.5.2 Sicurezza

In quanto sappiamo che l'unica nostra arma di sicurezza è quella di generare chiavi lunghe ($N \to \infty$) in modo tale che $\frac{1}{2^N} \to 0$, ovvero di generarle casualmente tramite un attacco di forza bruta, proviamo ad evidenziare due possibili metodi di attacco:

- Riuscire a fattorizzare n in modo da ottenere i valori di p e q primi che si sanno esistenti
- Dal valore di n ottengo $\Phi(n)$, ed in seguito riesco ad ottenere $d = e^{-1} \bmod \Phi(n)$, grazie ai dati che conosco dalla chiave pubblica

Possiamo quindi riscontrare che entrambi questi metodi hanno una complessità che è pari al costo di fattorizzare n, in quanto per ottenere $\Phi(n)$ devo saper fattorizzare n. Tuttavia fin'ora non è ancora stato trovato un risultato teorico che dimostri il costo computazionale del lower-bound di questo problema, e sono state portate unicamente delle evidenze empiriche sulla difficoltà strutturale nella risoluzione del problema: ci si affida unicamente al fatto che sia improbabile che qualcuno entro breve scopra il valore di suddetto lower-bound.

Come abbiamo potuto constatare dalle osservazioni precedenti, la sicurezza degli algoritmi di crittografia fa affidamento sulle esigue risorse a disposizione degli attaccanti: per questo, con l'evolversi della tecnologia, si trova sempre più necessario trovare dei valori di numeri primi sempre più grandi. Altra garanzia di sicurezza è dovuta al fatto che, riuscire a decomporre un numero n in due numeri primi, non implica essere in grado di effettuare la fattorizzazzione di $n' \neq n$ in un minor tempo, sempre considerando che n' non è composto dagli stessi fattori primi di n.

2.4.5.3 Efficienza

Ora vogliamo trovare delle tecniche per rendere più efficienti i calcoli del nostro algoritmo:

Esponenziale per computare n^{32}, invece di effettuare 31 operazioni di moltiplicazione, potremo operare nel modo seguente:

$$x^{2y} = x^y \cdot x^y \quad x^{y+1} = xx^y$$

In questo modo possiamo semplificare i conti se consideriamo l'esponente in binario: considerando le cifre da sinistra verso destra, eleveremo il risultato precedente al quadrato se il bit successivo è zero (sarebbe moltiplicare per due), altrimenti moltiplichiamo un'altra volta per x. In questo caso, detto y l'esponente, nel caso pessimo effettueremo sempre $2 \log_2 y$ moltiplicazioni

Modulo in quanto i nostri calcoli sono sempre sotto modulo n, allora possiamo verificare che:

$$(a \bmod n)(b \bmod n) \bmod n = (ab) \bmod n$$

In questo modo, invece di effettuare sempre il calcolo del modulo alla fine, lo possiamo effettuare per ogni passo intermedio, in modo da evitare l'overflow, in quanto appunto il risultato finale che otterremo non potrà mai essere più grande di n.

2.4.5.4 Test di primalità

Sempre correlato alla sicurezza dell'algoritmo, parliamo ora dei test di primalità che possono essere ricondotti alla sicurezza dell'algoritmo. Come prima cosa, abbiamo una probabilità di $\frac{1}{\ln(n)}$ che un dato numero primo n sia primo. Tuttavia questa probabilità, sebbene sia molto bassa, non è zero, e quindi l'evento può sempre avvenire.

Possiamo quindi effettuare dei test di primalità, che rispondono se un numero è primo o meno. In un metodo Naïf potremo scandire tutti gli interi da 2 a $n-1$, ma un risultato teorico ci consente di limitare ulteriormente questo numero con $\sqrt{n}$. Detto quindi $m = \log n$ la dimensione del nostro numero in input, otteniamo complessivamente una complessità $O(\sqrt{n}) = O(2^{\frac{1}{2}n})$. Non si sono trovati inoltre, se non recentemente, dei risultati polinomial per il test di primalità: utilizzando l'*Ipotesi di Riemann* si arriva ad un $O((\log n)^4)$, mentre con l'algoritmo AKS si arriva ad un $O((\log n)^6)$: tuttavia, anche se il risultato è polinomiale, abbiamo che è lo stesso un risultato inefficiente, in quanto la costante è considerevolmente elevata.

Abbiamo inoltre il seguente risultato dal *Teorema di Fermat*[3]:

$$\texttt{prime}(n), 0 < a < n \Rightarrow a^{n-1} \bmod n = 1$$

Pomerance ha inoltre ottenuto la probabilità che venga superato il test $a^{n-1} \bmod n = 1$ da un numero non primo con $\forall.a < n$, che è $\frac{1}{10^{13}}$. Possiamo quindi generare un algoritmo del tipo:

```
int isprime(int *n) {
  *n = random_dispari();
  a = random() % *n;
  return  ((a**(*n-1) % *n)==1);
}
```

Tuttavia, se il test è passato, è ancora presente $\frac{1}{10^{13}}$ di probabilità che il numero non sia primo: in questo modo possiamo ripetere il nostro test di cui sopra per tanti a generati casualmente, in quanto testo il mio numero n per k volte, in questo modo avrò effettuato prove del test per tanti valori di a, abbassando ulteriormente ad un $\frac{1}{10^{13^k}}$ la probabilità di fallimento del test. Si testeranno quindi $\frac{\log(n)}{2}$ numeri prima di accettare il numero come primo.

2.5 Identificazione, autentiticazione e firma digitale

In quanto ci possiamo accorgere che con la sola crittografia non possiamo aver garantita la confidenzialità, sono state introdotte le segueti funzionalità:

Identificazione dobbiamo essere in grado di accertare l'identità di un utente che vuole accedere ai suoi servizi

Autentificazione vogliamo accertare che il messaggio ricevuto sia di un predefinito utente, e che il messaggio ricevuto sia giunto integro

Firma digitale vogliamo accertarci che un documento sia stato scritto da un prefissato mittente (da utilizzare quando non c'è reciproca fiducia tra le due estremita della comunicazione).

[3]Come conseguenza del teorema di eulero

2.5.1 Funzioni di Hash

Per introdurre i concetti di firma dei messaggi che seguiranno, è importante definire che cosa siano le funzioni di Hash. Dato un dominio X contentente le *pre-image*, ovvero i valori che dovranno essere mappati nel codominio Y degli hash-value; tale valore risultante è chiamato **digest** o **fingerprint**. Il nostro requisito è che, data una funzione $f : X \to Y$, del tipo "many-to-one", ovvero $|X| > |Y|$ di molti ordini di grandezza.

Questa sarebbe una proprietà propria di ogni funzione di hasing. Tuttavia nel campo della crittografia e della sicurezza richiediamo delle altre caratteristiche:

◇ Visto che la funzione è del tipo "many-to-one", vogliamo che i nostri valori siano distribuiti all'interno di un'area uniforme. Dato un sottoinsieme del dominio degli elementi che hanno tutti la stessa immagine, ovvero $X_i = \{ x \in X \mid f(x) = y_i \}$, vogliamo che questi sottodomini siano distribuiti in una maniera uniforme, ovvero che:

$$\forall X_i . \exists c . |X_i| \simeq c$$

◇ Inoltre vogliamo che elementi $\forall x_i \in \mathbb{I}(x)$ appartengano a sottoinsiemi distinti e ben distanziati, ovvero:

$$\forall x_i, x_j \in \mathbb{I}(x) . x_i \in X_a \wedge x_j \in X_b,\ a \ll b \vee b \ll a$$

Questa dispersione all'interno delle hash table garantisce che non avrò mai la collisione per valori molto vicini all'interno della tabella, in modo da non rendere lenta la ricerca.

◇ Altre proprietà invece garantiscono che le funzioni di crittografia definite siano definibili come "one-way":

- $\forall x \in X . f(x)$ deve essere facilmente computabile
- $\forall y \in Y . \exists x . f(x) = y$ deve essere arduo da calcolare
- $\forall x_1 \in X . \exists x_2 \in X . f(x_1) = f(x_2)$ deve essere arduo da calcolare

Proviamo a dare un semplice (fin troppo) algoritmo di funzione di hash, detta *simple 8-bit block parity*: dato un qualunque messaggio m, disponiamo il messaggio su righe parallele previa suddivisione in blocchi da 8 bit, colmando con un padding di zero l'ultimo blocco nel caso in cui avesse una dimensione inferiore a quella prefissata. Il numero di digest si effettua calcolando il bit di parità per quella colonna, ovvero effettuando lo XOR tra ogni bit. Da ciò segue naturalmente che il fingerprint sarà sempre di una dimensione di 8 bit, con una possibilità di 2^8 valori possibili (è quindi il valore di $|Y|$), mentre $|X|$ è arbitraria. Si può tuttavia dimostrare facilmente che questo non soddisfa le ultime due proprietà dell'ultimo punto di cui sopra:

- in quanto il bit di parità mi fornisce un'informazione sul valore precedente di quel bit, posso ridurre la casistica dei valori possibili che seguono dimezzandola
- è conseguentemente facile ottenere un messaggio della stessa partità

2.5.2 Firma digitale

Introduciamo ora la questione della firma digitale, facendo un parallelo con quella manuale:

Authentic Deve essere generabile solamente da una data persona, e quindi deve essere unica (un esempio è la calligrafia di chi scrive la firma).

Unforgeable Non deve essere facilmente falsificabile: non possiamo quindi utilizzare una firma come una stringa di bit da apporre in un documento, poiché altrimenti sarebbe evidente il plagio. (Nel mondo reale si distinguerebbe dalla calligrafia)

Not Reusable Segue che non deve essere riutilizzabile per altri documenti, impedendone sostanzialmente la copiabilità

Unalterable Una volta che il documento è stato firmato, non deve essere possibile cambiare il contenuto del documento.

Non repudiation Segue inoltre che, garantendo con la firma che l'utente mittente esiste, esso non potrà mai, una volta riconosciuto il documento come integro, ripudiarne il contenuto.

In quanto possiamo garantire con alcune funzioni la loro commutatività, possiamo definire delle funzioni in questo modo:

$$A : s = D(m, k_A[priv]) \Rightarrow \langle A, m, s\rangle \qquad B : \texttt{cmp}(C(s, k_A[pub]), m)$$

In questo modo avremo che il mittente firmerà il messaggio con la sua chiave privata che lo identifica precisamente (e quindi non può ripudiare il documento non manipolato), è inoltre possibile applicare D ad un qualsiasi valore numerico in virtù della proprietà di cui sopra, mentre conoscendo A il destinatario potrà conoscere la chiave pubblica del nostro mittente, che è disponibile a chiunque in modo gratuito. Inoltre, in questo modo, la firma non è riutilizzabile, poichè strettamente legata al documento che è stato firmato, che quindi non risulta modificabile a patto di minare l'integrità della codifica. Tuttavia con questa tecnica avremo un messaggio spedito dell'ordine di grandezza di $2|m|$, ed il messaggio non risulta confidenziale poiché ognuno lo potrebbe decodificare.[4]

Per ovviare a questo fatto viene utilizzato un secondo modello:

$$A : s = D(m, k_A[priv]) \wedge c = C(s, k_B[pub]) \Rightarrow \langle A, c\rangle \qquad B : m = C((\mathbf{s} = \mathbf{D}(\mathbf{c}, \mathbf{k_B}[\mathbf{priv}])), k_A[pub])$$

In questo caso non sono in grado di controllare se il messaggio è stato inviato correttamente oppure no, se non tramite la correttezza del messaggio dal punto di vista linguistico. A questo punto abbiamo che, se il messaggio è una sequenza generica di bit, non possiamo in alcun modo controllare l'integrità del messaggio o meno.

Segue che possiamo utilizzare un terzo protocollo, dove utilizziamo anche il digest cifrato del nostro messaggio (per effettuare un controllo sul messaggio) nel modo seguente.

$$A : s = D(f_h(m), k_A[priv]) \wedge c = C(m, k_B[pub]) \Rightarrow \langle A, c, s\rangle \qquad B : f(D(c, k_B[priv])) == C(s, k_A[pub])$$

In questo modo autentico il messaggio con la mia chiave pubblica, ma firmo il digest tramite una chiave privata che manitene tutte e cinque le proprietà di una firma.

[4] In quanto il messaggio è presente in chiaro, e perché l'altro è ottenibile in lettura tramite la chiave pubblica.

2.5.3 Autentificazione

Per quanto concerne il *Reply Attack*, dobbiamo inoltre considerare la validità di un messaggi in base al tempo che si impiega tra invio e ricezione dello stesso. Il problema è che un man in the middle, potrebbe ricevere il messaggio ed inoltrare ripetutamente il messaggio alla destinazione: si rende quindi necessario applicare al messaggio un *time stamp* che possa indicare la scadenza.

Ora, per stabilire l'identità di un utente, abbiamo che il messaggio deve essere riconosciuto come fattura di un determinato utente, e conseguentemente deve essere riconosciuta l'integrità del nostro messaggio.

Tramite una chiave segreta, condivisa da mittente e destinatario, si può ottenere un'immagine di dimensione costante rispetto alla potenziale lunghezza del messaggio, che può essere utilizzata per autenticare il messaggio, senza tuttavia possedere la proprietà di **non-repudiation**.

In questo caso possiamo utilizzare quindi un MAC (*Message Authenticatin Code*), tramite un messaggio breve e di lunghezza fissata, ottenibile tramite una chiave segreta condivisa tra mittente e destinatario.

Dopo aver definito come mk la concatenazione tra messaggio e chiave segreta alla fine dello stesso (che tuttavia entrambi conoscono), possiamo ottenere

$$A : \langle \mu = C_k(m), \omega = f_h(mk) \rangle \qquad \texttt{cmp}(\omega, f_k(D_k(\mu)k));$$

Abbiamo quindi identificazione poiché l'intruso non conosce k, ed aggiungendo il fatto che $f_h()$ è one-way, ammettiamo anche confidenzialità, ed in quanto non può modificare m, preserviamo l'integrità.

Capitolo 3

Gestione delle chiavi

3.1 Gestione delle chiavi pubbliche - Certificati

In quanto non dobbiamo trattare la gestione delle chiavi private nella crittografia asimmetrica, in quanto queste rimarranno sempre, dovremmo porre l'attenzione sulla divulgazione delle chiavi pubbliche.

Se un utente `A` effettua un annuncio pubblico della sua chiave pubblica (es.) per poter raggiungere anche un possibile interlocutore `B`, questa forma di comunicazione non richiede la presenza di intermediari, ma si ha il problema di un possibile *man-in-the-middle attack*: se lo comunichiamo direttamente all'interno di un canale di comunicazione, un ascoltatore malevolo `X` potrebbe cambiare tale chiave con la propria di modo che, quando `B` vuole aver garanzia di comunicare con `A`, utilizzerà la chiave pubblica che crederà di `B` ma che è in realtà di `X`, di modo che `X` riceverà la comunicazione da `B` che crederà di parlare con `A`. Questo potrebbe anche provocare il seguente disagio: entrambi i lati della comunicazione potrebbero credere di comunicare con il diretto interessato, senza aver modo di controllare che, effettivamente, la comunicazione è mediata da `X`. Si può quindi vedere che ciò è stato possibile perché **la comunicazione non garantisce la confidenzialità**.

Una soluzione possibile è quella di ritornare ad un ente centralizzato e fidato, che è responsabile della pubblicazione delle chiavi: tuttavia anche in questo contesto abbiamo che dei possibili "elenchi pubblici" possano essere falsificati, violandone quindi le informazioni. Inoltre, dobbiamo assicurarci che anche la registrazione delle informazioni debba avvenire in un modo sicuro, altrimenti l'informazione potrebbe essere (es.) compromessa persino qualora `A` chieda all'ente di aggiornare l'informazione; inoltre la stessa directory dove possono esssere presenti queste chiavi può essere al contempo non sicura, compromettendo tutti i dati contenuti al suo interno.

Possiamo quindi concludere che una soluzione praticabile è quella dei **certificati**, all'interno dei quali vengono trasportate le chiavi pubbliche: queste sono quindi sono rese autentiche dalla **firma** dell'autorità garante:

- In questo modo si verifica l'eliminazione dell'attacco *man-in-the-middle*, in quanto in questo modo non è possibile modificare la chiave pubblica nel certificato senza alterarlo

- Si garantisce l'attualità delle chiavi, mitigando il problema del furto della chiave privata

Dobbiamo ora generare un oggetto che possa prevenire l'attacco *man-in-the-middle*, ovvero dobbiamo fare in modo, durante lo scambio delle chiavi, che un terzo possa spacciarsi per un altro utente del sistema, cambiando semplicemente la chiave pubblica: come abbiamo già visto, possiamo avere questa garanzia tramite un **certificato** rilasciato (es.) da un'autorità garante (che applica la sua firma sul documento) verso la quale riponiamo fiducia: questo deve quindi incaricarsi di garantire che l'oggetto venga consegnato direttamente nelle mani di chi lo richiede, e deve essere in grado di accertare l'identità dell'individuo, che deve essere presente (assieme alla sua chiave pubblica) all'interno del certificato.

Per fare un esempio pratico, all'interno di un ambito ristretto, si può ritenere la mail del richiedente una garanzia sufficiente per ottenere un certificato, in quanto si ha la garanzia che solamente quel possibile destinatario sia in grado di poter leggere la mail (garanzia d'identità *out of band*). Tuttavia ciò non si rivela più sufficiente qualora siano richiesti ulteriori dati sul soggetto.

Definiamo inoltre **PKI** (*Public Key Infrastructure*) l'insieme dei servizi logicamente indipendenti fornibili da una rete di:

- **Registration Authority**: effettua l'ottenimento di una richiesta di un certificato da parte di un utente.
- **Certification Authority**: effettua il controllo sull'identità del richiedente, ed ha quindi il compito di rilasciare i certificati; si può realizzare tramite procedure automatizzate e non richiedendo il contatto diretto con l'interessato.

In genere le *PKI* contengono un insieme di servizi e di protocolli per fornire (tramite il processo di certificazione), immagazinare, **validare** (ovvero, controllare che il certificato sia da considerarsi tutt'ora valido) e revocare certificati. I *certificate servers*, che contengono al loro interno database di chiavi accessibili dalla rete, permettono ai loro utenti di effettuare una richiesta di inserimento di un nuovo certificato personale (processo di **certificazione**), o di ottenere quello di qualcun altro.

3.1.1 X.509

Questo certificato si basa sul concetto di "trust", che è presente anche nell'ambito umano: chi accetta come valido un certificato come un passaporto, implicitamente ripone fiducia anche nell'ente, detto *Certification Authority* che lo ha rilasciato.

La *X.509* è un ente di standardizzazione che utilizza un certificato con particolari campi:

- ◇ `Subject`: *Distringuished Name* (Common Name, Organization or Company, Oraganizational Unit, City/Locality, State/Province, Country (ISO code)), *Public Key*
- ◇ `Issuer` (colui che ha rilasciato il certificato): *Distinguished name*, *Signature*
- ◇ `Validity`: *Not Before Date*, *Not After Date*
- ◇ `Administrative Info`: *Version*, *Serial Number*
- ◇ `Extended Info`: ...

Come possiamo vedere dal campo `Subject`, non è sufficiente avere il solo *Common Name*, in quanto in questo modo non si potrebbero gestire i casi di omonimia. Inoltre, aggiungendo le

ulteriori caratteristiche di cui sotto, uno stesso soggetto può avere diversi certificati, in base al contesto nel quale li deve utilizzare.

Dobbiamo inoltre notare che, all'interno di uno stesso *PKI*, possono essere presenti anche più *Certification Authority* (CA) decentralizzati, creando in questo modo una gerarchia diffusa di fiducia all'interno del sistema. Inoltre, uno stesso CA può essere delegato da un PKI per certificare altri CA: in questo modo si ha a che fare con una gerarchia di CA, fino ad arrivare agli utenti, che costituiscono le foglie del sistema. Si può inoltre sottolineare come, uno stesso ente, possa rilasciare anche più tipologie di certificati, ad esempio, ad uno stesso utente.

Supponiamo ora di creare un certificato nella forma:

<Distinguished Name U, Public Key U, Signature CA X>

Per convincermi che questo certificato è valido, devo essere certo che anche la firma di X sia valida: dovrò quindi ottenere un ulteriore certificato per X firmato da un altro CA, strutturato in modo analogo a sopra ma con la firma di un altro CA; questa catena di certificati prima o poi dovrà essere interrotta con un'autocertificato di un'entità *Root Authority*, che ovvero pone la sua stessa firma sul suo Distinguished name e la sua chiave pubblica.

In genere questi certificati sono cablati all'interno di applicativi, di modo che possiamo conoscere a priori chi ritenere una Root Authority affidabile; in questo modo, è sufficiente che arrivi un certificato di una CA rilasciato da un Root noto, per ottenere garanzia sulla chiave pubblica in esso contenuta.

In seguito, dobbiamo considerare i casi con i quali possiamo ottenere la validazione o la revoca di un certificato:

Validazione tale operazione è necessaria in quanto l'informazione di un certificato può cambiare col tempo, e quindi abbiamo l'esigenza di controllare se tali certificati sono da considerarsi o meno *aggiornati* ed *autentici*. Si può effettuare questa operazione **offline**, ma a questo punto si può effettuare il controllo unicamente andando ad analizzare il certificato, e controllando il suo periodo di validità; l'operazione è inoltre fattibile anche **online**, ed in questo modo si può sempre chiedere ad ogni istante al CA che l'ha rilasciato se il certificato corrente è valido (uno strumento simile si potrebbe avere nel controllo offline tramite una lista dei certificati non più validi, anche se questo elenco non può essere tenuto in costante aggiornamento).

Revoca La revoca del certificato può avvenire qualora si reputi che le informazioni in esso contenute, non sono più valide. Questo può avvenire (*i*) perché la firma del CA, per un motivo grave, non è più valida, (*ii*) perché l'utente ha cambiato la sua chiave segreta - e contemporaneamente quella pubblica, (*iii*) le informazioni riguardanti l'utente cambiano. La revoca del certificato in modalità **offline** diventa critica, mentre il problema risulta triviale in modalità **online**.

3.1.2 PGP (Pretty Good Privacy)

Il sistema PGP introduce un nuovo sistema per l'attribuzione dell'affidabilità in un modello non gerarchico: in quello gerarchico invece, abbiamo che una root authority può assumere questo privilegio unicamente tramite una conoscenza *out of band*, la quale implica una conoscenza di come questo abbia operato nel passato.

Questo sistema invece, da la possibilità ad ognuno di autocerficarsi, garantendo inoltre relazioni di fiducia tra utenti, di modo da creare un grafo di relazioni di affidabilità, che può cambiare contestualmente al cambiamento delle relazioni tra utenti. Tale pacchetto software (disponibile anche freeware), consente l'uso di:

- ◇ Generazione, e gestione delle chiavi
- ◇ Cifratura e decifratura, firma e verifica di documenti digitali
- ◇ $\Rightarrow$ gestione di una VPN sicura.
- ◇ Creazione di Self-Decrypting Arhchives
- ◇ Cancellazione permanente di file, directory e spazio sul disco.

Il sistema di mantenimento delle chiavi, prevede di conservare per noi criptate secondo una *passphrase* ben conosciuta, le chiavi pubbliche e private di un utente che, al contrario delle password numeriche, possono essere ricordate facilmente dall'utente, ma sono ottenibili difficilmente tramite un attacco di forza bruta. In questo modo, l'utente può anche generare ma non vedere mai le sue chiavi pubbliche e private, ma è necessario che non si dimentichi tale *passphrase*.

Possiamo ora vedere quali azioni dovrà compiere il mittente per inviare un documento: una volta compresso il documento, egli genererà una chiave di sessione con la quale cripterà il compresso, codificandolo assieme a detta chiave con quella pubblica del destinatario. Quest'ultimo potrà aprire il suo documento tramite la sua chiave privata, che gli permetterà di accedere alla chiave di sessione per poi poter decifrare il messaggio a lui solo destinato.

Possiamo notare che, tramite la compressione, abbiamo un rafforzamento del protocollo poiché, in questo modo, si può evitare l'attacco statistico del plaintext: comprimendo il testo apparirà di natura casuale, in quanto non si mantiene l'ordinamento del documento, poiché le lettere verranno mappate in un modo differente. Altro vantaggio che si ottiene tramite la compressione è la ridotta dimensione del messaggio da inviare a destinazione.

All'interno dei cifrari a chiavi pubbliche, in genere queste chiavi vengono sempre comunicati tramite certificati, che sono strutturati in un modo differente rispetto a *X.509*:

- ◇ Chiave pubblica di U e caratteristiche della chiave (lunghezza, algoritmo, data di creazione e sua durata)
- ◇ Informazioni sull'identità di U
- ◇ Auto-firma di U
- ◇ Indicazione dell'algoritmo simmetrico di codifica

Inoltre il sistema PGP è in grado di gestire (e quindi riconoscere) il sistema X.509, anche se non è supportata l'interoperabilità dei due sistemi, nel senso del riconoscimento nei certificati PGP quelli X.509. Mettiamo ora in luce le differenze tra questi due sistemi di Key Management:

- In PGP non esiste il concetto di "Registration Authority", come invece avviene per l'altro standard;
- In PGP il certificato nasce autofirmato, mentre in X.509 viene firmato dalla suddetta "Registration Authority";

- In PGP possono essere ammesse identità multiple per uno stesso utente, mentre X.509 deve essere in grado di identificare singolarmente ciascuna di quelle entità per evitare l'ambiguità

- Si ripone fiducia nei confronti di un certificato non solo dal numero di firme in esse presenti, ma anche da quante firme sono presenti nei certificati dei firmatari.

Per conesntire le identità multiple, all'interno di uno stesso certificato possono essere presenti coppie multiple chiavi-identità, ciascuna delle volte firmate un numero non precisato di volte e ciascuna con ruoli differenti, e utilizzi in differenti contesti.

Il certificato quindi nasce già da subito come autocertificato, e a questo punto ogni utente diventa in un certo modo come "autority" di sè stesso, senza essersi rivolto ad alcuna autorità; per testare l'autorevolezza del certificato, verrano esposte delle altre firme all'interno dello stesso di altri utenti, alle quali si può assegnare i seguenti gradi di attendibilità: (*i*) **complete** trust, (*ii*) **marginal** trust e (*iii*) **untrusted**. Inoltre, all'interno di questo contesto, dobbiamo essere in grado di dividere la **fiducia** che riponiamo su di un utente, dalla **validità** del certificato che esso utilizza: potrei riporre fiducia in un certificato perché esso mi viene presentato da quella persona in persona, attestando quindi la validità del certificato, e sostituendo in questo modo la certificazione di un singolo utente. Possiamo quindi distinguere tre modelli differenti di affidabilità:

Direct Trust ottengo la fiducia solamente delle persone con le quali interagisco direttamente

Hiearchical Trust la fiducia viene riposta nei confronti di un ente emittente in modo assoluto, tramite una conoscenza out-of-band della sua affidabilità

Web of Trust Si viene a creare un grafo arbitrario di affidabilità. Non esistendo una CA, tutti potranno essere CA in quanto tutti hanno la possibilità di firmare i certificati presenti. Inoltre, il primo utente che mette una firma in un certificato viene definito come **introducer**, ed ognuno può considerare più o meno affidabile quel certificato, in base alla firme delle entità presenti all'interno di quel certificato. La struttura di una "rete" è anche dovuta all'assenza di autorità centrali di certificazione. Tuttavia le relazioni del grafo nè sono da considerarsi simmetriche, nè sono da considerarsi transitive, in quanto non sempre la fiducia, anche nella vita reale, è una relazione di quel tipo, e contemporaneamente quei valori settati all'interno del certificato per ogni firma, possono cambiare con il tempo.

Abbiamo inoltre che, se una persona riceve un'affidabilità del tipo *Completely trust*, essa diventerà come una "CA", mentre una chiave è considerata come valida se almeno un utente l'ha indicato come *Completely trusted*, o più di due firme *Marginally trusted.*

Possiamo fornire alcuni dati:

- *Le chiavi segrete da 80b sono equivalenti a quelle pubbliche da 1024b; le chiavi segrete da 1028b invece sono equivalenti a quelle pubbliche a 3000b*

- *In quanto le chiavi segrete a 56b oggi non sono sicure, e quelle a 128b magari non lo saranno nel futuro coi computer quantistici, quelle a 256b sono ritenute sufficientemente sicure anche in un futuro lontato*

- *Gli algoritmi a chiave privata utilizzata sono CAST, Triple-DES, IDEA ed AES; quelli a chiave pubblica sono DSS ed RSA, mentre come algoritmo di HASH si usa SHA1.*

3.2 Gestione delle chiavi private di crittografia assimetrica

In quanto dobbiamo ricordare che entrambe le parti della comunicazione devono essere a conoscenza della sua chiave privata, tramite la quale si assicura la sicurezza della comunicazione all'interno del canale; tuttavia si ha il problema che, entro un certo periodo di tempo, bisogna considerare l'eventualità che questa chiave possa essere stata scoperta, rendendosi quindi necessario il suo cambiamento. Da ciò pertanto nasce l'esigenza di gestire opportunamente le chiavi associate al canale di comunicazione.

3.2.1 Protocollo di Needham-Schroeder

Assumiamo pertanto che esista un'autorità fidata che indicheremo con **KDS** (acronimo per *Key Distribution Server*), che conosce a priori una chiave privata associata a ciascun utente del sistema[1], ed è quindi è riconosciuta come autorevole da tutti gli utenti del sistema (si preoccupa quindi di non divulgare le informazioni degli altri utenti o comunque di renderle sicure, e che quindi potrà subire un attacco con successo con una bassa probabilità). Se vogliamo che `A` e `B` possano comunicare privatamente, possiamo fare in modo che tale `KDS` fornisca una nuova chiave di sessione, ogni volta che si vuole stabilire una nuova comunicazione. Per evitare attacchi di replica di messaggi (ovvero, un intruso potrebbe benissmo entrare in possesso di un messaggio, ed inviarlo ripetutamente a destinazione), verranno utilizzati come segnatempo dei **nonces** (*number user once*) che indicheremo con N_i.

1. `A` invia un messaggio al `KDS` nella forma $\{ A, B, N_1 \}$
2. `il KDS` invia la chiave di sessione k_S ad `A` nella forma:

   $$C(k_A, \{ k_S, A, B, N_1, C(k_B, \{ k_S, A \}) \})$$

   In questo modo il messaggio potrà essere solamente letto da A, ed inoltre il contenuto $\{ k_S, A \}$ potrà essere letto solo da `B` e non da `A` in quanto solo `B` conoscerà la sua chiave privata. A questo punto, il valore di k_A può anche essere dimenticato da `A`, ovvero può uscire dal sistema.
3. `B` invia ad `A` una *challenge* $C(k_S, N_2)$
4. `A` risponde a `B` con una *response* $C(k_S, N_2 + 1)$. Con $N_2 + 1$ intendiamo una qualsiasi modifica al N_2 prevista dal protocollo in qualche modo, di modo da evitare attacchi che prevedono unicamente la risposta con ricopiatura del messaggio.
5. A questo punto la comunicazione all'interno del canale è da ritenersi sicura

Ora risulta ovvio che se `A` vuole comunicare con un altro `C`, debba rieffettuare il meccanismo di cui sopra per ottenere un'altra chiave segreta per la comunicazione.

3.2.2 Kerberos

Un'ulteriore modifica allo schema precedente è presentato dallo schema **Kerberos**, con l'unica differenza che si interpone un `TGS` (*Ticket Granting Server*), con il quale, comunicando

[1] Se così non fosse, si avrebbe un problema ulteriore da gestire per quanto riguarda lo scambio di quella chiave primaria all'interno del sistema.

col TGS con la chiave k_S del KDS (per tutta la sessione si scambierà k_s e non il segreto k_A) , si tende a minimizzare la vulnerabilità iniziale della diffusione della chiave k_{AB} (fornita da quest'ultimo) possa essere intercettata da un ulteriore uditore malintenzionato. Perché k_s è generato casualmente, spesso KDS è all'interno dello stessa implementazione dove è anche presente TGS, ma giocano due ruoli differenti all'interno del sistema. Ulteriori differenze sono le seguenti:

- A, per stabilire una nuova comunicazione con un elemento differente del sistema, dovrà ricominciare la sua comunicazione con TGS utilizzando la chiave k_S generata automaticamente, e non con il segreto k_A.
- Per evitare in un sistema imponente che sia proprio il KDS il collo di bottiglia (inoltre, se si attaccasse questo unico punto, il sistema salterebbe), si possono replicare tali elementi secondo un principio di *master-slave*:
  - si può decentralizzare per evitare il DoS, e quindi far utilizzare il server KDS più vicino; può essere anche utilizzato per effettuare load-balancing delle richieste, qualora ci siano più KDS attivi, in modo da evitare che uno stesso server risulti subissato dalle richieste.
  - si deve inoltre creare un server *master* e gli altri *slave* in quanto, se i dati che vengono gestiti sono del tipo *Read-Write*, come in questo caso la gestione delle chiavi (lettura ed aggiornamento), possiamo avere problemi di concorrenza sulla modifica dei valori, in quanto in parti differenti del sistema, potrei in uno stesso momento ottenere valori differenti per uno stesso dato. In quanto l'atto della modifica della chiave è più rara della sua lettura o comunque controllo, seguendo un approccio *master-slave*, non si introducono sostanziali colli di bottiglia.

Sebbene questo sistema garantisca la riservatezza e l'autenticazione, e mantiene buone prestazioni anche con un frequente cambiamento delle chiavi, riducendo inoltre il numero delle chiavi da mantenere da n^2 ad n, rimane il grosso svantaggio della solidità del server KDS. Possiamo quindi mantenere un (primo) approccio ibrido nel seguente modo:

- A genera una coppia $(pub_A, priv_A)$
- A invia a B pub_A, A: *questa parte rimane soggetta a man-in-the-middle attack*, ovvero non garantisce autenticazione.
- B genera casualmente la chiave di sessione k_S e la invia ad A: $C(pub_A, k_S)$
- A ottiene quindi la chiave pubblica dal messaggio ricevuto adoperando la sua chiave privata
- A questo punto A può dimenticare la sua coppia di chiavi, utilizzando all'interno del canale unicamente la crittografia asimmetrica

I problemi di cui sopra possono essere risolti nel seguente modo, supponendo che le chiavi private e pubbliche siano già distribuite:

◇ Le chiavi pubbliche sono già state distribuite

◇ Si garantisce autentificazione (per evitare il man in the middle attack) tramite il TWH $C(pub_B, \{ A, N_1 \}) \rightarrow C(pub_A, \{ N_1, N_2 \}) \rightarrow C(pub_B, N_2)$

◇ Si garantisce riservatezza tramite la chiave privata e pubblica: A $\rightarrow$ B : $C(pub_B, D(piv_A, S))$

3.3 Key Escrow and Secret Sharing

Con il termine **escrow** si vuole intender un concetto che va oltre al semplice deposito di un bene (in questo caso la chiave): con questo termine si intende infatti la consegna di questo ad un terzo, dal quale possiamo riottenerlo in base ad una condizione che si è verificata. Vedremo ora quali problemi ponga la custodia di tali "segreti" (le chiavi), in base all'affidamento ad una o più persone.

Supponiamo inizialmente che la chiave o una password per accedere ad alcune risorse condivise sia conosciuta da una sola persona; questi possono essere la chiave privata di crittografia asimmetrica, o la chiave di crittografia simmetrica, o dei dati presenti all'interno di un calcolatore che, non verranno mai comunicati, ma che non vogliamo che siano accedibili dall'esterno. Vogliamo inoltre che l'utente possa accedere in un modo trasparente ai dati, senza conoscere necessariamente tale segreto.

Se il segreto rimane custodito unicamente da quella persona, ci si pone il problema di come accedere alle informazioni in caso di sua assenza, se per una serie di motivi queste non sono conseguentemente più disponibili. Vorremo inseguito che anche altri individui autorizzati possano accedere a quei dati, anche in mancanza del personale irreperibile. Introdurremo quindi questo problema per gradi.

3.3.1 Security officer

Una possibile soluzione al problema si potrebbe avere con un **secutiry officer**, che custodisce per l'azienda il segreto che è stato affidato ad un altro utente, in modo tale che si possano sempre recuperare i dati anche in mancanza di quel preciso utente. Questo segreto potrà essere inserito all'interno di un qualsiasi supporto, di modo che il security officier non conosca tale segreto se non in quella forma: ciò potrebbe essere utile per effettuare controlli sul sistema, in modo da controllare le operazioni svolte dal sistema quando l'utente privilegiato è assente, dando a quest'ultimo eventualmente un avviso, qualora il controllore agisca con il suo stesso segreto quando invece è in servizio.

In questo modo, tuttavia, si riavrebbe da capo lo stesso problema: la mancanza di questo security officier potrebbe causare a sua volta la rovina dell'intero sistema: si potrebbe quindi dividere la nostra chiave in n parti ed affidarla a rispettivi security officiers, criptando quei pezzi con le loro chiavi pubbliche. In questo modo vogliamo far diminuire esponenzialmente la probabilità che, alla mancanza di un agente, il segreto non sia più custodibile.

Possiamo inoltre ben vedere che non potremo dividere esattamente in k parti la chiave, poiché in questo modo sarebbe possibile ricostruire con poco sforzo la parte rimanente: se $k = 2$, saremmo a conoscenza di metà del segreto, facendo così scendere la possibilità di riconoscimento della chiave da una sua parte da p^k a $p^{\frac{1}{2}k}$ (con questo appunto intendiamo una qualsiasi suddivisione della risorsa in senso effettivo, ovvero un suo partizionamento).

Passiamo ora ad analizzare un caso immediato di "suddivisione", con $k = 2$: in quanto sappiamo che $x \oplus x \oplus y = x \oplus y \oplus x = y$, allora generato un valore random $\mathtt{R}$ della stessa lunghezza del segreto $\mathtt{S}$, possiamo consegnare ad un agente $\mathtt{R}$ ed all'altro $\mathtt{R} \oplus \mathtt{S}$. In questo modo, solamente con la compresenza dei due, si potrà ricostruire il segreto iniziale. In questo modo inoltre non si rileva più necessario mantenere il segreto iniziale $\mathtt{S}$. Si può dimostrare inoltre che questo metodo è di fatti **confidenziale**, in quanto nessun agente trae alcun vantaggio dalla conoscenza della loro parte di chiave; sebbene questo sia evidente per il detentore di $\mathtt{R}$, questo risulta meno ovvio per $\mathtt{R} \oplus \mathtt{S}$, in quanto ad ogni modo si intravede un legame

con $\mathtt{S}$ ma, essendo ottenuto $\mathtt{R}$ con una stringa casuale, ci si ricondurrebbe alla ricostruzione della stringa casuale $\mathtt{R}$, esattamente come nel cifrario perfetto *one-time-pad*. Possiamo generalizzare questo comportamento tra n agenti assegnando ad $n-1$ agenti i valori random $\mathtt{R}_1 \dots \mathtt{R}_{n-1}$, ed assegnando all'ultimo il segreto $\mathtt{R}_1 \oplus \dots \mathtt{R}_{n-1} \oplus \mathtt{S}$. Tuttavia in questo modo non garantiamo la disponibilità, in quanto dobbiamo ora garantire che **tutti** i detentori del segreto siano sempre presenti, peggiorando quindi la probabilità che, alla mancanza di uno, il segreto risulti irrecuperabile.

Segue necessariamente che, per garantire anche la **disponibilità**, dobbiamo indebolire lo schema inizilae, in modo che per conoscere il segreto non siano necessari tutti gli n agenti, ma che siano presenti in un numero $t < n$ utilizzando uno schema a soglia (*theresold scheme*): in questo modo si impone che siano necessari solamente t utenti per poter accedere al segreto.

Ciò è risolubile non generando gli $\mathtt{R}_i$ in modo casuale, ma dei polinomi casuali di grado $t-1$:

$$g(x) = (\sum_{i=1}^{t-1} a_i x^i + \mathtt{S}) \bmod p$$

dove p è un numero *primo* più grande dei valori costanti scelti arbitrariamente a_i, ovvero maggiore del $\max \{ a_i \}_{i<t}$; in seguito ad ogni agente verrà associato un segreto:

$$\forall i \leq n . \mathtt{S}_i = g(i)$$

in questo modo garantiamo che la curva $g(i)$ sia ricostruibile conoscendo solamente t valori che stanno su di essa.

Capitolo 4

Autentificazione

4.1 Internet Security: SSL

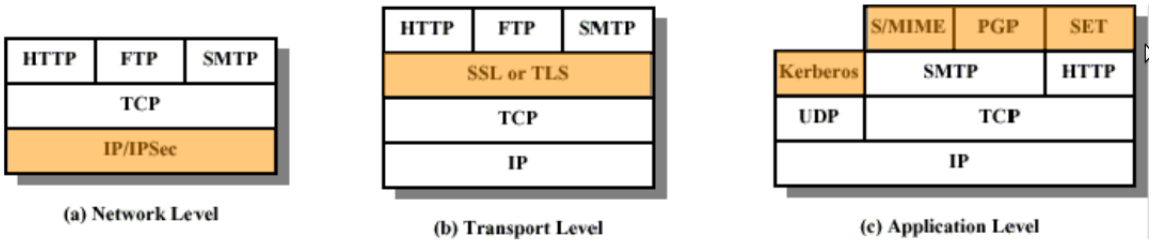


Figura 4.1: *Strati a livello OSI ove è possibile introduree SSL.*

Nel caso della crittografia di chiave pubblica, abbiamo che, per l'utente medio, sarebbe necessaria un'infrastruttura talmente complessa che sarebbe impossibile da mantenere. Come si può vedere dalla figura 4.1, in quanto a livello TCP/IP non è stato introdotto alcun criterio di sicurezza, questa è introducibile nei seguenti livelli OSI:

Network Level grazie ad **IPSec**, si inserisce la sicurezza a livello IP. Tuttavia, con questo metodo, anche se si può garantire la sicurezza per quelle applicazioni che non la richiedono, assicurando così comunicazioni protette all'interno di una rete potenzialmente insicura come quella pubblica, si dovrebbero effettuare molteplici modifiche a livello di sistema operativo. Inoltre, non è detto che i routers dispongano dell'architettura necessaria per poter gestire questi pacchetti: i router(s) infatti hanno la necessità di leggere il loro contenuto, per poter instradarli successivamente: se il router non riconosce quindi il pachcetto, la comunicazione con un altro componente della rete non sarebbe possibile, in quanto questo scarterebbe tale pacchetto.

Transport Level inserendo lo strato di sicurezza tra livello applicativo (HTTP) e di trasporto (TCP), si ha il vantaggio di poter dialogare con un nuovo livello (non più direttamente con i socket TCP) che garantisce in un modo immediato (simile a quello fornito dai socket) la sicurezza, anche se risulta necessario modificare gli applicativi, ma in una misura molto minore rispetto alle modifiche che è necessario apportare a livello di sistema operativo.

Application Level inserendo lo strato di sicurezza a questo livello, si ha la possibilità di garantire la sicurezza ai programmi, customizzandola per la particolare applicazione che

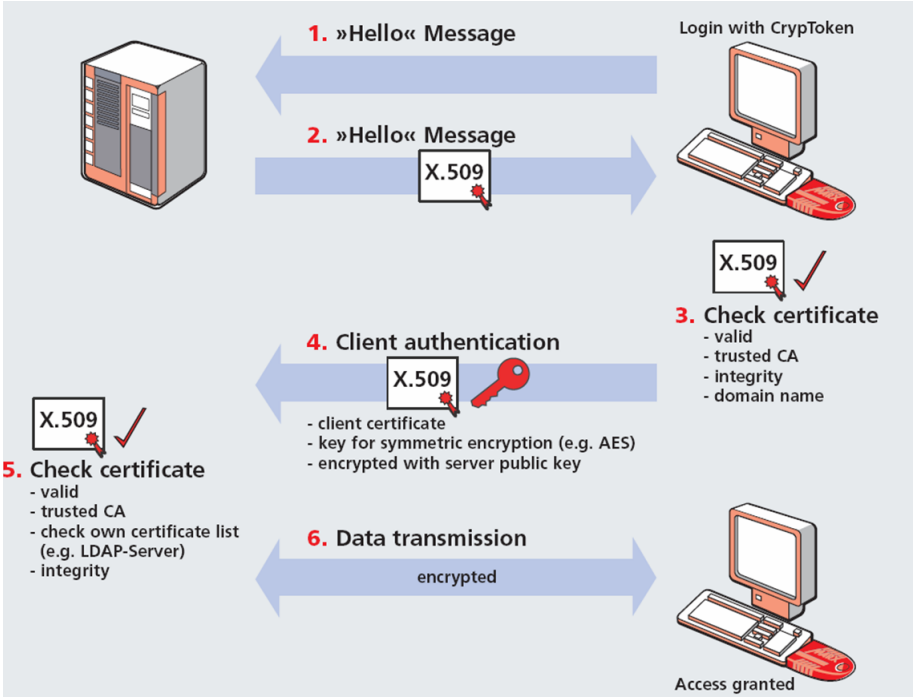


Figura 4.2: *Protocollo di instaurazione della comunicazione SSL.*

la richiede, anche se questo richiede la riscrittura della sola applicazione, e l'adozione di molteplici meccanismi di sicurezza.

Parliamo ora di come garantire sicurezza a livello di trasporto mediante SSL. Questo nasce come protocollo di Netscape, allo scopo di garantire sicurezza all'interno delle comunicazioni di e-commerce, che fino a quel momento non erano possibili in quanto Internet era nato come uno strumento di comunicazione insicuro. Questo strumento ha quindi lo scopo di garantire **confidenzialità** e **autentificazione** della comunicazione in Internet. Questa è inoltre una semplice reimplementazione di tecnologie preesistenti che vengono tuttavia automatizzate, in modo da rendere immediata la comunicazione all'utente, senza che questo debba preoccuparsi dei meccanismi di comunicazione che avvengono durante una normale interazione, analoga a quella che avviene in assenza di comunicazione sicura, dando ad esempio origine alle seguenti applicazioni che avvengono in maniera protetta, come HTTPS, SSMTP, SFTP e SSH.

L'implementazione di SSL prevede una fase di **SSL Handshake** che è utilizzata per creare un canale sicuro, affidabile ed autentificato tra client e server, fornendo i meccanismi di *cifratura* ed *autentificazione* delle chiavi, all'interno del quale poi **SSL Record** farà viaggiare i messaggi incapsulandoli in blocchi cifrati ed autentificati, utilizzando la suite fornita dallo Handshake, e fornendo i dati così ottenuti allo strato sottostante di TCP. Tuttavia, dobbiamo essere in grado di fornire queste garanzie di sicurezza anche se il client ed il server, attori della comunicazione, non si conoscono tra loro.

Viene quindi utilizzato un protocollo descritto dall'immagine 4.2: come possiamo vedere:

1. Il client invia al server un messaggio di *hello*

2. Il server risponde con un *hello server*, fornendo il suo certificato (ed in rari casi richiedendo quello del client: di massima questo non viene effettuato, in quanto i certi-

ficati SSL sono costosi da ottenere da un ente certificante). In questo caso avviene l'**autentificazione** del server verso il client.

3. Il client a questo punto analizza il certificato del server: a questo punto l'applicativo dell'utente potrebbe anche non riconoscere come valido il certificato ottenuto, in quanto potrebbe non essere riconosciuta la Root Certification Authority (che potrebbe fare riferimento allo stesso server con il quale stiamo instaurando la connessione) che ha fornito il certificato. A questo punto l'utente potrebbe o decidere di rifiutare di instaurare la comunicazione, oppure decidere di accettarlo ritenendolo validoo grazie ad una conoscenza da lui posseduta "out-of-band". Se il client non dispone di un certificato SSL, allora può essere richiesta una password (assieme ad un login name), con la quale potrà comunicare con il server senza esporre un certificato. Tramite questo sergreto condiviso si garantisce la **confidenzialità** del protocollo

4. In seguito, se la comunicazione è accettata dal server, la crittografia diventa di tipo simmetrico: d'ora in poi, per rendere la comunicazione sicura, si utilizzerà sempre la chiave privata fornita dal client.

A questo punto dobbiamo distinguere la **SSL Session**, la quale è un'associazione tra client e server a lungo termine, che si instaura con lo *Handshake protocol* e termina con l'uscita dal dominio server, dalla **SSL Connection**, che è scatenata da ogni interazione che avviene con il server all'interno di una stessa sessione.

4.2 Sistemi Operativi

Gli utenti umani non sono in grado di ricordare delle chiavi (pubbliche o private) sicure, mentre ciò potrebbe essere valido per applicazioni, come nel caso precedente, che dialogano tra loro, e nemmeno sono in grado di effettuare dei calcoli complicati, come quelli richiesti dagli algoritmi di crittazione e decifrazione.

Si rende quindi opportuno utilizzare delle regole di autentificazione, in modo da permettere gli accessi solo ad utenti umani, o solamente alcuni di questi. In prima istanza la coppia login-password serve proprio per garantire l'**autentificazione**: infatti una e-mail (come esempio di login) può **identificare** un solo essere umano, di cui la password associata mi garantisce che colui che vuole accedere, è proprio il soggetto che possiede quel dato account. Ricordando ora i *Golden Standard of security*, possiamo verificare che l'**autentificazione** è necessaria per fornire **autorizzazione** all'accesso di determinati utenti, tramite il quale posso anche controllare con **auditing** le azioni che vengono compiute.

Possiamo quindi utilizzare differenti tecniche per identificare esseri umani, chiedendo loro:

- ◇ *qualcosa che solo loro conoscono*: questo potrebbe essere un codice PIN o una passphrase
- ◇ *qualcosa che solo loro possiedono*: questo potrebbe essere un token, come ad esempio una smartcard, o comunque un oggetto che deve essere esibito.
- ◇ *qualcosa che solo loro fanno*: gli esseri umani possono compiere le stesse azioni richieste in un modo leggermente differente tra loro
- ◇ *qualcosa che solo loro sono*: gli esseri umani hanno delle caratteristiche che li distinguono tra loro, e conseguentemente possono essere utilizzate per effettuare una qualche forma di distinzione tra di loro.

- ◇ *dove ti trovi*: possiamo anche garantire l'autentificazione dicendo che un dato utente si trova in una determinata posizione geografica.

L'unica alternativa che non necessita dello hardware aggiuntivo è la prima opzione, in quanto le caratteristiche necessarie saranno comunque ovunque disponibili, mentre lo hardware per la rilevazione biometrica non è sempre presente.

Dopo aver scelto la prima alternativa, bisogna effettuare opportuni accorgimenti: se avviene una violazione della politica di sicurezza, questo metodo non lascia nessuna traccia dell'abuso, e quindi non garantisce la distinzione tra utente acceduto lecitamente e quello acceduto indebitamente. Questo perché le password dei login associati possono essere carpite in un modo più meno semplificato:

- ◇ Password banale od indovinata
- ◇ Password appuntata in un luogo molto sicuro
- ◇ Login spoofing
- ◇ Sniffing di rete (il segreto viaggia comunque in una rete)

4.2.1 Metodi d'attacco delle password

Possiamo distinguere principalmente due metodi d'attacco delle password:

Attacco OffLine questo tipo d'attacco utilizza un altro sistema, all'interno del quale si tenta di minare l'integrità del sistema da attaccare. Per effettuare questo tipo di attacco tuttavia, è necessario avere accesso ad una qualche informazione sulle password del sistema.

A questo punto ci preoccupiamo della modalità con la quale effettuo il salvataggio delle password: se si memorizzassero queste all'interno di un file in chiaro, protetto unicamente dai metodi di controllo dei permessi all'interno del sistema operativo, potremo avere che utenti privilegiati malintenzionati potrebbero accedervi abusando delle garanzie di privacy.

Conseguentemente dobbiamo garantire in un qualche modo la **criptazione** delle password, utilizzando una funzone *one-way hash* ottenendo così il fingerprint delle nostre funzioni. In questo modo si può controllare la correttezza della password immessa se, per un dato utente, il suo fingerprint coincide con quello generato dalla password immessa. Tuttavi in questo modo è semplice ottenere un **dictionary attack** con il quale, una volta ottenuta l'associazione tra parola e digest, si possa controllare se quel dato valore è presente all'interno di `/etc/passwd`. In questo modo, se ho trovato un match, posso dire di aver trovato una password che ha lo stesso hashing di quella inserita all'interno, riuscendo quindi ad accedere con quel login senza necessariamente conoscere LA password di quell'utente.

A questo punto, possiamo effettuare modifiche banali sulle parole del dizionario, quali permutazioni, per ottenere altre possibili password in chiaro presenti all'interno del sistema.

Questa tecnica è inoltre possibile in quanto l'attaccante è a conoscenza del metodo di codifica che viene utilizzato dal sistema, in quanto questa infomrazione è di fatti pubblica.

In quanto non possiamo gestire in alcun modo il tempo che può occorrere tra un tentativo di attacco ed il successivo, in quanto questi avvengono all'interno del sistema dell'attaccante, possiamo andare ad agire sulle potenze di calcolo di quest'ultimo, rallentando la performance di calcolo delle funzioni di hash utilizzando (es. in Unix) 25 volte di seguito la funzione DES. Altri modi per aumentare la sicurezza possono essere quelli di adottare una politica di "salting", o dell'ospitare i codici digest in un altro file, differente dall'`/etc/password`, ovvero `/etc/digest`, leggibile solamente a root: in questo modo nel primo file, il digest della password viene sostituito da una `x`.

Attacco Online questo tipo di attacco utilizza lo stesso sistema da minare per poter determinare la correttezza del segreto che si vuole indovinare. Per limitare questo tipo di attacco si può effettuare:

- ◇ Si può effettuare un ritardo del tempo fornito per poter effettuare i tentativi successivi, in modo da rallentare l'esecuzione del programma che effettua l'attacco.
- ◇ Si può limitare il numero di tentativi errati, di modo da bloccare i tentativi successivi; in questo modo tuttavia si impediscono futuri tentativi di accessi legittimi di uno stesso login, rendendo quindi possibile un attacco DoS nei confronti di un utente.
- ◇ Visualizzare all'atto del login l'ultimo tempo di accesso effettuato.

Di seguito proponiamo alcune tecniche di attacco OnLine:

Login Spoofing All'utente in questo caso, viene fatto credere di dialogare con il sistema anche se si trova di fronte ad un "terminale fasullo", il quale memorizza le sue credenziali e poi gli rifiuta l'accesso al sistema che l'utente crede reale, effettuando un redirect a questo in un secondo momento.

A questo punto, bisogna ricorrere alla nozione di **trusted path**, con la quale si indica un cammino, che si garantisce sicuro e quindi inattaccabile, tramite il quale è possibile effettuare la comunicazione delle password. Un esempio di questo particolare trusted path all'interno dei sistemi Windows si ha con la combinazione di tasti `CTRL-ALT-DEL`: in questo modo generiamo un particolare segnale, tramite la quale possiamo effettuare il lancio del login utente, in modo da essere certi di dialogare con il sistema o con uno spoofer. Questa gestione viene effettuata ad un livello più basso rispetto a quello dei driver di sistema, e per tanto non è intercettabile dagli applicativi che sono in esecuzione all'interno del sistema operativo.

Tuttavia, proseguendo con un procedimento di tipo paranoico, potremmo arrivare ad immaginare a questo punto che sullo stesso processore ci possa essere presente un altro sistema operativo diverso da Windows, che è stato tuttavia modificato dallo spoofer in modo da apparire identico al precedente: per uscire da questo sarebbe sufficiente effettuare il reboot, a patto che lo spoofer abbia modificato anche il Master Boot Record. A questo punto l'unica soluzione possibile sarebbe quella di riformattare il disco e reinstallare il sistema operativo, ma da un CD originale, e non da uno "modificato".

A questo punto è impossibile essere in grado di stabilire una regola precisa per avere una garanzia in sicureza: tutto si basa conseguentemente su di una mera considerazione personale.

Per poter premunirsi da questi attacchi, si rileva necessario il meccanismo di **mutua autentificazione**: mentre classicamente è solamente un utente ad autenticarsi verso il

sistema, che tuttavia poteva essere fittizzio, ora un meccanismo di **challenge-response**; il client, prima di rispondere al login, effettua una "sfida" con il sistema e, solamente se questo risponde correttamente, procederà con l'effettuare la procedura di login per accedervi, in un modo analogo a quello che avviene tramite i certificati SSL.

Phishing Oggigiorno, il login spoofing può anche essere effettuato tramite siti web che effettuano phishing, i quali mimano l'interfaccia del sito al quale l'utente effettuerà il login, esattamente come sopra, allo scopo di carpirne le informazioni. Questa forma di "login spoofing" oggi giorno di solito avviene tramite mail: tuttavia, grazie ai sistemi di DNS gratuiti quali **OpenDNS** e `8.8.8.8` di Google, oltre a fornire una notevole cache (grazie alle ricerche effettuate in precedenza) per la conversione automatica l'indirizzo mnemonico in indirizzo IP, possono fornire un ottimo supporto per l'antiphishing: di fatti i siti che effettuano questa politica di spoofing non cambiano frequentemente, ed è quindi possibile rintracciarli (questi servizi lavorano inoltre ad un livello superiore a quello dei standard Root DNS di Internet), ed avvisare i visitatori del sito del tipo di contenuti in esso presenti.

Spear-Phishing In questo caso si intende "lanciare un arpione per pescare pesci più grossi": il senso di questo metodo di attacco è quello di effettuare uno spam mirato all'interno di un azienda o comunque di un ambiente ben noto, con un'impostazione di contenuti molto simile a quella che si potrebbe ricevere all'interno di un certo ambito lavorativo. Questi attacchi sono conseguentemente molto più difficili da identificare

Keyloggers Il *keylogger* in genere è un porgramma che viene installato dall'amministratore, e permette di temere una traccia di cosa l'utente ha immesso all'interno del sistema da tastiera o da mouse, in modo da poter tracciare tutte le operazioni che lui stesso ha compiuto. Questi comportamenti infine sono analizzabili tramite un file di log finale, in modo (es.) da controllare a quali siti si effettua l'accesso.

Tuttavia, nell'ambito del **malaware/spyware**, questi applicativi possono essere installati anche a nostra insaputa, effetutando quindi l'inoltre delle informazioni a riguardo ad un hacker sconosciuto, che verrà a conoscenza delle password che abbiamo immesso all'interno del nostro sistema, tramite installazione di programmi di:

- FileSharing: molti di questi programmi contengono un payload nascosto, ovvero un keylogger che si introduce all'interno del sistema
- Attachment alle mail di file binari, quali PDF o immagini

Per evitare l'installazione di questi Malaware/Sypware, si possono installare dei sistemi per cancellare questi programmi una volta riconosciuti.

Oggigiorno, molti siti, tra cui quello della VISA, hanno modificato l'interfaccia utente per l'inserimento dei dati: questa presuppone la potenziale presenza di malaware all'interno dei sistemi, mettendo conseguentemente la possibilità di inserire il segreto all'interno del sito, non tramite utilizzo della tastiera, ma tramite una tastiera virtuale dalla quale selezionare i bottoni. Tuttavia, in quanto i keyloggers possono anche riconoscere in quale posizione dello schermo è stato effettuato il click, si sarebbe in grado (ad ogni modo) di ricostruire la sequenza esatta dei bottoni premuti. Tuttavia, onde evitare questa possibilità, le sopracitate tastiere permettono la ricombinazione casuale dei tasti in essa contenuti.

In quanto abbiamo detto che tale keylogger raccoglierà le sue informazioni all'interno di un file, ed in quanto certamente lo hacher sarà collegato in remoto con l'attaccato, potendo, quando questo si collega ad internet, ottenere le informazioni contenute all'interno del file. A questo scopo potremmo impedire l'accesso in internet verso indirizzi sconosciuti o da programmi sconosciuti.

Packet Sniffing In quanto il nostro pacchetto può essere intercettato all'interno di una rete pubblica che, in quanto permette lo scorrere in chiaro delle informazioni in esso presenti, può permettere un attacco passivo da parte di un uditore all'interno della rete. In questo modo, se all'interno della rete transita un pacchetto in chiaro contentente una password, lo sniffer può immediatamente attaccare quell'account. Per evitare questo, si possono utilizzare, oltre a quelle crittografiche, le seguenti tecniche:

- Invece di effettuare una immissione unilaterale dell'utente, si può fare in modo che l'utente immetterà la sua password solamente dopo che il sistema abbia risposto correttamente alla sua sfida. (**Challenge-response**)
- Invece di immettere ogni volta una stessa password, si può fare in modo da utilizzare **password monouso** di modo che, se queste venissero intercettate, non sarebbero più utilizzabili da un attaccante.

Ecco alcuni consigli che sono utili da impartire ad un amministratore di sistema:

◇ Mai lasciare delle password di default all'interno del sistema: si avrebbe altrimenti una grossa vulnerabilità in quanto la password sarebbe pubblica, e quindi riconoscibile da tutti.

◇ Gli utenti non degono utilizzare delle password banali, perchè sarebbero facilmente privedibili da un attaccante, conoscendo alcuni pochi dati sensibili dell'utente.

◇ Ogni periodo di tempo l'amministratode del sistema dovrebbe provare a minare l'integrità del sistema tramite emulazioni di possibili attacchi, per comprovarne la robustezza. In questo modo, l'amministratore deve unicamente disabilitare gli account, finchè gli utenti corrispondenti non abbiano cambiato la loro password. A questo punto, è opportuno inserire all'interno del sistema dei vincoli tali da non far accettare delle password che possono risultare banali all'atto della registrazione:

  – Non accettare password eccessivamente corte
  – Anticipare i dictionary attack, non accettando talune password
  – Accettare password con caratteri a contenuto misto, ovvero con elementi alfanumerici e con caratteri speciali.

◇ Non bisogna consentire di poter effettuare dei login da remoto tramite programmi che chiedono un'autentificazione con la richiesta di password in chiaro: di solito infatti tali password viaggeranno all'interno della rete pubblica, e possono quindi essere soggette ad attacchi di spoofing.

Tuttavia, se è richiesto mantenere più password casuali, ciò sarebbe estremamente difficile da memorizzare: da ciò è nata la soluzione di creare dei speciali software chiamati *Password Manager*, i quali hanno lo scopo di memorizzare delle password scelte dall'utente in maniera cifrata all'interno del computer. Si può inoltre effettuare il password aging, tramite il quale ogni tanto è opportuno cambiare la chiave associata ad un dato login, in quanto le chiavi private in questione potrebbero essere già state cifrate.

4.3 IPSec - VPN

Riprendiamo il tema affrontato nella Sezione 4.1 a pagina 33, tornando a chiederci come possiamo effettuare delle comunicazioni sicure all'interno di Internet. Come abbiamo già notato, la sicurezza a livello *network* ha il notevole vantaggio di rendere qualsiasi applicativo sicuro senza dover apportarvi alcun accorgimento di sicurezza, anche se comporta di dover effettuare cambiamenti su tutti i sistemi operativi presenti sulla rete mondiale, in quanto sia i nodi terminali, sia quelli intermedi (vedi i routers), devono essere in grado di comunicare tramite questo protocollo.

IPSec è un insieme di protocolli[1] basati su tecniche crittografiche, allo scopo di fornire autentificazione e confidenzialità. Queste caratteristiche, che per ora con IPv4 sono facoltative, saranno obbligatorie con l'introduzione di IPv6, che verrà introdotto all'interno del protocollo di comunicazione. Una delle possibili applicazioni di questa tecnologia sono le *Virtual Private Networks* (VPN), e fornisce un accesso sicuro ad internet paragonabile a quello fornito a quello SSL. Tramite questo meccanismo, è possibile garantire sicurezza tramite due differenti modalità di trasmissione:

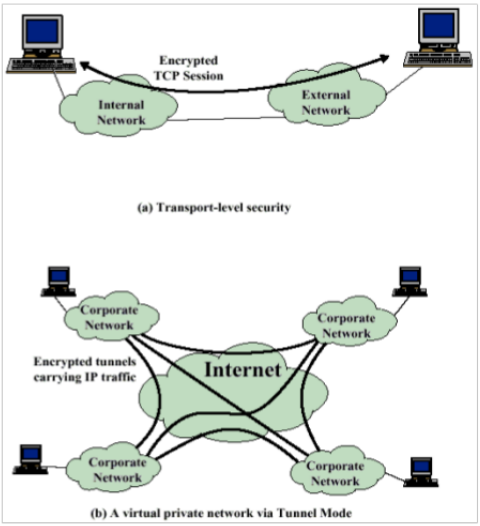
Tunnel mode fornisce una protezione dell'interno pacchetto IP, e quindi risulta utile per una comunicazione Gateway-Gateway come può avvenire nel caso delle VPN, permettendo la comunicazione sicura tra le due reti. In questo modo, nessuno dei suoi campi, nemmeno lo header, diventa leggibile all'interno della rete: è come se quindi viaggiasse all'interno di un tubo opaco, facendo in modo che la rete pubblica effettui l'inoltro del pacchetto unicamente verso il gateway di destinazione.

Tramite la VPN di IPSec è inoltre possibile comunicare tra due parti della rete aziendale connesse alla rete, che parlerà tra router a livello gateway quel protocollo (emulando quindi una comunicazione PPP tra i due router scollegati), mantenendo invece quello IP all'interno della rete: questo implica che IPSec effettua una forma di incapsulamento della informazione in chiaro fornita dal semplice pacchetto IP. Questo implica che gli stessi gateway devono essere in grado di tradurre i protocolli da IP a IPSec e viceversa.

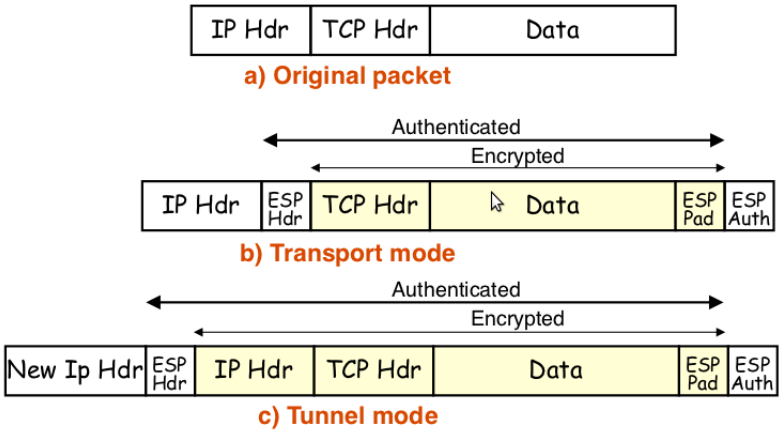
Con questa tecnica, il gateway di destinazione ha il compito di estrarre, decrittare ed autentificare il pacchetto originario che è stato trasportato tramite incapsulamento, ed inviato dal gateway mittente. In questo modo viene creato un nuovo header IP, dove la comunicazione sembra avvenire unicamentre tramite i due gateway.

Transport Mode fornisce sicurezza a livello applicativo, e spesso è utile per un utilizzo end-to-end, e quindi per `TCP`: con questa modalità, al contrario di quella precedente, non si critta anche lo header, ma unicamente il payload costituito dal solo pacchetto TCP: in questo modo non si critta anche chi sia il mittente e la destinazione, in quanto queste informazioni devono essere riconoscibili proprio da chi effettua l'inoltro del pacchetto.

[1] Questi sono AH (*Authentication Header*), ESP (*Encapsulating Security Payload*), ed *Internet Security and Key Management Protocol*.



(a) *Rappresentazione grafica*.



(b) *Incapsulamento*.

Figura 4.3: Rappresentazione delle differnze tra **Tunnel** e **Transport mode**.

Capitolo 5

Meccanismi di accesso

L'autentificazione è necessaria per realizzare l'autorizzazione: infatti non vogliamo solamente autentificare quegli utenti che avrebbero le credenziali per accedere al sistema, ma vogliamo anche essere in grado di controllare le azioni che questi compiono all'interno del sistema.

In questa sezione parleremo come strutturare un sistema informatico in modo da garantire il controllo degli accessi tramite il *principio di privilegio minimo*, in base al quale si concedono agli utenti quei privilegi minimi, in modo che questi possano effettuare tutti i compiti che sono loro permessi. Questo è inoltre necessario nella misura in cui potrei abusare dei privilegi aggiuntivi che mi sono stati concessi, anche perché non possiamo prevedere se quel dato utente in un futuro presenterà o meno un comportamento che potrebbe a violare l'integrità del sistema. Infatti, anche se non direttamente quell'utente potrebbe portare a minare l'integrità del sistema, tramite i programmi che ha installato. In questo modo si potrebbero inoltre prevenire eventuali programmi di spyware o malaware che utilizzano appunto i privilegi degli utenti. Utilizzando quindi i meccanismi di **complete mediation** creiamo un monitor detto **reference monitor** inglobato all'interno del **trusted computing base** (ovvero un'area del sistema operativo di cui si è comprovata la sicurezza) tramite il quale possiamo effettuare il controllo degli accessi all'interno del *kernel mode*. Inoltre possiamo garantire che l'utente non possa accedere direttamente alla periferica utilizzando l'interazione fornita dalle *system call*: queste verranno quindi intercettate ed eseguite all'interno del Reference Monitor, in modo da poter controllare ogni azione del programma che richieda un'interazione con il mondo esterno.

Bisogna inoltre sottolineare che l'**autorizzazione** non è solo un meccanismo mediante il quale si può permettere di garantire o bloccare gli accessi, ma è un insieme di politiche o di meccanismi che sono mirati a controllare se un utente è autorizzato a compiere o meno delle date azioni su di un determinato oggetto. Inoltre i meccnismi di protezione quali l'autentificazione sono necessari per poter garantire, a questo punto, l'autorizzazione.

Valuteremo quindi la presenza di tre possibili attori; si ha:

- ◇ un insieme di soggetti S, che compiono determinati accessi ed azioni permessi: è un soggetto **attivo**, anche se pure i soggetti possono subire delle azioni (possono essere altri oggetti): un esempio di soggetti sono i processi nei sistemi operativi e nei linguaggi object oriented, e le transazione all'interno dei DBMS.

- ◇ un insieme di oggetti O, che subiscono delle azioni da parte di un soggetto, e soddisfa delle sue richieste: un esempio di oggetti sono le risorse nei sistemi operativi, gli oggetti e le classi nei linguaggi object oriented, e le tabelle nei DBMS.

- ◇ un insieme di accessi dei soggetti sugli oggetti: nei sistemi operativi sono le azioni Read/Write/Execute, sono i metodi definiti nelle classi per i linguaggi object oriented, e le operazioni primitive di SQL per i DBMS.

Si definiscono inoltre i seguenti principi fondamentali:

Fail Safe Defaults secondo questo principio, nessun processo e conseguentemente nessun utente, possiedono già dei permessi di default, ed inizialmente questi hanno permessi di accessi nulli. Dovrà essere l'amministratore in una fase successiva, a fornire agli utenti gli opportuni accessi.

Mediated Access Utilizzo dei Reference Monitor come descritto sopra.

Separazione dei privilegi un sistema non dovrebbe consentire un accesso incondizionato ai privilegi di superutente tramite la conoscenza di una sola informazione: ad esempio Unix non permette di effettuare l'accesso come superutente solamente conoscendone la password, ma anche tramite l'appartenenza al gruppo dei *sudoers*. In questo modo si effettua prima l'accesso come utente normale, conoscendo quindi la propria password, e poi si effettuerà l'accesso come ROOT tramite un'altra password e l'appartenenza al gruppo dei sudoers. Si può quindi fare un'analogia con la strong authentication, tramite la quale non si ha accesso al token solamente possedendolo, ma anche conoscendone un pin.

Possiamo inoltre definire un **dominio di protezione** tramite una coppia costituita dall'oggetto, e dalle azioni che si possono effettuare su di questo: un soggetto può quindi lavorare all'interno di un *dominio di protezione*, che può contenere più coppie di quel tipo. Banalmente possiamo avere due domini di protezione quali il **kernel mode**, dove sono possibili tutte le operazioni su tutti gli oggetti, e lo **user mode**, con il quale l'utente può gestire unicamente gli oggetti che lui stesso ha creato. Tuttavia questa organizzazione è estremamente grossolana: se un processo di fatti nascesse e morisse in un dominio senza poter effettuare trasferimenti, sarebbe poco utile, mentre è maggiormente utile un sistema dinamico, ovvero che può muoversi di dominio: tramite le systemcall inoltre, si può permettere il passaggio da modo user a modo kernel, unicamente per effettuare la verifica sulla possibilità da parte dell'utente di effettuare quell'operazione.

Possiamo raggruppare tutti queste credenzialità tramite una tabella, dove l'elemento $access(i, j)$ rappresentano gli accessi possibili per quel dominio D_i gli accessi permessi su di un oggetto O_j, dove se tale cella è vuota, allora si indicherà che quell'utente non ha permessi di accesso per quell'oggetto. Possiamo quindi specificare che, nella suddetta tabella, gli indici dei domini sono disposti sulle righe, mentre sulle colonne sono disposti i nomi degli oggetti dei quali si vogliono pilotare gli accessi. Supponiamo tuttavia di avere sue processi su domini differenti (anche se più processi possono operare sullo stesso dominio contemporaneamente): vogliamo implementare una codifica tale che il processo utente effettui un accesso (es.) ad un dizionario, senza però accedere a tutto il file. Possiamo quindi introdurre un nuovo dominio, all'interno del quale sia unicamente possibile effettuare l'accesso in lettura su tutto il file: gli altri processi che vorranno accedere al dizionario, potranno spostarsi dal loro dominio in questo unicamente per la fase di accesso al contenuto del dizionario effettuando un'operazione di **switch**, durante la quale il processo eredita i permessi che aveva precedentemente per quello stesso file, ed aggiungendo quelli del nuovo dominio verso il quale ha transitato. L'operazione di switch prevede anche di ritornare, dopo aver

effettuato l'operazione che richiedeva il cambio di dominio, a quello posseduto precedentemente. Bisogna inoltre sottolineare che, quando si effettua lo switch, viene aggiunta un'altra riga alla tabella dei permmessi

In quel modo tuttavia, i due processi prenderanno il permesso di effettuare la lettura completa su tutto il file: tuttavia bisogna precisare che questo passaggio di accesso al dizionario avverrà tramite system call, e quindi verranno concessi quei nuovi permessi tramite shift solamente per la durata di quella operazione, che verrà fornita da una apposita libreria che provvederà a garantire che non vengano effettuate letture complete di tutto il file, ottenendo solamente da questo i valori richiesti da quell'interrogazione. Inoltre bisogna evitare il caso in cui un altro processo, che appartiene allo stesso dominio di un altro, si appropri indebitamente dai privilegi ottenuti dal precedente: è quindi opportuno che ogni processo effettui la copia dei suoi privilegi e li aggiorni eventualmente con quelli che gli vengono concessi tramite gli shift, in modo da mantenere distinte le "estensioni" dei vari domini dei processi.

Tramite il comando `setuid` ad esempio, un processo può temporaneamnete assumere un determinato dominio di protezione, esclusivamente per la durata di quel processo; essendo inoltre quell'azione scritta da root per gli utenti, saranno azioni sicure ad esempio, anche se il comando `passwd` richiede i privilegi di superutente, sarà possibile modificare le password degli utenti che hanno invocato quel dato comando: questo è inoltre possibile perché tale programma, mantiene in memoria le informazioni dell'utente che lo ha invocato. Da ciò segue che il meccanismo di switch nei sistemi Unix è fornito proprio dal comando `setuid`.

5.1 ACL e Capability

Continuando la discussione nell'ambito dei sistemi operativi, ci chiediamo a questo punto quali sono le modalità di possibile rappresentazione della tabella in questione.

- ◇ Possiamo memorizzare l'intera tabella come struttura dati matrice all'interno del sistema. Tuttavia questa soluzione non è facilmente implementabile, in quanto le dimensioni della tabella sono variabili, in quanto i sistemi che prendiamo in considerazione sono dinamici: possono ovvero avvenire shift, essere aggiunti o tolti oggetti e/o domini. Risulta anche poco scalabile in quanto, all'interno del sistema, si potrebbero avere un numero elevato di soggetti ed oggetti.

- ◇ Possiamo organizzare la memorizzazione della tabella per *colonne* tramite **ACL**, distribuendo detta tabella su tutti gli oggetti del sistema e, in particolare nei sistemi Unix, ciò significa inserire queste credenziali d'accesso per ogni singolo file, all'interno dei suoi metadati corrispondenti. In questo modo otterremo delle associazioni Dominio/Diritti, consentendo all'utente corrente, al gruppo a cui esso appartiene o agli altri particolari permessi di accesso: questi sono quindi modificabili direttamente dal possessore del file, in particolare ad ogni dominio sono associati tre bit per settare i permessi di lettura/scrittura/esecuzione, di cui il decimo indica se è possibile effettuare lo setuid per quell'oggetto.

- ◇ Possiamo organizzare la memorizzazione della tabella per *righe* tramite **Capability**, distribuendo i permessi sui domini. Con questa tecnica, associamo per ogni utente un insieme di oggetti con i suoi permessi, descrivendo quell'utente quali operazioni gli sono concesse.

Descriviamo ora, alla luce delle considerazioni precedenti, quali sarebbero le possibili implementazioni del reference monitor: possiamo rappresentare questo come una costruzione che limita l'accesso dell'oggetto, che risulta accessibile solamente tramite un'entrata sorvegliata da un controllore, che appunto media l'accesso all'oggetto (che eventualmente può essere automatizzato tramite un meccanismo chiave-serratura, lasciando quindi il rischio nella creazione della chiave):

- Tramite le *ACL*, il controllore media l'accesso all'oggetto tramite le stesse caratteristiche possedute dall'oggetto: al soggetto sarà dunque consentito l'accesso unicamente se esso sarà registrato all'interno della lista dei diritti.

- Tramite le *Capability*, si fornisce un certificato di accesso per ogni oggetto che si vuole accedere, che quindi contiene anche l'identificativo del particolare oggetto al quale si vuole accedere. A questo punto non importa più conoscere chi sia colui che vuole accedere all'oggetto, in quanto è sufficiente sapere che esso possiede le credenziali sufficienti per accedervi. Da ciò segue che il controllore dovrà unicamente verificare la validità del "certificato", in quanto su di questo non saranno presenti le credenziali dell'utente che lo detiene.

Vogliamo ora evitare il caso per il quale la capability possa essere genereata da un altro processo che non avrebbe i diritti di possederla, generando semplicemente dei bit: siamo costretti ad affermare che tale processo non deve sapere quale sia la forma di detta capability: questo tuttavia si può ottenere criptando con la chiave privata di crittografia simmetrica del sistema tale capability. Tuttavia si dovrà mantenere in chiaro l'informazione, in quanto altrimenti il processo non sarebbe in grado di conoscere a quale risorsa accedere: in questo modo possiamo garantire un metodo per accorgerci delle eventuali modifiche che verranno apportate nella capability in chiaro. Inoltre per evitare che sia possibile usare le nostre credenziali da ciò che è stato memorizzato in chiaro, si aggiunge un **codice di controllo** - banalmente un certo numero di bit casuali, che dovrà essere fornito anch'esso per poter accedere all'oggetto: in questo modo si può effettuare la verifica che il codice di controllo sia corretto, e che la richiesta del processo sia effettivamente autorizzata dalla capability ricevuta. Questo codice di controllo potrebbe essere una serie di bit casuali, che il sistema associa ad un dato soggetto prima di fornirgli la capability, e memorizzando tale associazione per le future richieste di accesso con capability. Questi vengono inoltre aggiunti perché ci sarebbe una bassa probabilità, se non nulla, di azzeccare casualmente il valore del crittogramma della capability.

Le chiavi private delle capability possono essere generate in maniera dinamica: in questo modo tuttavia il sistema deve tenere traccia di queste chiavi (questa tecnica ha tuttavia implicazioni sulla sicurezza per la possibile compromissione della chiave, causando potenzialmente la compromissione dell'intero sistema), o può essere un'unica chiave per tutti gli elementi del sistema o ripartendola per tutti gli elementi in esso presenti.

Tuttavia a questo punto un processo, potrebbe fornire ad un altro le sue stesse credenziali d'accesso: a questo punto infatti si potrebbe minare l'integrità del sistema semplicemente copiando i permessi che erano stati forniti ad un altro, e ripresentarli per ottenerli indebitamente. Per rendere quindi non trasferibili i permessi (anche se ciò in certi casi può essere ammesso), si possono utilizzare i meccanismi di protezione del sistema operativo, perché tali capability non siano direttamente accessibili al processo, che quindi non sarà in grado di accedere alle informazioni in esso contenute. Questo può essere implementato mettendo tali informazioni all'interno della parte kernel corrispondente del processo utente, il quale potrà

utilizzarle unicamente fornendo l'indice al sistema (ad esempio il file descriptor), gestendo tramite accesso al file descriptor l'accesso alla risorsa, che potrà essere consentita o meno.

Concludendo, un sistema può possedere sia le Capability, sia le ACL: questa strategia può quindi essere utilizzata per aumentare le prestazioni del nostro sistema: la più prestante delle tecniche di controllo è la **capability**, in quanto non richiede una tecnica di autentificazione come le **ACL** (di fatti lo switch tra modo utente e kernel non influisce pesantemente sulle prestazioni del sistema). Inoltre tramite capability non è necessario effettuare nessun controllo sulla correttezza del permesso memorizzato, in quanto si vuole garantire che non sia acessibile direttamente dall'utente. Anche se le Capability sono efficienti, è necessario effettuare una prima autentificazione quando il soggetto effettua l'accesso ad un file utilizzando quindi le ACL, fornendo poi successivamente la capability per non dover richiederne l'accesso in futuro, se non al rilascio della risorsa. Questo avviene quindi all'interno di Unix all'atto della chiamata della system call `open`, con la quale si controlla che i permessi di accesso richiesti dall'utente corrispondono ai suoi privilegi associati nel sistema; nel caso in cui il controllo dovesse passare, verrà restituito al processo un file descriptor, che indicherà l'indice all'interno della parte kernel del processo della risorsa alla quale il processo potrà accedere.

5.2 Review e revocation

Possiamo inoltre distinguere due tipi diversi di sistemi: i **DAC** (*Discretionary access control*), tramite i quali il proprietario stesso decide i diritti di accesso da consentire al suo gruppo ed al mondo esterno. I **MAC** invece (*Mandatory access control*) impongono un unico amministratore di sistema, il quale può decidere le regole di accesso da stabilire per gli oggetti nel sistema. Questi meccanismi vengono inoltre richiesti all'interno di sistemi che richiedono un altissimo livello di sicurezza e dove le strutture relazionali degli utenti sono estremamente gerarchizzate, come quelle militari.

Visto che il sistema è dinamico, dobbiamo prevedere la possibilità che i permessi agli accessi, che erano stati garantiti precedentemente, possano essere revocati o rivisti.

Per quanto concerne la **review**, se abbiamo un'implementazione del sistema con le *ACL*, si dovrebbe ricostruire tutta la Access Control Matrix accedendo ad ogni singolo oggetto: questa tecnica è sì laboriosa, ma è semplice da effettuare. Tramite le *Capability* invece, si ha la necessità di ricostruire lo stato degli accessi ottenendo le informazioni, presenti all'interno della loro memoria, per ogni singolo processo. In quanto questi valori non sono associati ai metadati ed in quanto un processo può memorizzare questi permessi ovunque nella memoria che gli è associata (sempre se queste informazioni non sono memorizzate all'interno dello spazio kernel, ma in quello utente), non si è concretamente in grado di effettuare una modifica all'interno dei suoi dati.

Per quanto concerne la **revocation**, all'interno dei sistemi con *Capability* si ha a che fare con lo stesso problema proprio dei certificati, ovvero quello di revocare il certificato precedentemente rilasciato, e quindi valido. Essendo inoltre detta capability memorizzata all'interno dei processi, dovremo:

- implementare un sistema di capability a durata limitata
- concedere diritti di accesso non direttamente agli oggetti, ma ad una tabella che punta virtualmente agli oggetti. In questo modo possiamo fare in modo che nessuno che prima aveva la possibilità di accedere a quella risorsa lo possa fare in futuro rimuovendo la riga della tabella, o consentire comunque un accesso solamente a determinati

processi, inserendo in quel punto una nuova lista di processi che hanno il permesso di accedere a quell'oggetto.

Questo sistema è quindi costoso ma fattibile. All'interno dei sistemi *ACL* invece, sarà sufficiente cambiare i valori presenti all'interno dei metadati associati alla risorsa in questione, (es.) all'interno dei sistemi Unix tramite chiamata `chmod`. La revoca inoltre può essere:

- immediata o ritardata (subito o si può attendere)
- selettiva o generale (per alcuni i domini o per tutti)
- parziale o totale (tutti i diritti o solo alcuni)
- temporanea o permanente

5.3 IDS - Intrusion Detection System

Vogliamo ora avere un modo per ottenere delle informazioni di un attacco subito (anche se idealmente vorremo ottenere delle informazioni mentre questo sta avvenendo, con un sistema analogo ad un antifurto informatico): questi non si dimostrano una soluzione definitiva al problema. Inoltre è possibile definire un'**intrusione** solamente tramite la specifica di politiche di sicurezza desiderate dal sistema, definendo cosa è più o meno ammesso, e quindi si rileva necessario che l'amministratore debba conoscere quali comportamenti sono da considerarsi dannosi.

Una volta il *CERT* raccoglieva i possibili tipi di attacchi, e forniva in seguito delle istruzioni che servivano a descrivere come difendersi da questi: tuttavia questo implicava una conoscenza molto settoriale ed avanzata delle tematiche di sicurezza affrontate; oggi giorno si possono attivare degli script detti `cron`, tramite i quali si possono effettuare delle operazioni di manutenzione come `root`.

Tramite **IDS** vogliamo invece automatizzare il processo di riconoscimento degli attacchi, di modo che mi possa avvisare in seguito di qualcosa di sospetto. In quanto si basa su tecniche di riconoscimento automatico secondo degli schemi di comportamento, questo può essere affetto da falsi positivi o da falsi negativi: bisogna quindi cercare un compromesso tra questi due possibili scenari.

L'**IDS** può essere situato su di un singolo host (*host based*), può essere situato su più host (*multihost based*), o può essere completamente decentralizzato e sparso all'interno della rete (*network based*: in questo caso in genere è un oggetto fisico presente all'interno della rete, e serve anche per monitorare il traffico, oltre ad effettuare auditing dei dati ricevuti da un host). Si possono indivituare due metodi di riconoscimento degli attacchi:

Anomaly Based dato uno stato "tranquillo" dove non avvengono intrusioni, cerca di intuire un'intrusione come una deviazione del modello iniziale: questo può avvenire tramite la misurazione di alcuni parametri rilevabili all'interno del sistema operativo, come l'utilizzo di CPU.

Misuse Based si posseggono vari "pattern" di intrusioni da quelli riconosciuti, che si cerca di riconoscere dai comportamenti osservati all'interno del sistema: questo implica che questi sistemi devono essere continuamente aggiornati[1].

[1]Questo meccanismo è analogo a quello del riconoscimento dei virus. Questi meccanismi utilizzano reti neurali e machine learning.

Definiamo ora alcune caratteristiche che questi sistemi devono possedere:

- ◇ Questi meccanismi devono inoltre essere *fault-tolerant*, in quanto se il sistema IDS è vulnerabile alla disattivazione da parte di un intruso, si rileva completamente inutile: anche questo meccanismo deve essere quindi sicuro. Inoltre deve essere in grado di resistere agli attacchi che possono essere inflitti al sistema stesso.
- ◇ Non dovrebbero essere delle black-box, ovvero il suo comportamento dovrebbe comunque essere osservabile dall'esterno.
- ◇ Deve causare un overhead minimale nel sistema, altrimenti il sistema di riconoscimento è inutilizzabile
- ◇ Deve essere facilmente configurabile, in modo da adattarsi alle esigenze dei differenti utenti.
- ◇ Deve essere in grado di adattarsi al cambiamenti del sistema, (deve essere adattivo).

Analizziamo ora brevemente come possiamo effettuare una risposta all'attacco, dopo però che l'intrusione è già avvenuta. Come prima cosa è opportuno effettuare un'analisi di tutte le informazioni che sono ottenibili, tramite tecniche di "computer forencsis", eventualmente comunicare alle parti in causa che è avvenuto una forma di attacco, cercare di contenere l'intrusione se questa è tutt'ora in atto, eliminare le debolezze del sistema che hanno consentito agli attaccanti di sferrare l'attacco, e riportare il sistema alle condizioni precedenti (es. tramite un backup). È inoltre opportuno, una volta ottenute tutte le prove, conservare tali informazioni, salvaguardando le informazioni ottenute.

5.3.1 Honeypot

Come suggerisce il nome, essa è una trappola, in modo da attrarre un hacker con un computer poco sofisticato: tramite questo possiamo capire, tramite azioni concrete e campioni vere, capire come si comportano, essendo quindi utile per prevenire attacchi successivi di questo tipo. Devono quindi essere situati in un luogo non troppo difficile da raggiungere, come della *DMZ* o nella rete esterna, nella quale lo hacker non deve superare alcun tipo di attacco (alternativamente può essere situato anche all'interno della rete locale). Tale honeypot deve inoltre avere delle caratteristiche comune: se così non fosse, l'attaccante potrebbe non cadere in trappola, in quanto ovviamente non vuole rivelare le sue modalità di attacco: non deve quindi contenere pochi comandi e nessun software, e deve contenere dei dati potenzialmente sensibili, ma fasulli.

Capitolo 6

Gestione delle Reti

6.1 Denial of Service (DoS)

La minaccia maggiore per i sistemi informatici è quella della mancata fruizione di un servizio mediante "cyberextorsion", mirando ad una perdita economica da parte di chi detiene la fornitura del servizio stesso. Mentre nel mondo fisico questo metodo di attacco è molto ridotto perché per effettuare SPAM, è necessario sobbarcarsi dei costi troppo ingenti, questo non avviene nel mondo informatico, dove i costi sono molto ridotti se non nulli. Per risolvere un potenziale attacco si potrebbero effettuare le seguenti considerazioni:

- ◇ Si potrebbe cercare di evitare l'attacco mediante l'utilizzo di *blacklist*, non consentendo più l'accesso ad un dato utente: tuttavia in questo caso potrei escludere un accesso legittimo di un utente con (es.) lo stesso numero di IP.
- ◇ Potrei cassare una richiesta onerosa, anche se alcune potrebbero essere in buona fede: si potrebbe quindi richiedere al potenziale attaccante un "costo" sufficientemente elevato da demotivarlo nel perseguire l'attacco.

  Tuttavia questa situazione potrebbe essere aggirata richiedendo chiedendo a tanti clienti di effettuare moltissime richieste che stiano al di sotto della soglia d'allarme. Questo tipo d'attacco è chiamato un DDoS, ovvero un distributed DoS, che diventa quasi impossibile da distinguere da una situazione problematica reale, ma non causata da attacco.

Molti sistemi utilizzano quindi dei filtri anti-spam, allo scopo di evitare richieste da potenziali attaccanti.

Descriviamo ora varie tipologie di attacchi di tipo Denial of Service, che sfruttano la fase di handshake del protocollo TCP:

- **Denial of Service tramite Syn-Flooding** Tramite questa tecnica, si mira ad inviare alla vittima un pacchetto IP modificato, in modo che sia cambiato l'indirizzo dell'attaccante/mittente con quello di un altro host casuale nella rete (che può quindi essere anche non esistente), tramite una tecnica di *spoofing*; quest'ultimo non risponderà allo `SYN-ACK` della vittima che avrà ricevuto un `SYN`, con la quale non aveva iniziato ad instaurare nessuna comunicazione. In questo modo, finché non scadrà il nostro timeout, la vittima terrà occupata una sua risorsa, occupando quindi la memoria del suo sistema.

  Essendo quest'attacco di tipo flooding, l'attaccante potrebbe inviare sempre nuove richieste di tipo `SYN`, effettuando spoofing nel pacchetto e modificando il mittente

con differenti mittenti casuali. In questo modo si vogliono impedire anche eventuali collegamenti legittimi nei confronti del sistema attaccato, rendendo indisponibile il servizio.

- **Syn-flooding reflector**. Con questa tecnica invece effettuiamo l'invio del pacchetto, sul quale è stato effettuato spoofing e modificato il campo mittente con quella vittima, ad un host casuale intermedio che, ricevendo il `SYN`, invierà lo `SYN-ACK` alla vittima. In questo caso verranno replicati i riceventi, che sono ignari del vero attacco in corso dal vero mittente che però non conoscono, i quali invieranno tutti la lororisposta al sistema attaccato.

  In questo caso non verranno quindi consumate risorse del sistema attaccato, ma bensì verrà occupato tutto il canale di comunicazione verso questo: anche in questo modo si mina la fruibilità del servizio, in quanto si blocca l'accesso del canale di comunicazione.

- **Bot-net (DDos)**. Tramite questa tecnica si ha un computer "padrone", che ha la possibilità di inviare dei comandi ad una rete costituita da computers detti *zombie*(s), in modo da orchestrare l'attacco verso una stessa vittima: questo è possibile perché gli zombie(s) che costituiscono la bot-net, sono sempre in ascolto del "padrone". Tali zombie si ottengono tramite l'installazione inconscia di software, che opera all'interno di computer casalinghi senza dare segni evidenti agli utenti dei sistemi infettati.

In pratica, è quasi impossibile essere in grado di prevenire o/e difendersi da un attacco di tipo DoS: non esiste quindi un unico tipo di difesa possibile in merito, anche perché le tecnologie di attacco sono in continua evoluzione, parallelamente alle tecniche di difesa. All'interno dei routers possono comunque essere implementate delle tecniche di difesa contro DoS, anche se raramente sono divulgate le caratteristiche di previsione degli attacchi, poiché in questo campo la security by secrecy è l'unica arma possibile per non divulgare il modo con il quale viene sventato l'attacco, e per non minare il sistema che viene commercializzato.

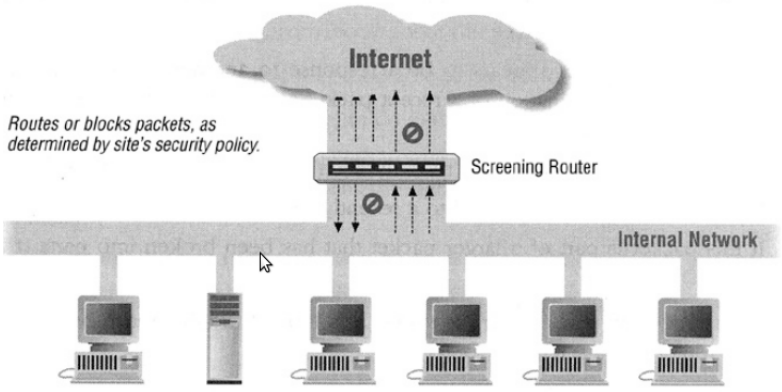
6.2 Firewall

Parliamo di **exo-structure** quando si fa riferimento agli oggetti esterni al sistema che stiamo considerando: ad esempio, all'interno di un sistema aziendale, vogliamo che tale exo-structure coincida unicamente con il contatto di un unico *firewall*; anche le *VPN* possono essere considerate come exo-structure(s). I firewall sono chiamati in questo modo poiché fanno riferimento alle porte tagliafuoco, che sono fatte per restistere agli incendi, fino ad un limite superiore di temperatura. In informatica questo richiama quindi il concetto del contenimento, sottolineando il fatto che si vogliono limitare i danni il più possibile; essi inoltre limitano il traffico da e verso l'esterno, tramite questo varco ben definito. Il firewall consente inoltre di:

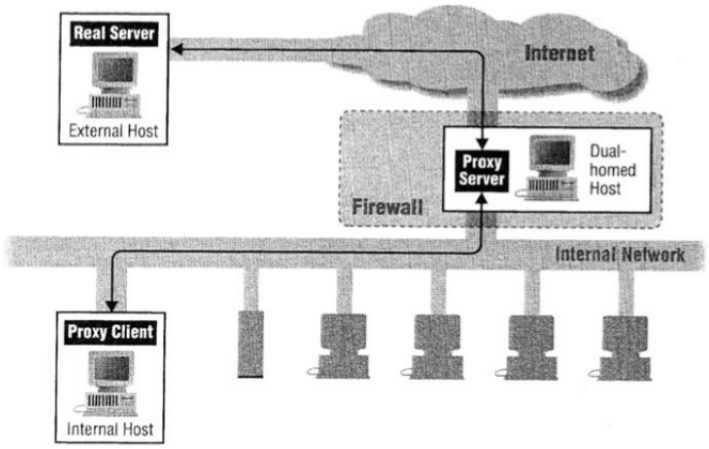
- concentrare i controlli di sicurezza su di pochi punti all'interno del sistema, forzando a mettere in pratica le politiche di sicurezza precedentemente enunciate

- possono essere utilizzate per effettuare "auditing"

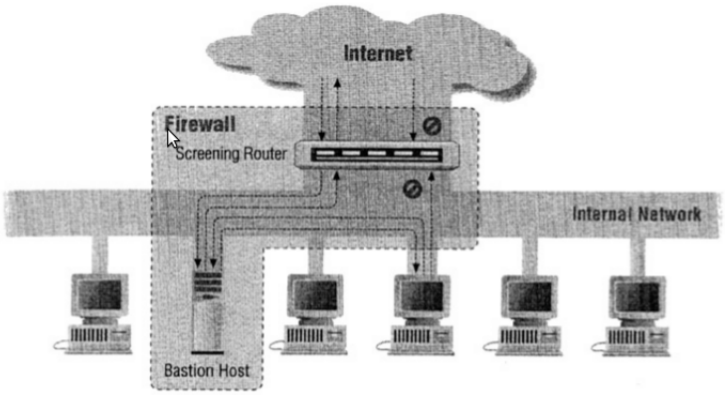
Tuttavia, tale firewall non ci può proteggere dalle minacce che provengono dall'interno della stessa intranet aziendale: a questo scopo risulta particolarmente inefficace se esiste una



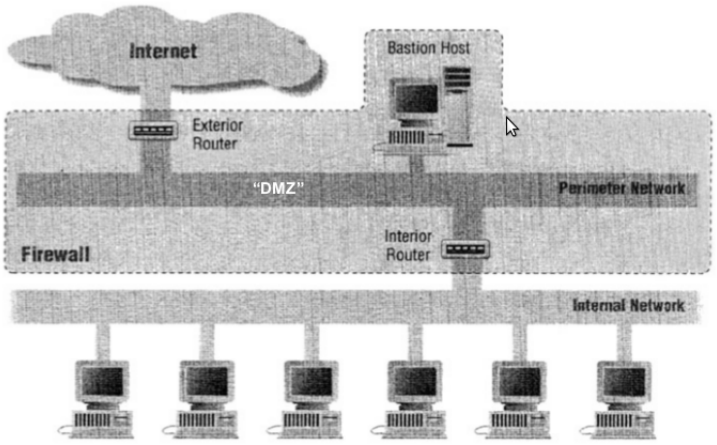
(a) *Screening Router*.



(b) *DualHomed Host*.



(c) *Screened Host*.



(d) *Screened Subnet*.

Figura 6.1: Esempi di Architetture Firewall.

possibilità che il traffico verso l'esterno possa anche non trasitare tramite il firewall[1]. Il firewall deve inoltre essere costantemente aggiornato, di modo da poter prevenire delle nuove minacce che non erano state previste precedentemente; bisogna inoltre sottolineare come, lavorando a livello IP, non è in grado di riconoscere le minacce relative a virus o a worm. Il firewall deve inoltre essere configurabile solamente dall'amministratore. Sono inoltre presenti differenti tipologie di firewall, tra le quali riconosciamo le seguenti:

◇ Packet Filter

◇ Proxy Server

◇ Network Address Translator

6.2.1 Packet Filter (o Screening Router)

Questi meccanismi in genere vengono implementati all'interno dei routers, i quali effettuano come compito base lo smistamento dei pacchetti, allo scopo di farli giungere ad una prefissata destinazione: in questo caso particolare parleremo di **screening router**. Mentre infatti un router semplice decide vome far avvicinare il pacchetto alla sua destinazione, questo tipo di router prima si chiede se quel dato pacchetto deve essere effettivamente avvicinato alla sua destinazione, ed in caso di risposta negativa verrà cestinato.

Questo risulta inoltre possibile tramite l'analisi del pacchetto a livello *network* (siamo quindi in grado di leggerne il solo header, senza andare ad indagare sul suo contenuto, disponibile solamente a livello applicativo). Considerando ciò, tale dispositivo può essere in grado di:

- *Consentire/bloccare la comunicazione da/verso un tale IP con numero di porta*: lo screening router può quindi essere configurato allo scopo di impedire un possibile attacco che proviene da un determinato IP, oppure bloccando una determinata porta.
- In quanto si ha la possibilità di avere molti possibili collegamenti in entrata o in uscita, si potrebbe anche stabilire di permettere il transito dei pacchetti di un certo tipo, tramite una sola uscita.
- Possibilmente devono avere anche una *memoria interna*, in modo da poter tracciare ciò che è avvenuto nel passato.

Si possono anche bloccare i pacchetti secondo le seguenti domande:

◇ Il pacchetto è stato ricevuto dall'esterno senza una precedente particolare richiesta, ovvero è un pacchetto "spontaneo"?

◇ Il pacchetto ricevuto da un particolare host esterno, ha superato un particolare più di un certo numero di pacchetti per unità di tempo?

◇ Si possono inoltre configurare degli stati, in modo da definire delle particolari politiche di sicurezza da attuare, in base ai quali effettuare o meno l'accettazione, ad esempio se si sono ricevuti tanti pacchetti con stesso contenuto del body.

[1] Ad esempio, se all'interno di questa rete esiste un computer con accesso diretto verso l'esterno tramite un modem seriale: in questo modo gli attaccanti potranno sfruttare le debolezze del sistema computer, per potere poi accedere potenzialmente a tutte le informazioni dell'intranet.

Come vantaggio dall'utilizzo di questi dispositivi, si potrebbe avere che tramite uno solo di questi dispositivi, che tra l'altro sono efficienti e facilmente reperibili (*COTS*), si potrebbe controllare un'intera intranet; tuttavia possono essere non banali da configurare, possono ridurre le prestazioni nella connessione di rete e sono "ignoranti" delle possoibili politiche che sono desumibili a livello applicativo, e quindi non riconducibili all'header IP.

6.2.2 Proxy Server

Il termine *proxy* significa "delega", ovvero che si concede a qualcuno di fare certe operazioni in vece di un altro: questi strumenti sono quindi utilizzati in modo da permettere al client di interfacciarsi con il proxy come se questo fosse il server, ed inoltre il proxy si comporterà con il server come se questo fosse il client. Questo è inoltre un servizio applicativo del server, è non è unicamente un packet filter.

Inizialmente questi dispositivi avevano solamente degli scopi di caching, o servivano per consentire agli utenti l'accesso solamente a determinati contenuti: inoltre il proxy viene contattato unicamente se la comunicazione necessita di una comunicazione verso l'esterno, e pertanto non verrà mai contattato per le comunicazioni dell'intranet.

In genere si utilizzano un **proxy client** all'interno dell'intranet, ed un **proxy server** utilizzato anche come firewall, che svolge la funzione di "dual-homed host": questo in genere è situato esternamente alla rete locale aziendale.

Anche in questo caso è opportuno evidenziare i suoi vantaggi con gli possibili svantaggi: tramite questo meccanismo è possibile autentificare gli utenti che utilizzano il servizio (possiamo quindi condcedere o meno il servizio al cliente, ed inoltre è possibile effettuare il logging del servizio), è possibile filtrare i pacchetti in base al loro contenuto applicativo ed effettuare caching. Gli svantaggi sono costituiti dal fatto che è necessario disporre di un proxy server, che ogni client (fungendo in quel momento da proxy client) deve essere in grado di contattare.

6.2.3 Network Address Translator

Una volta i NAT servivano unicamente per limitare il numero disponibile di indirizzi IP (2^{32}), allo scopo di non associare indirizzi IP all'infinito, senza rilasciarli quando questi non dovessero più essere utilizzati: lo scopo principale dei NAT è quindi quello di associare ad ognuno dei calcolatori all'interno della rete locale, un unico indirizzo IP che si collega ad Internet [2]. Questo meccanismo può quindi essere quindi implementato all'interno di uno stateful router.

Questa tecnologia ha vantaggi enormi: se i pacchetti spontanei in entrata non sono esplicitamente ammessi, vengono rifiutati: in questo modo non si rischia nemmeno che un potenziale attaccante possa conoscere come è strutturata la mia rete locale. Questo NAT, in quanto comporta la riscrittura degli indirizzi del mittente dei pacchetti, potrebbe causare dei problemi con pacchetti crittografati, che contengono come informazione di cifratura l'indirizzo IP: inoltre in quanto gli indirizzi IP assegnati all'interno della rete locale spesso sono dinamici, ci potrebbero essere dei problemi con il tracciamento delle informazioni (logging).

[2] Altro scopo per i Router con NAT è quello di associare ad ogni calcolatore un differente servizio di pacchetti, effettuando quindi un port-mapping dei servizi forniti in entrata: se non fosse possibile effettuare il port-mapping, non saremmo in grado di gestire i "pacchetti spontanei". Tale meccanismo si rivela inoltre necessario per fornire servizi verso l'esterno.

6.2.4 Architetture Firewall

Una volta descritti i componenti possibili che possono creare un firewall, possiamo descrivere alcune possibili architetture:

Screening Router questa è l'archiettura più semplice; include unicamente lo screening router che viene messo al posto di un router, ovvero sul limitare tra una rete interna ed una esterna: quest'ultima potrebbe anche non essere necessariamente Internet, ma benissimo un'altra intranet.

DualHomed Host in questo caso invece si mette (es.) un Proxy Server al posto di un router, che quindi viene rimpiazzato da un altro host.

Screened Host assieme ad uno Screening Router, si aggiunge un **Bastion host**, ovvero un host fortificato dove vivono tutti i servizi che sono accessibili tramite l'esterno, estremamente sicuro ed aggiornato sulle minacce possibili; possono essere inoltre specificate ulteriori politiche di gestione del traffico, ad esempio si può decidere di far passare tutto il flusso di informazioni in ingresso ed in uscita tramite il *bastion host*, od oppure si può decidere che una parte del traffico in uscita od in entrata, può passare tramite lo screening router, senza il ricorso del *bastion host* intermedio.

Screened Subnet si sposta il *bastion host* all'esterno dalla rete interna, mettendolo in una rete intermedia tra rete locale ed esterna chiamata **DMZ** (ovvero "zona demilitarizzata"), che è quindi delimitata da un *interior router* e da un *exterior router*: il primo può essere configurato in modo da far transitare il traffico in uscita solamente verso il *bastion host*, il quale successivamente provvederà ad attuare le politiche di packet filtring appropriate, mentre il secondo può essere configurato in modo da far transitare tutto il traffico in entrata verso lo stesso *bastion host*. In questo modo lo hacker deve superare due barriere minando entrambi i router.