SICUREZZA

Documento Realizzato da **Raffaele Mercurio** e **Vincenzo Salvatore**

Se questo documento vi è stato utile e vi ha aiutato a superare l'esame, potete offrirmi un caffè qui.
PAYPAL : @rafmer00 :D

CAPITOLO 1 - INTRODUZIONE

La **sicurezza informatica** è definita come le misure e i controlli che garantiscono **riservatezza**, **integrità** e **disponibilità** delle risorse del sistema (come hw, sw, firmware, ecc…). [**CIA**]

- **Riservatezza (confidentiality)**: Insieme di regole ad alto livello che impongono limitazioni di accesso e di divulgazione di informazioni. La riservatezza si concretizza nella ***riservatezza dei dati*** (garantisce che le informazioni private non vengano divulgate) e nella ***privacy*** (garantisce che gli individui possano scegliere con chi condividere quali informazioni).
  [*un esempio di perdita di riservatezza si ha nel caso di divulgazione non autorizzata di informazioni*];
- **Integrità (integrity)**: Garanzia che le informazioni siano affidabili ed accurate. L'integrità si misura sia sui ***dati*** (garantisce che le informazioni vengano modificate solo sotto autorizzazione) sia sul ***sistema*** (garantisce che il sistema svolga la funzione prevista senza essere manipolato da entità non autorizzate).
  [*un esempio di perdita di integrità si ha nel caso di modifica o distruzione non autorizzata di informazioni*];
- **Disponibilità (availability)**: Gestione del rischio per garantire un accesso affidabile alle informazioni da parte delle persone autorizzate.
  [*un esempio di perdita di disponibilità si ha nel caso di interruzione di accesso alle informazioni*].

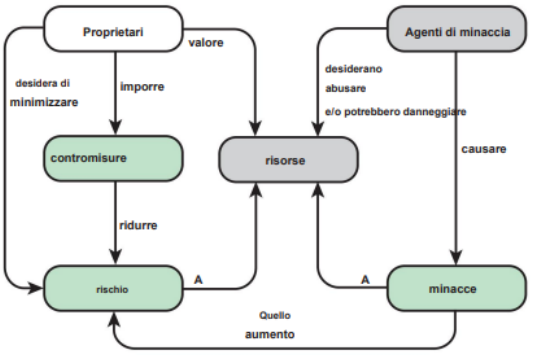
A questi tre fondamentali si aggiungono:

- **Autenticità (authenticity)**: Proprietà di un'informazione di essere verificata e affidabile e garanzia che quell'informazione provenga da canali attendibili;
- **Responsabilità (accountability)**: Esigenza di ricondurre una violazione della sicurezza ad un'unica entità.

La bassa CIA potrebbe avere diversi tipi di impatto in base alle condizioni:

- **Basso impatto**: efficacia delle funzioni dell'organizzazione notevolmente ridotta, nonostante vengano comunque consentite le funzioni primarie. Danni al patrimonio limitati e/o danni alle persone limitati;

- **Medio impatto**: efficacia delle funzioni dell'organizzazione notevolmente ridotta, nonostante vengano comunque consentite le funzioni primarie. Danni al patrimonio significativi e/o danni alle persone significativi ma che non comportino la perdita di vita o lesioni gravi;
- **Alto impatto**: perdita della capacità di svolgere le funzioni primarie. Danni al patrimonio gravi e/o danni alle persone catastrofici che possono comportarne la perdita di vita o lesioni gravi.



Nello sviluppo di un particolare meccanismo o algoritmo di sicurezza è sempre necessario considerare i potenziali attacchi. Tali meccanismi in genere implicano più di un particolare algoritmo o protocollo.

- **Risorsa di sistema (asset)**: hardware, software, struttura e rete di comunicazione, dati contenuti in un sistema, ecc….
- **Vulnerabilità di una risorsa (vulnerability)**: debolezza in un sistema. Un sistema vulnerabile può essere ***corrotto*** *(perdita di integrità)*, ***con perdite*** *(perdita di riservatezza)* o ***non disponibile*** *(perdita di disponibilità)*.
- **Minaccia (threat)**: qualsiasi circostanza o evento che possa potenzialmente avere un impatto negativo sul sistema sfruttando le vulnerabilità.
- **Avversario (threat agent)**: entità che attacca o costituisce una minaccia per il sistema.
- **Attacco (attack)**: sono minacce attuate e rappresentano qualsiasi tipo di attività dannosa che tenta di raccogliere, interrompere, negare, degradare o distruggere risorse o informazioni. Gli attacchi possono essere:
    - **Attacco passivo**: tentativo di apprendere o usare informazioni che non influiscono con le risorse del sistema. L'obiettivo è ottenere informazioni che vengono trasmesse. Un esempio è l'analisi del traffico;
    - **Attacco attivo**: tentativo di alterare le risorse del sistema o influenzarne il funzionamento. Implicano qualche modifica del flusso di dati o la creazione di un flusso falso. Ci sono quattro categorie (*replay, masquerade, modification of messages, denial of service*)
    - **Attacco dall'interno**: attacco avviato da un'entità interna al perimetro di sicurezza;
    - **Attacco dall'esterno**: attacco avviato da un'entità esterna al perimetro di sicurezza.

- **Contromisura (countermeasure)**: mezzi utilizzati per affrontare attacchi alla sicurezza e per ridurre al minimo il rischio. Constano di quattro fasi (*prevenire, individuare, rispondere e recuperare*). Possono introdurre nuove vulnerabilità.
- **Rischio (risk)**: misura delle potenziali minacce di un sistema. **R = PxD**, dove:
    - **P**: probabilità che si verifichi una minaccia
    - **D**: danno che porterebbe il concretizzarsi di una minaccia

Ci possono essere 4 tipi di **conseguenze alle minacce**:

Divulgazione non autorizzata (**unauthorized disclosure**): Circostanza o evento per cui un'entità non autorizzata ottiene accesso a dati per i quali non è autorizzata. Possibili attacchi sono:

- ***Exposure***: dati sensibili vengono rilasciati direttamente a un'entità non autorizzata
- ***Interception***: l'entità non autorizzata intercetta dati sensibili durante la trasmissione tra fonti e destinazioni autorizzate
- ***Inference***: l'entità non autorizzata carpisce dati sensibili non direttamente dalle comunicazioni ma ragionando su caratteristiche di queste ultime
- ***Intrusion***: l'entità non autorizzata ottiene l'accesso a dati sensibili eludendo le protezioni di sicurezza del sistema

Inganno (**deception**): Circostanza o evento per cui un'entità autorizzata ottiene dati falsi pensandoli veri. Possibili attacchi sono:

- ***Masquerade***: un'entità non autorizzata ottiene l'accesso a un sistema o esegue un atto dannoso fingendosi un'entità autorizzata
- ***Falsification***: i dati falsi ingannano un soggetto autorizzato
- ***Repudiation***: un'entità inganna un'altra negando falsamente la responsabilità di un atto

Interruzione (**disruption**): Circostanza o evento per cui vengono interrotte le funzioni e i servizi di un sistema. Possibili attacchi sono:

- ***Incapacitation***: un'entità disabilita un componente del sistema interrompendo il suo funzionamento
- ***Corruption***: un'entità altera in maniera indesiderata e negativa il funzionamento del sistema
- ***Obstruction***: un'entità ostacola il funzionamento del sistema, interrompendo la fornitura di servizi

Usurpazione (**usurpation**): Circostanza o evento per cui un'entità non autorizzata controlla il funzionamento dei servizi. Possibili attacchi sono:

- ***Misappropriation***: un'entità assume il controllo logico o fisico di una risorsa di sistema

- ***Misure***: un'entità fa sì che un componente del sistema esegua una funzione dannosa per la sicurezza del sistema

Esempi di minacce alle risorse

	Availability	**Confidentiality**	**Integrity**
Hardware	Attrezzatura rubata o disattivata, negandone il servizio	Viene rubato un CD-ROM non crittografato	
Software	Programmi eliminati, negandone l'accesso agli utenti	Copia non autorizzata del software	Programma funzionante viene modificato per provocarne il fallimento
Data	File eliminati, negandone l'accesso agli utenti	Lettura non autorizzata di dati	File esistenti vengono modificati
Networks	Messaggi distrutti o networks resi indisponibili	Messaggi letti	Messaggi modificati, ritardati o falsificati

Requisiti di sicurezza

Requisiti di sicurezza tecnici

- Controllo degli accessi (***access control***)
- Identificazione ed autenticazione (***identification and authentication***)
- Protezione dei sistemi e delle comunicazioni (***system and communication protection***)
- Integrità del sistema e delle informazioni (***system and information integrity***)

Requisiti di sicurezza funzionali

- Consapevolezza e formazione (***awareness and training***)
- Controllo e responsabilizzazione (***audit and accountability***)
- Certificazione, accreditamento e valutazione (***certification, accreditation and security assessment***)
- Manutenzione (***maintanance***)
- Protezione fisica e ambientale (***physical and environmental protection***)
- Pianificazione (***planning***)
- Sicurezza del personale (***personnel security***)

- Valutazione dei rischi (***risk assessment***)
- Acquisizione dei sistemi e servizi (***system and service acquisition***)

Requisiti di sicurezza tecnici e funzionali

- Gestione della configurazione (***configuration management***)
- Risposta agli incidenti (***incident response***)
- Protezione dai media (***media protection***)

Alberi d'attacco

Un **albero d'attacco** è una struttura dati gerarchica e ramificata che rappresenta un insieme di tecniche per sfruttare vulnerabilità e rischi di sicurezza.

L'obiettivo dell'attacco è rappresentato dalla radice, mentre le foglie sono i vari punti dove iniziare l'attacco. Ogni nodo intermedio rappresenta sotto-obiettivi uniti da AND oppure OR.

Sicurezza delle reti

Si sviluppa in:

- ***Sicurezza della comunicazione***: attraverso protocolli di sicurezza autonomi o parti di altri protocolli [*HTTPS, SSH, ecc…*]
- ***Sicurezza dei dispositivi***: attraverso firewall, intrusion detection (genera allarme) e intrusion prevention (rileva e blocca)

CAPITOLO 2 - STRUMENTI CRITTOGRAFICI

La **crittografia simmetrica** (o crittografia convenzionale o a chiave singola) è il primo sistema di crittografia creato ed è ancora il più utilizzato. Si compone di 5 ingredienti:

1. **Testo in chiaro**: messaggio che funge da input dell'algoritmo di crittografia;
2. **Algoritmo di crittografia**: algoritmo che esegue varie trasformazioni sul testo in chiaro;
3. **Chiave segreta**: altro input dell'algoritmo, da cui dipendono le trasformazioni effettuate;
4. **Testo cifrato**: output dell'algoritmo;
5. **Algoritmo di decrittografia**: algoritmo che, prendendo in input chiave e testo crittografato, restituisce il testo in chiaro.

Richiede due requisiti per il suo uso sicuro:

- ***Potente algoritmo di crittografia***. In tal modo l'avversario, nonostante abbia qualche messaggio in chiaro e i corrispettivi crittografati, non possa giungere facilmente alla chiave.
- ***Mittente e destinatario devono ottenere e mantenere la chiave in modo sicuro***

Esistono due tipo di attacchi a sistemi di crittografia simmetrica:

- **Crittoanalisi**: necessitano la conoscenza della natura dell'algoritmo, delle caratteristiche generali del testo in chiaro e di alcune coppie di testo in chiaro e relativo crittografato. Questo attacco sfrutta queste conoscenze per dedurre la chiave e riuscire a compromettere in questo modo tutti i messaggi crittografati con tale chiave;
- **Brutal force**: consiste nel provare ogni chiave possibile su un pezzo di testo cifrato fino ad ottenere una traduzione intelligibile. In media sarebbe necessario provare la metà delle chiave esistenti e soprattutto avere conoscenza di come il testo in chiaro dovrebbe essere dato che poi spetta all'attaccante capire se la decriptazione è andata a buon fine.

Algoritmi di crittografia a blocchi simmetrici

Gli algoritmi di crittografia simmetrica più usati sono i codici a blocchi, ossia quelli che elaborano il testo in chiaro in blocchi di dimensione fissa e producono altrettanti blocchi di testo cifrato. I più importanti sono il ***Data Encryption Standard*** (**DES**), il ***Triplo DES*** (**3DES**) e l'***Advanced Encryption Standard*** (**AES**)

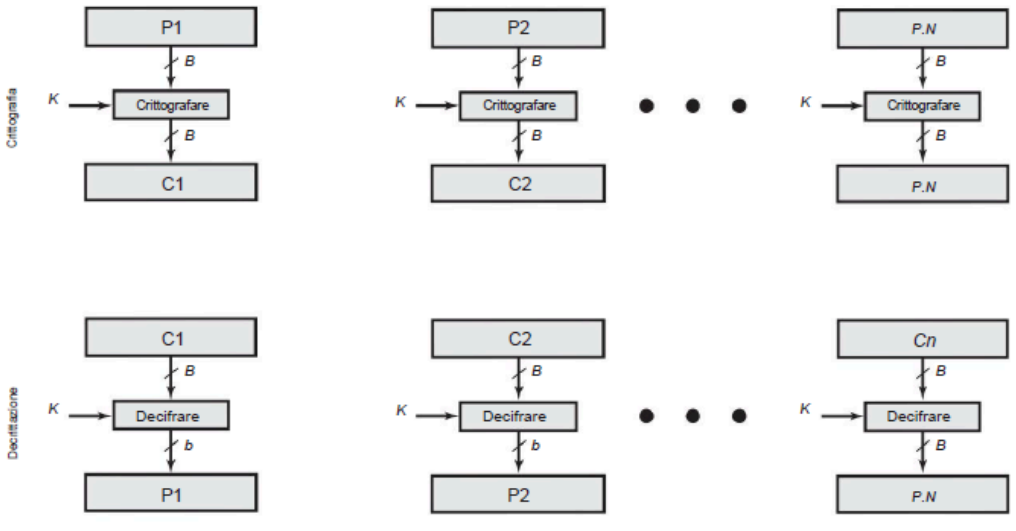
	DES	Triplo DES	AES
Dimensione del blocco di testo normale (bit)	64	64	128
Dimensioni del blocco di testo cifrato (bit)	64	64	128
Dimensione chiave (bit)	56	112 o 168	128, 192 o 256

La preoccupazione più seria riguardo al DES dipende dalla lunghezza della chiave (56 bit), la quale permette l'esistenza di troppe poche chiavi, rintracciabili anche con processori commerciali. Per risolvere tale problema è stato adottato il Triplo DES, che corrisponde semplicemente ad utilizzare due o tre volte l'algoritmo DES. In questo modo la chiave assume la lunghezza di 112 o 168 bit. L'unico svantaggio è che l'algoritmo di 3DES è relativamente lento. È da considerare poi che la lunghezza di blocco sarebbe auspicabile avere una lunghezza maggiore ai 64 bit imposti dal DES. Per questo motivo è stato introdotto l'algoritmo AES (algoritmo Rijndael - novembre 2001), con lunghezza di blocco di 128 bit e chiave a 128, 192 o 256 bit.

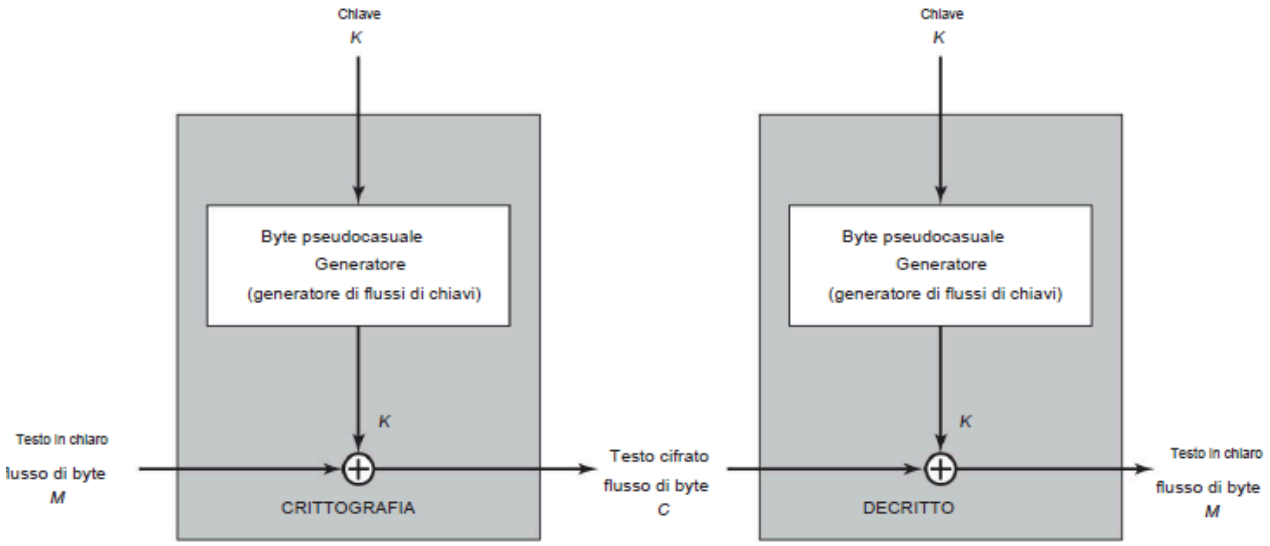
Dato che il testo in chiaro da crittografare è in genere più lungo della lunghezza del singolo blocco, si usa un approccio per la crittografia di blocchi multipli, di cui il più noto è ***l'Electronic Codebook*** (**ECB**), che consiste nel crittografare tutti i blocchi con lo stesso algoritmo e la stessa chiave. In questo modo però il testo crittografato potrebbe essere oggetto di crittoanalisi allo scopo di capirne la chiave utilizzata. Per evitare questo problema sono state sviluppate numerose tecniche, chiamate modalità operative.

Algoritmi di crittografia a flusso

Un algoritmo di crittografia a flusso elabora continuamente gli elementi di input, producendo l'output un elemento alla volta, man mano che procede. Sebbene i codici a blocchi siano più comuni, ci sono situazioni in cui quelli a flusso sono più favorevoli. In questo tipo di algoritmi la chiave produce uno stream di numeri pseudocasuali imprevedibile, utilizzati per crittografare e decriptare i vari bit o byte del messaggio.



(a) Crittografia a blocchi (modalità codebook elettronico)



(b) Crittografia del flusso

Autenticazione dei messaggi

La crittografia protegge dagli attacchi passivi (intercettazione); per evitare gli attacchi attivi (falsificazione dei dati) si procede con l'autenticazione dei messaggi. Un messaggio è autentico se proviene da una fonte sicura e non è stato contraffatto.

Autenticazione mediante crittografia simmetrica

Meccanismo di autenticazione nel quale solo il mittente e il destinatario condividono una chiave. Il messaggio può anche includere un meccanismo di ***controllo dell'errore*** ed un ***numero di sequenza***, in modo tale che il destinatario può verificare che non siano state apportate modifiche e che la sequenza sia corretta. Può essere inserito anche un timestamp per verificare che il messaggio non abbia subito ritardo.

La sola crittografia simmetrica non è un meccanismo sufficiente per l'autenticazione dei dati, ad esempio un attaccante potrebbe riordinare i blocchi, non facendo fallire il meccanismo di decriptazione, ma alterando il significato del messaggio.

Autenticazione dei messaggi senza crittografia dei messaggi:

In questo caso il messaggio non viene criptato e quindi la riservatezza non viene garantita. Invece, un tag di autenticazione viene generato e aggiunto a ciascun messaggio per la trasmissione. Esempi di situazioni nelle quali il messaggio viene autenticato senza badare alla riservatezza sono:

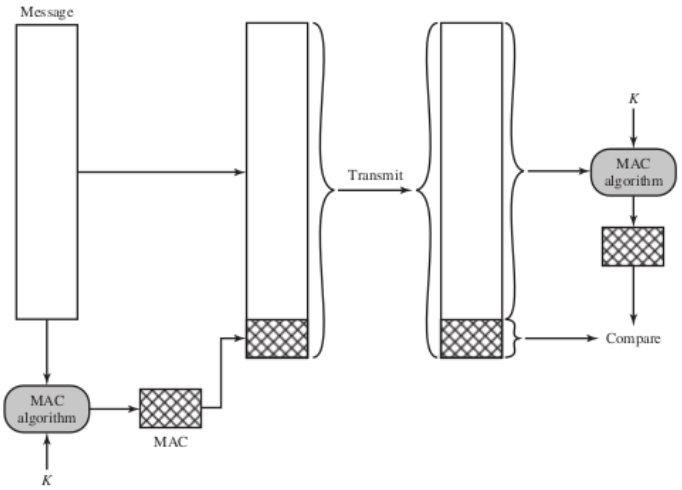
1. Quando viene mandato lo stesso messaggio in broadcast a più destinazioni. (*es. inviare una notifica di allarme in un centro di controllo*). Quindi il messaggio deve essere trasmesso in

chiaro con tag di autenticazione associato. Il sistema responsabile esegue l'autenticazione, che, nel caso fallisse, genera un allarme generale;

2. Quando si ha lo scambio di pacchetti di informazione molto pesanti (bytes), e non è possibile decriptare tutti i messaggi in arrivo. Si procede quindi tramite autenticazione, la quale viene effettuata a campione su alcuni messaggi casuali;
3. Quando si utilizza un programma per computer sarebbe bene non decriptarlo ogni volta, al fine di salvare risorse del processore. Tuttavia è necessario un tag di autenticazione allegato a tale programma.

Esistono varie tecniche di autenticazione:

- **Codice di autenticazione del messaggio (MAC):**



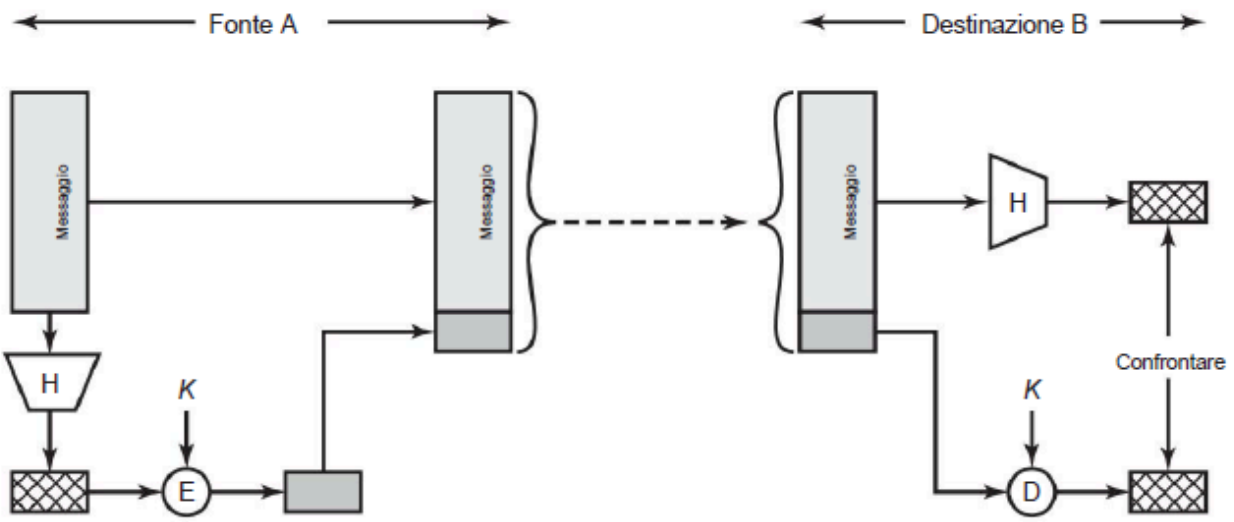
Tecnica utilizzata per generare una piccolo blocco di dati, il ***codice di autenticazione*** (**MAC)** utilizzando una ***chiave segreta*** (**K)** e una ***funzione complessa*** (**F(K, M))**. Il blocco di dati viene in seguito allegato al messaggio. Il destinatario può controllare se il messaggio sia stato corrotto dato che:

- Se un malintenzionato altera il messaggio, quando il destinatario riceverà il messaggio e calcolerà il *MAC* si renderà conto che c'è stata una manomissione.
- Siccome la chiave è segreta, nessuno potrebbe generare il *MAC* utilizzando un'altra chiave.

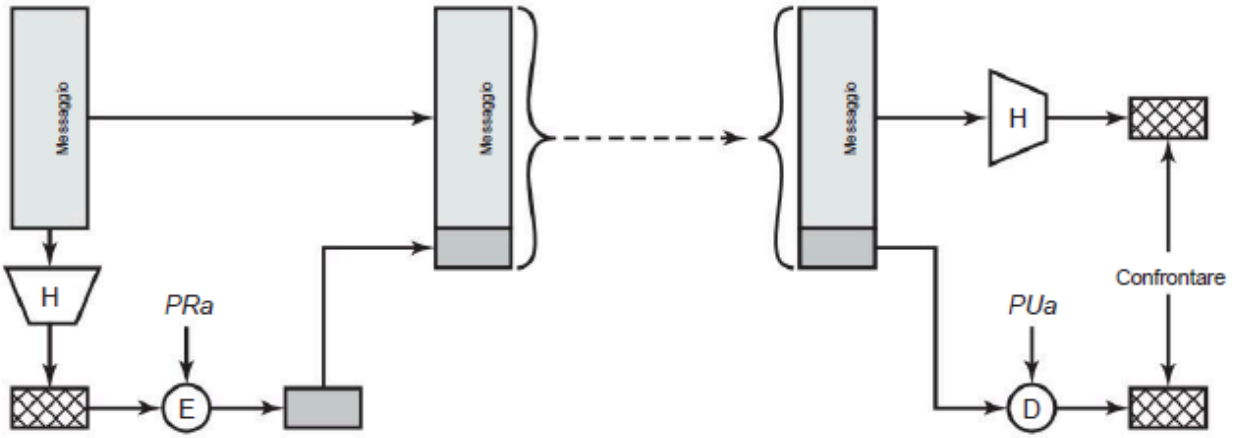
Grazie alle proprietà della funzione di autenticazione questa tecnica è meno vulnerabile rispetto alla criptazione.

- **One-Way Hash Function:**

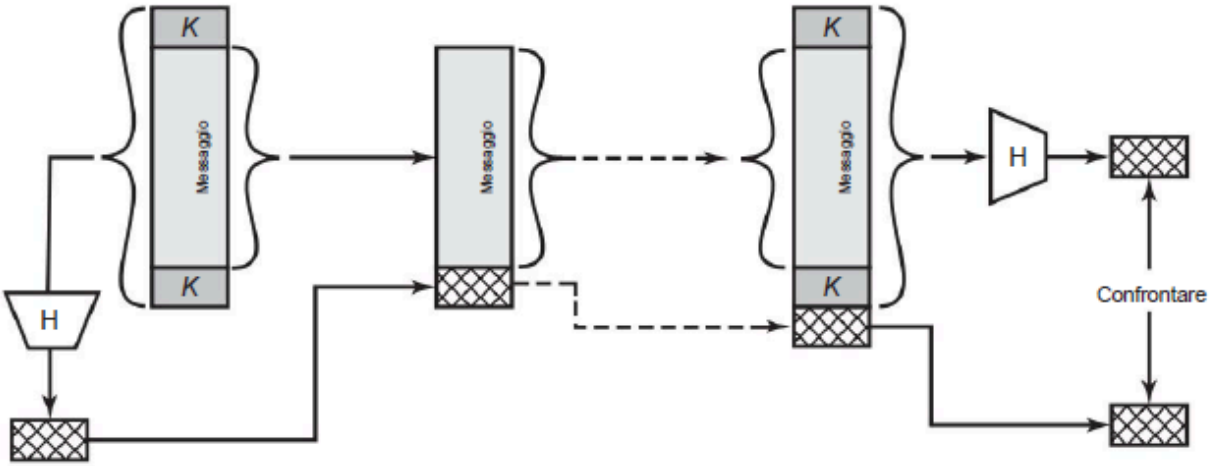
Questa tecnica utilizza una ***funzione di hash*** che, preso in input un messaggio di lunghezza variabile, produce una stringa di lunghezza fissa, detta ***digest***. Il digest può essere criptato utilizzando una ***crittografia simmetrica***, se si presuppone che src e dst condividano la chiave, una ***crittografia a chiave pubblica***, o ancora potrebbe non essere criptato. In questo ultimo caso la tecnica, nota come ***keyed hash MAC***, presuppone che le due parti condividano una chiave segreta



(a) Utilizzo della crittografia simmetrica



(b) Utilizzo della crittografia a chiave pubblica



(c) Utilizzo del valore segreto

Ovviamente per funzionare bene, le funzioni hash devono rispettare 6 requisiti:

1. H può essere applicata a un blocco di dati di qualsiasi dimensione;
2. H produce un output di lunghezza fissa;
3. Per ogni x, H(x) è relativamente semplice da calcolare;
4. Per ogni h = H(x) è computazionalmente impossibile trovare x (per questo motivo H è detta funzione unidirezionale); [*è impossibile ricavare il messaggio originale da uno codificato*]
5. Per ogni x, è computazionalmente impossibile trovare $y \neq x$, con H(y) = H(x) [*è impossibile sostituire il messaggio vero con uno falso ma con lo stesso hash code*]
6. È computazionalmente impossibile trovare una coppia (x, y) tale che H(y) = H(x) (per questo motivo H.

Se H soddisfa solo i primi cinque requisiti si parla di ***funzione di hash debole***, mentre se li soddisfa tutti viene detta ***funzione di hash forte***.

Crittografia a chiave pubblica

La **crittografia a chiave pubblica** è **asimmetrica** e prevede l'utilizzo di due chiavi separate. In questo modo ci sono ampi miglioramenti nella sicurezza e contro la crittoanalisi. Gli algoritmi caratteristici poi si basano su funzioni matematiche piuttosto che su semplici operazioni su bit. Proprio per questo motivo però viene preferita la crittografia a chiavi simmetriche talune volte, dato che il suo costo computazionale è minore.

Uno schema di crittografia a chiave pubblica ha 6 ingredienti:

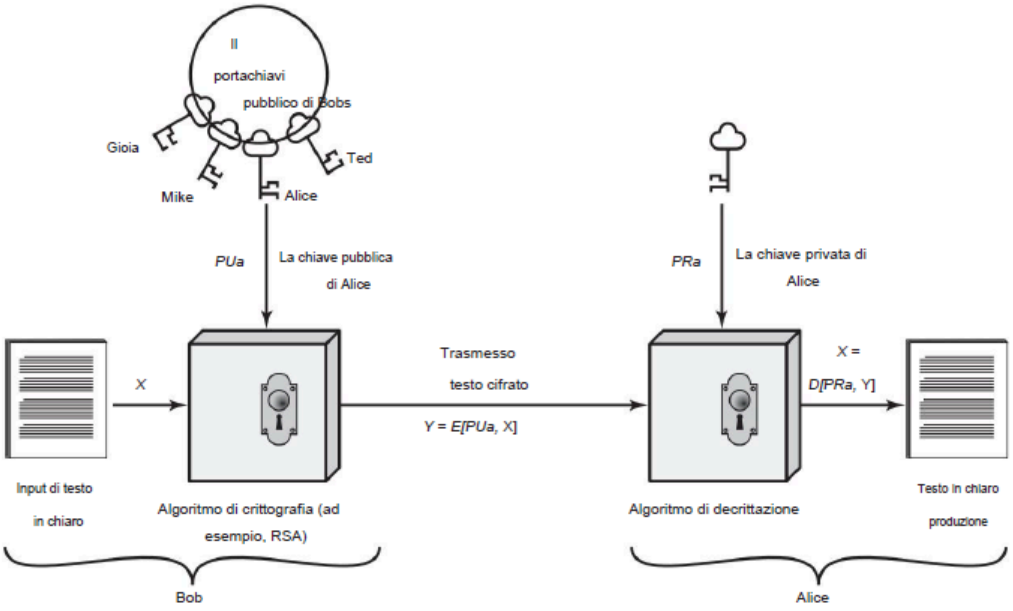
1. **Testo in chiaro**: messaggio che funge da input dell'algoritmo di crittografia;
2. **Algoritmo di crittografia**: algoritmo che esegue varie trasformazioni sul testo in chiaro;
3. **Chiave pubblica e privata**: altro input dell'algoritmo, da cui dipendono le trasformazioni effettuate. Una viene usata per la crittografia e una per la decriptazione;
4. **Testo cifrato**: output dell'algoritmo;
5. **Algoritmo di decriptazione**: algoritmo che, prendendo in input chiave e testo crittografato, restituisce il testo in chiaro.

I passaggi essenziali sono:

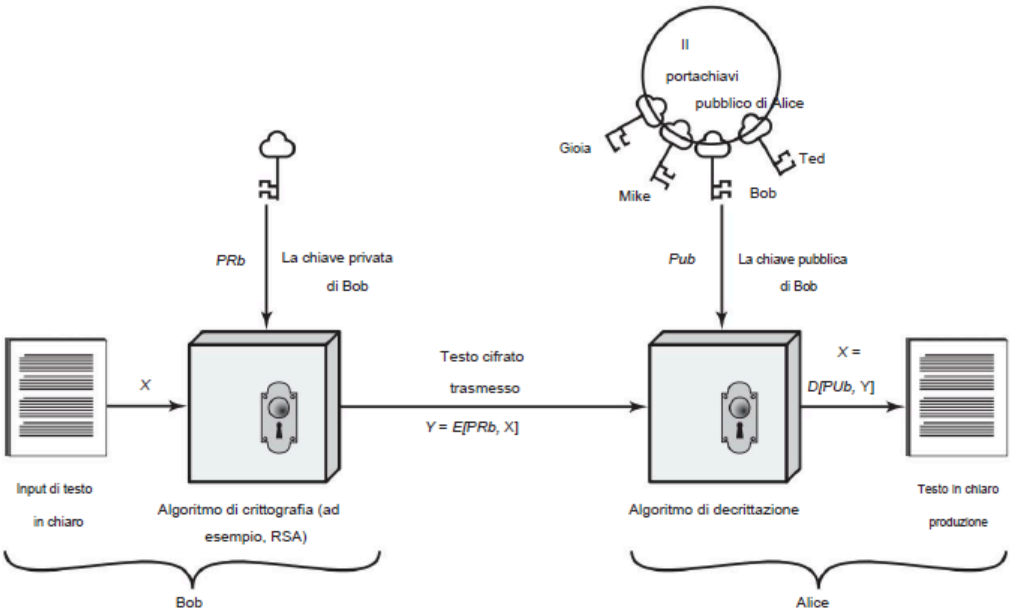
- Ogni utente genera una coppia di chiavi da utilizzare per la crittografia e per la decriptazione;

- Ciascun utente inserisce una delle due chiavi in un registro pubblico (***chiave pubblica***), mentre l'altra viene mantenuta privatamente;
- Il mittente, prima di inviare il messaggio, lo cripta con la chiave pubblica del destinatario;
- Il destinatario lo decripta utilizzando la sua chiave privata. Nessun altro destinatario può decriptare il messaggio dato che non ha la chiave privata.

Esiste una variante di questo meccanismo che prevede la criptazione con chiave privata e la decriptazione con chiave pubblica. In questo caso però chiunque abbia la chiave pubblica del mittente può decriptare il messaggio, quindi si perde la riservatezza.



(a) Crittografia con chiave pubblica



Il sistema crittografico a chiave pubblica deve soddisfare dei requisiti:

- È computazionalmente facile generare la coppia di chiavi;
- È computazionalmente facile per il mittente cifrare un messaggio conoscendo la chiave pubblica;
- È computazionalmente facile per il destinatario decifrare un messaggio utilizzando la chiave privata;
- È computazionalmente impossibile per un avversario determinare la chiave privata conoscendo quella pubblica;

Sono quattro i principali algoritmi:

- **RSA**: uno dei primi algoritmi sviluppato ed ancora quello più utilizzato. È un codice a blocchi nel quale il testo in chiaro e quello cifrato sono numeri interi compresi tra 0 e n-1 per qualche n;
- **Algoritmo Scambio delle Chiavi di Diffie-Hellman**: il primo algoritmo di questo tipo. Consente un accordo su un segreto condiviso tra mittente e destinatario che può essere usato per la successiva criptazione simmetrica. Tale caratteristica però è comunque limitante in termini di sicurezza;
- **Digital Signature Standard**: utilizza un algoritmo progettato per fornire solo la funzione di firma digitale;
- **Curva ellittica**: sembra offrire la stessa sicurezza dell'RSA ma con chiavi di lunghezza minore. Il livello di fiducia in questo algoritmo non è ancora elevato dato il costante interesse crittoanalitico nel sondarne i punti deboli.

Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
RSA	Yes	Yes	Yes
Diffie-Hellman	No	Yes	No
DSS	Yes	No	No
Elliptic Curve	Yes	Yes	Yes

Firma digitale

La crittografia pubblica può essere utilizzata anche per l'autenticazione. Ad esempio, nel caso in cui il mittente volesse inviare un messaggio non tenendo conto della riservatezza, ma volendo solo essere certo che il destinatario sappia che effettivamente il messaggio provenga da lui, può utilizzare una ***firma digitale***. Essa è generata a partire dal messaggio, il quale viene prima dato in

pasto ad una funzione di hash e poi criptato con la chiave privata del mittente. In questo modo il destinatario può decriptare la firma con la chiave pubblica del mittente e criptare il messaggio con la funzione di hash. Nel caso in cui il valore ottenuto sia uguale a quello inviato allora il destinatario sarà sicuro che il messaggio sia arrivato effettivamente da quel destinatario.

La caratteristica principale della crittografia a chiave pubblica è per l'appunto che la chiave è pubblica e quindi tutti possono trasmettere una chiave alla comunità. Ciò però rappresenta anche un problema, dato che chiunque si può spacciare per qualcun altro e pubblicare una chiave a suo nome. A quel punto, e fino a che il vero utente non se ne accorge e avverte tutti, l'avversario può leggere tutti i messaggi criptati e usare la chiave falsa per l'autenticazione. Per risolvere questo problema si usa il ***certificato a chiave pubblica***, costituito da una chiave pubblica e dall'ID utente del proprietario della chiave. L'intero blocco è poi firmato da una terza parte fidata, l'***autorità di certificazione*** (**CA**), ad esempio un'autorità governativa o comunque un ente ritenuto affidabile dalla comunità. In sintesi i passaggi chiave per produrre tale certificato sono:

- Il client crea la coppia di chiavi (pubblica e privata);
- Il client prepara un certificato che include la chiave e l'ID utente;
- L'utente fornisce tale bozza di certificato ad una CA in modo sicuro (magari tramite incontro fisico tra le parti o comunque tramite canali telematici sicuri);
- La CA crea una firma utilizzando una funzione hash sul certificato non firmato e criptando il valore ottenuto utilizzando la propria chiave privata;
- La CA allega la firma al certificato e restituisce tale certificato al client.

A questo punto qualsiasi utente può verificare la validità del certificato calcolando il codice hash del certificato (escludendo la firma) e decodificando la firma utilizzando la chiave pubblica della CA. Nel caso i valori ottenuti corrispondano a quelli del messaggio recapitato allora il certificato è valido.

Un'applicazione in cui la crittografia pubblica viene utilizzata per proteggere una chiave simmetrica è la ***busta digitale***. Essa si sviluppa come segue:

- Il mittente prepara il messaggio;
- Il mittente genera una chiave simmetrica casuale che verrà utilizzata solo questa volta;
- Il mittente cripta il messaggio utilizzando la chiave monouso e cripta la chiave monouso utilizzando la chiave pubblica del destinatario;
- Il mittente allega la chiave monouso al messaggio ed invia tutto al destinatario;
- Il destinatario è l'unico che può decriptare il messaggio dato che deve usare la propria chiave privata.

CAPITOLO 3 - AUTENTICAZIONE UTENTI

L'autenticazione digitale degli utenti è definita come il processo atto a stabilire la fiducia nelle identità degli utenti che sono presentate elettronicamente ad un sistema informativo. I sistemi possono quindi utilizzare l'identità autenticata per determinare se l'individuo autentico è autorizzato ad effettuare particolari funzioni e ad accedere a particolari risorse.

È importante distinguere l'***identificazione*** (meccanismo con cui l'utente presenta un'identità presunta) dall'***autenticazione*** (meccanismo per stabilire la validità dell'identità presunta). Fondamentale è chiarire che quest'ultima autenticazione differisce da quella riguardante i messaggi (quest'ultima infatti permette di verificare che la fonte di un messaggio sia autentica e che il suo contenuto non sia stato alterato).

I requisiti di sicurezza basici per servizi di identificazione e autenticazione sono fondamentalmente due:

1. Identificare gli utenti del sistema informativo, i processi che agiscono per conto degli stessi utenti e i dispositivi;
2. Verificare (autenticare) le identità di tali utenti, processi e dispositivi come prerequisito per concedere l'accesso alle informazioni di sistema.

Da questi requisiti ne scaturiscono altri derivati:

1. Usare l'autenticazione a più fattori per accessi locali o di rete ad account privilegiati e per accessi di rete ad account non privilegiati;
2. Impiegare meccanismi di autenticazione resistenti alla riproduzione per accessi ad utenti privilegiati e non;
3. Prevenire l'utilizzo di identificatori per un periodo definito;
4. Disabilitare identificatori dopo un tot di tempo;
5. Applicare requisiti di complessità minima alle password ed evitare che vengano aggiornate password con delle nuove che sono troppo simili alle vecchie;
6. Proibire riuso di password per uno specifico numero di generazioni;
7. Consentire l'uso di password temporanee per accessi una-tantum al sistema (password che devono poi essere cambiate immediatamente dopo);
8. Salvare e trasmettere password solo se protette crittograficamente;
9. Oscurare feedback con informazioni di autenticazione.

Il requisito iniziale per eseguire l'autenticazione è che l'utente sia registrato. Per fare ciò:

- Il ***richiedente*** presenta domanda ad un'***autorità di registrazione*** (**RA**) per diventare subscriber di un ***fornitore di servizi di credenziali*** (**CSP**);
- La RA, entità fidata, garantisce l'identità del richiedente al CSP facendo avviare quindi una comunicazione tra questi due soggetti;
- Il CSP emette una sorta di credenziale elettronica al subscriber. La ***credenziale*** è una struttura dati che lega in modo autorevole un'identità ad un token ad un subscriber.
- Il token e le credenziali possono essere utilizzati per le successive autenticazioni. Esso può essere generato dal subscriber, dal CSP o fornito da terze parti.

Una volta registrato, un utente può autenticarsi come segue:

- La parte da autenticare è detta ***richiedente***, mentre quella che deve verificare è detta ***verificatore***;
- Quando il richiedente dimostra con successo il possesso e controllo di un token al verificatore, quest'ultimo può verificare che il richiedente sia effettivamente il subscriber indicato dalle credenziali;
- Il verificatore trasmette un'asserzione alla ***aware party*** (**RP**) contenente informazioni sull'identità del richiedente;
- L'RP utilizza tali informazioni autenticate fornite dal verificatore per prendere decisioni sul controllo degli accessi o sull'autorizzazione.

I sistemi di autenticazione possono ovviamente essere più complicati di questo modello, ma la base rimane questa.

Generalmente esistono quattro modi per autenticare l'identità di un utente:

- ***Qualcosa che l'individuo conosce*** (ad es. password, PIN, ecc…)
- ***Qualcosa che l'individuo possiede*** (ad es. token fisici)
- ***Qualcosa che l'individuo è*** [**biometria statica**] (ad es. impronte digitali, retina, ecc…)
- ***Qualcosa che l'individuo fa*** [**biometria dinamica**] (ad es. voce, firma, ecc…)

Ovviamente ognuno di questi metodi presenta dei problemi. Un ***livello di garanzia*** è definito come:

- Grado di fiducia che la persona che richiede la registrazione sia effettivamente la persona che dice di essere;

- Grado di fiducia che la persona che richiede l'accesso sia effettivamente la persona a cui appartengono le credenziali.

Esistono 4 livelli:

Livello 1: Poca o nessuna fiducia (es. login e password);

Livello 2: Una certa fiducia (es. OTP);

Livello 3: Elevata fiducia (es. caratteristiche biometriche);

Livello 4: Massima fiducia (es. riconoscimento fisico).

Un concetto correlato a quello del livello di garanzia è l'***impatto potenziale***, cioè il livello del potenziale impatto su soggetti in caso di errore sull'autenticazione dell'utente:

1. **Basso**: effetto negativo limitato. L'errore potrebbe:
    - causare un degrado della capacità della missione in misura e durata tali da consentire all'organizzazione di svolgere le sue funzioni primarie, ma riducendone l'efficacia;
    - provocare danni minori alle risorse organizzative;
    - comportare minori risultati finanziari per l'organizzazione o per gli individui;
    - provocare danni minori alle persone;
2. **Moderato**: effetto negativo grave. L'errore potrebbe:
    - causare un significativo degrado della capacità della missione in misura e durata tali da consentire all'organizzazione di svolgere le sue funzioni primarie, ma riducendone significativamente l'efficacia;
    - provocare gravi danni alle risorse organizzative;
    - comportare gravi perdite finanziarie per l'organizzazione o per gli individui;
    - provocare danni significativi alle persone (escludendo lesioni letali);
3. **Alto**: effetto negativo catastrofico. L'errore potrebbe:
    - perdita della capacità della missione in misura e durata tali da non consentire all'organizzazione di svolgere le sue funzioni primarie;
    - provocare gravi danni alle risorse organizzative;
    - comportare gravi perdite finanziarie per l'organizzazione o per gli individui;
    - provocare danni significativi alle persone (possibili lesioni letali);

Per valutare i rischi quindi si elencano delle ***aree di rischio*** (ad es. sicurezza delle persone) e in base all'*impatto potenziale* si va a decidere il *livello di garanzia* che si vuole attuare.

	Assurance Level Impact Profiles			
Potential Impact Categories for Authentication Errors	**1**	**2**	**3**	**4**
Inconvenience, distress, or damage to standing or reputation	Low	Mod	Mod	High
Financial loss or organization liability	Low	Mod	Mod	High
Harm to organization programs or interests	None	Low	Mod	High
Unauthorized release of sensitive information	None	Low	Mod	High
Personal safety	None	None	Low	Mod/ High
Civil or criminal violations	None	Low	Mod	High

Autenticazione basata su password

Quasi tutti i sistemi informativi odierni richiedono agli utenti un identificativo e una password. Il sistema confronta la password ottenuta dall'utente con quella memorizzata per quell'identificativo. L'identificativo fornisce sicurezza determinando:

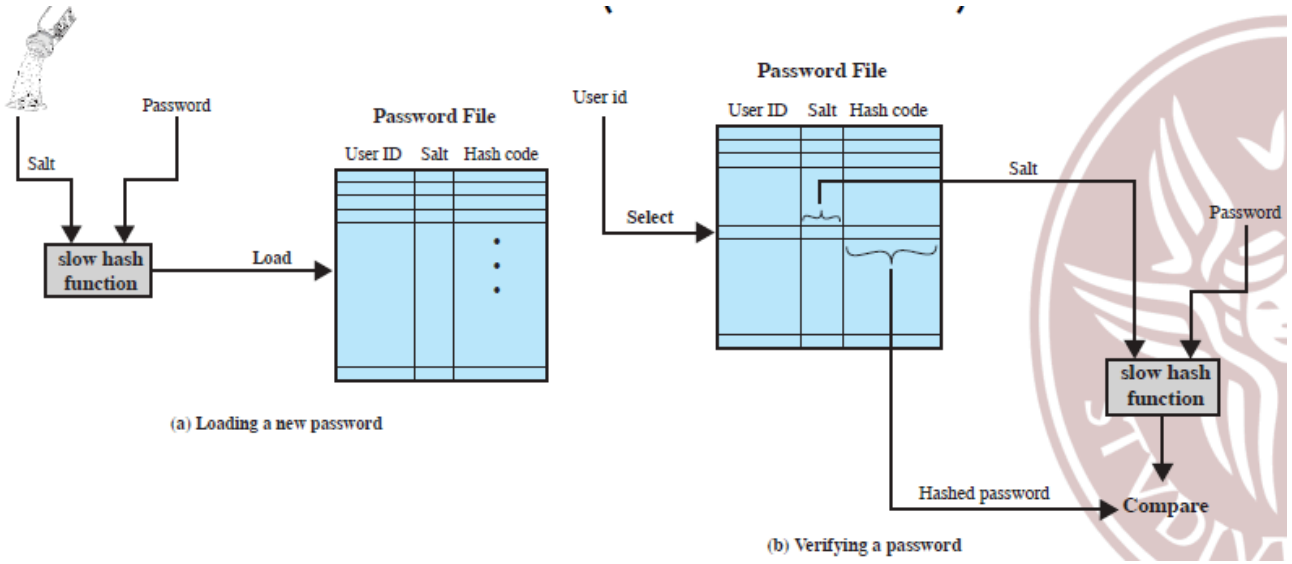
- se l'utente è autorizzato ad accedere al sistema (se l'utente possiede un ID);
- i privilegi concessi all'utente;
- l'accesso discrezionale (cioè la concessione del permesso di leggere i propri file ad altri utenti).

Questo tipo di autenticazione ovviamente presta il fianco a molte forme di attacco:

- **Attacco con dizionario offline**: l'attaccante ottiene accesso al file delle password e può confrontare gli hash delle password con gli hash di password comunemente utilizzate. Se viene trovata una corrispondenza, l'attaccante può ottenere l'accesso al sistema. [*Le contromisure includono controlli per prevenire accessi al file delle password e intrusion detection per identificarne compromissioni*];
- **Attacco ad account specifico**: l'attaccante prende di mira un account specifico e tenta varie password. [*Le contromisure includono un meccanismo di blocco dell'account dopo un tot di tentativi sbagliati*];
- **Attacco con password popolare**: l'attaccante seleziona una password utilizzata comunemente e la testa con vari ID. [*Le contromisure includono politiche per inibire la scelta di password comuni e scansioni IP per capire da dove provengono le richieste di autenticazione*];

- **Indovinare la password di un singolo utente**: l'attaccante acquisisce conoscenze riguardo al titolare dell'account selezionato e cerca di capire così la sua password. [*Le contromisure includono politiche di gestione password che le rendano difficili da indovinare*];
- **Sequestro della postazione di lavoro**: l'attaccante attende che la postazione di lavoro con login attivo sia lasciata incustodita. [*Le contromisure includono logout automatico e intrusion detection per rilevare comportamenti anomali degli utenti*];
- **Sfruttare errori dell'utente**: l'attaccante sfrutta cattive abitudini degli utenti (password scritte su fogli, condivisione di informazioni di accesso con colleghi, phishing, ecc…). [*Le contromisure includono l'educazione degli utenti (awareness), l'intrusion detection e l'autenticazione a più fattori*];
- **Sfruttare l'utilizzo di password multiple**: l'attaccante sfrutta situazioni nelle quali si utilizzano le stesse password per più servizi. [*Le contromisure includono politiche per vietare l'uso di password uguali o simili. In questo caso però ciò è possibile solo se le password sono usate su sistemi dello stesso proprietario*];
- **Monitoraggio elettronico**: l'attaccante intercetta una password trasmessa tramite rete. [*Le contromisure includono qualcosa in più della sola cifratura e protocolli di comunicazione sicura*].

Una tecnica di sicurezza delle password ampiamente utilizzata è l'uso di password con hash e un valore di salt di lunghezza fissa (un numero pseudo-casuale).



Il sale ha tre scopi:

- Impedire che password uguali siano visibili (avendo salt diversi, hanno valori hash diversi);

- Aumentare la difficoltà negli attacchi offline di un fattore 2^b, dove b è la lunghezza del salt;
- Rendere impossibile scoprire se un utente ha usato la stessa password più volte.

Cracking delle password

Esistono vari approcci per cercare di indovinare la password (***password cracking***). I tradizionali sono:

- **Attacco con dizionario**: si sviluppa un ampio dizionario di possibili password, le quali vengono sottoposte a funzioni di hash con tutti i possibili salt per trovare una corrispondenza;
- **Tabella arcobaleno**: si sviluppa un'immensa tabella con le possibili password pre-processate e che devono solo essere confrontate. Una possibile difesa sarebbe quella di usare un salt abbastanza lungo ma un possibile problema potrebbe presentarsi nel caso in cui la password sia semplice da indovinare (contenga informazioni conosciute sull'utente come il nome);
- **Combinazione di forza bruta e dizionari**: un esempio è l'algoritmo di John the Ripper.

Con gli anni la password cracking si è sviluppata e ha richiesto una politica delle password sempre più complessa, la quale nega agli utenti la possibilità di scegliere password semplici e veloci da carpire. Le tecniche di cracking sono migliorate grazie soprattutto a:

- La capacità di elaborazione disponibile per il cracking delle password è aumentata notevolmente;
- L'uso di algoritmi sofisticati per generare potenziali password, utilizzando la probabilità di caratteri nel linguaggio naturale e il modello di Markov;
- Lo studio di esempi e strutture delle password attualmente in uso

Uno dei modi migliore per contrastare un attacco con password è negare all'avversario l'accesso al file delle password, magari distinguendo la tabella degli ID da quella delle password (***file delle password shadow***) e consentendo l'accesso solo ad utenti privilegiati. Rimangono però alcune vulnerabilità:

- Debolezze del sistema che consentono comunque l'accesso al file:
- Problemi con i permessi che rendono il file leggibile;
- Utenti con la stessa password su diversi sistemi;
- Accesso tramite file di backup, meno controllato;

- Sniffing del traffico di rete.

L'obiettivo delle politiche delle password è rendere le password difficili da indovinare e nel contempo non complicatissime da ricordare per gli utenti. Per fare ciò si usano quattro tecniche di base:

- ***Educazione degli utenti***: dare delle linee guida agli utenti per la scelta della password. Questa tecnica però non funzionerà quasi mai, dato che la maggior parte delle persone semplicemente ignoreranno le linee guida e qualcun'altro invece penserà che usando lettere grandi possa risolvere il problema [*genio del male*]. Una soluzione potrebbe essere usare le iniziali di una frase, ammesso che essa non sia una frase nota;
- ***Password generate da computer***: in questo caso il problema sta nel ricordare tale password. Una soluzione potrebbe essere quella di usare algoritmi che producono password fatte da sillabe pronunciabili e quindi memorizzabili;
- ***Controllo reattivo della password***: il sistema esegue periodicamente il proprio password cracker per mostrare delle password deboli e richiederne la modifica all'utente di riferimento;
- ***Politica di password complessa***: obbligare l'utente a scegliere una password con caratteristiche che la rendano complicata da indovinare.

Controllo proattivo delle password

Può essere svolto attraverso tre approcci:

1. **Applicazione delle regole**: consiste nell'imporre delle regole per le password, come quella per cui la lunghezza deve essere maggiore o uguale a 8 oppure come quella per cui nei primi otto caratteri ci devono essere maiuscole, minuscole, cifre e segni di punteggiatura;
2. **Controllo password**: consiste nel creare un dizionario con tutte le possibili password facili da indovinare, in modo tale da negare ad un utente di scegliere una di quelle memorizzate. Questo approccio però ha problemi di spazio (il dizionario dovrebbe essere immenso per essere efficace) e di tempo (il tempo necessario per controllare una password potrebbe essere elevato;
3. **Filtro Bloom**: viene usato per rifiutare password presenti in una struttura dati. Un filtro di Bloom di ordine k è costituito da k funzioni di hash indipendenti, ciascuna delle quali mappa una password in un valore hash. Quando una nuova password viene presentata, vengono calcolati i valori con tutte le k funzioni di hash. Viene controllata all'interno della matrice del filtro e se tutti i bit corrispondenti nella tabella hash sono uguali a 1 allora la

password viene rifiutata. Il filtro di Bloom è molto conveniente in termini di efficienza, ma può presentare falsi positivi, ossia password che non sono presenti nella matrice ma che vengono comunque scartate. Non può avvenire però il contrario, cioè che una password presente non sia rivelata.

Autenticazione basata su token

I ***token*** sono oggetti che l'utente possiede e utilizza ai fini dell'autenticazione. Esempi di token sono:

- **Token in rilievo**, *come le vecchie carte di credito*;
- **Token con banda magnetica**, *come le carte di credito odierne non contacltess*;
- **Token con memoria**, *come le SIM*;
- **Token contactless**, *come le carte di credito contactless*.

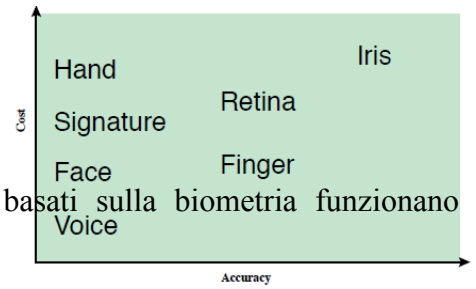
Autenticazione biometrica

Un sistema di autenticazione biometrica utilizza le caratteristiche fisiche di un utente per identificarlo. Essa si basa sul riconoscimento di pattern e proprio per questo, rispetto a password e token, rappresentano una modalità più complessa e costosa. Le principali caratteristiche fisiche utilizzate in applicazioni biometriche sono:

- ***Caratteristiche della faccia***: sono ciò che usiamo nella realtà per identificare le persone. L'approccio più naturale è quello di prendere in considerazione le posizioni e le forme delle principali caratteristiche facciali (occhi, labbra, naso, ecc…). Un'alternativa sarebbe quella di usare una telecamera a infrarossi per produrre un termogramma correlato al sistema vascolare della faccia;
- ***Impronte digitali***: anche questo metodo è usato anche nella realtà. Si ritiene che le impronte digitali siano uniche in tutta la popolazione umana;
- ***Geometrica della mano***: usano caratteristiche come forma della mano, lunghezza e forma delle dita;
- ***Schema della retina***: anche lo schema delle vene sotto la superficie della retina è unico e quindi utilizzabile. Un sistema del genere ottiene un'immagine della retina tramite infrarossi a bassa intensità;
- ***Iride***: utilizza la struttura dettagliata dell'iride;
- ***Firma***: ogni individuo ha una scrittura unica e ciò si riflette anche sulla firma. Tuttavia uno stesso individuo non firma mai in modo identico;

- ***Voce***: anche in questo caso la voce potrebbe cambiare nel tempo e rendere complicata l'autenticazione degli utenti.

Il funzionamento di sistemi basati sulla biometria funzionano similmente a quelli basati su password o token:



- ***Iscrizione*** - L'utente si iscrive tipicamente con un nome e con una caratteristica biometrica. Quest'ultima viene processata, estraendo delle caratteristiche principali che vengono salvate in un database;
- ***Verifica*** - L'utente si identifica fornendo un nome e la caratteristica biometrica richiesta. Quest'ultima viene processata, estraendo delle caratteristiche principali che vengono confrontate con quelle salvate nel database, fornendo la risposta all'autenticazione;
- ***Identificazione*** - L'utente si identifica fornendo la caratteristica biometrica richiesta. Quest'ultima viene processata, estraendo delle caratteristiche principali che vengono confrontate a quelle salvate nel database per ottenere l'identità dell'utente;

Nel confronto menzionato precedentemente è da considerare che la caratteristica biometrica salvata non è precisa al 100% e che quindi deve essere usata una soglia di tolleranza minima. Ovviamente aumentare troppo la soglia di tolleranza potrebbe dare atto a falsi positivi, mentre abbassarla esageratamente produrrebbe casi di falsi negativi. È importante quindi equilibrare il tutto.

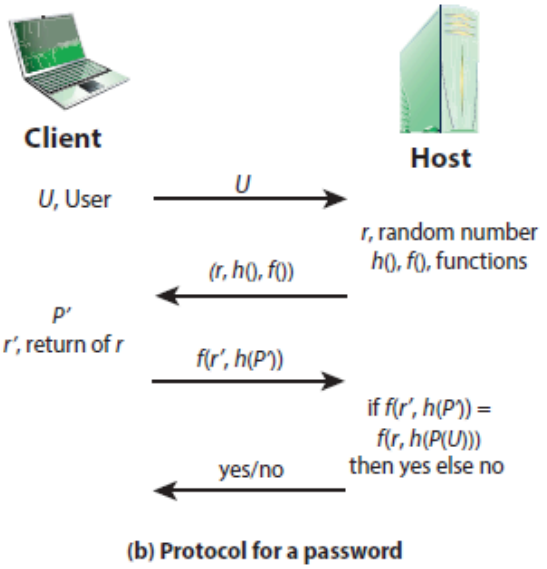
Autenticazione utente remoto

Il caso più complesso di autenticazione si ha quando l'utente si connette da remoto. Essa avviene tramite la rete, ad esempio Internet. Questo tipo di autenticazione è complessa dato che prevede ulteriori problemi di sicurezza come la possibilità che qualcuno possa intercettare una password o una sequenza di autenticazione. Per contrastare tali minacce i sistemi generalmente si affidano a protocolli challenge-response.

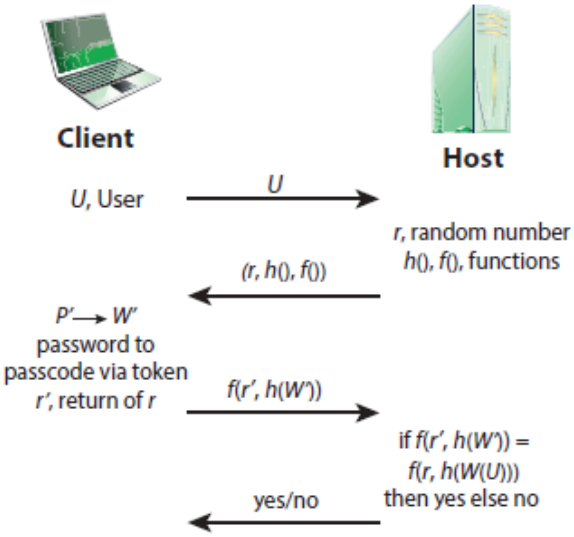
Un esempio dell'uso di questo protocollo per l'autenticazione tramite password da remoto è il seguente:

- L'utente trasmette all'host remoto la propria identità;
- L'host genera un numero casuale r (chiamata **nonce**) e lo restituisce all'utente, specificando due funzioni h() e f();

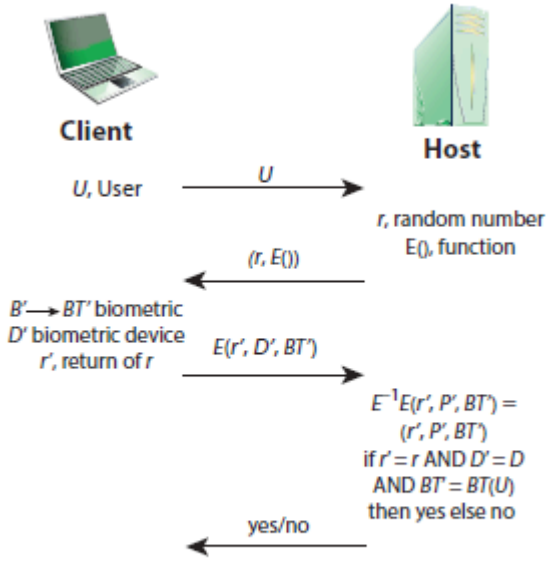
- L'utente risponde all'host con il valore f(r', h(P')), dove r'=r è il numero casuale e P' è la password;
- L'host confronta il messaggio dell'utente con f(r, h((P(U))), dove P(U) è la password dell'utente salvata. Se il confronto va a buon fine conferma l'autenticazione.



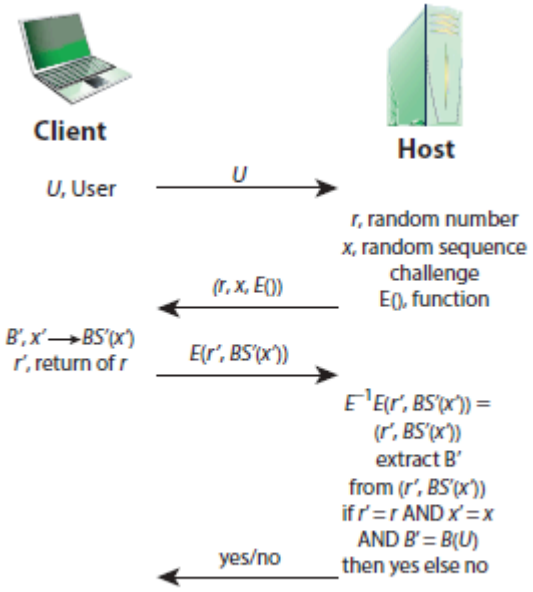
(b) Protocol for a password



(b) Protocol for a token



(c) Protocol for static biometric



(d) Protocol for dynamic biometric

Tipologie comuni di attacchi

L’autenticazione degli utenti, in particolare quella da remoto, è soggetta a numerosi attacchi. Alcuni esempi sono:

- **Attacchi Client**: sono quelli in cui l’avversario tenta di ottenere l'autenticazione dell’utente, senza passare dall’host. Esempi sono quando l’attaccante prova ad indovinare la password o a sequenziare tutte le possibili password. Un modo per contrastare ciò sarebbe quello di selezionare una password lunga e imprevedibile (grande entropia), oppure quello di limitare i tentativi di accesso;
- **Attacchi Host**: sono quelli diretti ai file dell’host in cui sono memorizzate informazioni chiave (ad es. password). Nella sezione dell’autenticazione basata su password sono citati i principali metodi per contrastare attacchi di questo genere, mentre per quanto riguarda i token un’ulteriore difesa potrebbe essere quella di usare passcode monouso. Le caratteristiche biometriche invece sono più difficili da proteggere dato che sono caratteristiche fisiche dell’utente. Si usano in questo caso il protocollo challenge-response;
- **Intercettazione**: sono attacchi mirati ad apprendere la password dell’utente sfruttando la vicinanza ad esso oppure tramite keylogging. Un modo per difendersi sarebbe quello di utilizzare un’autenticazione a più fattori;
- **Furto**: sono quelli diretti a rubare il token o una sua copia fisica. Il metodo più semplice sarebbe quella di utilizzare un’autenticazione a più fattori;
- **Copia**: sono quelli volti ad imitare caratteristiche biometriche dell’utente. Un modo per difendersi sarebbe quello di autenticare anche il dispositivo;
- **Riproduzione**: sono attacchi che coinvolgono un avversario che ripete una risposta dell’utente. La contromisura più comune sarebbe il protocollo challenge-response;
- **Trojan Horse**: sono quelli volti a mascherare un’applicazione o dispositivo come autentico, ma con il reale scopo di acquisire informazioni dall’utente. In questo modo l’attaccante può utilizzare le informazioni carpite per spacciarsi per l’utente;
- **Negazione del Servizio**: sono attacchi che tendono tentano di disabilitare un servizio di autenticazione inondandolo di richieste. Un protocollo di autenticazione a più fattori che consiste anche di un token può inibire tale attacco.

CAPITOLO 4 - CONTROLLO DEGLI ACCESSI

Il **controllo degli accessi** è definito come il *processo* di concessione o rifiuto di richieste specifiche per:

- ottenere ed utilizzare informazioni e relativi servizi;
- accedere a strutture fisiche specifiche.

Una seconda definizione descrive il **controllo degli accessi** come il *processo* mediante il quale l'uso delle risorse di sistema è regolato secondo una politica di sicurezza ed è consentito solo da entità autorizzate (utenti, programmi, processi o altri sistemi) secondo tale politica.

(In senso lato, la sicurezza informatica si basa sul controllo degli accessi. Essa infatti è definita come le misure che assicurano servizi di sicurezza in un sistema informatico, in particolare quelli che assicurano il servizio di controllo degli accessi.)

I ***requisiti*** base di sicurezza nel controllo degli accessi sono:

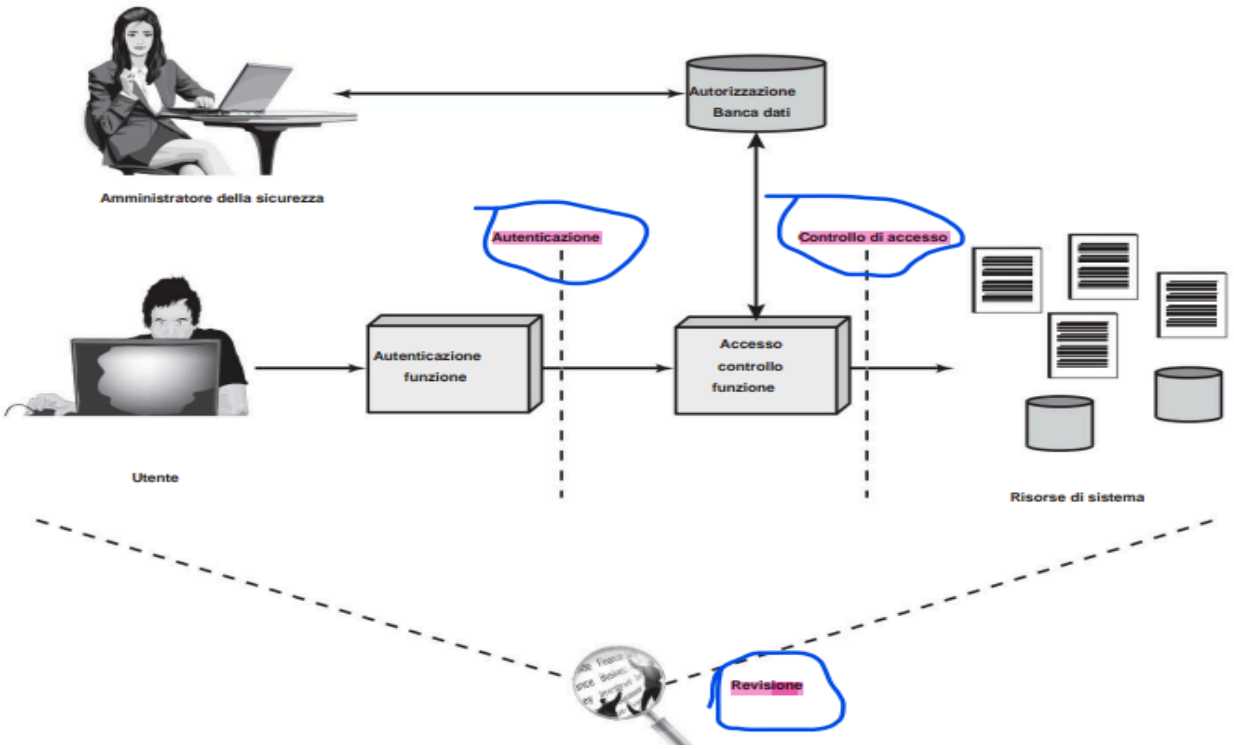
1. Limitare l'accesso al sistema a:
    a. i soli utenti autorizzati;
    b. i processi che operano per conto di utenti autorizzati;
    c. i dispositivi autorizzati.
2. Limitare l'accesso al sistema ai tipi di transizioni e funzioni che gli utenti autorizzati sono autorizzati ad eseguire.

In poche parole, gli ***obiettivi*** principali della sicurezza informatica sono impedire agli utenti non autorizzati di accedere alle risorse, impedire agli utenti legittimi di accedere alle risorse in modo non autorizzato e consentire agli utenti legittimi di accedere alle risorse in modo autorizzato.

Questo contesto coinvolge:

- **Autenticazione:** verifica delle credenziali dell'utente;
- **Autorizzazione:** concessione di un diritto o di un'autorizzazione ad un'entità di sistema per accedere a una risorsa di sistema. Rappresenta quindi la funzione con la quale si definiscono quali entità sono considerate attendibile per quali scopi;
- **Audit:** revisione ed esame delle registrazioni e delle attività del sistema al fine di verificare l'adeguatezza dei controlli di sistema, in modo da rilevare violazioni della sicurezza e raccomandare eventuali modifiche.

In poche parole l'autenticazione verifica se un utente è autorizzato ad accedere al sistema, mentre il controllo degli accessi determina se l'accesso specifico ad una risorsa da parte dell'utente è legale.



Una **politica di controllo** degli accessi stabilisce quali tipi di accesso sono consentiti. Le policy di controllo degli accessi possono essere suddivise nelle seguenti categorie:

- **Controllo discrezionale dell'accesso (DAC)**;
- **Controllo di accesso obbligatori (MAC)**: controlla l'accesso in base al confronto delle etichette di sicurezza (che indicano la sensibilità delle risorse) con le autorizzazioni di sicurezza (che indicano la tipologia di risorse a cui possono accedere gli utenti). Questa politica è definita *obbligatoria* perché, a differenza della precedente, perché un'entità non ha potere di concedere a sua volta permessi ad altre entità su alcune risorse
- **Controllo degli accessi basati sui ruoli (RBAC)**;
- **Controllo dell'accesso basato sugli attributi (ABAC)**.

All'interno di un sistema non è detto che non possano essere usate più di una di queste politiche.

Gli elementi base del controllo degli accessi sono:

- Il **soggetto** è un'entità capace di accedere ad una risorsa. Generalmente il soggetto è associato ad un processo, il quale assume i diritti dell'utente che lo ha invocato. Il soggetto è ovviamente ritenuto responsabile delle azioni che ha avviato. Si hanno tre classi di soggetti:
  - ***Proprietario***: potrebbe essere il creatore della risorsa;
  - ***Gruppo***: un utente può utilizzare i privilegi assegnati ad un gruppo al quale appartiene. Il soggetto può appartenere anche a più di un gruppo;

- ***Mondo***: tutti i soggetti hanno un certo set di privilegi, il quale deve essere ovviamente quanto più piccolo possibile;

- Un **oggetto** è una risorsa il cui accesso è controllato. Il numero e il tipo di oggetti da tenere in considerazione in un sistema di controllo degli accessi può dipendere dall'ambiente in cui opera il sistema;

- Un **diritto di accesso** descrive il modo in cui un soggetto può accedere ad un oggetto. I diritti di accesso possono includere:
  - ***Lettura*** : l'utente può visualizzare le informazioni in una risorsa di sistema (*questo diritto comprende anche la copia e la stampa dell'oggetto*);
  - ***Scrittura***: l'utente può modificare la risorsa, ma non eliminarla (questo diritto comprende anche il diritto di lettura);
  - ***Esecuzione***: l'utente può eseguire determinati programmi;
  - ***Eliminazione***: l'utente può eliminare una determinata risorsa;
  - ***Creazione***: l'utente può creare nuovi file;
  - ***Ricerca***: l'utente può elencare i file di una directory o effettuare una ricerca nella directory.

Controllo discrezionale degli accessi (DAC)

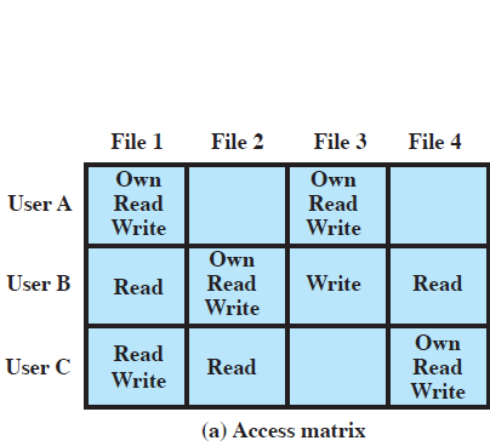
Uno schema di ***controllo discrezionale degli accessi*** controlla l'accesso in base all'identità del richiedente e alle autorizzazioni. Questo tipo di controllo è detto *discrezionale* perché un'entità potrebbe avere diritti tali per cui potrebbe a sua volta concedere permessi ad altre entità su alcune risorse.

Il DAC viene implementato attraverso una matrice (***Access Matrix***) la quale ha dimensione NxM, dove N è il numero di soggetti di cui si vuole definire i privilegi e M è il numero di oggetti a cui è possibile accedere. In ogni cella della matrice degli accessi quindi si avrà il diritto di accesso per un determinato soggetto ad un determinato oggetto.

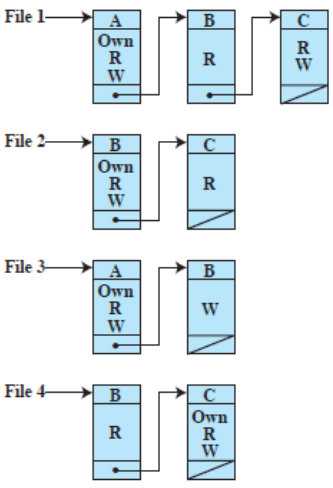
Tale matrice è generalmente sparsa e quindi scomposta in due modi:

- ***Access Control Lists*** (**ACL**): viene prodotta una lista per ogni oggetto nel quale sono presenti i diritti per ogni soggetto (*decomposizione per colonne*).
  - *Vantaggi*:
    - vantaggiosa per determinare tutti i permessi su una risorsa;
    - può contenere una voce di default per garantire permessi di accesso agli utenti non elencati (*least privilege*).
  - Svantaggi:
    - svantaggiosa per determinare tutti i permessi associati ad un soggetto;

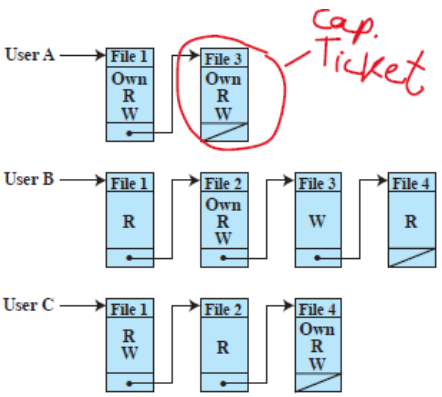
- ***Capability Lists*** (**CL**): viene prodotta una lista per ogni soggetto nel quale sono presenti i suoi diritti per ogni oggetto (*decomposizione per righe*). In questo tipo di scomposizione si produce un ***ticket di capacità***, il quale specifica gli oggetti e le operazioni autorizzate per un particolare utente. Ogni utente dispone di un certo numero di ticket e potrebbe essere autorizzato a prestarli o cederli ad altri soggetti.
  - *Vantaggi*:
    - vantaggiosa per determinare tutti i permessi associati ad un soggetto;
    - potrebbe essere autorizzato a prestare o cedere i ticket ad altri soggetti.
  - Svantaggi:
    - svantaggiosa per determinare tutti i permessi su una risorsa;
    - la possibile disseminazione di ticket di capacità potrebbe essere un problema.



(a) Access matrix



(b) Access control lists for files of part (a)



(c) Capability lists for files of part (a)

Per risolvere i problemi di efficienza della matrice degli accessi, si è provveduto ad introdurre una struttura dati non sparsa, la **tabella di autorizzazione**. Questa struttura dati presenta una riga per ogni diritto di accesso di un soggetto ad un oggetto.

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Ordinare tale tabella per soggetto equivale ad avere una CL, mentre ordinarla per oggetti produrrebbe una ACL.

Un modello per il controllo degli accessi discrezionale è quello proposto da Lampson, Graham e Denning. Esso presuppone un insieme di soggetti, oggetti e diritti di accesso.

---

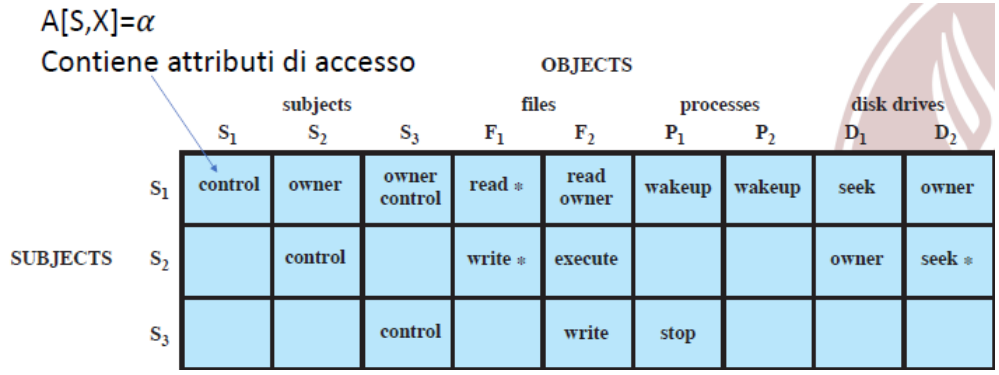
*Lo **stato di protezione di un sistema** è l'insieme di informazioni, in un dato momento, che specificano i diritti di accesso di ogni soggetto relativamente ad ogni oggetto.*

Il modello presenta tre requisiti:

1. Rappresentare dello stato di protezione del sistema;
2. Far rispettare i permessi di accesso;
3. Permettere a soggetti di modificare lo stato di protezione secondo determinate modalità.

Per il primo punto viene utilizzata una ***matrice di controllo estesa***, nella quale i diritti di accesso includono:

- **Processi**: possibilità di eliminare, arrestare e riattivare un processo;
- **Dispositivi**: possibilità di leggere/scrivere il dispositivo, controllarne il funzionamento, bloccarlo e sbloccarlo;
- **Locazione o regione di memoria**: possibilità di leggere/scrivere determinate regioni di memoria la cui impostazione di default vieti l'accesso;
- **Soggetti**: possibilità di concedere/cancellare permessi di accesso relativi ad altri utenti.



Per il secondo punto invece il modello utilizza un controller degli accessi per ogni risorsa. I tentativi di accesso ad un oggetto quindi seguono il seguente iter:

- Il soggetto S_i emette una richiesta di tipo α per l'oggetto X;
- Il sistema genera un messaggio del formato (S_i , α, X) per il controller di X;
- Il controller interroga la matrice di accesso per determinare se α è in A[S_i , X]. In tal caso l'accesso è consentito, altrimenti si verifica una violazione della protezione che dovrebbe innescare un avvertimento e un'azione conseguente.

Per il terzo punto il modello permette ad alcuni soggetti di modificare la matrice degli accessi, trattando tale operazione come un qualsiasi tentativo di accesso (in questo caso l'oggetto è rappresentato dalle varie celle della matrice). Il modello inoltre prevede un insieme di regole che disciplinano la modifica della matrice, come mostrato di seguito (sia A la matrice degli accessi):

- **R1** [***trasferimento di permesso di accesso***]:
  - *Autorizzazione necessaria*: il permesso (con asterisco, che indica che il permesso può essere trasferito) in A[S_0 , X];
  - *Conseguenza*: salva il permesso in A[S, X];

- **R2** [***concessione di permesso di accesso***]:
    - *Autorizzazione necessaria*: *'owner'* in A[S_0 , X];
    - *Conseguenza*: salva il permesso in A[S, X];
- **R3** [***eliminazione di permesso di accesso***]:
    - *Autorizzazione necessaria*: *'owner'* in A[S_0 , X] e *'control'* in A[S_0 , S];
    - *Conseguenza*: cancella il permesso in A[S, X];
- **R4** [***lettura di permesso di accesso***]:
    - *Autorizzazione necessaria*: *'owner'* in A[S_0 , X] e *'control'* in A[S_0 , S];
    - *Conseguenza*: legge il permesso in A[S, X];
- **R5** [***creazione di oggetto***]:
    - *Autorizzazione necessaria*: //;
    - *Conseguenza*: aggiunge *'owner'* in A[S_0 , X];
- **R6** [***distruzione di oggetto***]:
    - *Autorizzazione necessaria*: *'owner'* in A[S_0 , X];
    - *Conseguenza*: elimina la colonna X da A;
- **R7** [***creazione di soggetto***]:
    - *Autorizzazione necessaria*: //;
    - *Conseguenza*: aggiunge aggiunge la riga S, esegue la creazione dell'oggetto S e memorizza *'control'* in A[S , S];
- **R8** [***distruzione di soggetto***]:
    - *Autorizzazione necessaria*: *'owner'* in A[S_0 , X];
    - *Conseguenza*: elimina la colonna X da A;

Un **dominio di protezione** è un insieme di oggetti con i relativi permessi di accesso. Ad esempio nella matrice appena descritta ogni riga (cioè ogni utente) rappresenta un dominio di protezione. Volendo utilizzare un approccio più generale si potrebbe permettere all'utente di generare processi che non usino tutto il dominio di protezione dell'utente, ma che ne usino solo un sottoinsieme. In questo modo si limitano le capacità del processo ai fini di sicurezza. L'associazione tra un processo e un dominio di protezione può essere statica o dinamica. Una forma di dominio di protezione ha a che fare con la distinzione in molti OS, come UNIX, di ***modalità kernel*** e ***modalità utente***. Nella prima infatti è possibile eseguire istruzioni privilegiate e accedere ad aree protette della memoria, cosa infattibile nella seconda modalità.

Controllo degli accessi a file UNIX

Tutti i tipi di ***file*** in UNIX vengono gestiti dal sistema operativo tramite ***inode*** **(index nodes)**, cioè una struttura di controllo che contiene le informazioni chiave (come attributi del file, permessi e altre informazioni di controllo) necessarie per gestire quel file. Diversi nomi di file possono essere associati ad un singolo inode, ma un inode attivo è associato ad uno e un solo file. Sul disco è presente una tabella (o una lista) di inode, che contiene gli inode di tutti i file del file system.

Le ***directories*** sono strutturate in un albero gerarchico e ognuna di essa può contenere file (foglie) o altre directories (nodi). Una directory quindi non è altro che un file che contiene un elenco di nomi di file con puntatori per gli inode associati. Perciò anche ogni directory è associata ad un inode.

La maggior parte dei sistemi operativi UNIX sono basati sullo schema di controllo dell'accesso per il quale ad ogni utente viene assegnato un numero di identificazione univoco (***User ID***). Un utente è anche membro di uno o più gruppi (di cui uno primario), ciascuno identificato con un ***Group ID***.

Quando un file viene creato, esso viene designato come di proprietà di un utente dal quale prende l'User ID e il Group ID (che inizialmente è l'ID del gruppo principale del creatore oppure è quello della directory a cui appartiene, nel caso in cui quest'ultima abbia il SetGID impostato). Al file viene anche associato un ***insieme di 12 bit di protezione***, di questi:

- 3 bit specificano permessi di lettura, scrittura e esecuzione per il proprietario;
- 3 bit specificano permessi di lettura, scrittura e esecuzione per il gruppo del proprietario;
- 3 bit specificano permessi di lettura, scrittura e esecuzione per il resto degli utenti;
- 2 bit per SetUID e SetGID rappresentano un permesso speciale che modifica il comportamento del sistema nei confronti dell'utente che utilizza il file;
- 1 bit per StickyBit (applicabile solo a directory) specifica che solo il proprietario può rinominare, muovere o cancellare i file nella directory.

Queste informazioni sono presenti nell'inode del file.

Molti sistemi operativi moderni basati su UNIX supportano ACL, come FreeBSD e Linux. FreeBSD consente all'amministratore di assegnare ad un file, tramite comando ***setfacl***, un numero qualsiasi di utenti e gruppi (tramite i loro ID), ciascuno dei quali con 3 bit di protezione (r, w, e). Non è necessario che un file disponga di un ACL, esso potrebbe essere protetto dal tradizionale meccanismo di accesso di UNIX. I file però dispongono di un ulteriore bit di protezione che indica se il file ha un ACL esteso.

Quando un processo richiede accesso ad un oggetto si procede nel seguente modo:

1. Si seleziona la voce nell'ACL più appropriata, controllando nell'ordine quelle del proprietario, dell'utente designato e del gruppo ;
2. Si controlla che i permessi presenti in quel ACL siano sufficienti all'operazione richiesta.

Controllo degli accessi basato sui ruoli (RBAC)

Uno schema di ***controllo degli accessi basato sui ruoli*** controlla l'accesso in base ai ruoli che l'utente assume in un sistema. I modelli RBAC quindi definiscono un **ruolo** come una funzione all'interno di un'organizzazione e assegnano i diritti di accesso ad essi, invece che ai singoli utenti. Le relazioni tra utenti e ruoli e tra ruoli e risorse sono entrambe relazioni molti-a-molti.

L'assegnazione dei ruoli agli utenti è generalmente statica, cioè viene modificata raramente, così come l'assegnazione di diritti di accesso relativi ai ruoli. Ma ciò potrebbe trasformarsi anche in qualcosa di dinamico in alcuni ambienti.

RBAC si presta ad un'effettiva implementazione del principio del least privilege, dato che ciascun ruolo deve contenere un insieme di permessi tali per cui gli utenti possano svolgere il proprio compito e solo quello.

Un insieme di modelli per il controllo degli accessi basato sui ruoli prevede i modelli $RBAC_0$, $RBAC_1$, $RBAC_2$, $RBAC_3$.

Modello base ($RBAC_0$) presuppone quattro tipi di entità:

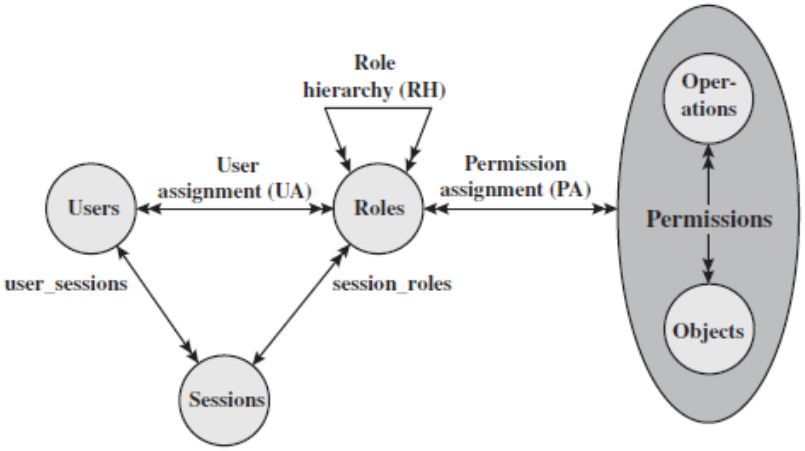
- **Utente**: individuo che ha accesso al sistema. Ogni utente ha un ID associato;
- **Ruolo**: funzione all'interno dell'organizzazione che controlla il sistema. In genere al ruolo si associa una descrizione delle responsabilità conferite a chi ne prende parte;
- **Autorizzazione**: il diritto di accesso;
- **Sessione**: mappatura tra un utente e un sottoinsieme attivato dell'insieme dei ruoli a cui appartiene l'utente.

Modello di gerarchie di ruoli ($RBAC_1$) fornisce il mezzo per rappresentare la **struttura gerarchica** dei ruoli nell'organizzazione. Viene inserito quindi anche il concetto di ereditarietà dei permessi lungo i rami della gerarchia, quindi ruoli più alti hanno anche i permessi dei ruoli sottostanti.

Modello con vincoli ($RBAC_2$) fornisce un mezzo per adattare RBAC alle specificità amministrative e le politiche di sicurezza di un'organizzazione. Un **vincolo** può essere rappresentato da una relazione tra ruoli o da una condizione di un ruolo specifico. Esempi di vincoli sono:

- ***Ruoli mutualmente esclusivi***: un utente può essere assegnato solo ad uno dei ruoli vincolati;
- ***Cardinalità***: un ruolo può contenere massimo un tot di utenti;
- ***Prerequisiti di ruolo***: un utente può avere un ruolo solo se ha anche un dato altro ruolo.

Modello completo ($RBAC_3$) coniuga i vantaggi del modello di gerarchia dei ruoli e del modello con vincoli.



Controllo degli accessi basato sugli attributi (ABAC)

Uno schema di ***controllo degli accessi basato sugli attributi*** controlla l'accesso in base agli attributi del soggetto, dell'oggetto o dell'ambiente. I modelli ABAC quindi definiscono un **attributo** come una caratteristica specifica che viene analizzata. In particolare:

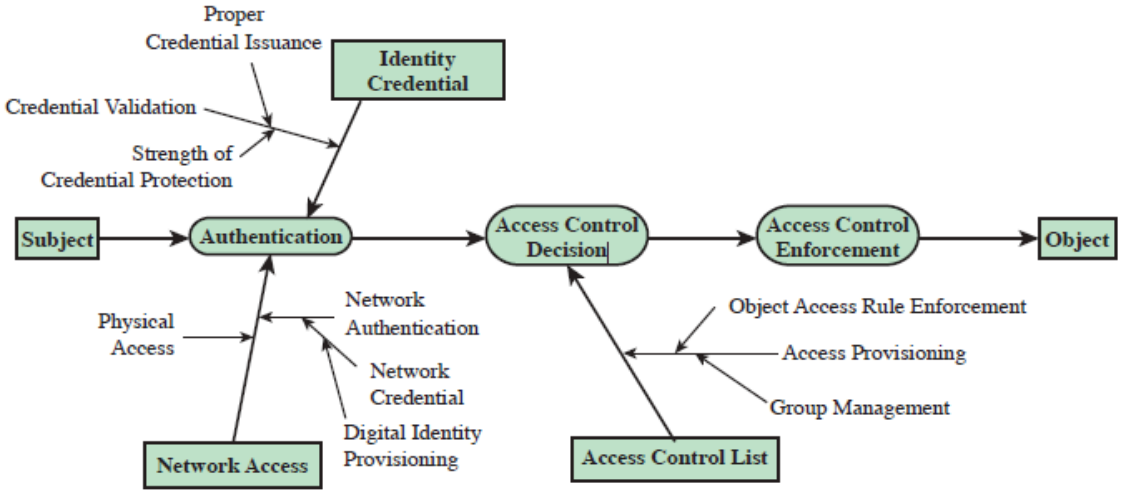
- ***Attributi del soggetto***: il soggetto è un'entità attiva che potrebbe alterare lo stato del sistema. Questo tipo di attributi definiscono l'identità e le caratteristiche del soggetto (es. nome, età, ecc…)
- ***Attributi dell'oggetto***: l'oggetto è un'entità passiva, il quale contiene o riceve informazioni. Questo tipo di attributi può essere ottenuto dai metadati della risorsa (es. titolo, dimensioni, ecc…)
- ***Attributi ambientali***: descrivono il contesto in cui avviene l'accesso al sistema (es. data, ora, ecc…)

L'accesso di un soggetto ad un oggetto, in un sistema con controllo ABAC, procede così:

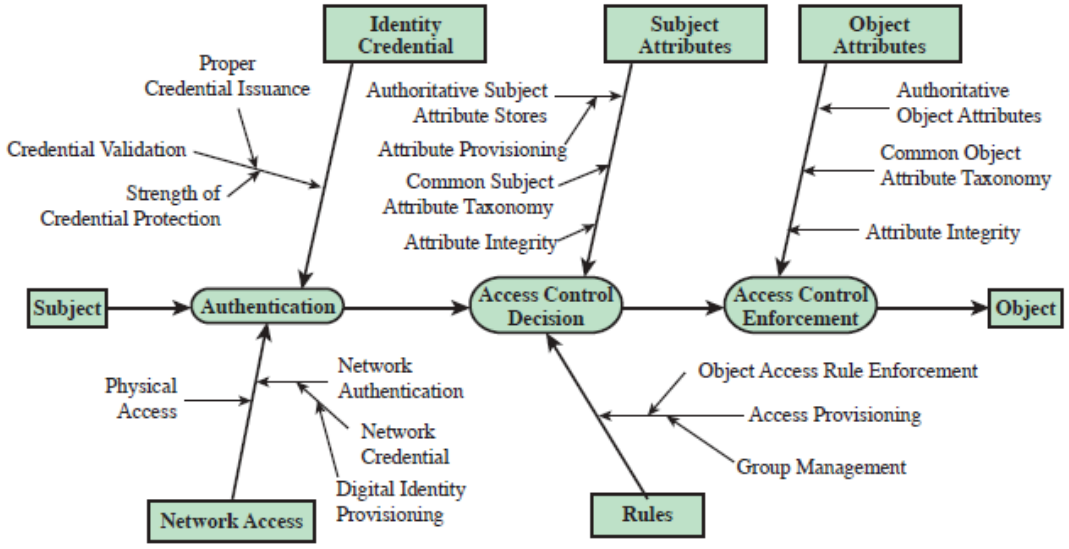
1. Il soggetto richiede l'accesso all'oggetto;

2. Il meccanismo di controllo degli accessi è disciplinato da un insieme di regole definite da una politica. Attraverso queste regole il sistema di controllo valuta gli attributi del soggetto, dell'oggetto e dell'ambiente per determinare l'autorizzazione;
3. Il meccanismo di controllo dell'accesso garantisce l'accesso del soggetto all'oggetto in caso se esso risulta autorizzato.

Come detto, una **politica** è un insieme di regole che governano il comportamento consentito all'interno di un sistema, in base ai privilegi dei soggetti e al modo in cui gli oggetti devono essere protetti. A loro volta i **privilegi** rappresentano il comportamento autorizzato di un soggetto. La politica è tipicamente scritta dal punto di vista dell'oggetto e dei privilegi ottenuti dal soggetto.



(a) ACL Trust Chain



(b) ABAC Trust Chain

Un esempio di politica ABAC è il seguente:

Classe del film	Età necessaria
R	≥17
PG-13	≥13
G	Tutti

La regola da seguire per controllare l'accesso quindi sarebbe la seguente:

R: authorized(s, o, e) □ (Classe(o)=R ∧ Età(s)≥17)∨(Classe(o)=PG-13 ∧ Età(s)≥13)∨(Classe(o)=G)

Gestione di identità, credenziali e accessi (ICAM)

ICAM è un approccio completo alla gestione e all'implementazione delle identità digitali, delle credenziali e del controllo degli accessi. Esso è progettato per:

- Creare rappresentazioni attendibili dell'identità digitale di individui e di entità non personali (NPE);
- Associare tali identità a credenziali che possano essere utilizzate dagli individui o dalle NPE;
- Utilizzare le credenziali per fornire l'accesso autorizzato alle risorse di un sistema.

La **gestione dell'identità** riguarda l'assegnazione di attributi a un'identità digitale e il collegamento di tale identità ad un individuo o ad una NPE. L'obiettivo è quello di stabilire un'identità digitale affidabile che sia indipendente dall'applicazione e dal contesto specifico. L'approccio più comune consiste nel creare una rappresentazione digitale per l'uso specifico dell'applicazione; di conseguenza il mantenimento e la protezione dell'identità vengono considerati secondari. Un'identità virtuale è spesso composta da un insieme di attributi che identificano in modo univoco un utente all'interno di un sistema. Un ultimo elemento per la gestione dell'identità è la gestione del ciclo di vita, il quale include:

- Meccanismi per la protezione delle informazioni dell'identità personale;
- Controllo degli accessi a tali informazioni;

- Tecniche per la condivisione di questi dati con le applicazioni che ne necessitano;
- Revoca dell'identità.

La **gestione delle credenziali** è la gestione del ciclo di vita della credenziale, la quale consiste nei seguenti passi:

- Un individuo autorizzato sponsorizza un individuo o un'entità per richiedere una credenziale;
- L'individuo sponsorizzato si iscrive per ottenere la credenziale. In questo processo si verifica l'identità del richiedente e si acquisiscono suoi dati biografici o biometrici;
- Viene prodotta una credenziale che viene rilasciata al richiedente;
- La credenziale deve essere mantenuta per tutto il suo ciclo di vita, che potrebbe includere revoca, sostituzione, sospensione e reintegro.

La **gestione degli accessi** è la gestione delle modalità con cui alle entità viene concesso l'accesso alle risorse. Lo scopo è garantire che venga effettuata la corretta verifica dell'identità quando un soggetto tenta di accedere ad un oggetto. Sono necessari tre elementi di supporto:

- ***Gestione delle risorse***: riguarda la definizione di regole per una risorsa che richiede un controllo degli accessi. Le regole includono requisiti delle credenziali e degli attributi di soggetto, oggetto e ambiente che determinano l'accesso alla risorsa;
- ***Gestione dei privilegi***: riguarda la definizione e il mantenimento di privilegi, basati sulle caratteristiche del soggetto. I privilegi sono considerati attributi collegabili a un'identità digitale;
- ***Gestione delle policy***: riguarda la regolazione di cosa è consentito e cosa no in una transazione di accesso. Le policy, in base all'identità e gli attributi del richiedente specifica quali azioni sono possibili su una certa richiesta.

Con il termine **Federazione delle identità** si descrive la tecnologia, gli standard, le politiche e i processi che consentono ad un'organizzazione di fidarsi delle identità digitali, degli attributi e delle credenziali emesse da un'altra organizzazione. L'obiettivo infatti è duplice:

- Potersi fidare delle identità di individui provenienti da altre organizzazioni e che vogliono accedere alla nostra organizzazione;
- Garantire l'identità delle persone della nostra organizzazione quando hanno bisogno di accedere ad altre organizzazioni.

L’approccio tradizionale prevede accordi tra utenti e fornitori di servizi di identità per l’ottenimento dell’identità e delle credenziali. Gli utenti poi stipulano anche contratti con fornitori di servizi di applicazione, le quali si fidano delle informazioni generate dal fornitore di servizi di identità.

CAPITOLO 5 - SICUREZZA DEI DATABASE

Sono vari i motivi per cui un’organizzazione abbia interesse nel proteggere i propri database:

- Esiste un drammatico squilibrio tra la complessità alta dei DBMS e le tecniche di sicurezza per proteggere questi sistemi critici;
- I database dispongono di un sofisticato e complesso protocollo di interazione (SQL). Una sicurezza efficace richiede una piena consapevolezza delle vulnerabilità di SQL;
- Le organizzazioni tipiche sono manchevoli di personale specializzato per la sicurezza dei database;
- La maggior parte delle aziende utilizzano una miscela eterogenea fatta di diverse piattaforme di database, diverse piattaforme aziendali e diverse piattaforme di sistemi operativi. Ciò aumenta la complessità della sicurezza;
- La crescente dipendenza dalle tecnologie cloud per ospitare tutto o parte di un database aziendale.

Attacco SQL injection

L’**attacco SQL injection** (**SQLi**) è una delle minacce di rete più diffuse e pericolose. Esso è pensato per sfruttare la natura delle applicazioni web, mandando comandi SQL malevoli. Gli obiettivi principali di questo attacco sono:

- Estrarre masse di dati;
- Modificare o eliminare dati;
- Eseguire comandi arbitrari del sistema operativo;
- Lanciare attacchi DoS (Denial-of-Service).

Un attacco del genere si sviluppa nel seguente modo:

1. L’hacker trova una vulnerabilità nell’applicazione web e inserisce un comando SQL che non viene bloccato dal firewall, in quanto è apparentemente non malevolo;
2. Il server web riceve il comando e lo inoltra al server dell’applicazione, il quale a sua volta interroga il server del database;
3. Il server del database esegue il codice sul database e invia la risposta all’utente seguendo gli stessi passaggi a ritroso.

Vie di attacco SQLi

- ***Input dell'utente***: gli attaccanti inseriscono comandi SQL come input da utente;
- ***Variabili server***: gli attaccanti prendono di mira il salvataggio di variabili server (variabili che contengono intestazioni HTTP, di protocollo di rete e variabili ambientali usate dal server) all'interno di database non adeguatamente protetti;
- ***Iniezione di secondo ordine***: gli attaccanti sfruttano l'incompletezza della prevenzione contro attacchi SQLi. In questo caso l'input che modifica la query non è da utente ma viene generato dal sistema stesso;
- ***Cookie***: gli attaccanti alterano i cookie per sfruttare query SQL gestite dal sistema e basate sul contenuto dei cookie stessi;
- ***Input fisico dell'utente***: gli attaccanti potrebbero utilizzare input fisici (come codici a barre particolari che vengono letti) per mandare richieste malevole al database.

Tipi di attacco SQLi

- ***Attacco in banda***: utilizza lo stesso canale di comunicazione sia per inserire il codice SQL che per recuperare i risultati. Le metodologie utilizzate sono:
  - ***Tautologia:*** modifica istruzioni condizionali in modo da renderle sempre vere;
  - ***Commento di fine riga***: termina la stringa di input con il segno di commento '--', in modo tale che eventuale codice successivo non venga eseguito;
  - ***Query piggyback***: aggiunge nuovi comandi SQL tramite la stessa stringa di input.

Esempi di attacchi in banda sono i seguenti:

1. La query prevista per ottenere il nome degli studenti di una università è la seguente:

```
SELECT nome FROM studenti WHERE università = '?' [...]
```

L'hacker inserisce come input il seguente:

```
Sapienza'; DROP table studenti; - -
```

La query risultante è la seguente e ha come effetto la distruzione della tabella degli studenti:

```
SELECT nome FROM studenti WHERE università = 'Sapienza';
        DROP table studenti; - -' [...]
```

2. La query prevista per ottenere le informazioni dello studente loggato:

```
SELECT info FROM studenti WHERE nome = '?' AND pwd=$_GET["pwd"];
```

L'hacker inserisce come input il seguente:

```
xx' OR 2=2 - -
```

La query risultante è la seguente e ha come effetto la distruzione della tabella degli studenti:

```
SELECT info FROM studenti WHERE nome = 'xx' OR 2=2 - -
' AND pwd=$_GET["pwd"];
```

- ***Attacco inferenziale***: non avviene un vero e proprio trasferimento di dati, ma l'attaccante riesce a ricostruire informazioni di sistema inviando particolari richieste e osservando il relativo comportamento del server. Questo tipo di attacchi includono:
  - ***Query illegali o logicamente errate***: l'attaccante invia delle query illegali o logicamente errate che producono un errore. Nel caso in cui la pagina di errore restituita sia troppo descrittiva, l'attaccante ha la possibilità di carpire informazioni riguardanti il sistema;
  - ***Blind SQLi***: l'attaccante pone al server delle domande vero/falso. Il sistema, anche se non rende informazioni all'aggressore, si comporta in un modo diverso nel caso in cui la query abbia dato risultato positivo o negativo.
- ***Attacco fuori banda***: i dati vengono ottenuti usando un canale diverso rispetto a quello dove è passata la query SQL (ad esempio email). Questo attacco può essere usato quando ci sono limitazioni nell'ottenimento di dati, ma la connettività in uscita dal database è scarsa.

Contromisure

Possono essere classificate in tre tipi. Ovviamente, data la pericolosità degli attacchi SQLi, c'è bisogno di adottare più di una contromisura:

- ***Coding difensivo***:
  - Convalida degli input;
  - Sviluppo di query con parametri (per limitare gli input da utente);
  - SQL DOM e successori (insieme di classi che consente la convalida dei dati;
- ***Detection***:
  - Basata sulla firma (cerca di confrontare la query con modelli di attacchi specifici);
  - Basata sulle anomalie (cerca di rilevare comportamenti anormali del sistema);
  - Anlisi del codice (analizza il codice con suite specifiche per limitare problemi);
- ***Prevenzione a run-time***:
  - Controlla le query in fase di esecuzione per verificare se sono conformi ad un modello di query previste.

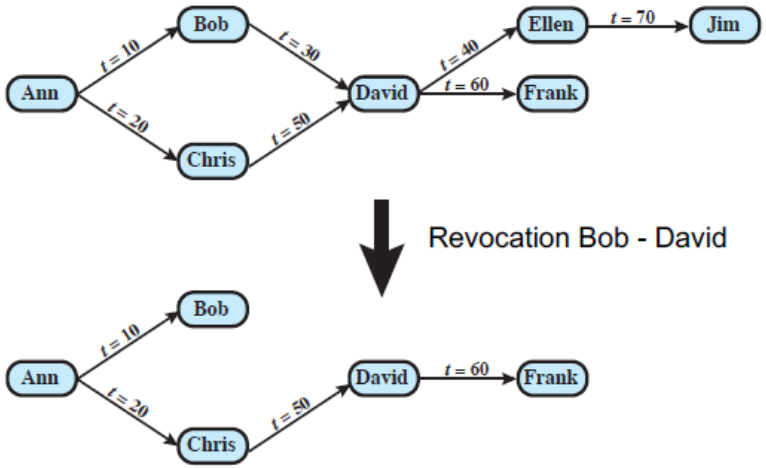
Controllo degli accessi al database

I **sistemi di controllo degli accessi al database** determinano se un utente ha accesso a parte o alla totalità del database e che privilegi abbia rispetto alle operazioni sulle tabelle. Tipicamente un DBMS può supportare una serie di politiche amministrative, come:

- ***Amministrazione centralizzata***: un numero limitato di utenti può concedere e revocare diritti di accesso agli altri utenti;
- ***Amministrazione basata sulla proprietà***: il proprietario di una tabella può concedere può concedere e revocare diritti di accesso agli altri utenti;
- ***Amministrazione decentralizzata***: funziona come l'amministrazione basata sulla proprietà, ma in questo caso il proprietario della tabella può dare diritto di concedere e revocare diritti sulla tabella anche ad altri utenti.

SQL fornisce due comandi per la gestione dei diritti di accesso **GRANT** (usato per dare uno o più permessi o per assegnare un ruolo ad un utente) e **REVOKE** (usato per revocare permessi). I permessi tipici sono *SELECT*, *INSERT*, *UPDATE*, *DELETE* e *REFERENCES*.

Il comando GRANT abilita un diritto di accesso a cascata attraversi gli utenti. Facendo l'esempio nell'immagine seguente, se un utente A garantisce un privilegio ad un utente B, allora B può garantire privilegi ad altri utenti. Se poi A revoca il privilegio a B, allora vengono revocati tutti i permessi garantiti da B utilizzando quello appena revocato.



Data la natura del comando GRANT, uno schema di controllo degli accessi basato sui ruoli sembra essere la soluzione migliore. In un tale ambiente di controllo, si possono classificare gli utenti del database in tre grandi categorie:

- ***Proprietario dell'applicazione***: end-user che è proprietario di oggetti del database;
- ***End-user***: Tutti gli altri utenti finali che operano sugli oggetti del database senza possederli;
- ***Amministratore***: utente che ha la responsabilità amministrativa di parte o tutto il db.

Un sistema RBAC deve inoltre garantire le seguenti funzionalità:

- Creare ed eliminare ruoli;
- Definire permessi per un ruolo
- Assegnare o annullare l'assegnazione di utenti a ruoli.

Un esempio dell'uso dei ruoli è l'RBAC in Microsoft SQL Server. Qui ci sono tre tipi di ruoli:

- ***Ruoli server fissi***: definiti a livello di server per facilitare il compito di amministrazione. Gli amministratori del database possono utilizzare questi ruoli per assegnare diversi compiti amministrativi al personale, concedendo loro solo i privilegi necessari;
- ***Ruoli fissi del database***: definiti a livello del singolo database. Alcuni di questi ruoli sono usati, come i precedenti, per affidare compiti amministrati, altri invece sono usati per concedere autorizzazioni ad utenti finali;
- ***Ruoli definiti dall'utente***: definiti dagli utenti stessi per assegnare privilegi rispetto a parti di db.

Inferenza

L'**inferenza** è il processo di esecuzione di query autorizzate al fine di dedurre informazioni non autorizzate dalle risposte legittime ricevute. Il problema dell'inferenza sorge quando la combinazione di dati risulta più sensibile del singolo dato oppure quando una combinazione di dati può essere utilizzata per dedurre altri dati di maggiore sensibilità.

Il percorso di trasferimento delle informazioni attraverso il quale vengono ottenuti dati non autorizzati viene detto **canale di inferenza**.

Esistono due tipi di approcci per rilevare una minaccia del genere:

- Durante la ***progettazione del database***: questo approccio rimuove i canali di inferenza alterando la struttura del db o cambiando le policy di accesso. In questo modo però spesso si vanno a porre limiti di accesso troppo rigidi;
- Durante le ***esecuzioni di query***: questo approccio cerca di rilevare un canale di inferenza durante una o più query. Se viene rilevata un'anomalia la query viene negata o modificata.

Per entrambi questi due approcci è necessario un algoritmo di rilevamento dell'inferenza.

Crittografia del database

Data la sensibilità dei database, essi devono essere protetti con più livelli di sicurezza, tra cui firewall, autenticazione, controllo degli accessi, ecc…. L'ultima spiaggia per la difesa dei db sembra essere quindi la **crittografia** degli stessi. Ci sono però due svantaggi:

- ***Gestione delle chiavi***: gli utenti autorizzati devono avere accesso alla chiave di decrittografia per i dati a cui hanno accesso. Fornire chiavi sicure a parti selezionati degli utenti che possono accedere a tale struttura dati, però, è un compito complesso;
- ***Inflessibilità***: quando parte o tutto il database è criptato, diventa più difficile eseguire una ricerca all'interno di esso.

La crittografia può essere applicata all'intero db, a livello di record, di attributi o del singolo campo.

Un caso di uso della crittografia potrebbe essere quello presente nelle piccole e medie imprese, le quali cercano di esternalizzare i database presso dei fornitori di servizi, in modo tale da limitare costi di manutenzione e di sistema. Ciò provoca però un problema a livello di riservatezza, in quanto ci sarebbe possibilità di lettura dei dati da parte dal fornitore di servizi. Una soluzione sarebbe quella di criptare i dati e non dare la chiave di decriptazione al fornitore di servizi. La flessibilità in questo caso però non sembra essere altissima, dato che per eseguire una query bisognerebbe scaricare intere tabelle e decriptarle. In questo tipo di approccio vi sono quattro entità:

- ***Proprietario dei dati***: l'organizzazione che produce i dati da rendere disponibili;
- ***Utente***: entità umana che presenta richieste (query) al sistema;
- ***Client***: sistema che trasforma le query degli utenti in query da eseguire sui dati crittografati;
- ***Server***: organizzazione che riceve i dati crittografati da un proprietario dei dati e li rende disponibili per la distribuzione ai client. Il server potrebbe essere di proprietà dello stesso proprietario dei dati, oppure essere di un fornitore esterno.

Un tipo di approccio di query potrebbe essere quindi il seguente:

- L'utente invia una query SQL con un valore specifico della chiave primaria;
- Il ***query processor*** sul client cripta la chiave primaria, modifica di conseguenza la query e la invia al server;
- Il server elabora la query e restituisce i record appropriati;
- Il query processor sul client decripta i record e li restituisce all'utente.

Un problema però potrebbe sorgere per query che non ricercano record in base a valori specifici (ad es. WHERE stipendio < 7000 non andrebbe bene con l'approccio precedente). Una possibile soluzione per aumentare la flessibilità sarebbe quella di criptare i dati partizionandoli e tradurre le query in base alla partizione conseguente. Ad esempio:

eid	ename	salary	addr	did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	River	50
875	Jerry	55K	Hopewell	92

E(*k*, *B*)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100110011001011…	1	10	3	7	4
0111000111001010…	5	7	2	7	8
1100010010001101…	2	5	1	9	5
0011010011111101…	5	5	2	4	9

In questo modo la query per cercare i dipendenti con stipendio sotto i €7000 si traduce con WHERE I(salary) <= 3.

Esistono due opzioni per la cifratura dei dati:

- Crittografia a chiave simmetrica: efficiente nell'eseguire la cifratura e la ricerca dei dati, ma le chiavi devono essere trasmesse su canali sicuri e la ricerca è limitata alla condizione di un attributo singolo;
- Crittografia a chiave pubblica: rende una maggiore flessibilità delle query, ma è inefficiente a causa dei calcoli complessi della cifratura a chiave pubblica.

Sicurezza dei cloud data services

Con il rapido sviluppo del cloud computing, sempre più aziende stanno iniziando ad esternalizzare i dati locali sui server cloud. Ovviamente si vanno così a palesare enormi rischi per la sicurezza della privacy.

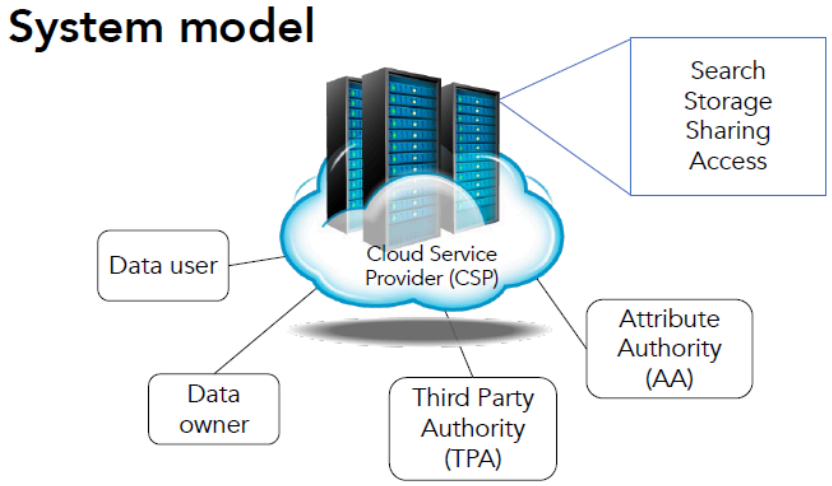
Il **cloud computing** è un nuovo modello di distribuzione di risorse informatiche che consente un comodo accesso tramite rete ad un pool virtualizzato di risorse remote. I suoi vantaggi includono:

- Self-service su richiesta;
- Pool illimitato di risorse;
- Scalabilità dinamica;
- Prezzi misurati in base al servizio;
- Riduzione dei costi e rischi di gestione.

I servizi di cloud data services racchiudono i seguenti:

- Ricerca;
- Computazione;
- Memorizzazione;
- Condivisione;
- Accesso.

Un modello di sistema di cloud computing è il seguente:



1. Il **CSP** è il fornitore di servizi cloud, che dispone di notevoli spazi di archiviazione, capacità di calcolo ed è responsabile della fornitura di tutti i servizi;
2. I **data owner** sono i proprietari dei dati, i quali caricano i propri dati locali sul cloud;
3. L'**utente** è la parte che consuma i dati sul cloud;
4. Il **Third Party Authority** (**TPA**) è una parte a cui potrebbe essere affidato il compito di verificare la correttezza dei dati archiviati nel cloud, ma che non ha il privilegio di accedere al contenuto effettivo dei dati;
5. L'**Attribute Authority** (AA) è un'autorità chiave affidabile in un sistema di controllo degli accessi basato su attributi. Si occupa di generare chiavi di attributo per gli utenti in base alla loro identità.

Oltre i tradizionali pericoli di sicurezza, come DoS o intercettazioni su rete, sono da attenzionare numerose minacce associate ai servizi di cloud, come:

- ***Fuga o divulgazione di dati***: una possibile soluzione sarebbe quella di usare CSP con riservatezza garantita, ad esempio tramite cifratura per la ricerca sicura di dati [come mostrato in precedenza];
- ***Accessi illegali***: una possibile soluzione sarebbe quella di far controllare la condivisione dei dati dal proprietario;
- ***Corruzione o perdita di dati***: una possibile soluzione sarebbe quella di usare CSP con integrità garantita, ad esempio tramite duplicazione dei dati locali per il controllo;
- ***Calamità naturali***: una possibile soluzione sarebbe quella di usare CSP con server dislocati;
- ***Malware***;
- ***Cyber-spionaggio***;
- ***Violazione della privacy dell'utente***: una possibile soluzione sarebbe quella di usare CSP con privacy garantita;

Il controllo degli accessi ai servizi di cloud data services prevede la **cifratura basata su attributi**, nella quale svolge un ruolo principale l'Attribute Authority. Quest'ultima setta dei parametri pubblici e crea una chiave master. A questo punto crea una chiave privata in base ai parametri pubblici, la chiave master e un set di attributi. I messaggi cifrati con quella chiave saranno decifrabili solamente dagli utenti che posseggono gli attributi con cui la chiave è stata definita. In questo caso si parla di ***politica basata sulla chiave***, dato che la politica di accesso è definita nella chiave stessa. Inoltre esiste la ***politica basata su testo cifrato***, in cui la politica di accesso è definita nel testo stesso.

Per quanto riguarda invece il controllo di integrità, si procede con due prove:

- ***Prova di possesso dei dati***: per controllare che effettivamente il cloud server memorizzi i dati corretti. Tale prova consiste in challenge casuali in cui confronta dei metadati salvati in locale durante il salvataggio dei dati sul server con i metadati effettivamente salvati sul server;
- ***Prova di recuperabilità***: per provare che un file di destinazione è intatto. Tale prova consiste in challenge simili alle precedenti, ma nelle quali si confrontano anche dei tag di integrità. In questo modo si è sicuri che anche il contenuto sia affidabile.

CAPITOLO 6 - MALWARE

Un **malware** è un programma che viene inserito in un sistema, tipicamente di nascosto, con l'intento di compromettere la riservatezza, l'integrità o la disponibilità dei dati, della applicazioni o del sistema operativo della vittima, oppure allo scopo di infastidire o interrompere la vittima in altro modo.

I malware possono essere suddivisi in base a vari aspetti, tra cui i *meccanismi con cui si propagano*:

- ***Infezione di contenuti esistenti*** da parte di virus che vengono successivamente diffusi ad altri sistemi;
- ***Sfruttamento delle vulnerabilità del software*** da parte di worm o drive-by-downloads per consentire la replicazione del malware;
- ***Attacchi di ingegneria sociale*** che convincono gli utenti a bypassare meccanismi di sicurezza per installare trojan o abboccare ad attacchi di phishing.

Altre classificazioni di malware invece distinguono:

- Malware che ***necessitano di un programma host***, essendo codice parassita (come i virus);
- Malware che ***sono indipendenti*** e rappresentano programmi autonomi (come i worm e i trojan).

Oppure:

- Malware che ***si replicano*** (trojan e email spam);
- Malware che ***non si replicano*** (virus e worm).

I malware differiscono poi anche per i **payload** (cioè le azioni malevoli che compiono), come:

- Corruzione di file di sistema o file di dati;
- Furto di servizio per rendere il sistema uno agente di attacco zombie;
- Furto di informazioni dal sistema;
- Occultamento all'interno del sistema.

In realtà i malware odierni non si fermano ad un unico meccanismo di propagazione e di payload.

Inizialmente lo sviluppo di malware richiedeva notevoli competenze tecniche da parte degli autori. La situazione è cambiata con l'introduzione di toolkit per la creazione di malware, noti come ***crimeware***. Essi contengono una varietà di moduli di meccanismi di propagazione e di payload.

Un altro motivo di sviluppo dei malware è stato il passaggio da attaccanti individuali a ***fonti di attacco*** molto più organizzate, come organizzazioni mosse da motivi politici, organizzazioni criminali, organizzazioni governative, ecc…

Un attacco può essere ***one-shot***, cioè inteso per colpire il sistema in maniera rapida, oppure ***duraturo*** nel tempo. Esempi di attacchi duraturi sono le ***minacce persistenti avanzate*** (**APT**). Tipicamente essi sono sponsorizzati da Stati o da organizzazioni criminali. Differiscono dagli altri attacchi per la selezione attenta del target e l'intrusione occulta per periodi estesi di tempo. Le APT prendono il nome dalle seguenti caratteristiche:

- **Avanzate**: usano di un'ampia varietà di tecnologie di intrusione e malware;
- **Persistenti**: sono attacchi intesi ad essere prolungati nel tempo contro il bersaglio scelto per massimizzare le possibilità di riuscita. È possibile che tali minacce attuino una serie di attacchi nel tempo fino a quando il bersaglio non venga compromesso;
- **Minacce**: sono minacce verso obiettivi specifici organizzati da attaccanti capaci e ben finanziati.

Lo scopo di questi attacchi varia dal furto della proprietà intellettuale o dei dati relativi alla sicurezza e alle infrastrutture. Le tecniche utilizzate includono l'ingegneria sociale, le email di phishing, e attacchi drive-by-downloads tramite siti compromessi e che potrebbero essere visitati da utenti appartenenti al target dell'attacco. L'intento risulta quindi essere quello di infettare l'obiettivo con un malware sofisticato, con meccanismi multipli di propagazione e di payload. Una volta ottenuto l'accesso iniziale ai sistemi target, viene utilizzata una serie ulteriore di strumenti di attacco per mantenere ed estendere tale accesso.

Metodo di propagazione: *Infezione di contenuti esistenti*

Virus

I **virus** sono frammenti di software parassiti che infettano programmi, al fine di modificarli per includere una copia del virus, replicarsi ed andare ad infettare ulteriori contenuti e propagarsi facilmente attraverso la rete. Quando un virus è attaccato ad un programma eseguibile esso assume gli stessi permessi del programma stesso.

I virus sono composti da tre parti:

- ***Meccanismo di infezione*** **[vettore dell'infezione]**: il mezzo attraverso il quale il virus si diffonde;
- ***Trigger*** [**bomba logica**]: l'evento o la condizione che determina l'attivazione del payload;

- ***Payload***: l'azione malevola del virus che comporta danni al sistema.

Durante la sua vita, il virus attraversa quattro fasi:

- **Fase dormiente**: il virus è inattivo e aspetta che venga attivato da un trigger. Non tutti i virus attraversano questa fase;
- **Fase di propagazione**: il virus inserisce una copia di sé stesso in altri programmi o in altre locazioni di memoria. Ciascun programma infetto ora conterrà un clone del virus che attraverserà a sua volta una fase di propagazione;
- **Fase di attivazione**: il virus viene attivato da un trigger, che può essere un qualsiasi evento o condizione del sistema;
- **Fase di esecuzione**: il virus esegue il payload, che può produrre effetti innocui o dannosi.

Virus macro

Una tipologia di virus è il **virus macro** (o ***virus eseguibile***), cioè un virus che si attacca ad un documento in modo tale da modificare il funzionamento del file in fase di esecuzione e da propagarsi. Sono minacciosi per una serie di motivi:

- Sono indipendenti dalla piattaforma;
- Infettano documenti e non porzioni di codice eseguibile, facendo sì che i tradizionali controlli di accesso non abbiano tanto effetto nella difesa da questi malware;
- Si diffonde facilmente;

I virus, come il resto dei malware, possono distinguersi in base a vari aspetti. Un primo potrebbe essere il loro *target*:

- ***Settore di Boot***: infettano un record di boot in modo tale da diffondersi ogni quando il sistema viene avviato dal disco;
- ***File***: infetta i file che il sistema operativo o la shell considerano eseguibili;
- ***Virus macro***;
- Multipartito: infetta più tipi di file in diversi modi.

I virus poi possono distinguersi in base alla loro *strategia di occultamento*:

- ***Virus criptato***: una porzione del virus cripta la restante parte in modo da renderla impossibile da rilevare. In fase di payload poi la prima parte del virus decodifica la seconda e si esegue;
- ***Virus invisibile***: una forma di virus progettata esplicitamente per nascondersi dal rilevamento da parte di software antivirus. Per raggiungere questo tipo di occultamento c'è bisogno di conoscenze ampie e di continui aggiornamenti;

- ***Virus polimorfici***: una forma di virus che crea copie diverse di sé stesso in maniera da risultare non rintracciabile. Il funzionamento delle copie è però uguale al precedente;
- ***Virus metamorfici***: una forma di virus che crea copie diverse di sé stesso in maniera da risultare non rintracciabile. Il funzionamento delle copie risulta diverso dal precedente;

Metodo di propagazione: *Sfruttamento vulnerabilità software*

Worm

Un **worm** è un programma che cerca attivamente più macchine da infettare e da usare come trampolino per attaccare altre macchine. I worm sfruttano le vulnerabilità del software di programmi client e server. Oltre alla propagazione il worm di solito porta con sé una forma di payload.

Per replicarsi, un worm utilizza alcuni mezzi per accedere a sistemi remoti, tra cui:

- ***Posta elettronica o servizi di messaggistica istantanea***: un worm invia una copia di sé stesso attraverso posta elettronica o come allegato di un messaggio instantaneo;
- ***Condivisione di file***: un worm crea una copia di se stesso oppure infetta come virus altri file idonei su supporti rimovibili, come USB, CD, DVD, ecc…;
- ***Esecuzione remota***: un worm esegue una copia di sé stesso su un sistema remoto;
- ***Accesso o trasferimento di file remoti***: un worm utilizza un servizio di accesso o trasferimento di file remoti a un altro sistema per lanciarsi in esso;
- ***Accesso remoto***: un worm accede ad un sistema remoto come un utente e poi usa comandi per copiarsi in quel sistema e venire eseguito.

Una volta raggiunto un altro sistema, il worm potrebbe nascondersi nominando come processo di sistema o utilizzando qualche altro nome che potrebbe non essere notato da un operatore di sistema. Quelli più recenti poi potrebbero perfino inserire il proprio codice all'interno di altri processi.

Per quanto riguarda il ciclo di vita, un worm attraversa più o meno le stesse fasi di un virus.

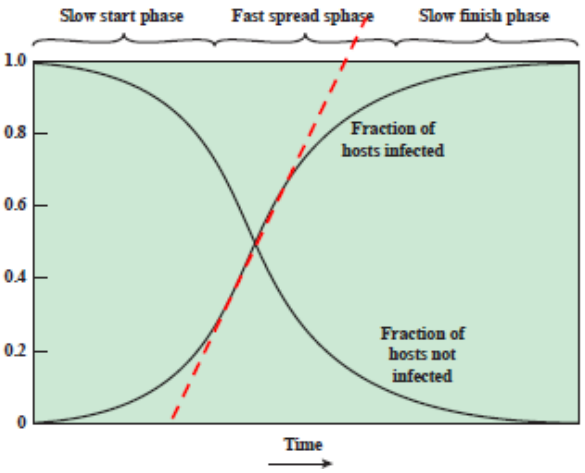
La prima funzione nella fase di propagazione di un worm è la ricerca di sistemi da infettare. Tale processo è detto **scansione** (o ***fingerprinting***). Sono varie le strategie di scansione degli indirizzi di rete che un warm potrebbe utilizzare:

- ***Casuale***: ciascun host compromesso analizza indirizzi casuali nello spazio degli indirizzi IP. Questa tecnica produce un volume elevato di traffico internet, che può causare rallentamenti;

- ***Hit-list***: innanzitutto l'attaccante compila un lungo elenco di macchine potenzialmente vulnerabili. Una volta completata la lista si cominciano con le infezioni di tali macchine. Ad ogni macchina infetta viene data poi una parte della lista di macchine da infettare;
- ***Topologico***: questo metodo utilizza le informazioni contenute sul computer infetto per trovare altre macchine da attaccare;
- ***Subnet locale***: in questo caso i computer infetti attaccano altri dispositivi all'interno della stessa rete locale, in modo da evitare controlli del firewall.

Un worm ben progettato può diffondersi rapidamente ed infettare un numero enorme di host. La velocità con cui tale malware si propaga segue un modello generale, descritto dal seguente grafico, nel quale si distinguono tre fasi:

- ***Fase di avvio***: *lenta*;
- ***Fase di diffusione***: *veloce*;
- ***Fase di finitura***: *lenta*.



Probabilmente la prima infezione da worm significativa fu quella rilasciata da Morris nel 1988. Il worm Morris fu progettato per diffondersi su sistemi UNIX utilizzando tre tipi di attacchi:

- Tentare di accedere ad un host remoto per violare il file delle password locali e utilizzarle per accedere a sistemi diversi. (Il presupposto di questo attacco è che un utente potesse usare la stessa password per più sistemi);
- Sfruttare un bug nel protocollo UNIX che segnala il luogo in cui si trova un utente;
- Sfruttare una botola nell'opzione di debug del processo remoto che riceve e invia mail.

Se uno qualsiasi di questi attacchi andava a buon fine, il worm riusciva ad eseguire delle istruzioni per scaricare il resto del worm.

Mobile code

Per mobile code si intende un programma che può essere spedito senza modifiche ad un insieme eterogeneo di piattaforme ed eseguiti con la stessa semantica. Il codice mobile viene trasmesso da un sistema remoto ad uno locale e quindi eseguito qui. I principali veicoli includono Java, Javascript, ActiveX, ecc…. I metodi più comuni di utilizzo del codice mobile per operazioni dannose sono lo scripting, i siti web interattivi e dinamici, gli allegati di posta elettronica e i download da siti o software poco attendibili.

Drive-by-Downloads

Un'altra tecnica comune di attacco sfrutta le vulnerabilità dei browser in modo che, quando l'utente sfrutta una pagina web controllata dall'attaccante, vengono scaricati ed installati malware all'insaputa dell'utente stesso. Questo approccio è noto come **drive-by-downloads**. Nella maggior parte dei casi questo malware non si propaga attivamente come fa un worm, ma aspetta che siano invece gli utenti a visitare la pagina web dannosa per diffondersi nei loro sistemi.

In generale questo tipo di attacchi sono rivolti a chiunque visiti un determinato sito. Esiste però una variante, chiamata **watering-hole**, che si distingue per il target ristretto di vittime. L'attaccante infatti ricerca i siti web che le vittime predefinite sono abituati a visitare, quindi scansiona tali siti per identificare quelli più vulnerabili e che possano consentire la compromissione con un attacco drive-by-downloads. A questo punto attendono che le vittime si uccidano con le loro stesse mani. Il codice di questo tipo di attacco potrebbe anche contemplare una sezione per capire se l'utente che ha visitato il sito sia effettivamente un utente target. Se ciò non fosse allora l'utente non avrebbe alcuna conseguenza negativa. Ciò abbassa notevolmente le probabilità che il sito compromesso sia notato.

Malvertising

Il **malvertising** è un'altra tecnica per inserire malware all'interno di siti web senza effettivamente comprometterli. L'attaccante paga per inserire annunci pubblicitari contenenti il malware. Anche in questo caso l'attaccante può generare codice per ridurre la possibilità di rilevamento. Questo tipo di attacchi si è esteso negli ultimi anni data la semplicità nell'inserire tali pubblicità all'interno dei siti desiderati e la difficoltà nel tracciarlo. Gli attaccanti potrebbero addirittura inserire ads per poche ore prestabilite della giornata, cioè quando pensano che i target effettivamente visitino il sito.

Clickjacking

Il **clickjacking**, anche noto come ***attacco di riparazione dell'interfaccia utente***, è un attacco volto a raccogliere i click di un utente infetto. Un tipico attacco usa più livelli trasparenti o opachi per indurre un utente a fare click su un pulsante o un collegamento ad altra pagina corrotta. Usando una tecnica simile l'utente potrebbe essere indotto a pensare di star digitando una password per un'autenticazione, quando invece la sta inserendo in un frame invisibile controllato dall'attaccante.

Metodo di propagazione: *Ingegneria sociale*

L'ultimo metodo di propagazione di un malware è l'ingegneria sociale, cioè ingannare l'utente per farsi aiutare a compromettere i suoi sistemi.

E-mail di spam

Con la crescente diffusione dell'utilizzo della posta elettronica, si è andato a diffondere anche l'uso di messaggi di posta elettronica non richiesti, noti come **spam**. Ma lo spam è anche un importante vettore di malware, grazie soprattutto ai suoi allegati. Oltre a diffondere worm, trojan e macro virus, tali messaggi possono essere utilizzati per attacchi di phishing, indirizzando l'utente verso un sito web falso che rispecchia però un servizio legittimo, come un home banking. Qui l'utente è portato ad inserire, e quindi condividere con l'attaccante, informazioni sensibili.

Trojan

Un **trojan** (***cavallo di troia***) è un programma che, quando richiamato, esegue alcune funzioni indesiderate o dannose, come la ricerca all'interno del file system dell'utente di informazioni sensibili. Il trojan viene inviato dall'attaccante tramite pagina web, email o messaggio testuale con l'intento di invogliare l'utente ad eseguire tale programma. I trojan rientrano in uno di tre modelli:

1. Continuano a svolgere la funzione del programma originale ed eseguono inoltre la funzione dannosa;
2. Continuano ad eseguire la funzione del programma originale ma modificandole in modo tale da effettuare o mascherare attività dannose;
3. Eseguono soltanto la funzione dannosa.

Payload

Una volta che il malware è attivo sul sistema, esso inizierà a compiere delle azioni su quel sistema (**payload**). Esistono diverse varianti di payload, ognuna dei quali con obiettivi diversi.

Corruzione di sistema

La prima categoria di payload riguarda le tecniche utilizzate dal malware per corrompere il sistema tramite il suo filesystem.

- ***Distruzione dei dati***: questi payload lavorano sulla distruzione e corruzione di dati:
    - Un primo esempio è stato il Chernobyl che infetta file eseguibili quando vengono aperti e cancella i dati sul sistema infetto sovrascrivendo il primo mb de disco rigido con zeri, provocando una massiccia corruzione del file system;
    - Il worm Klez, invece, si diffondeva inviando copie di sé via e-mail ad indirizzi trovati nella rubrica e nel file system. Può arrestare ed eliminare alcuni antivirus e svuotare i file sul disco rigido, utilizzando come trigger un determinato giorno del mese;
    - I ransomware invece non vanno a distruggere dati, ma li criptano e chiedono un riscatto ai proprietari per poter ottenere la chiave di decriptazione.
- ***Danni nel mondo reale***: questi payload mirano a causare danni all'hardware. Un esempio è ancora il virus Chernobyl, nel caso esso prendesse di mira file del BIOS rendendo inutilizzabile il chip di avvio di sistema;
- ***Bomba logica***: questi payload rappresentano del codice incorporato al malware, il quale viene esploso quando vengono soddisfatte determinate condizioni. Una volta attivata, una bomba può alterare o cancellare dati o interi file, causare l'arresto della macchina o altri danni.

Bot

Questi payload consentono al malware di sovvertire le risorse di rete e di sistema affinché possano essere usate dall'aggressore. Questi sistemi così colpiti vengono detti **bot**. Una collezione di bots capaci di agire in maniera coordinata formano un **botnet**. Essi possono essere usati per:

- Attacchi DDoS (DoS distribuito);
- Spamming;
- Sniffing del traffico;
- Keylogging.

La possibilità di controllo remoto è ciò che distingue un bot da un worm. Quest'ultimo infatti si propaga e si attiva mentre il bot è controllato tramite una rete di server di comando e controllo. Questo contatto non deve essere continuo.

Uno dei primi mezzi per implementare tale forma di controllo utilizzava server IRC, ai quali i bot si collegano e trattano i messaggi ricevuti come comandi. Le botnet più recenti invece utilizzano canali di comunicazione nascosti tramite protocolli come HTTP. Altra modalità richiede invece l'utilizzo di meccanismi di controllo distribuito tramite protocolli peer-to-peer per evitare problemi per singoli punti di guasto.

Furto di informazioni

Questa categoria di payload riguarda le tecniche utilizzate dal malware per rubare informazioni sensibili.

- ***Keylogger***: questi payload catturano i tasti premuti sulla macchina infetta per consentire all'attaccante di monitorare queste informazioni sensibili. Ovviamente i keylogger implementano una qualche forma di meccanismo di filtraggio che restituisce solo informazioni desiderate (ad esempio quelle successive a parole chiave come password);
- ***Spyware***: questi payload consentono di monitorare un più largo insieme di attività sul sistema, ad esempio si potrebbe monitorare la cronologia e il contenuto delle attività su browser, redirezionare l'utente a siti fake o ancora modificare dinamicamente dei dati;
- ***Phishing***: questi payload sfruttano l'ingegneria sociale mascherandosi da comunicazioni attendibili. Esempi di phishing sono includere in email spam dei link a pagine web false, ma che sembrano pagine attendibili e suggerire azioni urgenti che inducano gli utenti ad autenticarsi; Una variante più pericolosa è lo ***spear-phishing***. Anch'essa utilizza e-mail mascherate come attendibili, tuttavia i destinatari sono scelti con cura e ogni e-mail è creata per adattarsi al suo destinatario.

Occultamento

L'ultima categoria di payload riguarda le tecniche utilizzate dal malware per nascondere la propria presenza sul sistema infetto. Ne esistono due principali:

- ***Backdoor*** (o ***trapdoor***): è un punto di entrata segreto in un programma che consente a qualcuno che ne è a conoscenza di ottenere l'accesso senza passare attraverso le consuete procedure di sicurezza per l'accesso. Questa procedura è utilizzata anche da programmatori per eseguire debug e testare programmi (in questo caso si parla di ***hook di manutenzione***). È complicato implementare controlli di sistema per individuare backdoor in applicazioni;
- ***Rootkit***: è un insieme di programmi installati su un sistema per mantenere un accesso nascosto a quel sistema con privilegi da root. Un rootkit si nasconde sovvertendo i

meccanismi che monitorano e segnalano processi, file e registri. Essi possono essere classificati tramite le seguenti caratteristiche:

- ***Persistente***;
- ***Basato sulla memoria***;
- ***Modalità utente***;
- ***Modalità kernel***: apportano modifiche all'interno del kernel, coesistendo quindi con il sistema operativo e risultando molto più difficili da trovare. L'obiettivo principale è quello di modificare in qualche modo la gestione delle syscall:
  - **Modifica la tabella delle syscall**: modifica l'indirizzo a cui puntano le syscall, in modo tale da farli riferire a nuovi indirizzi di memoria dove è presente il codice malevolo;
  - **Modifica dei target della tabella delle syscall**: sovrascrive il codice da eseguire di una syscall con codice malevolo (in questo caso quindi la tabella non viene modificata);
  - **Redirezione della tabella delle syscall**: reindirizza l'intero sistema ad una nuova tabella delle syscall.
- ***Basato su macchine virtuali***;
- ***Modalità esterna***.

Contromisure

La migliore soluzione al problema dei malware è sicuramente la **prevenzione**, la quale prevede 4 elementi principali (policy, consapevolezza, mitigazione delle vulnerabilità, mitigazione delle minacce).

Se la prevenzione fallisce, è possibile utilizzare meccanismi tecnici per supportare le seguenti opzioni di mitigazione delle minacce:

- ***Rilevamento***: determinare se si è verificata un'infezione;
- ***Identificazione***: Identificare il malware specifico che ha infettato il sistema;
- ***Rimozione***: rimuovere tutte le tracce del malware in modo che non possa diffondersi.

Per svolgere questi compiti sono stati introdotti particolari software, gli **anti-virus**. Essi si sono evoluti nel tempo e hanno vissuto 4 generazioni:

1ª: *scanner semplici* - richiedono signatures per identificare il malware;

2ª: *scanner euristici* - richiedono euristiche per cercare eventuali malware;

3ª: *trappole di attività* - localizzati in memoria e cercano il malware in base alle sue attività;

4ª: *protezione completa* - sfruttano più delle precedenti tecniche insieme;

Una possibile contromisure nel caso non ci si fidasse di codice potenzialmente malevolo è poi quella di eseguirlo in contesti protetti (come ***sandbox*** o ***macchine virtuali***). In questo modo è possibile monitorare le conseguenze di tale codice senza che esso possa diventare una minaccia per il sistema reale. L'unico problema di questa contromisura è quello di stabilire quanto tempo bisogna controllare l'esecuzione del codice all'interno di una sandbox per interpretarlo come buono.

Ci sono poi i ***software di blocco del comportamento basato su host***. Essi sono integrati con il sistema operativo di un host e monitora il comportamento in tempo reale di azioni malevoli. In questo caso però il problema è che, essendo necessario eseguire il codice sulla macchina per capire se esso sia malevolo, potrebbe darsi che esso possa causare problemi prima che ce ne possiamo accorgere e bloccarlo.

Un altro luogo dove viene utilizzato il software antivirus è il firewall incluso anche in servizi di email e web proxies. Tuttavia tale approccio si limita alla scansione del contenuto del malware, dato che non ha possibilità di controllare la sua esecuzione. Possono essere utilizzati due tipi di monitoraggio:

- ***Monitor di ingresso***: si trovano al confine tra la rete aziendale e Internet. Un esempio di tecnica di rilevamento di questo tipo consiste nel cercare il traffico in entrata verso indirizzi IP locali non utilizzati;
- ***Monitor di uscita***: si trovano nei punti di uscita delle singole LAN sulla rete aziendale o ancora al confine tra la rete aziendale e Internet. Il monitoraggio in uscita è progettato per individuare segni di scansione o altri comportamenti strani.

CAPITOLO 7 - DENIAL OF SERVICE (DoS)

Un **Denial of Service** (**DoS**) è un'azione che impedisce o compromette l'uso autorizzato di reti, sistemi o applicazioni esaurendo risorse come CPU, memoria, larghezza di banda e spazio su disco.

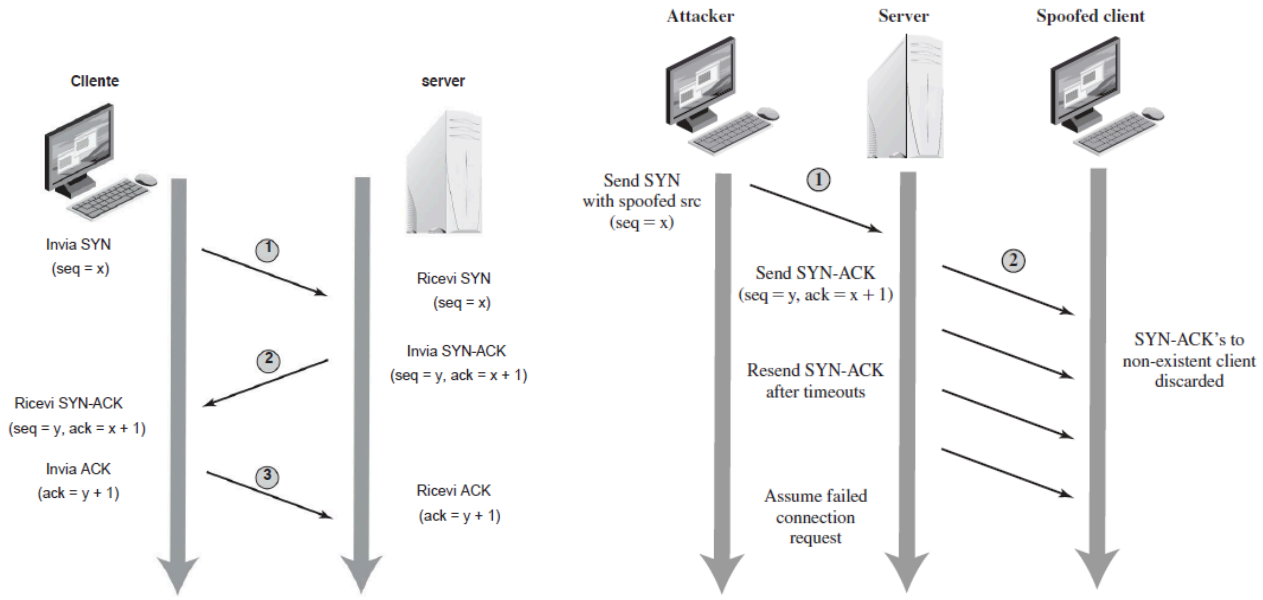
Da tale definizione si comprende come DoS sia una forma di attacco che intende compromettere la disponibilità di alcuni servizi. Le categorie di risorse che potrebbero essere attaccate sono:

- ***Larghezza di banda della rete***: cioè la capacità della rete tra server e Internet;
- ***Risorse di sistema***: in questo caso si mira a sovraccaricare o bloccare il software di gestione della rete;
- ***Risorse dell'applicazione***: tramite richieste valide che consumano un numero significativo di risorse.

Uno degli attacchi DoS classici è il **Flooding Ping Command**, il quale ha l'obiettivo di sopraffare la capacità della rete verso l'organizzazione target. L'attacco ha buone possibilità di successo soprattutto se l'attaccante ha accesso ad un sistema con una connessione di rete di capacità maggiore rispetto a quello della vittima; in questo modo genererà un volume di traffico maggiore a quello che è possibile gestire con conseguente perdita di pacchetti. Questo attacco però presenta lo svantaggio dell'assenza dell'anonimato: infatti la sorgente dell'attacco è nota a meno che non si usino indirizzi di rete falsificati. Inoltre anche la capacità di rete dell'attaccante può risentire dello stesso attacco, volendo rispondere a pacchetti di ritorno.

Per quanto riguarda il problema dell'indirizzo di origine noto, si potrebbe optare per l'utilizzo di uno falsificato. Questo metodo è noto come ***spoofing dell'indirizzo di origine***, e può essere utile sia per rendere l'attaccante difficile da individuare sia per liberare la rete dell'attaccante. Infatti in questo modo le risposte del server vittima arriveranno sugli indirizzi falsi e non all'attaccante.

Un altro attacco DoS classico è il **SYN Spoofing**, il quale attacca la capacità di un server di rete di rispondere alle richieste di connessione TCP, sovraccaricando le tabelle usate per gestire tali richieste. Questo rende impossibile successive richieste di connessione da utenti leciti. Questo attacco utilizza il sistema di handshake a tre vie per le connessioni TCP. L'attaccante però invia un numero elevato di richieste di connessione e lo fa specificando un indirizzo di rete contraffatto, al quale poi arriveranno i SYN-ACK. Nel caso esista qualcosa all'indirizzo contraffatto allora esso risponderà con reset e il server cancellerà i dati relativi a questa richiesta dalla tabella. Nel caso invece che non vi sia niente all'indirizzo falsificato (oppure che chi c'è è troppo occupato) allora il server continuerà a mandare SYN-ACK per un numero specifico di volte prima di cancellare le informazioni dalla tabella, la quale per richieste fallite multiple potrebbe sovraccaricarsi.



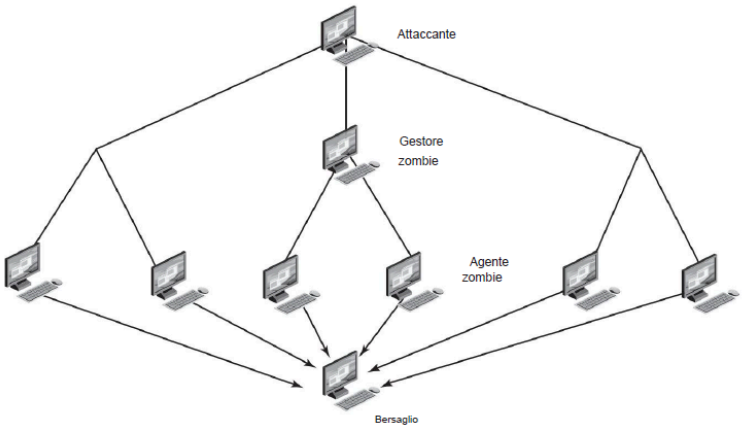
Attacchi di Flooding

Gli **attacchi di flooding** (*inondazione*) hanno l'intento di sovraccaricare la capacità di qualche connessione di rete verso un determinato server. Essi possono assumere diverse forme a seconda del protocollo di rete utilizzato:

- ***Flood ICMP***: sono gli attacchi di flooding ping descritti precedentemente. Essi usano il protocollo ICMP, il quale è utile agli attaccanti dato che gli amministratori di rete tradizionalmente consentono l'ingresso di tali pacchetti anche per strumento di diagnostica della rete;
- ***Flood UDP***: usa pacchetti direzionati verso un numero di porta (quindi un potenziale servizio) sul sistema di destinazione;
- ***Flood TCP SYN***: anche in questo caso vengono mandati pacchetti TCP al sistema target, ma, a differenza del SYN spoofing, in questo caso i pacchetti sono legittimi.

Attacchi DoS Distribuiti

Dati i limiti di attacchi flooding generati da un singolo sistema, una possibile soluzione potrebbe essere quella di usare molteplici sistemi per generare tali attacchi. L'attaccante potrebbe quindi usare una botnet (*rete*



*di sistemi **zombie***) creata tramite malware.

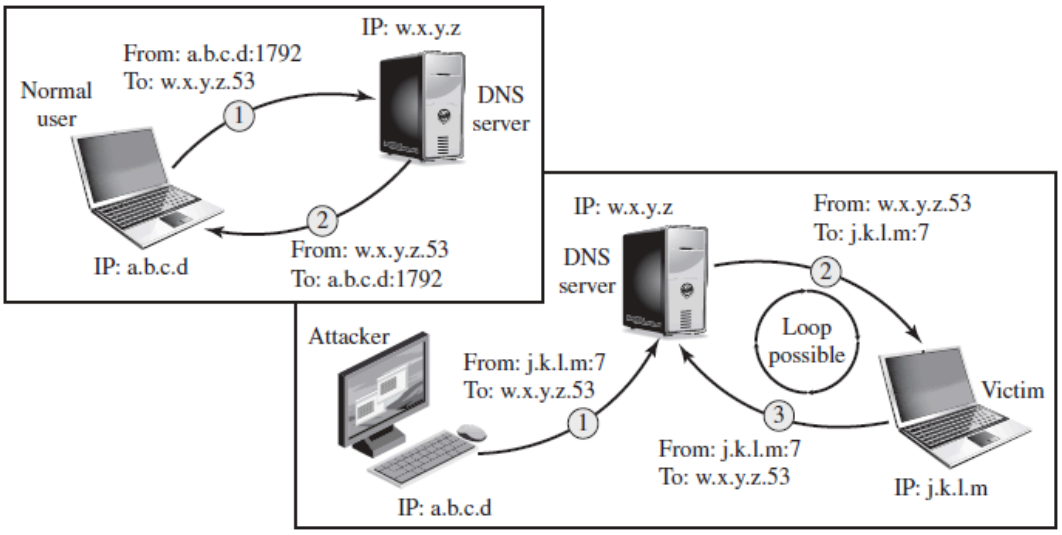
Attacchi basati su HTTP

Un altro modo per condurre attacchi DoS è quello di sfruttare il protocollo HTTP. Tali attacchi bombardando i web servers tramite richieste HTTP e consumando in questo modo un numero considerevole di risorse. Una variante, nota come ***Flood HTTP ricorsivo*** (o ***spidering***)utilizza dei bot che partono da un determinato indirizzo HTTP e poi seguono in modo ricorsivo tutti i collegamenti presenti su di esso.

Una forma interessante di attacco basato su HTTP è **Slowloris**. Esso cerca di monopolizzare tutti i thread di gestione delle richieste sul server web inviando richieste HTTP che non vengono mai completate. Ciò è possibile perché tali richieste sono formattate in un certo modo (come non inserendo la doppia riga vuota, simbolo di fine richiesta). Man mano che l'attacco si prolunga, esso consuma tutte le connessioni disponibili al server. Sistemi standard di rilevamento e prevenzione di intrusioni non riuscirebbero a identificare tale attacco. Una possibile contromisura potrebbe essere invece la variazione del timeout in base al numero di altre connessioni libere disponibili, in modo tale da tagliare corto nel caso di sovraccarico del server.

Attacchi di riflessione e amplificazione

L'**attacco di riflessione** è un attacco che prevede l'invio di pacchetti ad un servizio noto (*intermediario*) con un indirizzo di origine falsificato. In realtà in questo caso l'obiettivo dell'attacco è proprio l'indirizzo falsificato, al quale arriveranno di riflesso un numero di risposte in grado di inondare i collegamenti di rete del sistema target senza che il server a cui sono state fatte richieste sia allertato.



Gli **attacchi di amplificazione** sono una variante degli attacchi di riflessione. Differiscono per il fatto che per ogni pacchetto inviato, ne vengono generati più di uno. Ciò può essere ottenuto indirizzando la richiesta all'indirizzo di trasmissione per alcune reti. Quindi tutti gli host su quella rete possono potenzialmente rispondere, generando un'ondata di risposte.

Ancora un buon esempio potrebbe essere un attacco che sfrutti un server DNS come intermediario. L'attaccante crea una serie di richieste DNS contenenti l'indirizzo del sistema target. Si usa il comportamento dei DNS, i quali da richieste piccole producono richieste molto più grandi.

Contromisure

È possibile adottare una serie di contromisure per gli attacchi DoS, ma non è possibile prevenirli completamente. Un grande traffico di rete infatti potrebbe comunque essere legittimo. In generale esistono quattro linee di difesa:

- ***Prevenzione degli attacchi*** [*prima dell'attacco*]: tali tecniche includono l'applicazione di politiche per il consumo delle risorse e la modifica di protocolli in Internet per ridurre la possibilità di attacchi DoS;
- ***Rilevamento e filtraggio degli attacchi*** [*durante l'attacco*]: tali meccanismi tentano di rilevare quanto prima l'attacco per poter rispondere in modo adeguato. Si potrebbero usare ad esempio signatures e euristiche per il rilevamento e filtrare i pacchetti sospetti;
- ***Tracciamento e identificazione della fonte dell'attacco*** [*durante/dopo l'attacco*]: tali meccanismi hanno il principale scopo di evitare attacchi futuri dalla stessa fonte. Difficilmente si riesce a bloccarne uno in corso;
- ***Reazione all'attacco*** [*dopo l'attacco*]: tali meccanismi tentano di eliminare o ridurre effetti di attacchi DoS.

Una componente critica degli attacchi DoS è l'utilizzo di indirizzi falsificati. Una possibile contromisura potrebbe essere quella di effettuare un filtraggio di pacchetti proprio in base all'indirizzo specificato. Tali filtri dovrebbero essere posti quanto più vicini alla fonte e possibilmente effettuati da una ISP, la quale conosce gli indirizzi dei propri clienti e potrebbe rilevare pacchetti sospetti e bloccarli. D'altro canto un filtraggio del genere non potrebbe essere effettuato quando il pacchetto ha già raggiunto l'Internet.

La miglior difesa contro gli attacchi di amplificazione è invece bloccare l'uso delle trasmissioni dirette via IP. Entrano in gioco ancora le ISP, o anche gli intermediari in questo caso, i quali

potrebbero adottare pratiche di buona sicurezza generali, come l'utilizzo del CAPTCHA per distinguere richieste legittime da quelle inviate da bot.

Infine, se un'organizzazione dipende da servizi di rete, dovrebbe prendere in considerazione il mirroring e la replica dei server su più siti e con più reti.

Per rispondere con successo ad un attacco DoS è necessario un buon piano di risposta agli incidenti, il quale dovrebbe includere dettagli su come contattare personale tecnico della ISP tramite mezzi non di rete. In questo modo non si va a sovraccaricare la rete e si può richiedere di tracciare il flusso dei pacchetti in entrata per scovare la fonte. Un'altra difesa da utilizzare potrebbero essere i filtri anti-spoofing. È importante poi essere forniti monitor per identificare e notare pattern di traffico anormale, in modo tale da comprendere il tipo di attacco che si sta subendo.

Per rispondere ad un attacco DoS bisogna quindi implementare un piano di contingenza, che preveda ad esempio lo switch ad un server di backup. Fondamentale è aggiornare tale piano di difesa per migliorarlo in vista di attacchi futuri.

MITRE ATT&CK

La **matrice MITRE ATT&CK** è un framework per comprendere e categorizzare le varie tattiche, tecniche e procedure usati dagli attaccanti durante i cyber attacchi. Tali categorie sono molteplici:

- ***Ricognizione***: l'attaccante prova ad ottenere informazioni che possa usare per pianificare operazioni future;
- ***Sviluppo di risorse***: l'attaccante prova a stabilire risorse che possano supportare le sue operazioni;
- ***Accesso iniziale***: l'attaccante prova a entrare nella rete;
- ***Esecuzione***: l'attaccante prova ad eseguire codice malevolo;
- ***Persistenza***: l'attaccante prova a mantenere il suo punto di appoggio;
- ***Acquisizione dei privilegi***: l'attaccante prova ad ottenere permessi di alto livello;
- ***Evasione delle difese***: l'attaccante prova ad evitare di essere sgamato;
- ***Credenziali di accesso***: l'attaccante prova a rubare le credenziali dell'utente;
- ***Scoperta***: l'attaccante prova a capire l'ambiente;
- ***Movimento laterale***: l'attaccante prova a spostarsi all'interno dell'ambiente;
- ***Collezione***: l'attaccante prova a carpire dati interessanti per i suoi scopi;
- ***Command & Control***: l'attaccante prova a comunicare e controllare i sistemi corrotti;
- ***Estrazione***: l'attaccante prova ad estrarre dati;
- ***Impatto***: l'attaccante prova a manipolare, interrompere o distruggere sistema e dati.

CAPITOLO 8 - INTRUSION DETECTION

Conoscere le motivazioni per cui gli attaccanti conducono azioni malevoli può aiutare a sviluppare una strategia difensiva migliore. Si distinguono per questo motivo quattro classi di intrusi:

- ***Cyber criminali***, il cui obiettivo è il profitto finanziario. Le attività possono includere furto d'identità, furto di credenziali finanziarie, spionaggio aziendale, furto di dati, ecc…;
- ***Attivisti*** (o ***hacktivist***), il cui obiettivo è morale. Le attività possono includere deturpazione di siti web, DoS, furto e distribuzione di dati, ecc…;
- ***Organizzazioni sponsorizzate da Stati***, il cui obiettivo è condurre attività di spionaggio e sabotaggio;
- ***Altri***, il cui obiettivo è diverso da quelli elencati prima. Questa classe include hacker che agiscono perché motivati da prove di forza o da semplice passione.

Gli attaccanti si distinguono anche in base al loro livello di preparazione:

- ***Apprendista***: sono hacker con competenze tecniche limitate. Essi utilizzano principalmente toolkit e proprio per questo motivo sono più semplici da neutralizzare;
- ***Veterani***: sono hacker con competenze tecniche sufficienti per personalizzare i toolkit in modo da sfruttare vulnerabilità appena scoperte;
- ***Master***: sono hacker con competenze tecniche di alto livello, in grado di scoprire nuove categorie di vulnerabilità o di creare nuovi potenti toolkit di attacco.

Intrusion

Un'**intrusione di sicurezza** è un evento, o combinazione di più eventi, che costituisce un incidente di sicurezza in cui un intruso ottiene, o tenta di ottenere l'accesso ad un sistema (o risorsa) senza avere l'autorizzazione per farlo.

L'**intrusion detection** è un servizio di sicurezza che monitora e analizza gli eventi di sistema allo scopo di individuare e fornire avvisi in tempo reale (o quasi) sui tentativi di accesso alle risorse di sistema in modo non autorizzato,

Un **Intrusion Detection System** (**IDS**) è un sistema di intrusion detection, composto da tre componenti logiche:

- ***Sensori***: responsabili della raccolta dati;
- ***Analizzatori***: il cui obiettivo è quello di analizzare i dati forniti da uno o più sensori per determinare se è avvenuta un'intrusione;

- ***Interfaccia utente***: consente ad un utente di visualizzare l'output del sistema e controllare eventuali anomalie.

Gli IDS sono spesso classificati in base alla fonte e al tipo di dati analizzati:

- ***IDS basati su host*** (**HIDS**): monitora le caratteristiche di un singolo host e gli eventi che si verificano al suo interno per individuare attività sospette;
- ***IDS basati sulla rete*** (**NIDS**): monitora il traffico di rete per determinati segmenti o dispositivi di rete e analizza i protocolli di rete, trasporto e applicazione per identificare attività sospette;
- ***IDS distribuiti o ibridi***: combina le informazioni provenienti da una serie di sensori in un analizzatore centrale.
- ***IDS basati su host*** (**HIDS**): monitora le caratteristiche di un singolo host e gli eventi che si verificano al suo interno per individuare attività sospette;

L'utilizzo degli IDS è motivato da diversi fattori:

- Se un'intrusione è rilevata in tempo, l'intruso può essere identificato ed espulso prima che provochi danni;
- Se l'intrusione non è rilevata in tempo, comunque quanto prima si rileva tanti meno danni l'intruso è capace di provocare;
- Un IDS efficace può fungere da deterrente per possibili intrusi;
- L'intrusion detection consente la raccolta di informazioni sulle nuove tecniche di intrusione, in modo tale da aggiornare le difese in tal senso.

Nonostante sia vero che il comportamento degli intrusi differisce in parte da quello di utenti legittimi, questo non è sempre vero. Ci sono alcune azioni che si sovrappongono e che quindi possono portare gli IDS ad una serie di rilevamenti che rappresentano falsi positivi, in cui gli utenti autorizzati vengono identificati come intrusi. D'altra parte però, cercare di alleggerire le difese per abbassare i falsi positivi potrebbe portare ad un aumento pericoloso dei falsi negativi, cioè intrusioni non rilevate.

Inoltre, un importante studio sulle intrusioni ha affermato che si potrebbe distinguere con abbastanza certezza un attaccante outsider da un utente legittimo. Lo stesso studio ha però mostrato che il compito di individuare un attaccante interno è più difficile, in quanto la differenza tra i comportamenti anomali e legittimi può essere molto piccola. Per tale motivo l'intrusion detection in questo caso non può basarsi solo sulla ricerca di comportamenti anomali. Tuttavia gli insider

potrebbero comunque essere rilevabili tramite una classe di condizioni che suggeriscono azioni non autorizzate.

Gli IDS devono soddisfare dei requisiti:

- Funzionare continuamente con una supervisione umana minima;
- Essere tollerante agli errori, nel senso che deve essere in grado di ripristinare il sistema;
- Resistere alla sovversione e capire quindi se è stato manomesso senza autorizzazioni;
- Imporre un sovraccarico minimo al sistema su cui è in esecuzione;
- Essere configurabile secondo le politiche di sicurezza del sistema che monitora;
- Adattarsi ai cambiamenti nel comportamento del sistema e degli utenti nel tempo;
- Poter scalare per monitorare un gran numero di host;
- Fornire un degrado graduale del servizio, nel senso che se alcuni suoi componenti dovessero smettere di funzionare allora gli altri ne devono risentire quanto meno.
- Consentire la riconfigurazione dinamica in caso di riavvio.

Gli IDS utilizzano in genere uno dei seguenti approcci per analizzare i dati dei sensori:

- ***Rilevamento di anomalie***: comporta la raccolta di dati relativi al comportamento degli utenti legittimi per un certo periodo di tempo. Successivamente i comportamenti osservati sono usati per determinare se si tratti di utente legittimo o meno;
- ***Rilevamento tramite signatures o euristiche***: viene utilizzata una serie di modelli di dati riferiti ad attività dannose o a regole di attacco. Tali dati vengono usati per capire la natura di un determinato comportamento.

Rilevamento di anomalie

L'approccio di rilevamento delle anomalie utilizza diversi approcci di classificazione:

- ***Statistico***: analisi del comportamento osservato usando tecniche statistiche;
- ***Basato sulla conoscenza***: analisi del comportamento osservato in base ad un insieme di regole che modellano un comportamento legittimo;
- ***Machine-learning***: classificazione automatica dei comportamenti in base ad algoritmi di machine-learning addestrati sui modelli ottenuti dall'analisi dei comportamenti registrati inizialmente sull'utente legittimo.

Rilevamento tramite signatures

L'approccio basato su signatures confronta un'ampia raccolta di modelli noti di dati dannosi con i dati archiviati sul sistema o in transito sulla rete. Le signatures devono essere sufficienti da ridurre

al minimo i falsi positivi. Questo metodo è vantaggioso in termini di costi, ma al contempo includono la necessità di aggiornare costantemente il database di signatures.

Rilevamento tramite euristiche

L'approccio basato su euristiche prevede l'uso di regole per identificare penetrazioni note o che potrebbero comunque sfruttare punti deboli noti del sistema o della rete. Tali regole potrebbero essere definite anche per far sì che identifichino comportamenti sospetti, nonostante essi lo siano entro i limiti dei modelli di uso stabiliti.

Intrusion detection basato su host

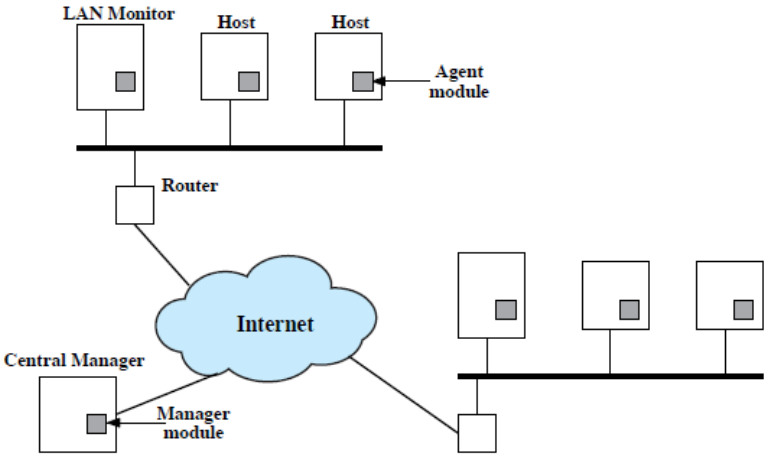
Gli **IDS basati su host** (**HIDS**) aggiungono uno strato specializzato di software di sicurezza a sistemi vulnerabili o sensibili. Gli HIDS monitorano l'attività del sistema tramite signatures ed euristiche o anche tramite rilevamento di anomalie. Il vantaggio degli HIDS è che può rilevare intrusioni sia esterne che interne.

Un componente fondamentale dell'intrusion detection è il sensore che raccoglie i dati. Le fonti di dati includono:

- ***Tracce delle syscall***: una registrazione della sequenza di chiamate di sistema da parte dei processi su un sistema è una delle fonti principali per l'analisi e il rilevamento di intrusioni, nonostante funzionino meglio su UNIX che su Windows (qui ci sono poche syscall);
- ***Log***: la maggior parte dei sistemi operativi moderni tiene traccia dell'attività dell'utente;
- ***Checksum per l'integrità dei file***: un approccio comune per rilevare intrusioni consiste nello scansionare periodicamente file critici per individuare eventuali modifiche strane;
- ***Accesso al registro***: specifico per Windows, anche se con successo limitato anche qui;

Un'organizzazione tipica deve difendere un insieme distribuito di host supportati da una rete. Sebbene sia possibile predisporre una difesa utilizzando HIDS autonomi su ciascun host, è possibile ottenere una maggiore efficacia con ***HIDS distribuiti***. Un'architettura del genere consiste in tre componenti principali:

- ***Modulo agente***: il suo scopo è raccogliere dati sugli eventi dell'host e trasmetterli al gestore centrale;
- ***Modulo monitor LAN***: il suo scopo è analogo a quello dell'agente host ma in questo caso si raccolgono eventi sulla rete;
- ***Modulo di gestione centrale***: riceve report dai componenti precedenti, li elabora e correla questi report per rilevare le intrusioni.



L'approccio generale adottato dall'agente è il seguente: l'agente acquisisce ogni record di controllo prodotto dal sistema, utilizza un filtro per mantenere solo i record di interesse per la sicurezza e li riformatta in un formato standard (HAR). Successivamente un modulo logico, basato su modelli, analizza i record per individuare attività sospette. L'agente esegue quindi la scansione di eventi importanti, sequenze di eventi che aderiscono a modelli di attacchi noti e ricerca comportamenti anomali rispetto a come l'utente storicamente si comporta. Quando viene rilevata un'attività sospetta, viene inviato un avviso al gestore centrale, il quale è in grado di trarre deduzioni dai dati ricevuti.

L'approccio adottato dal monitor LAN è molto simile a quello dell'agente. Esso fornisce inoltre informazioni quali volume di traffico, connessioni tra host, cambiamenti improvvisi del carico di rete, ecc…

Intrusion detection basato su rete

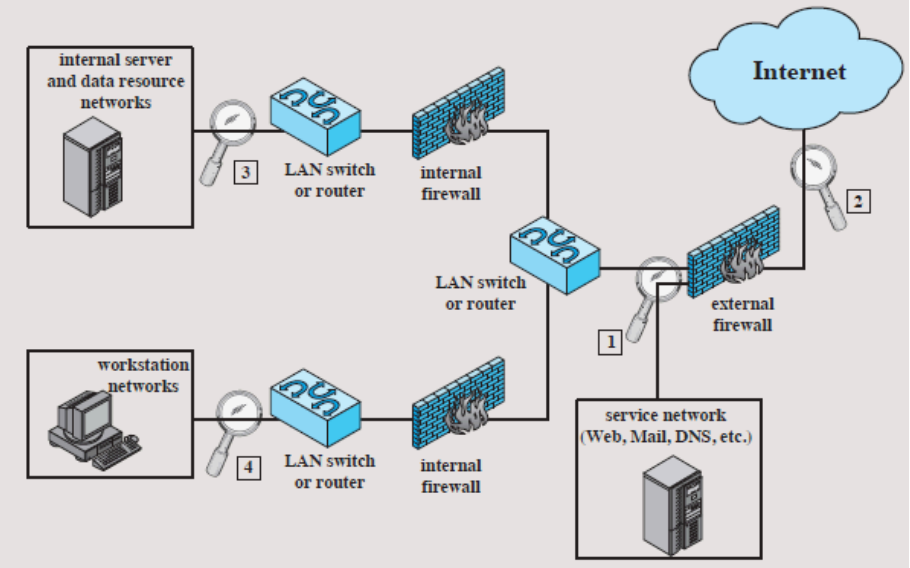
Un ***IDS basato sulla rete*** (**NIDS**) monitora il traffico in punti selezionati su una singola rete o su un insieme di reti interconnesse. Il NIDS esamina il traffico pacchetto per pacchetto in tempo reale (o quasi) esaminando l'attività di protocolli a livello di rete, trasporto e/o applicazione nel tentativo di rilevare modelli di intrusione. I NIDS sono generalmente inclusi nell'infrastruttura di sicurezza perimetrale di un'organizzazione; infatti essi solitamente si concentrano sul monitoraggio di tentativi di intrusione esterna. Una tipica struttura NIDS include una serie di sensori, uno o più server per le funzioni di gestione e una o più console per l'interfaccia utente.

I sensori possono essere implementati in due modi:

- ***Sensore Inline***: viene inserito in un segmento di rete in modo che il traffico debba passare al suo interno;

- ***Sensore Passivo***: monitora una copia del traffico di rete, mentre il traffico effettivo non passa attraverso questo dispositivo.

Consideriamo ad esempio un'organizzazione con più siti, ognuno dei quali dispone di una o più LAN interconnesse. Per una strategia NIDS completa si necessita di più sensori.



1. ***Sensore appena dietro il firewall esterno***: questa posizione vede gli attacchi provenienti dall'esterno e che riescono a superare il firewall, evidenziandone così le vulnerabilità;
2. ***Sensore tra firewall esterno e Internet***: questa posizione serve a monitorare tutto il traffico di rete che tenta di entrare, documentando in questo modo il numero e il tipo di attacchi che prendono di mira la rete;
3. ***Sensore prima di server interni***: questa posizione monitora una grande quantità di traffico di rete, dal quale può rilevare anche attività non autorizzate da parte di utenti interni. In questo modo tale sensore è sensibile sia ad attacchi esterni che a quelli interni;
4. ***Sensore prima di workstations***: questa posizione rileva attacchi contro sistemi e risorse critiche.

Gli ultimi due sensori possono essere sintonizzati su protocolli e tipi di attacchi specifici per ridurre il carico di elaborazione.

Come per gli HIDS, anche i NIDS utilizzano sia rilevamento di anomalie che rilevamento tramite signature ed euristiche. Per quanto riguarda il primo esso è particolarmente sensibile ad attacchi DoS, attacchi di scansione e worms; il secondo invece rintraccia ricognizioni e attacchi a livello di applicazione, di trasporto e di rete, oltre a violazioni delle policy.

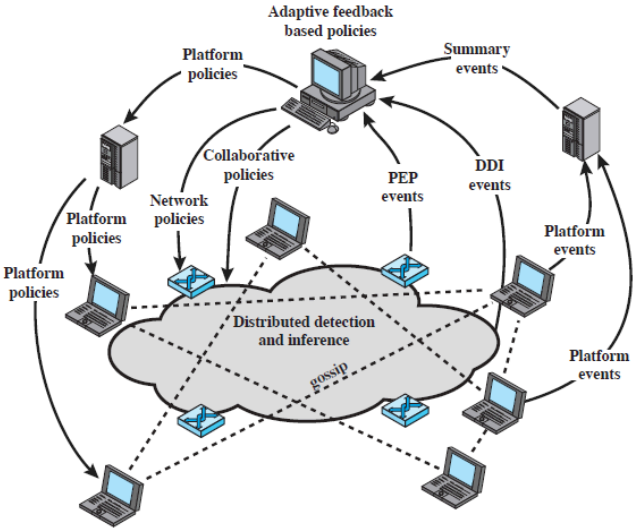
Un sottoinsieme di rilevamento di anomalie è rappresentato dallo ***Stateful Protocol Analysis*** (**SPA**). Esso, a differenza delle confronta il traffico di rete che sta avvenendo con modelli predefiniti di traffico benigno, forniti da un'entità esterna. Questo lo distingue dalle tecniche di rilevamento di anomalie basate su profili di traffico dell'organizzazione ma impone anche la necessità di un elevato uso di risorse. Lo SPA capisce inoltre lo stato (*ecco perché stateful*) delle comunicazioni di rete per essere sicuro che esse procedano come ci si aspetta.

Tipicamente, le informazioni raccolte dai sensori dei NIDS sono:

- Timestamp;
- ID di sessioni e connessioni;
- Tipo di evento o avviso;
- Protocolli di rete, trasporto e applicazione;
- Indirizzi IP di origine e destinazione;
- Porte TCP o UDP;
- Numero di bytes trasmessi;
- Informazioni relative allo stato della connessione.

<u>Intrusion detection ibridi</u>

Negli ultimi anni gli IDS hanno sviluppato schemi per combinare fonti di informazioni ibridi (HIDS e NIDS). Il problema principale però è che risulta difficile aggiornare gli schemi con sufficiente rapidità per far fronte ad attacchi che si sviluppano sempre più velocemente. Un modo per contrastare tali attacchi è sviluppare sistemi cooperativi in grado di riconoscere attacchi da indizi più sottili. In questo approccio i rilevatori di anomalie locali cercano prove di attività insolite. Per evitare però falsi positivi o falsi negativi utilizzano il protocollo peer-to-peer ***gossip***, il quale consiste nell'informare le altre macchine. Queste ultime, nel caso ricevano più di un certo numero di avvisi allora presumono che ci sia un attacco e rispondono.



Tramite questo approccio, il sistema centrale analizza tre tipi di eventi:

- ***Eventi di riepilogo***: gli eventi provenienti da varie origini vengono raccolti da punti di raccolta intermediari (come firewall e IDS). Questi eventi vengono riepilogati e consegnati al sistema centrale;
- ***Eventi DDI*** [***Distributed Detection and Interface***]: sono eventi generati quando il traffico di gossip consente di concludere che è in corso un attacco;
- ***Eventi PEP*** [***Policy Enforcement Points***]: risiedono su piattaforme autodifese. Questi sistemi mettono in correlazione informazioni distribuite e azioni dei singoli dispositivi per rilevare intrusioni che potrebbero non essere evidenti a livello host.

Per favorire lo sviluppo di IDS distribuiti sono necessari degli standard. Questo problema è al centro dell'attenzione della ***Internet Engineering Task Force*** (**IETF**), il cui scopo è quello di definire formati di dati e procedure di condivisione di informazioni di interesse al fine di rilevare e rispondere ad intrusioni. A questo proposito sono state emesse le seguenti RFC (documenti che identificano standard per Internet):

- ***Requisiti per lo scambio di messaggi di intrusion detection***;
- ***Formato di scambio dei messaggi di intrusion detection***;
- ***Intrusion Detection Exchange Protocol***: un protocollo a livello applicativo per lo scambio di dati tra entità di intrusion detection.

Honeypot

Gli **honeypot** sono sistemi esca progettati per attirare potenziali aggressori lontano da sistemi critici. Il loro scopo è molteplice:

- Impedire ad un utente malintenzionato di accedere a sistemi critici;
- Raccogliere informazioni sull'attaccante;
- Incoraggiare l'attaccante a rimanere nel sistema per consentire una buona risposta;

Questi sistemi sono pieni di informazioni progettate per apparire preziose, ma a cui in realtà un utente legittimo non potrebbe neanche accedere. Proprio per questo motivo, qualsiasi tentativo di accesso ad un honeypot può essere solo causato da un'indagine, una scansione o un attacco.
Gli honeypot si classificano in:

- ***Honeypot a bassa interazione***: consiste in un pacchetto software che emula particolari servizi in maniera realistica, ma non esegue mai un servizio completo;
- ***Honeypot ad alta interazione***: è un sistema reale, con sistema operativo, servizi e applicazioni che vengono distribuiti dove possono essere accessibili agli aggressori. Ovviamente è più attrattivo rispetto all'altro tipo, ma richiede anche molte più risorse.

Snort

Snort è un IDS open source, altamente configurabile e portabile basato su host o su rete. Presenta le seguenti caratteristiche:

- Facilmente implementabile sulla maggior parte dei nodi di una rete;
- Molto efficiente in termini di memoria e tempo del processore;
- Facilmente configurabile dagli amministratori di sistema che hanno bisogno di una determinata soluzione di sistema in breve tempo;
- Possibilità di eseguire in tempo reale acquisizione di pacchetti, analisi dei protocolli e confronto con modelli di attacco noti ed euristiche.

Un'installazione di Snort presenta quattro componenti logici:

- ***Decoder di pacchetti***: elabora ciascun pacchetto per estrarre le informazioni utili (ad es. intestazione dei protocolli);
- ***Motore di rilevamento***: esegue il lavoro vero e proprio di intrusion detection. Analizza le informazioni decodificate tramite le regole specificate;
- ***Logger***: per ogni pacchetto che avvera una regola, il logger lo memorizza in un formato leggibile dall'uomo;
- ***Alerter***: per ogni pacchetto rilevato viene inviato un alert.

Le regole di Snort sono definite in modo semplice e flessibile, in modo tale da essere abbastanza potenti da rilevare un'ampia varietà di traffico ostile e sospetto. L'intestazione presenta:

- ***Azione***: cosa fare quando un pacchetto soddisfa la regola;
- ***Protocollo***: individua quali pacchetti devono essere analizzati in base al loro protocollo;
- ***Indirizzo IP di origine***: se presente, indica quali indirizzi IP devono essere analizzati;
- ***Porta di origine***: se presente, indica quali porte di origine analizzare;
- ***Direzione*** [*unidirezionale* o *bidirezionale*]: indica quali lati delle trasmissioni analizzare;
- ***Indirizzo IP di destinazione***: se presente, indica quali indirizzi IP devono essere analizzati;
- ***Porta di destinazione***: se presente, indica quali porte di destinazione analizzare;

In aggiunta all'intestazione, possono essere aggiunte delle opzioni alla regola:

- ***Metadati***: forniscono informazioni sulla regola;
- ***Payload***: cerca dati all'interno del payload del pacchetto;
- ***Non payload***: cerca dati tra quelli non utili;
- ***Post-rilevamento***: trigger che si verificano dopo che una regola trova corrispondenza.

CAPITOLO 9 - FIREWALL E INTRUSION PREVENTION SYSTEMS

Firewall

La connessione Internet è diventata un must per le organizzazioni, ma questo porta a dei pericoli. Per questo motivo c'è bisogno di una protezione per le reti LAN a livello perimetrale, in modo da isolare il sistema interno dalla rete esterna. Il **firewall** è quindi progettato per i seguenti obiettivi:

- Tutto il traffico, dall'esterno all'interno e viceversa, deve passare dal firewall;
- Solo il traffico autorizzato secondo una politica di sicurezza definita localmente può passare attraverso il firewall;
- Il firewall è immune da penetrazioni.

Una componente critica nella pianificazione e implementazione di un firewall è la definizione di una ***politica di accesso*** adeguata. Essa deve includere le caratteristiche del traffico autorizzato ad entrare dal firewall, incluso range di indirizzi, protocolli, applicazioni e tipi di contenuto. Questa politica dovrebbe essere sviluppata in base al traffico che l'organizzazione deve supportare.

Una politica di accesso al firewall deve avere le seguenti caratteristiche:

- ***Indirizzo IP e valori del protocollo***: controlla l'accesso in base agli indirizzi di origine e destinazione, al numero di porte di origine e destinazione, alla direzione del flusso, ecc…. Questo tipo di filtro viene usato per limitare l'accesso a servizi specifici;
- ***Protocollo applicativo***: controlla l'accesso in base a dati del protocollo applicativo. Questo tipo di filtro viene usato da gateways a livello di applicazione per consentire solo protocolli specifici;
- ***Identità utente***: controlla l'accesso in base all'identità dell'utente;
- ***Attività di rete***: controlla l'accesso in base a considerazioni come ora e tipo di richiesta. Questo tipo di filtro intende rilevare tentativi di scansione o attività illecite.

Riassumendo le caratteristiche di un firewall:

- Un firewall definisce un singolo punto di strozzatura dove si tenta di bloccare utenti non autorizzati. L'utilizzo di un solo punto di verifica semplifica la gestione della sicurezza;
- Un firewall fornisce una posizione per monitorare gli eventi relativi alla sicurezza;
- Un firewall è una piattaforma che può presentare anche funzionalità non correlate con la sicurezza;
- Un firewall può fungere da piattaforma per IPSec, implementando reti private virtuali.

I firewall presentano anche dei limiti:

- Possono presentarsi attacchi che aggirino il firewall;
- Un firewall potrebbe non proteggere completamente dalle minacce interne;
- È possibile che connessioni wireless non siano ben gestite;
- Dispositivi mobili potrebbero essere infettati all'esterno e poi collegati e utilizzati internamente (aggirando in questo modo il firewall).

Esistono quattro tipi di firewall:

- ***Firewall con filtraggio dei pacchetti***;
- ***Firewall con ispezione stateful***;
- ***Gateway a livello di applicazione***;
- ***Gateway a livello di circuito***.

Firewall con filtraggio dei pacchetti

Un **firewall con filtraggio dei pacchetti** applica una serie di regole a ciascun pacchetto IP in entrata e in uscita per determinare se bisogna inoltrarlo o bloccarlo. Le regole di filtraggio di basano su:

- Indirizzo IP di origine;
- Indirizzo IP di destinazione;
- Indirizzo di origine e destinazione a livello di trasporto;
- Protocollo di trasporto;
- Interfaccia del firewall da cui proviene il pacchetto.

Nel caso il pacchetto non abbia corrispondenza con nessuna regola viene eseguita un'azione predefinita. Può quindi succedere che ciò che non è espressamente consentito venga vietato o viceversa.

Firewall di questo tipo sono molto semplici e veloci e quindi tipicamente trasparenti agli utenti. D'altro canto però presentano delle controindicazioni:

- Non riescono a prevenire attacchi a vulnerabilità a livelli superiori rispetto a quello di trasporto;
- Presentano funzionalità di logging limitate;
- Non supportano schemi di autenticazione utente avanzati;
- Tipicamente vulnerabili ad attacchi che sfruttano problemi di protocolli TCP/IP;
- Configurazioni improprie potrebbero portare a violazioni di sicurezza.

Firewall con ispezione stateful

Un **firewall con ispezione stateful** prende decisioni sulla base del singolo pacchetto e non prendendo in considerazione nessun contesto di livello superiore. Esso rafforza le regole per il traffico TCP creando una directory di connessioni TCP in uscita. Per ogni connessione attualmente stabilita esiste una voce. Il filtro dei pacchetti con stato ora consentirà il traffico in entrata verso porte con numero elevato solo per quei pacchetti che corrispondono al profilo di una delle voci in questa directory.

Questo tipo di firewall registra anche le informazioni sulle connessioni TCP, tenendo traccia del numero di sequenza TCP per prevenire attacchi che dipendano da essi e ispezionando i dati per alcuni protocolli al fine di identificare e tenere traccia di connessioni correlate.

Gateway a livello di applicazione

Un **gateway a livello di applicazione** (o ***proxy di applicazione***) agisce come inoltro del traffico a livello applicazione. L'utente contatta tramite TCP/IP il gateway, il quale richiede un'autenticazione all'utente stesso. Il gateway a questo punto contatta l'applicazione sull'host remoto e inoltra i segmenti TCP contenenti i dati. Inoltre il gateway potrebbe essere configurato per supportare solo funzionalità specifiche di un'applicazione, negando tutte le altre richieste. Questo tipo di firewall sembra essere più sicuro dei filtri su pacchetti, ma necessita di codice proxy per ogni applicazione.

Gateway a livello di circuito

Un **gateway a livello di circuito**, come per quelli di applicazione, non consente connessioni TCP dirette end-to-end, ma ne stabilisce due intermedie. In genere il gateway inoltra i segmenti TCP da una connessione all'altra senza esaminarne il contenuto. La funzione di sicurezza consiste invece nel determinare quali connessioni saranno consentite. Questo tipo di gateway però è consentito solo quando ci si fida degli utenti interni. Il gateway potrebbe anche essere configurato per supportare funzioni di livello applicazione in entrata e funzioni di livello di circuito per le connessioni in uscita. In questo modo può sostenere il sovraccarico di elaborazione derivante dall'esame dei dati a livello applicazione.

Basi per firewall

Un **bastion host** è un sistema identificato dall'amministratore del firewall come un punto di forza cruciale per la sicurezza della rete. Il bastion host funge da piattaforma per un gateway a livello applicazione o di circuito. Le caratteristiche comuni ai bastion host sono:

- Eseguono una versione sicura del proprio sistema operativo, rendendolo un sistema rafforzato;
- Potrebbero richiedere un'ulteriore autenticazione per accedere ai servizi proxy;
- Prevedono solo servizi che l'amministratore di rete considera necessari;
- Supportano solo un sottoinsieme di funzionalità e di host specifici;
- Ciascun modulo proxy è un pacchetto software molto piccolo e semplice, progettato specificatamente per la sicurezza di rete;
- Ciascun proxy è indipendente dagli altri proxy del bastion host e non ha privilegi, se non quello di lettura del proprio file di configurazione.

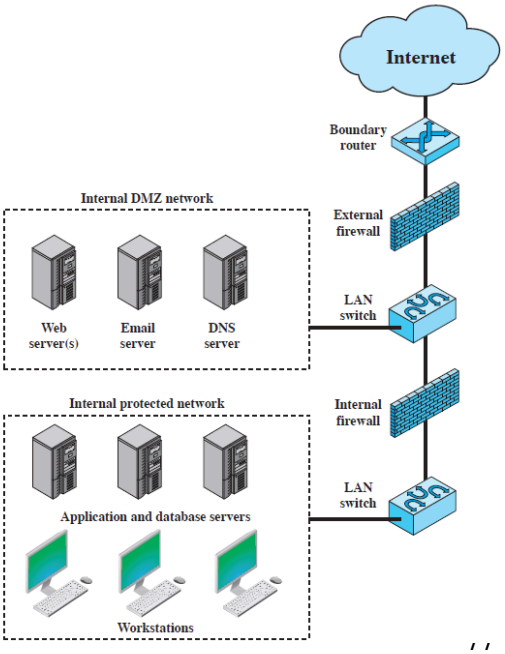
Un **firewall basato su host** è un modulo software utilizzato per proteggere un singolo host. Tali moduli sono disponibili in molti sistemi operativi o possono essere forniti come pacchetti aggiuntivi. Una locazione tipica per questi firewall è un server. Ciò presenta i seguenti vantaggi:

- Le regole di filtraggio sono adattate allo specifico host.
- La protezione è fornita indipendentemente dalla topologia. Quindi, sia interno che esterno, tutto il traffico deve passare attraverso il firewall;
- Può essere usato in aggiunta ad altri firewall.

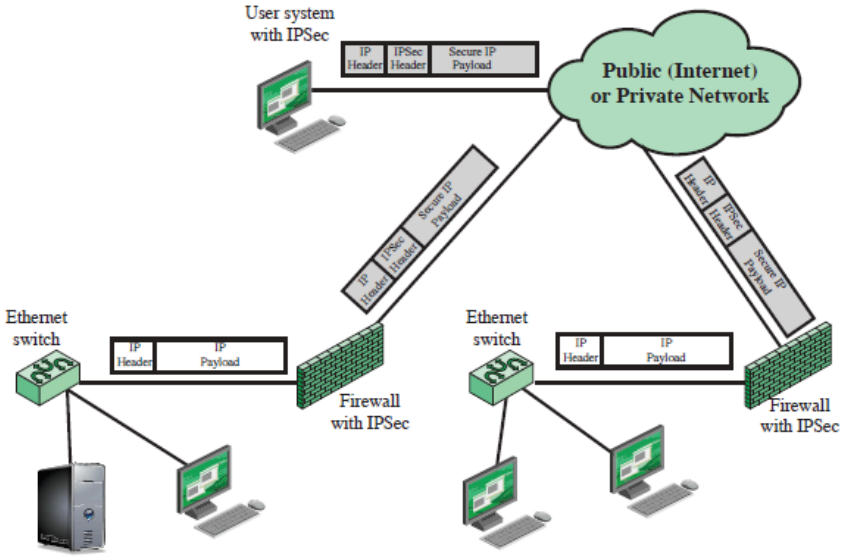
Un **firewall personale** è un modulo software che controlla il traffico tra un pc e Internet o una rete aziendale. Esso può essere ospitato su un router, un modem o un'altra interfaccia Internet. I firewall personali sono in genere molto meno complessi rispetto a quelli basati su server o autonomi. Il loro ruolo principale è quello di negare l'accesso remoto non autorizzato al pc, ma può anche monitorare l'attività in uscita cercando di rilevare e bloccare worm o altri malware.

Posizione e configurazione dei firewall

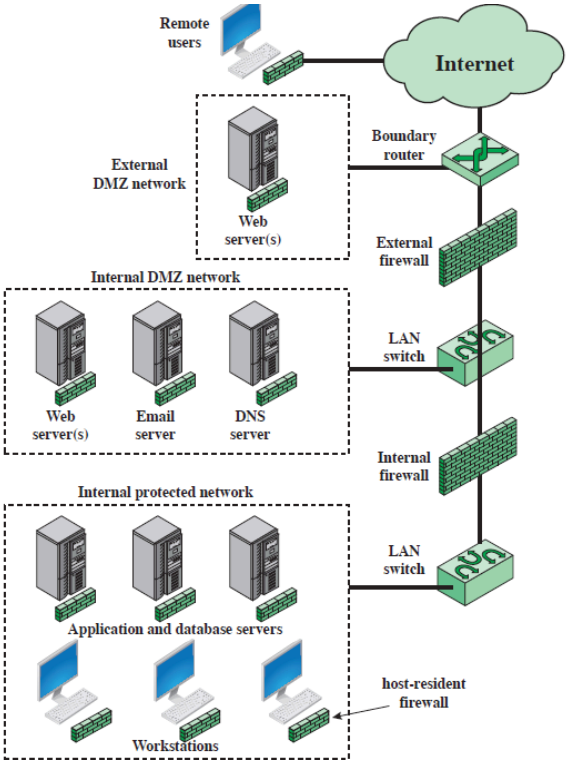
Una configurazione comune include un segmento di rete aggiuntivo tra un firewall interno ed uno esterno, chiamato **zona demilitarizzata** (***DMZ***). Tipicamente i sistemi nella DMZ favoriscono la connettività esterna (ad es. e-mail server). Il firewall esterno quindi fornisce una misura di controllo di base, che permetta il funzionamento dei sistemi suddetti. Il firewall interno invece aggiunge capacità di filtraggio più rigorose per proteggere le workstation da attacchi esterni e i sistemi della DMZ da attacchi interni. È comunque possibile usare più firewall interni.



Altra configurazione comune nell'odierno ambiente informatico è la **rete privata virtuale** (**VPN**), la quale è costituita da un insieme di computer che si interconnettono tramite una rete relativamente non sicura, ma utilizzando crittografia e protocolli speciali per garantire la sicurezza. Quando si usa una VPN infatti, tutti i dati inviati e ricevuti vengono criptati da un firewall o da un router. Il meccanismo più comune per questo scopo è il protocollo ***IPSec***. In questo caso, come nell'immagine seguente, il traffico in uscita viene criptato da un firewall e fatto passare tramite Internet verso la sua destinazione. Qui un altro firewall con lo stesso protocollo lo decripta e lo recapita all'host di destinazione.



Una **configurazione firewall distribuita** prevede firewall autonomi che si affiancano a firewall basati su host. Tutti tali dispositivi lavorano sotto un controllo amministrativo centrale. Con questo tipo di configurazione può avere senso avere una DMZ anche interna per quei sistemi che necessitano di una sicurezza maggiore rispetto a quella di base fornita dal firewall esterno.



Topologia dei firewall

Riassumendo si possono distinguere le seguenti alternative di firewall:

- ***Firewall basato su host***: firewall personali o basati su server. Possono essere usati da soli o come parte di una distribuzione;
- ***Router di screening***: singolo router tra reti interne ed esterne con filtraggio dei pacchetti stateless o completo;
- ***Single bastion inline***: singolo firewall tra router interno ed esterno. Tipica configurazione dell'applicazione di firewall per le organizzazione di piccole e medie dimensioni;
- ***Single bastion T***: simile al precedente ma dispone di una terza interfaccia di rete verso una DMZ;
- ***Doppio bastion inline***: doppio firewall, uno interno ed uno esterno (come visto nella figura). Tipica configurazione dell'applicazione di firewall per grandi aziende e organizzazioni governative;
- ***Doppio bastion T***: simile al precedente ma la DMZ si trova in un segmento di rete separato dal firewall interno. Essa infatti è coperta da un altro firewall (vedi figura);
- ***Firewall distribuito***: come descritto in precedenza e mostrato nell'ultima figura.

Intrusion Prevention System (IPS)

Un **sistema di prevenzione delle intrusioni** (**IPS**), anche noto come ***sistema di rilevazione e prevenzione delle intrusioni*** (***IDPS***), è un'estensione di un IDS, che include la capacità di bloccare o prevenire attività dannose rilevate. Anch'esso può basarsi su host, sulla rete o distribuito

Un IPS, per il rilevamento di comportamenti non legittimi, può utilizzare l'anomaly detection o le signatures e le euristiche.

Un **IPS basato su host** (**HIPS**) può fronteggiare i seguenti tipi di comportamento dannoso:

- ***Modifica delle risorse di sistema***;
- ***Escalation dei privilegi***;
- ***Buffer overflow***;
- ***Accesso alla rubrica dei contatti***;
- ***Attraversamento delle directory*** (ad esempio l'hacker accede a file al di fuori di quelli che vengono serviti dall'applicazione web).

Alcuni pacchetti HIPS sono progettati per proteggere specifici tipi di server, come server web e db.

Gli HIPS possono utilizzare, oltre ad anomaliy e signatures detection, anche l'approccio sandbox, il quale prevede la quarantena del codice in un'area di sistema isolata, in modo tale da monitorarla. I sandbox sono particolarmente adatti al codice mobile, come applet Java.

Gli HIPS tipicamente offrono protezione riguardo a:

- System calls;
- Accesso al file system;
- Impostazioni del registro di sistema;
- Host I/O

Si potrebbe pensare che un IPS abbastanza complesso possa determinare una sicurezza abbastanza alta per il sistema. Un approccio più prudente però consiste comunque nell'utilizzare l'HIPS come parte di una difesa più strutturata, la quale prevede anche dispositivi a livello di rete.

Un **IPS basato su rete** (**NIPS**) è essenzialmente un NIDS con autorità di modificare o scartare pacchetti e interrompere connessioni TCP. I NIPS utilizzano i seguenti metodi per identificare pacchetti dannosi:

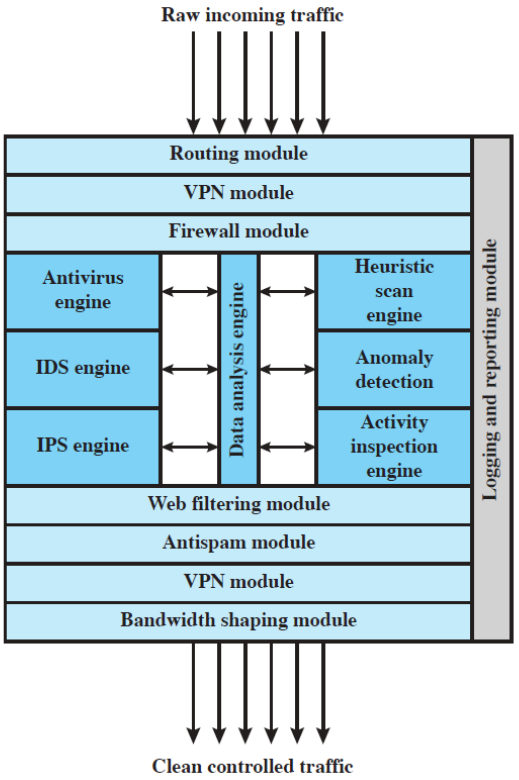
- ***Pattern matching***: analizza i pacchetti in entrata cercando corrispondenze con quelli di attacchi noti;
- ***Stateful matching***: esegue la scansione delle firme degli attacchi nel contesto di flusso di traffico invece che del singolo pacchetto;
- ***Anomalia di protocollo***: cerca incongruenze con gli standard RFC;
- ***Anomali di traffico***: rileva attività di traffico insolite, come un'inondazione di pacchetti;
- ***Anomalia statistica***: sviluppa delle statistiche di normale attività di traffico e avverte nel caso il traffico devii da esse.

Il **sistema immunitario digitale** è una difesa completa contro comportamenti dannosi causati da malware. La motivazione di tale sistema risiede nella crescente minaccia da malware, nella crescente velocità della loro propagazione e nella necessità di acquisire una visione globale della situazione. Il successo di tale sistema dipende dalla capacità del sistema di analisi del malware di rilevare ceppi nuovi. ***Snort***, introdotto come IDS open source, presenta una sua versione modificata (**Snort inline**), la quale aggiunge tre nuove funzionalità di prevenzione delle intrusioni:

- ***Drop***: rifiuta un pacchetto in base alle opzioni definite nelle regole;
- ***Reject***: rifiuta un pacchetto e registra il risultato, restituendo un messaggio di errore;
- ***Sdrop***: rifiuta un pacchetto senza registrare nulla.

Snort può quindi modificare i pacchetti senza eliminarli. Questa funzionalità è utile per l'implementazione di honeypot.

L'implementazione di tutti gli approcci analizzati nei capitoli precedenti può fornire ad un'organizzazione una difesa profonda, utilizzando più filtri di sicurezza. Un approccio potrebbe essere quello di introdurre un unico dispositivo che integri queste protezioni, definito ***sistema di gestione unificata delle minacce*** (**UTM**). Un problema significativo però risiede nella decadenza delle prestazioni di tale dispositivo, quali throughput e latenza.



1. Il traffico in entrata viene decriptato se necessario dal ***modulo VPN*** (IPSec);
2. Un ***modulo firewall*** iniziale filtra il traffico, scartando i pacchetti che violano le regole previste dalla policy;
3. Una ***serie di moduli*** elaborano i singoli pacchetti e i flussi di pacchetti a vari livelli di protocollo. Contestualmente un ***motore di analisi dei dati*** è responsabile di tenere traccia dei flussi di pacchetti e di coordinare ***IDS***, ***IPS*** e ***antivirus***;
4. Il ***motore di analisi dei dati*** riassembla i payload multipacchetto per l'analisi del contenuto da parte dei ***moduli di filtraggio antivirus***, ***web*** e ***antispam***;
5. Potrebbe essere necessario criptare nuovamente parte del traffico in entrata per mantenere in sicurezza il flusso all'interno della rete aziendale;
6. Tutte le minacce rilevate vengono segnalate per emettere avvisi e per il report;
7. Il ***modulo di modellazione della larghezza di banda*** può utilizzare diverse priorità e qualità del servizio per ottimizzare le prestazioni e ridurre il problema precedentemente analizzato.

CAPITOLO 10 - BUFFER OVERFLOW

Il **buffer overflow** (o ***buffer overrun***) è la condizione su un'interfaccia in cui è possibile inserire in un buffer, o altra area di conservazione dei dati, più dati rispetto alla capacità di allocazione, sovrascrivendo in questo modo altre informazioni. Gli attaccanti sfruttano quindi tale condizione per mandare in crash un sistema o per inserire del codice malevolo allo scopo di ottenere il controllo del sistema.

Un buffer overflow quindi può verificarsi per un errore di programmazione che permette, in caso di sovraccarico del buffer, di scrivere su locazioni di memoria adiacenti. Il buffer può essere collocato in diverse posizioni, come stack, heap o nella sezione data di un processo.

Le possibili conseguenze di un attacco del genere includono la corruzione dei dati utilizzati da un programma, il trasferimento inatteso del controllo del programma, la violazione di accesso alla memoria e l'esecuzione di codice introdotto dall'attaccante.

Per sfruttare un attacco di buffer overflow, l'attaccante ha bisogno:

- identificare nei programmi vulnerabilità a buffer overflow che possono essere attivate utilizzando dati provenienti dall'esterno e sotto il controllo dell'attaccante;
- comprendere come quel buffer è salvato in memoria e quindi capire il potenziale di corruzione.

L'identificazione dei programmi vulnerabili può essere fatta:

- ispezionando il codice del programma;
- tracciando l'esecuzione del programma mentre elaborano input sovradimensionati;
- usando strumenti che automaticamente riscontrano tali vulnerabilità.

Il problema del buffer overflow è stato affrontato anche nello sviluppo di nuovi linguaggi di programmazione. I più recenti linguaggi ad alto livello, infatti, hanno una nozione molto forte del tipo di variabili e delle operazioni consentite su di esse. Essi non sono soggetti a pericoli di questo tipo, non consentendo di salvare nel buffer più dati di quelli che possa contenere. Per quanto riguarda invece linguaggi come C e derivati, che hanno molte strutture di controllo di alto livello e astrazioni del tipo di dati, essi forniscono la possibilità di usare i puntatori di memoria, mostrando il fianco ad attacchi di buffer overflow.

Uno **stack buffer overflow** (*o **stack smashing***) si verifica quando il buffer attaccato si trova nello stack. I primi attacchi di questo genere creati sfruttavano il funzionamento incontrollato della funzione goes() di C.

Uno stack buffer overflow spesso utilizza il cosiddetto ***stack frame***, cioè una struttura dati in cui una funzione che chiama un'altra funzione salva i propri dati (input, output, argomenti, ecc…) per lasciarle posto in memoria.

Una componente importante di molti attacchi di buffer overflow è il trasferimento dell'esecuzione di codice fornito dall'attaccante e spesso salvato nel buffer in overflow. Tale codice è noto come **shellcode**, dato che tradizionalmente la sua funzione era quella di trasferire il controllo ad un interprete di linea di comando dell'utente (shell). Lo shellcode quindi è essenzialmente del codice macchina e, proprio per questo motivo, è specifico per una particolare architettura e un particolare sistema operativo. In effetti sviluppare uno shellcode non è compito semplice e necessita sia di buone abilità di assembly sia di una conoscenza del sistema attaccato. Recentemente un buon numero di siti e di strumenti sono stati sviluppati per automatizzare questo processo.

Il compito di sviluppare uno shellcode è reso ancora più arduo a causa di alcune restrizioni:

1. Indipendenza dalla posizione: lo shellcode non può contenere indirizzi assoluti dato che l'attaccante ha in anticipo solo un'idea approssimativa della locazione del buffer nello stack, ma non la sua posizione precisa. Questo vuol dire che l'attaccante non sarà in grado di specificare con esattezza dove le istruzioni verranno eseguite;
2. Inutilizzabilità dei valori NULL: l'attaccante di solito sfrutta funzioni per la manipolazione delle stringhe. Proprio per questo motivo il carattere NULL (che indica la terminazione della stringa) può comparire solo alla fine.

Tali restrizioni però possono essere superato utilizzando degli stratagemmi:

1. Si può referenziare la stringa "/bin/sh", che si riferisce alla shell di sistema, tramite un uso alternativo delle istruzioni JUMP e CALL per mettere il suo indirizzo in stack e poi estrarlo. Tale indirizzo permetterà di accedere a tutte le altre costanti (args);
2. Si può ottenere un valore equivalente a NULL facendo lo XOR del valore di un registro con sé stesso.

Per far fronte all'impossibilità di determinare con precisione l'indirizzo iniziale del codice, l'attaccante può sfruttare il fatto che il codice è spesso molto più piccolo rispetto allo spazio disponibile nel buffer. Posizionando il codice vicino alla fine del buffer, l'attaccante può riempire lo spazio precedente con istruzioni NOP, che sono istruzioni di no operation. L'attaccante quindi, per aggirare il problema di non sapere precisamente dove il buffer si trovi in memoria, crea una slitta NOP al cui interno viene posizionato lo shellcode. In questo modo si aumenta lo *spazio di*

atterraggio per il flusso di esecuzione. Il sistema quindi eseguirà tutti i NOP che sono stati inseriti nel buffer senza fare nulla, fino a giungere alla locazione dello shellcode, che viene quindi eseguito.

Programmi **target** di un attacco shellcode potrebbero essere ***sistemi affidabili di utilità***, ***network service daemon*** (sono processi in background che forniscono servizi di rete su un sistema operativo) e ***programmi di libreria comunemente utilizzati***.

Generalmente lo shellcode effettua le seguenti operazioni: quando si collega esegue una shell remota e una shell inversa che si connette all'hacker. Lo shellcode inoltre potrebbe utilizzare exploit locali per rendere il firewall sensibile ad altri attacchi e per terminare l'esecuzione limitata e dare pieno accesso al sistema.

Difesa dai buffer overflow

Trovare e sfruttare un stack buffer overflow non è così difficile e il gran numero di exploit avvenuti negli ultimi due decenni lo dimostra chiaramente. Di conseguenza è necessario difendere i propri sistemi ed è possibile farlo tramite due tipologie di difesa:

- ***Difesa in fase di compilazione***: mirano a rafforzare i programmi per renderli resistenti ad attacchi di questo genere;
- ***Difesa in fase di esecuzione***: mirano a rilevare e interrompere gli attacchi in corso sul sistema.

Per quanto riguarda la **difesa in fase di compilazione**, sono varie le modalità di intervento.

Prima fra tutte è consigliabile utilizzare ***linguaggi di programmazione ad alto livello***, con una forte consistenza dei tipi delle variabili e con compilatori che includono del codice aggiuntivo per i controlli dei limiti del buffer. Tale scelta però presenta anche degli svantaggi, che però con il passare degli anni stanno diventando sempre meno pesanti. Un primo problema deriva dalla necessità di eseguire il codice aggiuntivo in fase di esecuzione. La distanza dal linguaggio macchina e dall'architettura sottostante determina poi la perdita di accesso ad istruzioni e risorse hardware. Si rende quindi necessaria la presenza di una parte di codice scritta in un linguaggio di più basso livello e perciò meno sicuro.

Bisogna utilizzare sicuramente delle ***tecniche di codifica sicure***. È stato notato, ad esempio, che i progettisti di C hanno dato molta più enfasi all'efficienza di spazio e alle performance rispetto alla sicurezza, assumendo che sarebbero stati i programmatori ad averne cura nella scrittura di codice. Così però evidentemente non è stato; sono, infatti, molti i sistemi che sono stati sviluppati tramite codice potenzialmente non sicuro, come Linux, UNIX, ecc… Per rafforzare questi sistemi, il

programmatore deve ispezionare il codice e riscrivere eventuali costrutti di codifica non sicuri in modo sicuro. Un esempio di ciò è il progetto OpenBSD, durante il quale i programmatori hanno controllato il codice esistente, incluso il sistema operativo, le librerie standard e le utilità comuni. Ciò ha portato alla creazione di quello che oggi è ampiamente considerato come uno dei sistemi operativi più sicuri e ampiamente utilizzato.

Non da sottovalutare è poi l'***uso di estensioni di linguaggio e di librerie sicure***. Considerati i problemi che possono verificarsi in C sono state avanzate proposte di potenziamento dei compilatori in modo da inserire controlli automatici di intervallo su riferimenti ad array e puntatori. La gestione della memoria allocata dinamicamente però potrebbe essere problematica dato che le informazioni sulla dimensione non sono disponibili in fase di compilazione. La gestione di ciò quindi richiede un'estensione della semantica di un puntatore per includere informazioni sui limiti e l'uso di routine di libreria per garantire che tali valori siano impostati correttamente. Tuttavia ciò comporta una penalità in termini di prestazioni data soprattutto dalla necessità di ricompilare tutti i programmi e le librerie che richiedono questa funzionalità. Ciò potrebbe inoltre rendere possibili problemi con applicazioni di terze parti. Altro problema comune con il C deriva poi dall'uso di routine di libreria standard non sicure, in particolare quelle per la manipolazione di stringhe. Un approccio per migliorare la sicurezza dei sistemi è quello di sostituirli con varianti più sicure, come Libsafe.

Per salvaguardare lo stack da attacchi di tipo buffer overflow è importante utilizzare dei ***meccanismi di protezione dello stack*** stesso. Un metodo efficace consiste nell'aggiungere codice all'entrata e all'uscita delle funzioni per controllare l'esistenza di eventuali segni di corruzione. ***Stackguard*** è uno dei meccanismi di protezione più conosciuti in questo senso. Esso utilizza un valore canarino casuale che viene aggiunto nello stack all'avvio della funzione. Prima di ritornare dalla funzione si va quindi a controllare tale valore per capire se esso possa essere stato sovrascritto da qualche sorta di attacco buffer overflow. Per fare ciò è necessario che tale valore sia imprevedibile e diverso da sistema a sistema. Altre varianti di protezione sono Stackguard e Return Address Defender (RAD), le quali invece, all'avvio della funzione, salvano il return address in una zona della memoria molto difficile da corrompere. Al termine della funzione quindi viene confrontato tale valore con quello salvato nello stack per scoprire se esso sia stato corrotto.

Per quanto riguarda la **difesa in fase di esecuzione**, a differenza di quelle a tempo di compilazione, non si concentrano sulla ricompilazione dei programmi, ma sulla gestione della memoria.

Un primo approccio è la ***protezione dello spazio degli indirizzi eseguibili***. Molti attacchi di buffer overflow infatti consistono nel copiare del codice macchina nel buffer e poi trasferire l'esecuzione ad esso. Una possibile difesa potrebbe essere quella di bloccare l'esecuzione del codice sullo stack, presupponendo che il codice eseguibile dovrebbe essere trovato altrove nello spazio degli indirizzi del processo. Per fare ciò, tramite il supporto dell'unità di gestione della memoria del processore (***MMU***), si vanno ad etichettare pagine della memoria virtuale come non eseguibili. Tale approccio è considerato uno dei metodi migliori per proteggere i programmi da alcuni attacchi; tuttavia si palesa il problema per il supporto di programmi che necessitano di inserire codice eseguibile nello stack, come ad esempio per i compilatori just-in-time tipo Java Runtime. Per essi vanno implementate delle disposizioni speciali.

Altra idea potrebbe essere quella di ***randomizzare lo spazio degli indirizzi***. Questa tecnica consiste nel modificare l'indirizzo in cui si trova lo stack in modo casuale per ogni processo. La gamma di indirizzi disponibili sui processori moderni è ampia e ciò minimizza l'impatto di questa operazione sulla disponibilità della memoria (lo spreco di alcuni indirizzi infatti non influisce troppo). Discorso analogo può essere fatto con la posizione del buffer dell'heap o delle routine delle librerie standard.

Ulteriore tecnica di difesa a runtime riguarda l'uso di ***pagine di guardia*** tra regioni critiche della memoria. Tali pagine vengono contrassegnate dalla MMU come indirizzi illegali e qualsiasi tentativo di accedervi comporta l'interruzione del processo. Un'ulteriore estensione di questo approccio posiziona pagine di guardia tra stack frame e buffer dell'heap, ma ciò comporta un costo maggiore in termini di tempi di esecuzione, per l'elevato numero di mappature di pagine necessarie.

CAPITOLO 11 - SICUREZZA DEL SOFTWARE

Molte vulnerabilità della sicurezza informatica derivano da pratiche di programmazione inadeguate. L'elenco dei 25 errori software più pericolosi distingue tre categorie:

- Interazione non sicura tra componenti;
- Gestione rischiosa delle risorse;
- Difese porose.

Già la consapevolezza dei difetti di sicurezza e dei problemi derivanti è un passo iniziale fondamentale nella scrittura di programmi più sicuri.

L'elenco dei dieci difetti critici per applicazioni web ne include ben cinque relativi alla poca sicurezza del codice. Essi includono l'assenza di convalida degli input, scripting cross-site, buffer overflow, injection e gestione impropria degli errori.

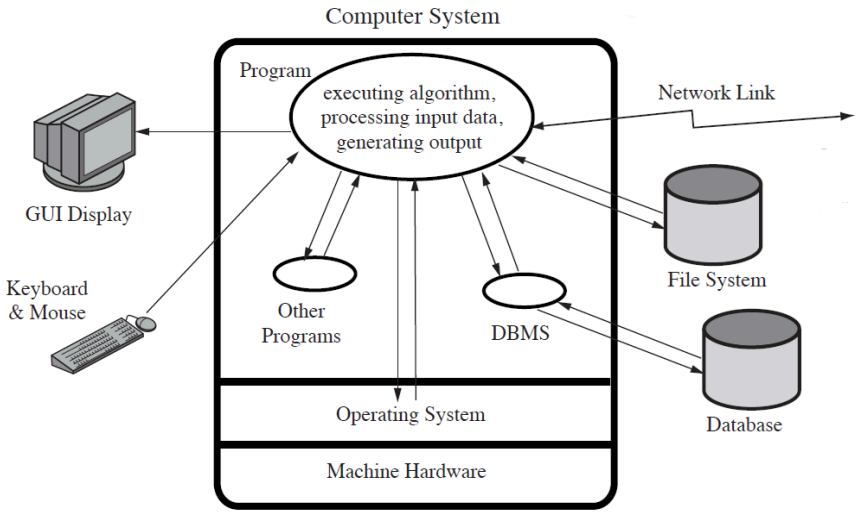
Sono stati presentati una serie di approcci per ridurre il numero di vulnerabilità software:

- Usare metodi migliorati per la costruzione di software;
- Usare tecniche di testing migliori e più efficaci;
- Costruire delle architetture software più resilienti per ridurre l'impatto delle vulnerabilità.

La sicurezza del software è strettamente correlata alla sua qualità e alla sua affidabilità. A queste caratteristiche bisogna affiancare però una buona considerazione di tutti i possibili input e delle relative conseguenze. Molto spesso infatti gli attaccanti sfruttano input molto diversi rispetto a quelli che generalmente vengono usati. Questo quindi rende le tecniche di testing tradizionali (che si concentrano per l'appunto su errori di input generici) inefficaci nell'eliminare bug pericolosi.

La scrittura di codice sicuro richiede attenzione a tutti gli aspetti, dal come e dove viene eseguito un programma al tipo di dati che elabora. Non si può dare per scontato nulla e tutti i potenziali errori devono essere controllati.

Si definisce quindi **programmazione difensiva** (o ***sicura***) il processo di progettazione e implementazione del software in modo che continui a funzionare anche in caso di attacco. Il software scritto in questo modo è in grado di rilevare condizioni errate risultanti da attacchi e continuare l'esecuzione in modo sicuro o di fallire in modo corretto. La regola chiave nella programmazione difensiva è non dare nulla per scontato, ma verificare tutte le ipotesi e gestire ogni possibile stato di errore (ad esempio ipotizzare come verrà eseguito un programma e che tipi di input elaborerà).



Quando si scrive un programma, generalmente ci si concentra sulla risoluzione del problema che il software deve affrontare. Eseguire una programmazione difensiva vuol dire invece tenere in considerazione anche tutti i possibili punti di vulnerabilità del codice. Ai programmatori è quindi richiesto un cambiamento di mentalità rispetto alle pratiche di programmazione tradizionali. Essi devono capire come possono verificarsi guasti e quali sono le misure necessarie per ridurre la possibilità che si verifichino nei loro programmi. E questo deve essere fatto sin dalla fase di progettazione.

Gestione degli input

La **gestione errata degli input** è uno degli errori più comuni nella sicurezza del software. Per input si intende qualsiasi fonte di dati esterna al programma e il cui valore non è predicibile dal programmatore. È importante quindi convalidare tali dati in termini di dimensioni e tipo prima dell'uso.

Per quanto riguarda le ***dimensioni degli input***, spesso i programmatori fanno ipotesi sulla dimensione massima prevista di essi, allocando in questo modo un buffer di una certa grandezza. Nel caso però in cui la dimensione degli input non venga poi controllata si va a cadere nel pericolo di buffer overflow.

I test tradizionali per il controllo dei software potrebbero non identificare un tipo di errore del genere, in quanto gli input usati per testare il codice di solito rispecchiano la gamma di input che i programmatori si aspettano che gli utenti forniscano.

Scrivere codice sicuro significa considerare qualsiasi input come pericoloso.

Nel caso di difesa contro il buffer overflow si potrebbe procedere con un buffer con dimensioni dinamiche o ancora elaborare l'input dividendolo in blocchi delle dimensioni del buffer.

L'altra preoccupazione fondamentale relativa agli input è il loro significato e la loro interpretazione. I dati di input possono essere generalmente classificabili come testuali o binari. L'interpretazione di questi ultimi dipende dalla codifica che solitamente è specificata nell'applicazione. È fondamentale determinare anche il significato dei dati in input man mano che essi vengono letti e interpretati.

Gli input sono sorgenti potenti di attacchi injection. Tale termine si riferisce ad un'ampia varietà di problemi risultanti da una cattiva gestione degli input, la quale può influenzare il flusso di esecuzione del programma. Uno dei più comuni attacchi di injection avviene quando i dati in input vengono passati come parametri ad altri programmi di supporto al sistema. Si verifica più spesso in linguaggi di scripting (come PHP, Python, ecc…).

È necessario garantire che i dati siano conformi a quanto il programmatore ha presupposto, prima che essi vengano utilizzati. I dati in input dovrebbero essere confrontati con ciò che si desidera o con valori pericolosi noti, accettando in questo modo solo sati che sono notevolmente sicuri. In questo modo è più probabile che il programma rimanga sicuro. Questo tipo di confronto viene comunemente eseguito utilizzando le ***espressioni regolari***.

È importante anche considerare che gli input possano essere codificati utilizzando differenti set di caratteri. Tradizionalmente infatti i programmatori presupponevano l'uso di un unico set di carattere (ASCII), il quale però non è in grado di rappresentare caratteri appartenenti in altre lingue, come il cinese e il giapponese. La crescente esigenza di supportare utenti da tutto il mondo ha fatto sì che venissero introdotti altri set di caratteri, come l'Unicode. Esso utilizza 16 bit per carattere. Considerare più codifiche durante la convalida degli input potrebbe fornire campo aperto ad attaccanti, i quali potrebbero sfruttare ad esempio la ridondanza durante la conversione di caratteri. Data la possibilità di codifiche multiple, gli input dovrebbero essere prima portati ad una rappresentazione standard e minima. Tale processo è detto ***canonicalizzazione***.

Altro discorso va affrontato per gli input di tipo numerico. Essi vengono rappresentati da valori di dimensione fissa, ma diversa da sistema a sistema. Ciò vuol dire che un problema comunque persiste in fase di conversione, soprattutto ad esempio se a cambiare è il tipo di rappresentazione (con segno o meno).

Una modalità di testing dei software molto potente per la verifica della gestione degli input è il **fuzzing**, il quale genera input casuali e li dà in pasto al programma. In tal modo si va a capire se il

software gestisce correttamente input del tutto anormali. In realtà gli input potrebbero anche non essere del tutto casuali, ma essere generati tramite templates di problemi conosciuti. In questo ultimo modo però si vanno comunque a fare ipotesi sui possibili input e questo potrebbe essere poco efficace per testare input di attacchi non conosciuti. Sarebbe quindi da utilizzare entrambi gli approcci.

Anche se il fuzzing è un metodo di test concettualmente molto semplice, presenta dei limiti. In generale identifica solo semplici tipi di guasti. Se ad esempio un problema è sviluppabile tramite un insieme ridotto di input, il fuzzing non lo individuerebbe.

Scripting cross-site (XSS)

Gli attacchi di scripting cross-site (XSS) sono attacchi comunemente riscontrati nelle applicazioni web basate su script. Essi rappresentano un'altra ampia classe di vulnerabilità in cui l'input fornito da un utente viene successivamente inviato ad un altro utente.

Gli XSS implicano l'inclusione di codice script nel contenuto HTML che potrebbe dover accedere a dati associati ad altre pagine. I browser di default impongono controlli di sicurezza e limitano l'accesso ai dati di pagine provenienti dallo stesso sito (SOP). Il presupposto è che tutto il contenuto di un sito sia ugualmente attendibile e quindi sia consentito interagire con esso. Gli XSS sfruttano questo presupposto per aggirare i controlli dei browser ed ottenere privilegi di accesso elevati a dati sensibili appartenenti ad altro sito. La variante più comune è la ***vulnerabilità di riflessione XSS***, la quale prevede che un attaccante includa lo script dannoso nei dati forniti a un sito. Se questo contenuto viene successivamente visualizzato da altri utenti senza un controllo sufficiente allora essi eseguiranno lo script.

Codice sicuro

La seconda componente importante nella sicurezza di un software è la fase di elaborazione dei dati in input secondo un algoritmo. Le questioni chiave sono capire:

1. se l'algoritmo risolve correttamente il problema posto;
2. se le istruzioni macchina in cui viene convertito il codice siano equivalenti;
3. se l'interpretazione e la manipolazione dei dati è valida.

Un primo problema derivante dalla questione della corretta implementazione dell'algoritmo potrebbe nascere qualora esso non gestisca correttamente tutti i casi e le varianti del problema dato. Eventuali bug potrebbero quindi essere utilizzati per concedere ad utenti malintenzionati delle funzionalità aggiuntive.

Un altro problema potrebbe invece nascere dalla presenza di codice di debug all'interno della versione di produzione del programma. I programmatori usano queste sezioni per poter carpire informazioni e correggere eventuali bug, ma se tale codice non viene eliminato in fase di produzione, esso potrebbe portare anche altri utenti malintenzionati ad ottenere tali informazioni.

La seconda questione è ampiamente ignorata dai programmatori, i quali presuppongono che l'interprete generi ed esegua effettivamente codice macchina equivalente all'algoritmo scritto. Nel caso in cui però il compilatore viene corrotto per modificare il codice macchina generato in modo da renderlo malevolo, allora prove di tale corruzione potrebbero trovarsi solo nel codice generato.

L'ultima questione riguarda la gestione dei dati. Essi vengono salvati in memoria come gruppi di bit e vengono manipolati a seguito di una interpretazione che dipende dalle istruzioni dell'algoritmo e quindi anche dal linguaggio di programmazione. La difesa migliore in tal senso sarebbe quella di usare linguaggi di programmazione fortemente tipizzati, in modo tale da evitare problemi nell'interpretazione sbagliata di dati.

Legata a questo ultimo problema è anche la gestione della memoria dinamica, presente in programmi che manipolano quantità sconosciute di dati. La memoria in questo caso deve essere allocata e liberata secondo necessità. Se questo non avviene correttamente, si presenta la cosiddetta ***perdita di memoria***, cioè l'heap riduce la propria memoria fino al suo completo esaurimento, portando a problemi DoS.

Altra preoccupazione deriva dalla gestione dell'accesso alla memoria condivisa da più processi o thread (***race condition***). Una cattiva (o peggio mancata) gestione potrebbe portare alla corruzione di dati o anche all'innesco di una situazione di deadlock (stallo) da parte di un utente malintenzionato. La tecnica più antica e generale consiste nell'utilizzare un file di blocco. Un processo quando ottiene l'accesso ad una risorsa ne crea uno, in modo tale che qualsiasi altro processo, in presenza di tale file, se ha bisogno di quella stessa risorsa deve attendere quel il file di blocco sia eliminato.

Tuttavia l'azione del controllo di esistenza del file di blocco e la conseguente creazione devono essere considerati un'operazione atomica, in quanto se il processo venisse bloccato tra queste due operazioni si creerebbero i presupposti per la creazione di più file di blocco per la stessa risorsa. Infatti il primo processo potrebbe aver capito che può creare un file di blocco ma viene bloccato prima di farlo; nel frattempo viene scatenato un altro processo che verifica e crea un altro file di blocco.

I software potrebbero poi utilizzare funzionalità e servizi di altri programmi. È importante, in questo caso, gestire tale interazione. Soprattutto nel caso in cui il programma utilizzato non è del tutto sicuro. In questo caso spetta all'autore del programma più recente di fornire una gestione oculata del programma secondario, in modo da non inibire la sicurezza del proprio. Allo stesso modo in questi casi sono la questione relativa alla gestione dei dati da parte del programma utilizzato e il rilevamento e la gestione di nuovi errori generabili da questa interazione.

Interazione con il sistema operativo

Nella scrittura di software sicuro è fondamentale tenere presente l'interazione di tale codice con il sistema operativo sul quale viene eseguito. Generalmente tali sistemi hanno il concetto di più utenti. Le risorse quindi sono di proprietà di un utente e dispongono di autorizzazioni diverse per gruppi di utenti diversi.

I sistemi prevedono poi le ***variabili ambientali***, ossia delle variabili che influenzano il comportamento dei processi in esecuzione su di essi. Il sistema le include nella memoria del processo quando lo costruisce e possono essere modificate da esso in qualsiasi momento. Rappresentano comunque un altro tipo di input da gestire dato che sono dati esterni al programma.

I primi attacchi che utilizzano variabili ambientali prendevano di mira script di shell eseguiti con privilegi di root. Il loro utilizzo infatti è ad oggi pesantemente scoraggiato dato che è generalmente riconosciuto che scriverne di sicuri e privilegiati è molto difficile. Al massimo si consiglia di modificare solo i gruppi di utenti e non i singoli utenti e di non reimpostare le variabili ambientali critiche. Se poi dovesse presentarsi la necessità di un'applicazione basata su script, la soluzione migliore è utilizzare un programma wrapper per chiamarla, il quale crea un insieme di variabili d'ambiente sicura e persistente.

I programmi compilati potrebbero essere vulnerabili a manipolazioni della variabile ambientale ***PATH***.

Una caratteristica fondamentale per la sicurezza dei software è l'utilizzo del cosiddetto **principio del privilegio minimo**, al fine di evitare un'escalation dei privilegi da parte di un attaccante. Infatti i programmi che rappresentano i principali target di attacchi sono quelli con privilegi di root/admin. Una buona progettazione distingue varie sezioni del codice e assegna ad ognuno di esse i privilegi minimi necessari. In tal modo è fornito un maggior grado di isolamento dei componenti e quindi riduce le possibili conseguenze derivanti da una violazione del programma.

Altra vulnerabilità potrebbe scaturire dal fatto che i programmi in generale non contengono tutto il codice necessario, ma presentano varie ***chiamate di sistema*** e ***funzioni di librerie***. I programmatori in questo senso tendono a pensare al proprio programma come processo isolato, mentre in realtà andrà ad essere eseguito insieme ad altri processi con cui dovrà condividere risorse.

Molti programmi necessitano di archiviare una copia temporanea di dati mentre li elaborano. A questo scopo vengono creati dei ***file temporanei***. La maggior parte dei sistemi operativi offre posizioni note per tali file (come la cartella tmp di UNIX). Per nominare tali file viene usato il PID del processo che lo crea. In questo modo il file risulta unico, ma ciò non impedisce che sia anche prevedibile. Tale vulnerabilità potrebbe essere usata da un attaccante, il quale potrebbe indovinare il nome di un file tmp e usarlo a proprio piacimento. Per evitare un simile problema si dovrebbero usare dei nomi casuali.

Gestione degli output

Ultima componente della sicurezza dei software è la generazione dell'output, il quale deve essere conforme e ben interpretabile. Tali dati potrebbero essere archiviati per uso futuro, trasmessi tramite rete o destinati alla visualizzazione da parte di qualche utente. Come per gli input, possono essere in forma binaria o di testo. Nel primo caso il set di caratteri di interpretazione deve essere specificato. Ogni software deve identificare quale sia il contenuto dell'output consentito e deve filtrare tutti i possibili dati non attendibili per garantire che venga visualizzato solo l'output valido.

CAPITOLO 12 - CRITTOGRAFIA SIMMETRICA

La **crittografia simmetrica** (oppure ***crittografia convenzionale***, ***a chiave segreta*** o ***a chiave singola***) è il primo e di gran lunga il più diffuso tipo di crittografia. Il suo schema presenta cinque ingredienti:

- **Testo in chiaro**: messaggio che funge da input dell'algoritmo di crittografia;
- **Algoritmo di crittografia**: algoritmo che esegue varie trasformazioni sul testo in chiaro;
- **Chiave segreta**: altro input dell'algoritmo, da cui dipendono le trasformazioni effettuate;
- **Testo cifrato**: output dell'algoritmo;
- **Algoritmo di decrittografia**: algoritmo che, prendendo in input chiave e testo crittografato, restituisce il testo in chiaro.

I sistemi crittografici sono generalmente classificabili lungo tre dimensioni indipendenti:

- ***Tipo di operazioni usate per trasformare il testo in chiaro in testo cifrato***: tutti gli algoritmi di crittografia sono basati su due principi generali:
  - ***Sostituzione***: ogni elemento del testo in chiaro è mappato ad un altro elemento;
  - ***Trasposizione***: gli elementi del testo in chiaro vengono riorganizzati.

  L'importante è che non vengano perse informazioni e che quindi le operazioni siano reversibili.

- ***Numero di chiavi usate***: sono distinte per questo motivo la crittografia ***simmetrica*** (a chiave singola) e quella ***asimmetrica*** (a due chiavi o a chiave pubblica);
- ***Modalità di elaborazione del testo in chiaro***: si distinguono in questo senso:
  - ***Cifratura a blocchi***: elabora un blocco di testo in chiaro alla volta e produce un blocco di output;
  - ***Cifratura a flusso***: elabora ogni elemento del testo in chiaro in maniera continua, producendo come output un elemento alla volta.

Si definisce come **crittoanalisi** il processo atto a scoprire il testo in chiaro o la chiave all'interno di un sistema di crittografia.

La strategia utilizzata dal crittoanalista dipende dalla natura dello schema di crittografia e dalle informazioni in suo possesso. Il problema più complicato, ad esempio, si presenta quando si ha solo il testo cifrato. Un approccio in questo caso potrebbe essere quello di forza bruta, il quale consiste nel provare varie chiavi, assumendo che l'attaccante conosca l'algoritmo di crittografia e che sappia più o meno che tipo di testo in chiaro possa dover intercettare. Molto spesso però il crittoanalista dispone di più informazioni ed è quindi più complicato contrastarlo.

Uno schema di crittografia è ***computazionalmente sicuro*** se il testo cifrato generato soddisfa almeno uno dei seguenti criteri:

- Il costo per violare la cifratura supera il valore delle informazioni crittografate;
- Il tempo necessario per decifrare il testo supera la vita utile dell'informazione.

Sfortunatamente è molto difficile stimare l'entità dello sforzo richiesto per crittoanalizzare con successo un testo cifrato, a meno che non si parli di un attacco di forza bruta. In questo caso bisogna provare in media la metà di tutte le chiavi possibili.

Molti algoritmi di crittografia a blocchi simmetrici hanno una struttura descritta per la prima volta da Feistel (IBM) nel 1973. Gli input di tali algoritmi sono un blocco di testo in chiaro di lunghezza 2w e una chiave K . Il blocco del testo in chiaro è diviso in due metà, L0 e R0, che passano attraverso n cicli di elaborazione e poi si combinano per produrre il blocco di testo cifrato. Tutti gli n round hanno la stessa struttura. Alla parte destra viene applicata una funzione di round F, il cui output va in XOR con la parte sinistra. F ha la stessa struttura generale per ogni round, ma è parametrizzata sulla sottochiave K_i. Il risultato di questa operazione nel round i va a costituire R_i , mentre la parte destra viene spostata a sinistra e diventa L_i.

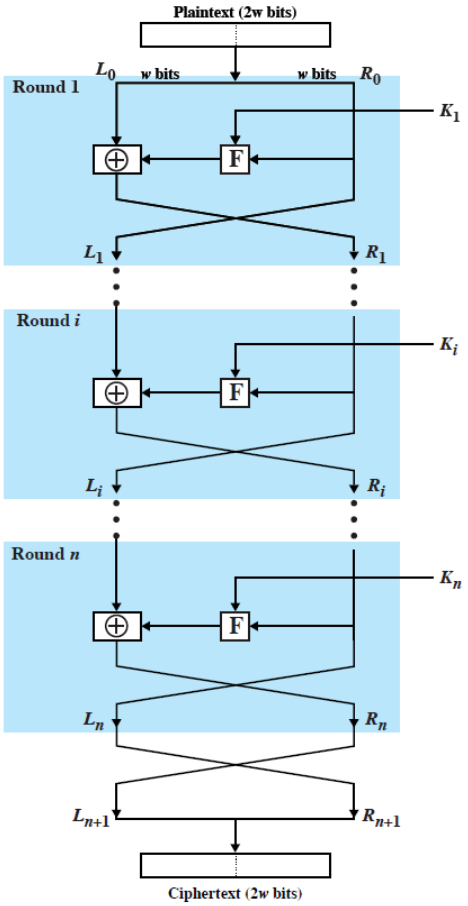


Figure 20.1 Classical Feistel Network

La struttura di Feistel dipende dalla scelta di alcuni parametri. In generale maggiori sono i valori di tali parametri e maggiore è la sicurezza, a scapito però della velocità di crittografia/decrittografia:

- ***Dimensione del blocco***: la dimensione più utilizzata è 128 bit;
- ***Dimensione della chiave***: la dimensione più utilizzata è 128 bit;
- ***Numero di cicli***: la dimensione tipica è di 16 cicli;

Altra variabile in gioco per misurare la sicurezza rispetto alla crittoanalisi sono i seguenti algoritmi, la cui complessità è direttamente proporzionale alla difficoltà della crittoanalisi:

- ***Algoritmo di generazione della sottochiave***;
- ***Funzione di round***;

Infine, ci sono altre due considerazioni nella progettazione di un codice a blocchi simmetrici:

- La ***velocità dei software*** di crittografia/decrittografia è un fattore da non sottovalutare;
- La ***facilità di analisi*** dell'algoritmo non è per forza un male. La conoscenza del solo algoritmo non porta facilmente al successo della crittoanalisi, ma può portare al capire più velocemente le sue vulnerabilità e gestire il sistema in modo tale da limitarne le possibili conseguenze.

Data Encryption Standard (DES)

Lo schema di crittografia più utilizzato è basato sul **Data Encryption Standard** (**DES**). L'algoritmo prende invece il nome di ***Data Encryption Algorithm*** (***DEA***). Esso può essere descritto come segue:

- Il testo in chiaro è diviso in blocchi lunghi 64 bit;
- Ci sono 16 cicli di elaborazione;
- La chiave è lunga 56 bit e viene suddivisa in 16 sottochiavi, una per ogni round;
- La decrittazione è l'operazione inversa della crittazione e può essere fatta semplicemente invertendo l'ordine delle chiavi.

Esiste anche una variante più complessa del DES, il **Triple DES** (**3DES**). Esso utilizza tre chiavi e tre esecuzioni dell'algoritmo DES. Il funzionamento descritto in figura può essere immaginato con le seguenti funzioni, rispettivamente di encoding e decoding:

$$C = E(K_3, D(K_2, E(K_1, P)))$$
$$P = D(K_1, E(K_2, D(K_3, C)))$$

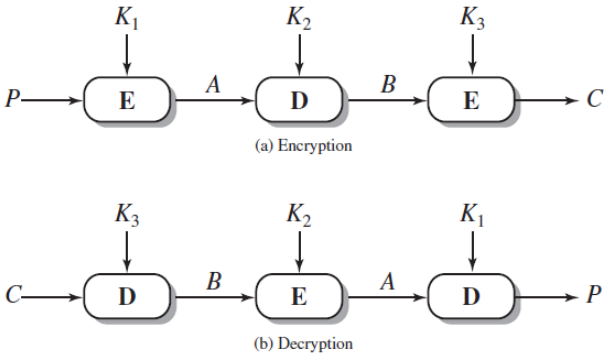


Figure 20.2 **Triple DES**

Il 3DES è un algoritmo fortemente consigliato, in quanto si basa sul DEA e quindi ne assume la stessa resistenza alla crittoanalisi. La maggiore complessità però fa sì che sia ancora più resistente alle poche vulnerabilità del DES.

Advanced Encryption Standard (AES)

L'**Advanced Encryption Standard** (**AES**) è l'algoritmo destinato probabilmente a sostituire DES e 3DES, in quanto più sicuro ed efficiente. AES utilizza una lunghezza di blocco di 128 bit e una chiave di 128/192/256 bit. In base a quest'ultimo parametro, il numero di round varia da 10 a 14.

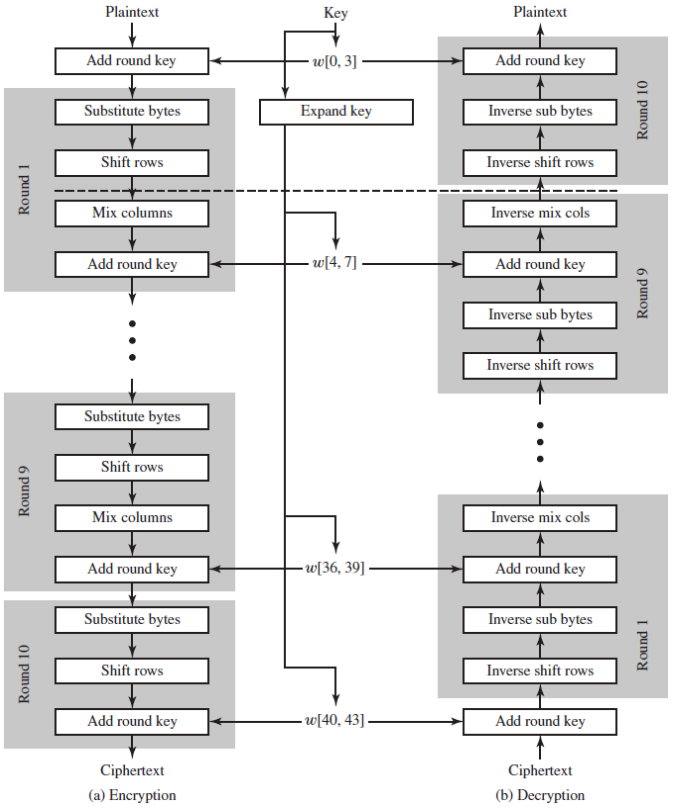
L'input degli algoritmi di crittazione/decrittazione è un singolo blocco da 128 bit, il quale viene rappresentato sotto forma di matrice quadrata 4byte x 4byte. Tale matrice viene copiata nell'**array di stato**, il quale viene modificato ad ogni fase. Al termine della fase finale, l'array è copiato nella matrice di output.

Allo stesso modo la chiave fornita in input viene espansa in un array di 44 parole a 4 byte (nel caso di chiave a 128 bit) e rappresentata sotto forma di matrice quadrata. Quattro parole distinte (in totale 128 bit) fungono da chiave per ogni round. Nel caso di chiave a 128 bit ci sono in realtà 10 round; le rimanenti 4 parole servono per la fase iniziale, prima del primo round. L'ordinamento delle matrici, sia di input che di chiave, avvengono per colonna.

Questo algoritmo non ricade nella struttura Feistel menzionata precedentemente, dato che in questo caso ad ogni round viene processato l'intero blocco di dati (e non una sua metà) usando sostituzioni e permutazioni.

Vengono utilizzate quattro diverse fasi, una di permutazione e tre di sostituzione:

- ***Sostituzione di byte***: si usa una tabella per eseguire una sostituzione byte a byte del blocco;
- ***Shift delle righe***: viene eseguita una semplice permutazione riga per riga;
- ***Mix delle colonne***: viene eseguita una sostituzione che altera ciascun byte di una colonna in funzione di tutti i byte della colonna;
- ***Aggiunta della chiave del round***: viene eseguito un semplice XOR bit a bit del blocco corrente con una porzione della chiave espansa.



La struttura complessiva dell'algoritmo prevede una fase iniziale in cui viene solo aggiunta la chiave per il round successivo, 9 round comprensivi delle quattro operazioni e un round finale con le sole operazioni di sostituzione per produrre l'output.

Solo l'operazione di aggiunta della chiave del round prevede l'utilizzo della chiave. Proprio per questo motivo questa è sia la prima che l'ultima operazione che viene effettuata, dato che tutte le altre sarebbero reversibili senza la conoscenza della chiave. Le altre operazioni sono invece quelle

che modificano i blocchi di dati (l'operazione sulla chiave infatti da sola non produrrebbe alcuna sicurezza).

Ciascuna operazione è facilmente reversibile. Infatti l'operazione sulla chiave si inverte semplicemente usando la stessa chiave di round, dato che $A \oplus A \oplus B = B$. Le altre operazioni invece hanno delle proprie funzioni inverse. In questo senso però c'è una differenza tra AES e DES. Quest'ultimo infatti utilizzava lo stesso algoritmo sia per la crittografia che per la decrittazione, l'AES invece usa due algoritmi diversi (nonostante venga mantenuta la struttura dei round [1 x operazione chiave, 9 x 4 operazioni, 1 x 3 operazioni]).

SubBytes

La ***trasformazione dei byte*** (**SubBytes**) è una semplice ricerca in tabella. AES definisce una matrice 16x16 di bytes, chiamata S-box, la quale contiene le permutazioni di tutti i 256 possibili valori di un byte. La mappatura di un byte avviene semplicemente dividendolo in due gruppi da 4 bit. I primi identificano la riga e i secondi la colonna della cella contenente il valore con cui sostituire il byte.

(a) S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

La sostituzione ***inversa***, chiamata **InvSubBytes**, fa uso della S-box inversa.

La S-box è progettata per resistere ai noti attacchi id crittoanalisi; infatti l'idea principale della S-box è quella di non avere alcuna relazione matematica tra il byte sostituto e quello sostitutivo.

ShiftRows

La ***trasformazione dello spostamento di riga*** (**ShiftRows**) avviene come segue:

- La 1ª riga di stato non viene alterata;
- Nella 2ª riga ogni cella viene spostata di 1 bit a sinistra;
- Nella 3ª riga ogni cella viene spostata di 2 bit a sinistra;
- Nella 4ª riga ogni cella viene spostata di 3 bit a sinistra;

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Per quanto riguarda la trasformazione ***inversa*** (**InvShiftRows**) avviene semplicemente invertendo il verso di shifting.

Una trasformazione siffatta sembrerebbe non produrre chissà che sicurezza, ma in realtà è molto potente. Basti pensare ad esempio che l'operazione sulle colonne va ad eseguire una sostituzione basandosi su tutti i valori delle colonne. Se ognuno di essi viene spostato di un numero diverso di bit allora tale sostituzione avrà risultati imprevedibili, a meno che non si conosca l'array di stato.

MixColumns

La ***trasformazione di mixaggio della colonna*** (**MixColumns**) opera su ciascuna colonna singolarmente. Ogni byte di una colonna viene mappato in un nuovo valore tramite una funzione che prende in input tutti i 4 byte della colonna. La funzione utilizzata fa uso di equazioni sui campi finiti, le quali assicurano un buon mix tra i valori della colonna.

MixColumns e ShiftRows, come sono stati pensati, garantiscono che, dopo soli pochi round, ogni bit di output dipendano da tutti i bit di input.

AddRoundKey

La ***trasformazione dell'aggiunta della chiave del round*** (**AddRoundKey**) esegue uno XOR tra i 128 bit di stato e i 128 bit della chiave di round. L'operazione viene svolta tra 4 byte di una colonna dell'array di stato e 4 byte di una parola della chiave di round.

La trasformazione inversa, come detto, è identica alla trasformazione diretta, dato che l'operazione di XOR è l'inversa di sé stessa. Questo è uno dei motivi per cui tale operazione è ritenuta così utile in crittografia.

Espansione della chiave AES

Un'operazione preliminare per questo algoritmo è l'**espansione della chiave** da 4 parole (16 byte) a 44 parole (156 byte). L'operazione è eseguita copiando le prime 4 parole in un array. Quest'ultimo viene poi riempito usando un complesso algoritmo a campi finiti che prende in input la parola precedente e quella 4 posizioni prima. Cioè w[i] dipende da w[i-1] e da w[i-4].

RC4

RC4 è un algoritmo di cifratura a flusso. A differenza della cifratura a blocco, esso elabora l'input procedendo byte per byte. Un sistema tipico di cifratura a flusso crea, per ogni byte da processare, un byte apparentemente casuale (**keystream**), ma che in realtà dipende dalla chiave.

RC4 utilizza una chiave di dimensione variabile (da 1 a 256 byte) e si basa su permutazioni casuali. Sono necessarie dalle 8 alle 16 operazioni macchina per ogni byte dell'output e ciò rende tale algoritmo molto rapido.

Viene ad esempio utilizzato negli standard SSL/TLS, WEP e WPA. Esso è stato tenuto come segreto commerciale dalla RSA Security per poi essere pubblicato anonimamente nel 1994.

RS4 è un algoritmo molto semplice, che prevede 3 fasi:

```
/* Initialization */
for i = 0 to 255 do
S[i] =i;
T[i] = K[i mod keylen];
```

- ***Fase di inizializzazione***: viene usata la chiave K per inizializzare un vettore di stato S di 256 byte. In ogni momento S contiene una permutazione di tutti i numeri da 0 a 255. S viene poi utilizzato per elaborare un byte selezionando una dei 256 valori. Fatto ciò, le voci di S vengono nuovamente permutate. Inizialmente S contiene i valori in maniera ordinata (S[0]=0, S[1]=1, ecc…). Viene creato inoltre un vettore temporaneo T, il quale viene riempito tramite la chiave K (nel caso in cui K sia più piccolo di 256 bytes allora essa viene ripetuta fino a riempire T);

```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
   j = (j + S[i] + T[i]) mod 256
   Swap (S[i], S[j]);
```

- ***Prima permutazione di S***: viene usato T per produrre la prima permutazione di S, scorrendo quest'ultimo e scambiando via via i valori nella posizione visionata con i valori nella posizione j = j + S[i] + S[j] mod 256. Fatto ciò la chiave non verrà più utilizzata oltre;

```
/* Stream Generation */
i, j = 0;
while (true)
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 256
    k = S[t];
```

- ***Generazione del flusso***: si cicla sugli elementi di S e nuovamente vengono scambiati con altri elementi di S in base alla sua configurazione corrente. Per crittare/decrittare si esegue uno XOR tra k e il prossimo byte del testo in chiaro/testo cifrato.

<u>Modalità operative della cifratura a blocco</u>

Per applicare una cifratura a blocchi in una varietà di applicazioni, sono state definite cinque modalità operative al fine di regolare il modo in cui viene processato un testo in chiaro più lungo rispetto alla lunghezza dei blocchi. È applicato a DES, 3DES, AES e molti altri algoritmi simili.

Il modo più semplice è la modalità **Electronic Codebook** (**ECB**), in cui il testo in chiaro viene gestito un certo numero di bit alla volta usando sempre la stessa chiave. Con la ECB però, se lo

stesso testo viene ripetuto all'interno del messaggio, esso produrrà lo stesso testo cifrato. Per questo motivo tale modalità non sembra essere sicura per messaggi lunghi.

Nella modalità **Cipher Block Chaining** (**CBC**), invece, l'input dell'algoritmo di crittografia è il risultato dello XOR tra il blocco corrente e l'output del blocco precedente. In questo modo si evita il problema evidenziato con la precedente modalità. Data la natura dell'operazione di XOR, per la fase di decrittazione basta semplicemente mettere in XOR l'output della decrittazione del blocco corrente con il testo cifrato del blocco precedente. Dato che per il primo blocco non è possibile eseguire lo XOR con alcun blocco precedente, allora si utilizza un ***vettore di inizializzazione*** (***IV***), che deve essere noto sia al mittente che al destinatario e che viene messo in XOR con il primo blocco di testo in chiaro (crittazione) o con il risultato della decrittazione del primo blocco di testo cifrato (decrittazione). Ovviamente anche l'IV deve essere protetto come la chiave, dato che, se intercettato, potrebbe essere modificato per rendere impossibile ad esempio la decrittazione del primo blocco del messaggio. Nel CBC sorge però un altro problema. Mentre nell'ECB un errore nel blocco di testo cifrato corrisponde ad un errore nel solo rispettivo blocco di testo in chiaro, nel CBC tale errore si propaga ai blocchi successivi. Nello specifico, nel caso di errore nella trasmissione del blocco i-esimo del testo codificato, esso si ripercuoterà soltanto su quel blocco di testo in chiaro e su quello successivo. Nel caso invece di errore nel testo in un blocco del testo in chiaro, questo errore si propagherà anche a tutti i blocchi successivi. Infatti quel blocco influenza il blocco successivo, che poi influenza il blocco dopo, ecc… Però, in quest'ultimo caso, in fase di decifratura la propagazione dell'errore viene risanato. Questo è dato dal fatto che in fase di decifratura del blocco i-esimo vado a fare lo XOR con il blocco cifrato i-1, lo stesso che mi ha creato il problema nel blocco che sto analizzando. Sempre per le proprietà dello XOR vado ad eliminare quindi l'attributo dannoso del blocco i-1 e quindi risano tutto il testo in chiaro. L'unico blocco che rimane in errore è proprio il blocco del testo in chiaro che conteneva l'errore.

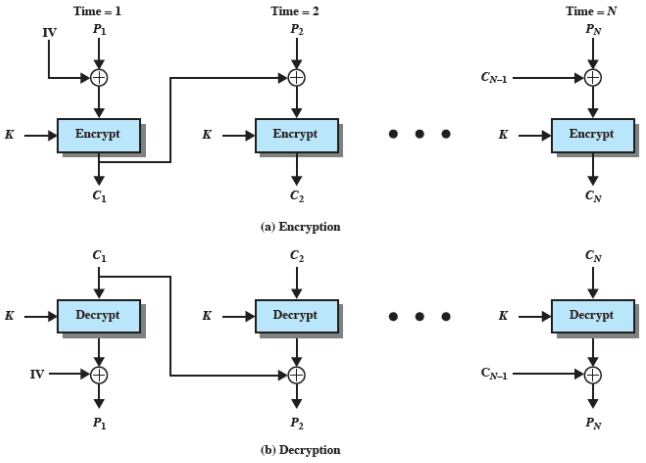
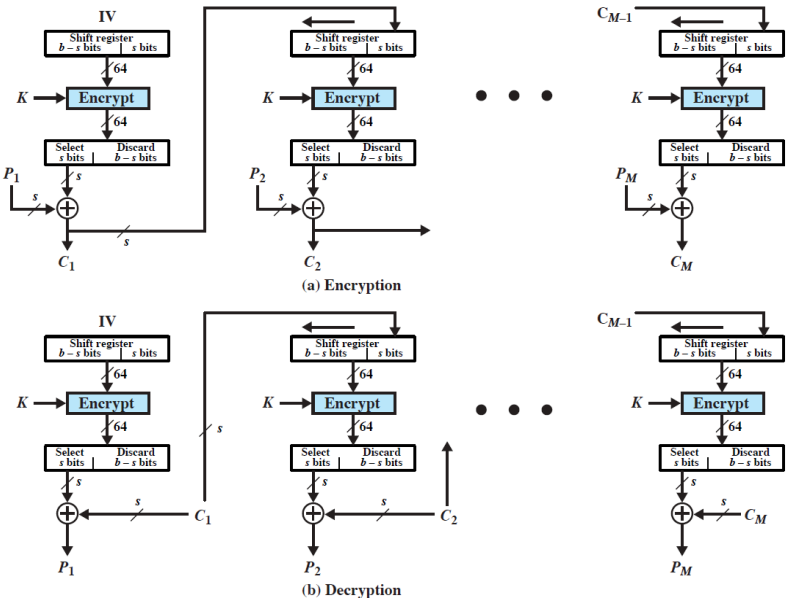


Figure 20.7 Cipher Block Chaining (CBC) Mode

È possibile convertire qualsiasi cifrario a blocchi in un cifrario a flusso utilizzando la modalità **Chiper Feedback** (**CFB**). Tale stratagemma elimina la necessità di riempire il messaggio per rendere la sua lunghezza un multiplo intero della dimensione di un blocco. Una proprietà

desiderabile di una codifica a flusso sarebbe quella di avere la stessa lunghezza sia del testo in chiaro che di quello cifrato.



L'ultima modalità è la modalità **Contatore** (**CTR**), su cui l'interesse è ultimamente crescente ed è applicata alla sicurezza, tra gli altri, di sistemi ATM e IPSec. Viene utilizzato un contatore uguale alla dimensione del blocco di testo in chiaro. Tale valore deve però essere diverso per ogni blocco che viene criptato. In genere, il contatore viene inizializzato su un valore e poi incrementato di 1 per ogni blocco successivo. A livello di crittografia, tale valore è utile perché viene criptato e unito tramite XOR al blocco di testo in chiaro. A livello di decrittografia avviene la stessa operazione. Si ricordi che lo XOR è l'inverso di sè stesso. I vantaggi della modalità CTR sono:

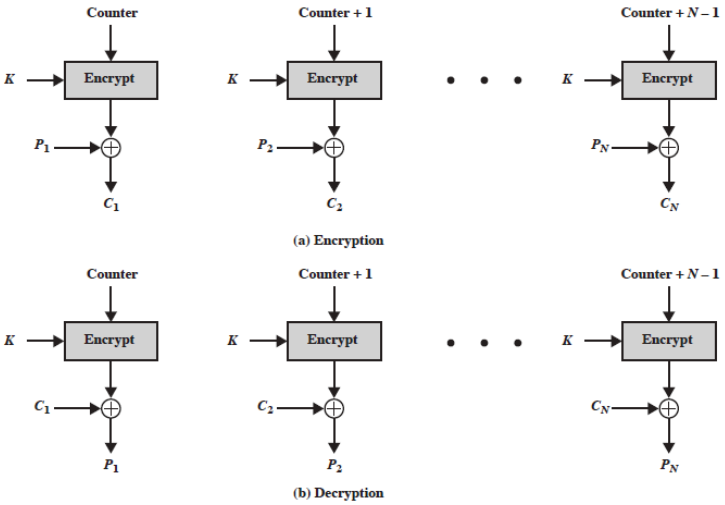


Figure 20.9 Counter (CTR) Mode

- ***Efficacia hardware e software***: questo tipo di crittografia può essere fatta in parallelo su più blocchi. La produttività quindi è limitata solo dal numero di operazioni eseguibili in parallelo;

- ***Preprocessing***: dato che il testo in chiaro non partecipa al processo di crittazione, allora è possibile, se si ha abbastanza memoria, processare preliminarmente la parte cifrata che va ad alimentare gli XOR;
- ***Accesso random***: dato che ogni blocco è processato in maniera autonoma rispetto agli altri, la modalità CTR può permettere di crittare/decrittare i blocchi in qualsiasi ordine o di processare solo alcuni blocchi;
- ***Sicurezza dimostrata***: CTR è sicuro almeno quanto le altre modalità citate;
- ***Semplicità***: CTR richiede solo l'implementazione dell'algoritmo di cifratura. Non vi è decrittazione.

Distribuzione delle chiavi

Affinché la crittografia simmetrica funzioni, le due parti della comunicazione devono condividere la stessa chiave e tale chiave deve essere protetta. Inoltre, sono generalmente auspicabili modifiche frequenti della chiave, per limitare la quantità di dati compromessi nel caso di intercetto della chiave. Quindi la forza di qualsiasi sistema crittografico risiede nella tecnica di distribuzione delle chiavi, la quale può essere ottenuta in diversi modi (siano A e B le due parti coinvolte):

- La chiave potrebbe essere scelta da A e consegnata fisicamente a B;
- Una terza persona potrebbe selezionare la chiave e consegnarla fisicamente ad A e B;
- Se A e B hanno finora usato una chiave, allora una delle parti potrebbe scegliere una nuova chiave e comunicarla all'altra utilizzando la crittografia con la vecchia chiave;
- Se A e B hanno entrambi una connessione criptata con una terza parte, allora essa potrebbe consegnare la nuova chiave tramite tali collegamenti alle due parti.

Le prime due opzioni sono sicuramente le più sicure, ma creerebbero disagi nel caso di crittografia end-to-end. In un sistema distribuito, infatti, sono molti gli attori coinvolti. Anche la terza opzione potrebbe dare adito a problemi. Potrebbe infatti darsi che un attaccante possa intercettare una chiave e con essa eseguire crittoanalisi per scoprire anche tutte le chiavi seguenti, compromettendo di fatti tale collegamento. Per fornire chiavi per la crittografia end-to-end quindi l'opzione 4 sembra essere la migliore. Esso consiste in due tipi di chiavi:

- ***Chiave di sessione***: quando due end system desiderano comunicare, stabiliscono una connesione logica. Per tutta la durata di tale sessione, tutti i dati vengono crittografati con una chiave di sessione monouso, che scade al termine della sessione;
- ***Chiave permanente***: è la chiave utilizzata per lo scambio di chiavi di sessione.

La configurazione è poi composta dai seguenti elementi:

- ***Centro di distribuzione chiavi*** (***KDC***): determina quali sistemi possono comunicare tra loro. Quando viene concessa l'autorizzazione a stabilire una connessione allora il KDC fornisce la chiave di sessione;
- ***Modulo di servizio di sicurezza*** (***SSM***): può consistere in funzionalità a un livello di protocollo, esegue la crittografia end-to-end ed ottiene le chiavi di sessione per conto degli utenti.

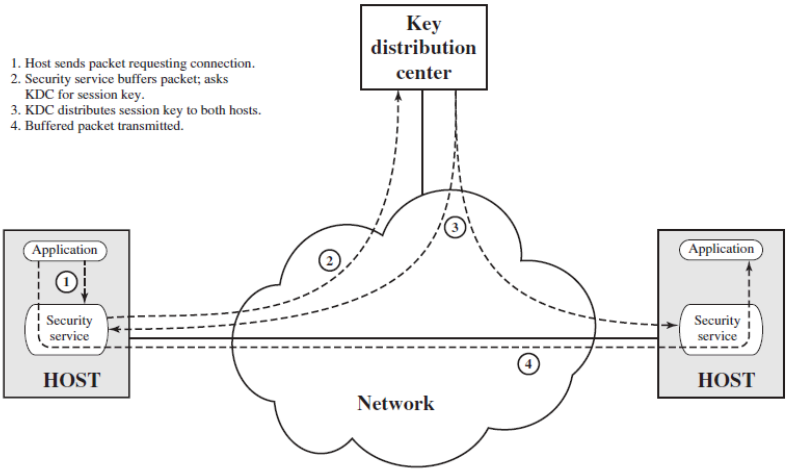


Figure 20.10 **Automatic Key Distribution for Connection-Oriented Protocd**

Quindi, quando un host desidera stabilire una connessione con un altro host, trasmette il pacchetto all'SSM, il quale chiede l'autorizzazione al KDC. Quest'ultima comunicazione avviene su un collegamento criptato con chiave master condivisa. Se il KDC approva la richiesta, genera una chiave di sessione e la manda ad entrambe le parti.

CAPITOLO 12 - FUNZIONI DI HASH E CRITTOGRAFIA ASIMMETRICA

Le **funzioni hash unidirezionali** (o **sicure**) sono meccanismi di sicurezza importanti sia per l'autenticazione dei messaggi che per le firme digitali. I loro requisiti sono stati precedentemente analizzati.

Tutte le funzioni hash operano utilizzando i seguenti principi generali. L'input viene visualizzato come una sequenza di blocchi di n bit, i quali vengono elaborati uno alla volta in modo iterativo per produrre una funzione hash a n bit.

La **funzione hash semplice** prevede soltanto un'operazione di XOR bit a bit di ogni blocco, che può essere espresso come segue:

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

dove:

- C_i: i-esimo bit dell'hash code [i appartiene a [0, n]]
- m: numero di blocchi da n bit;
- b_{ij}: i-esimo bit del j-esimo blocco.

	Bit 1	Bit 2	• • •	Bit n
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
Block m	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

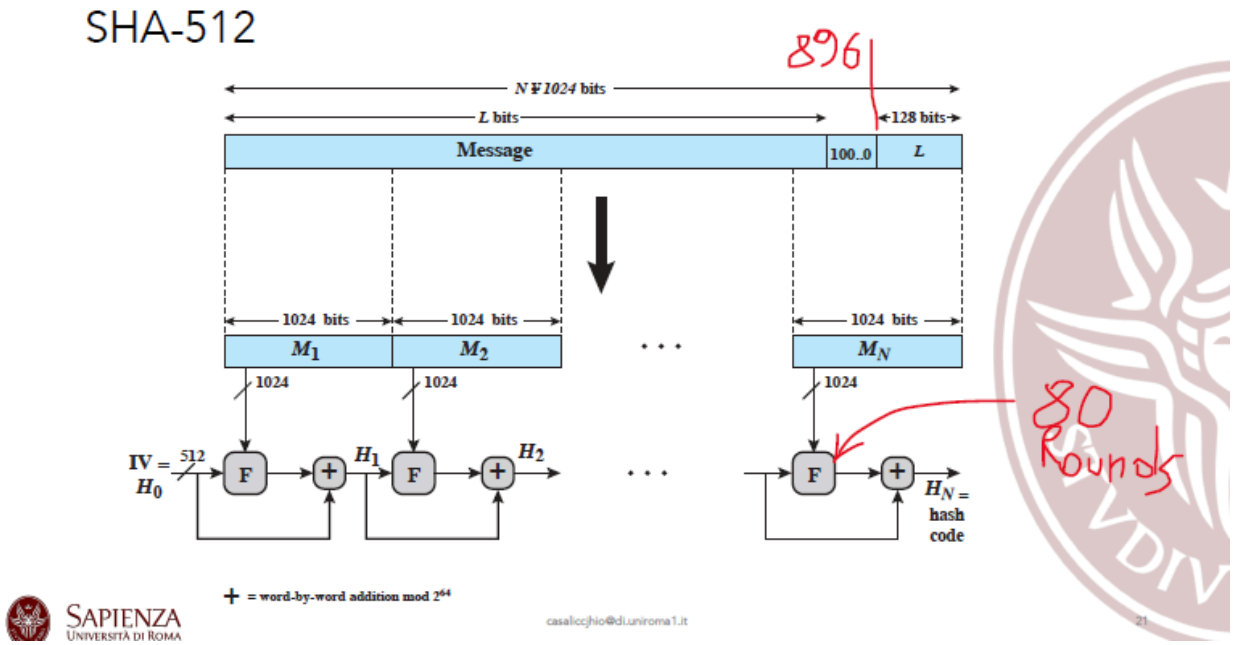
Tale algoritmo quindi produce un bit di parità per ogni posizione da 0 a n, il quale è noto come controllo di ridondanza longitudinale. È ragionevolmente efficace per il controllo di integrità di dati casuali, dato che la probabilità che un errore nei dati non produca cambiamenti sull'hash code è di solo 2^{-n}. La funzione però diventa meno efficace per dati non casuali.

Un modo semplice per migliorare la situazione è eseguire una rotazione di 1 bit sul valore hash dopo l'elaborazione di ciascun blocco. In pratica si impostano inizialmente tutti gli n bit dell'hash code a 0 per poi elaborare ciascuno dei blocchi eseguendo due operazioni: si ruota il valore hash corrente a sinistra di 1 bit e si esegue lo XOR del blocco con tale valore shiftato. Tale algoritmo sarebbe utile nel caso solo nel caso in cui sia cifrato. È stato mostrato però che, nel caso la cifratura avvenga utilizzando la modalità CBC, il codice hash non cambierebbe se l'ordine dei blocchi fosse modificato.

Il **Secure Hash Algorithm** (**SHA**) è un algoritmo di hash più potente, di cui esistono diverse versioni. Di seguito è spiegato il funzionamento dell'algoritmo SHA-512, il quale prende in input

un messaggio di lunghezza massima di 2^{128} bit e produce un digest di 512 bit. L'input viene elaborato in blocchi da 1024 bit seguendo le seguenti fasi:

1. ***Aggiunta di bit di riempimento***: si riempie il messaggio in modo tale che la sua lunghezza sia congruente a 896 in modulo 1024. Il padding è composto da un singolo bit di valore 1 seguito dal numero necessario di 0;
2. ***Aggiunta di lunghezza***: viene aggiunto al messaggio un blocco di 128 bit che indicano la lunghezza originale del messaggio;
3. ***Inizializzazione del buffer hash***: viene usato un buffer a 512 bit per contenere i risultati intermedi e finali delle operazioni. Questi 512 bit sono divisi in otto registri a 64 bit (*a, b, c, d, e, f, g, h*) inizializzati sempre con determinati valori, che corrispondono ai primi 64 bit della parte frazionaria delle radici quadrate dei primi otto numeri primi;
4. ***Elaborazione dei blocchi***: ogni blocco viene processato usando il modulo F, il quale consiste in 80 round;
5. ***Output***: dopo che tutti i blocchi da 1024 bit sono stati processati, viene prodotto un digest lungo 512 bit.



Il cuore delle operazioni quindi sta nel modulo F. Esso consiste in 80 round, ognuno dei quali prende come input il valore dei registri del buffer e li aggiorna. Ogni round t fa uso di un valore W_t, che deriva dal blocco in processo (W_i), e di una costante K_t, che rappresenta i primi 64 bit della parte frazionaria della radice quadrata del t-esimo numero primo. Quest'ultimo valore viene usato per eliminare ogni regolarità che possa presentarsi dei dati di input. Le operazioni di ogni round

consistono in shift circolari, OR, AND, NOT e XOR. L'output dell'ottantesimo round viene sommato dall'input del primo round per produrre i valori dei registri del buffer che verranno trasmessi poi alla fase successiva.

L'algoritmo SHA-512 ha la proprietà che ogni bit dell'hash code è in funzione di ogni bit dell'input. Le ripetizioni delle operazioni all'interno dei moduli F fa sì che sia impossibile che delle regolarità all'interno degli input possano avere lo stesso hash code. Infatti si stima che la difficoltà di trovare due messaggi diversi con lo stesso digest sia nell'ordine di 2^{256} operazioni, mentre la difficoltà di trovare un messaggio tramite un dato digest sia nell'ordine di 2^{512} operazioni.

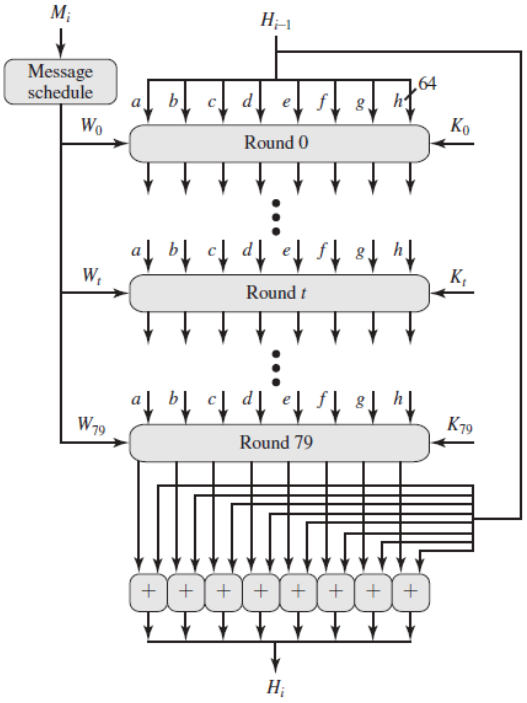


Figure 21.3 **SHA-512 Processing of a Single 1024-Bit Block**

SHA-512 e i suoi simili (famiglia SHA-2) sembrerebbero fornire una sicurezza inattaccabile se non fosse che condividono la stessa struttura e le stesse operazioni dei loro predecessori. Proprio per questo motivo è stato indetto nel 2007 un concorso per lo sviluppo di SHA-3, i cui requisiti devono essere:

- Deve supportare le stesse lunghezze di SHA-2, dato che la sostituzione deve avvenire in drop-in;
- L'algoritmo deve elaborare piccoli blocchi alla volta allo scopo di non richiedere troppo spazio per l'esecuzione delle operazioni.

Hash-based MAC (HMAC)

Negli ultimi anni è cresciuto l'interesse per lo sviluppo di un MAC derivante da codice hash crittografico, come SHA-1 (anche se quest'ultimo non è stato creato a questo scopo, infatti non si basa su una chiave segreta). Le motivazioni sono le seguenti:

- Le funzioni hash crittografiche sono generalmente più veloci rispetto ai tradizionali algoritmi di crittografia (tipo DES);
- Codice di libreria per funzioni hash crittografiche è ampiamente disponibile.

L'approccio più utilizzato per questo tipo di funzione è l'**HMAC**, la cui implementazione è stata resa obbligatoria per la sicurezza IP ed è anche utilizzata per altri protocolli Internet, come il TLS.

HMAC è stato disegnato con i seguenti obiettivi:

- Utilizzare funzioni hash già esistenti senza modifiche;
- Consentire una facile sostituibilità della funzione hash in caso di necessità;
- Preservare le performance originali della funzione hash;
- Usare e gestire chiavi in modo semplice;
- Avere un'analisi crittografica sulla forza del meccanismo di autenticazione, basata su assunzioni rispetto alla funzione hash.

HMAC funziona nel seguente modo:

- Aggiungere zeri alla chiave ***K*** per renderla lunga quanto il numero ***b*** di bit in un blocco. La stringa siffatta prende il nome di K^+;
- XOR tra K^+ e ***ipad*** (costante) per produrre il blocco S_i di b bit;
- Aggiunta del messaggio ***M***, diviso in blocchi da b bit, a destra del blocco S_i;
- Applicazione della funzione hash ***H*** sui blocchi precedentemente ottenuti;
- XOR tra K^+ e ***opad*** (costante) per produrre il blocco S_0 di b bit;
- Aggiunta del risultato del risultato di H, che è un solo blocco di b bit, a destra del blocco S_i;

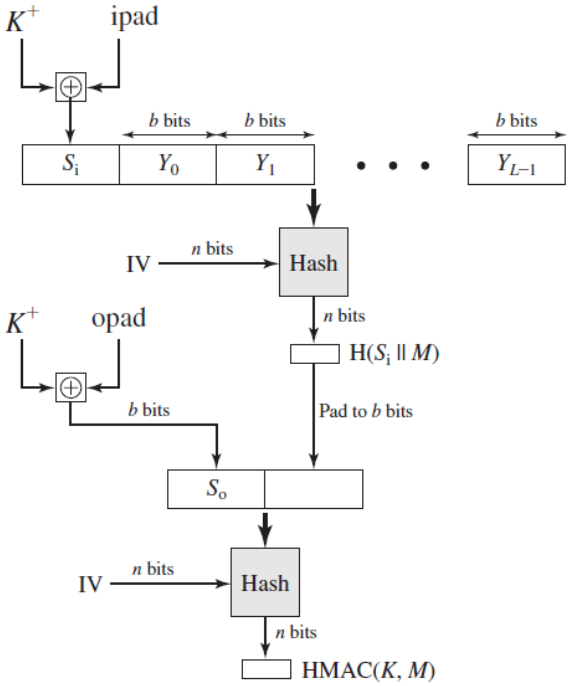


Figure 21.4 **HMAC Structure**

Per la scelta delle costanti ipad e opad, in realtà i due XOR hanno l'effetto di specchiare due parti diverse della chiave. I due blocchi risultanti (S_i e S_0) fanno sì che vengano usate delle chiavi pseudocasuali.

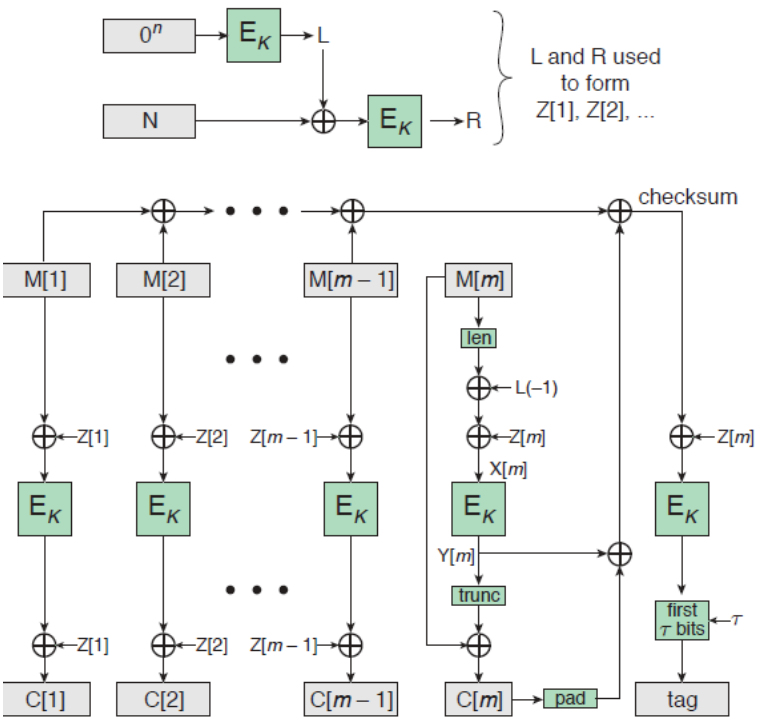
Come detto, la sicurezza di qualsiasi funzione HMAC dipende dalla sicurezza della funzione H. Inoltre la probabilità che l'algoritmo HMAC possa essere attaccato con successo è equivalente ad uno dei seguenti eventi:

- L'attaccante è in grado di calcolare un output della funzione di compressione nonostante il valore del vettore di inizializzazione IV sia random, segreto e sconosciuto;
- L'attaccante trova collisioni nella funzione hash anche quando il vettore di inizializzazione IV è random e segreto.

Authenticated Encryption (AE)

L'algoritmo di **Authenticated Encryption** (**AE**) è un algoritmo che provvede sia alla cifratura del messaggio che alla produzione di un tag finalizzato all'autenticazione.

Tale algoritmo lavora su blocchi, usando una modalità simile all'ECB, ma con qualche variante. Il blocco infatti, prima di essere cifrato, viene combinato in XOR con un valore Z[i], che è diverso da blocco a blocco. In questo modo si evita il problema dell'ECB per cui si potrebbero ripetere parti uguali nel testo cifrato. La produzione del vettore Z richiede il calcolo di due altri vettori, L e R. Il primo è semplicemente la cifratura (tramite prodotto su campi finiti) di un blocco composto da soli zeri, mentre il secondo proviene dalla cifratura del risultato dello XOR tra L e un vettore N. Z viene prodotto partendo da Z[1] = L XOR R e producendo tutti gli altri Z[i] mettendo in XOR l'elemento precedente con un elemento di L.



n **= block length in bits**
N = nonce
len(M[*m*]) = length of M[*m*] represented as an n-bit integer
trunc(Y[*m*]) = deletes least significant bits so that result is same length as M[*m*]
pad = pad with least significant 0 bits to length n
τ = length of authentication tag

L'algoritmo quindi divide il messaggio in m blocchi da n bit (in realtà l'ultimo quasi sicuramente conterrà meno di n bit, dato che in questo caso non si esegue alcun padding). I primi m-1 blocchi vengono messi in XOR con Z[i], poi cifrati e poi messi nuovamente in XOR ancora con Z[i].

Il blocco finale di testo in chiaro invece subisce una lavorazione diversa. Viene prima calcolarta la sua lunghezza. Tale valore viene messo in XOR con un valore di L e poi con Z[m]. A questo punto viene traoncato alla lunghezza del blocco, cifrato e messo in XOR con il blocco stesso. Il risultato viene sottoposto a padding, per renderlo di lunghezza uguale agli altri blocchi.

A questo punto viene eseguito uno XOR tra il valore ottenuto da quest'utltima operazione e gli altri m-1 blocchi. Il risultato viene messo in XOR con Z[m], criptato e troncato per ottenere solo il numero di bit necessari per il tag di autenticazione per il messaggio. La fase di decifratura è sommariamente uguale a quella di cifratura.

Crittografia asimmetrica

Gli **algoritmi di crittografia asimmetrica** (o **a chiave pubblica**) più utilizzati sono RSA e Diffie-Hellman.

RSA è uno dei primi schemi a chiave pubblica ed è anche l'approccio maggiormente implementato. Si tratta di un codice a blocchi, in cui testo in chiaro e testo cifrato sono numeri interi compresi tra 0 e n-1, per qualche n.

La cifratura e la decifratura hanno le seguenti forme, dati alcuni blocchi di testo in chiaro M e di testo cifrato C:

$$C = M^e \bmod n \qquad M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Ovviamente, sia mittente che destinatario devono conoscere sia n che e; invece solo il destinatario deve conoscere d.

Si parla quindi di chiave asimmetrica perchè c'è sia una ***chiave pubblica PU*** ({e, n}), che una ***privata PR*** ({d, n}).

Affinchè RSA sia efficiente, devono essere soddisfatti dei requisiti:

- È possibile trovare valori di e, d, n tali che $M^{ed} \bmod n = M$ per ogni $M < n$;
- È relativamente facile calcolare M^e e C^d per ogni $M < n$;
- Non è possibile determinare d da e ed n;

Questo ultimo requisito necessita di grandi valori di e ed n per essere soddisfatto. Per quanto riguarda invece il primo requisito, esso vale se d ed e sono inversi moltiplicativi modulo Φ(n), dove Φ(n) è la funzione di Eulero, cioè il numero di interi positivi minori di n e primi rispetto ad n stesso. Algebricamente parlando, e (e anche d) soddisfa il requisito se è primo rispetto a Φ(n) (cioè MCD(e, Φ(n)) = 1).

Per quanto riguarda invece la scelta della chiave, si selezionano due numeri primi p e q (ovviamente diversi) e si calcola il loro prodotto n = p*q. Successivamente si calcola Φ(n), il quale sarà uguale a (p-1)*(q-1), dato che p e q sono primi. A questo punto si sceglie e, che deve essere primo rispetto a Φ(n). Infine si calcola d come l'inverso moltiplicativo di e rispetto a Φ(n). A questo punto le chiavi pubblica e privata sono pronte.

Sono quattro i possibili approcci per attaccare l'algoritmo RSA:

- ***Brutal force***: provare tutte le possibili chiavi private;

- ***Attacco matematico***: prevede diversi approcci, accomunati dal fine di trovare la giusta fattorizzazione di n;
- ***Attacco temporale***: dipende dal tempo di esecuzione dell'algoritmo di decifratura;
- ***Attacco con testo cifrato scelto***: sfrutta le proprietà dell'algoritmo RSA.

Per quanto riguarda l'**attacco matematico**, possiamo identificare tre approcci:

- Fattorizzare n nei suoi due fattori primi (p e q). Ciò consente di eseguire tutte le operazioni di sviluppo della chiave e di comprendere d (e è pubblico);
- Determinare direttamente $\Phi(n)$. Anche in questo caso si può determinare d tramite e;
- Determinare d direttamente.

Dato che tutto si basa sulla fattorizzazione, possiamo misurare la sicurezza di RSA tramite il costo temporale di eseguire tale operazione. Essa in effetti è un'operazione difficile ma non impossibile. Per questo ci sono delle raccomandazioni per la scelta di p e q:

- p e q dovrebbero differire in lunghezza di soli pochi caratteri;
- Sia (p-1) che (q-1) dovrebbero contenere un fattore primo grande;
- MCD(p-1, q-1) dovrebbe essere piccolo.

Inoltre è stato dimostrato che se $e < n$ e $d < n/4$, allora d è facilmente determinabile.

Per quanto riguarda invece l'attacco temporale, è stato dimostrato che si uò determinare la chiave privata tenendo traccia del tempo impiegato dall'algoritmo di decifratura. Questi attacchi sono allarmanti per due ragioni:

- proviene da una direzione assolutamente inaspettata;
- è un attacco che si basa esclusivamente sul testo cifrato.

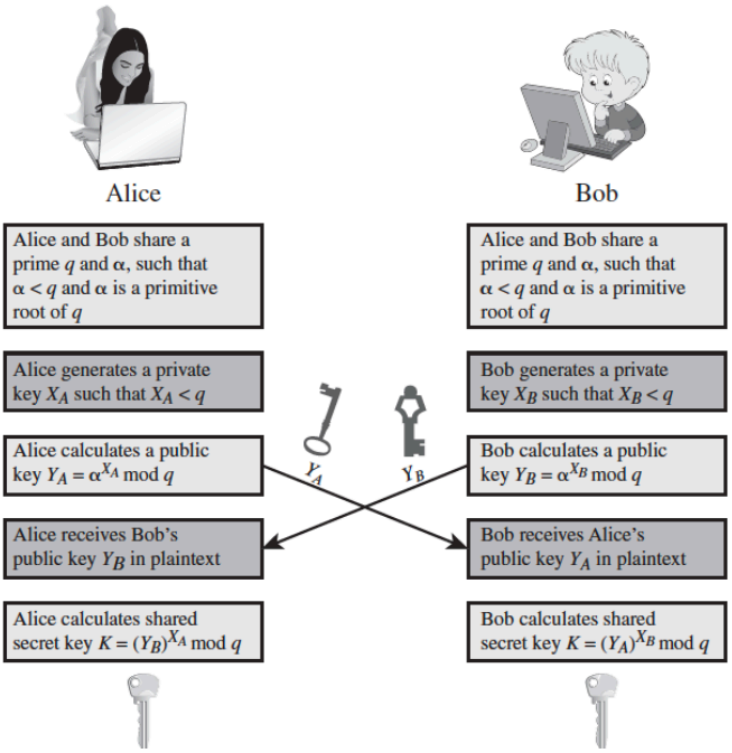
L'ipotesi alla base dell'attacco è che il sistema target utilizza una funzione di moltiplicazione modulare che è generalmente molto veloce, a meno di alcuni casi specifici. L'attacco procede con una decifratura bit a bit e si calcola il tempo impiegato. Se la decifratura del bit è lenta allora esso sarà 1, altrimenti esso sarà 0. Sebbene tale attacco rappresenti una minaccia seria, esistono delle semplici contromisure:

- Tempo di esponenzializzazione costante: assicurarsi che tutte le elevazioni a potenza richiedano lo stesso tempo prima di restituire il resultato. Questa è una soluzione semplice ma che riduce le prestazioni;
- Ritardo casuale: aggiungere un ritardo casuale, abbastanza rumoroso, per rendere indeterminabile la differenza di tempi tra le esponenzializzazioni;

- Accecamento: moltiplicare il testo cifrato per un numero casuale prima di eseguire l'esponenzializzazione. In questo modo si impedice l'analisi bit a bit essenziale per l'attaco.

Il primo algoritmo a chiave pubblica pubblicato è il **Diffie-Hellman**. Lo scopo di tale algoritmo è quello di scambiare in modo sicuro una chiave segreta che possa poi essere utilizzata per le successive crittografie di messaggi. L'efficacia di questo algoritmo dipende dalla difficoltà di calcolare logaritmi discreti.

L'algoritmo prevede inzialmente la scelta di un numero primo q e di un numero α, radice primitiva di q. In seguito, le due parti della comunicazione, A e B, generano delle chiavi. A seleziona una chiave privata X_a, che sia minore di q, e poi calcola la chiave pubblica $Y_a = \alpha^{Xa} \bmod q$. Stessa cosa fa B generando le chiavi X_b e Y_b. A questo punto A e B calcolano la chiave privata comune calcolata rispettivamente come $K = Y_b^{Xa}$ e $K = Y_a^{Xb}$.



La sicurezza dello scambio di chiavi tramite Diffie-Hellman risiede nel fatto che è molto compleso calcolare i logaritmi discreti. Tale compito è considerato irrealizzabile nel caso di numeri primi grandi.

Un'altra vulnerabilità deriva anche dal protocollo di scambio delle chiavi. Un possibile pericolo potrebbe essere l'attacco ***Man-in-the-middle***, il quale, dati A e B i due capi della comunicazione e D l'attaccante, si comporta così:

- D genera le chiavi private XD_1 e XD_2 e le rispettive chiavi pubbliche YD_1 e YD_2;
- A trasmette Y_A a B;
- D intercetta Y_A e trasmette al suo posto YD_1 a B. Contestualmente calcola K_2;
- B riceve YD_1 e calcola K_1;
- B trasmette Y_B a A;
- D intercetta Y_B e trasmette al suo posto YD_1 a A. Contestualmente calcola K_1;
- A riceve YD_2 e calcola K_2;

A questo punto tutte le successive comunicazioni saranno compromesse, dato che A e B pensano di condividere una chiave segreta, che però tale non è. Infatti, di fatto, A e D condividono la chiave segreta K_2 mentre B e D condividono la chiave segreta K_1. Tutte le seguenti comunicazioni funzioneranno nel seguente modo:

- A manda un messaggio criptato con chiave K_2;
- D intercetta il messaggio, lo decifra per recuperare il messaggio in chiaro M. Quest'ultimo, o un qualsiasi testo in chiaro deciso da D, viene quindi cifrato usando la chiave K_1 e mandato a B.

Il protocollo di scambio delle chiavi è vulnerabile rispetto tale attacco dato che non autentica i partecipanti. Questo problema può essere aggirato con l'uso di firme digitali e certificati a chiave pubblica.

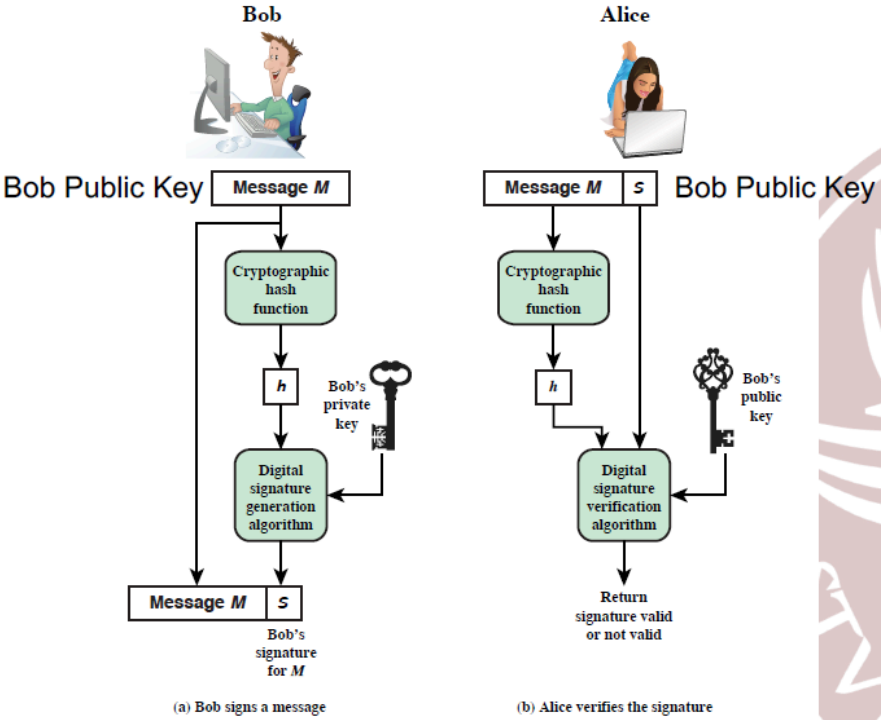


Figure 2.7 Simplified Depiction of Essential Elements of Digital Signature Process