Appunti Sicurezza dei dati

Corso di Sicurezza dei Dati

14 ottobre 2025

Indice

1 Introduzione e Caratterizzazione dei Sistemi Distribuiti

Definizione 1 (Sistema Distribuito)**.** *Un Sistema Distribuito è un insieme di componenti software, localizzati su computer interconnessi in rete, che comunicano e coordinano le loro azioni esclusivamente tramite lo scambio di messaggi.*

1.1 Aspetti Fondamentali

La principale sfida nella progettazione di questi sistemi è l'assenza di un controllo centrale e di un orologio globale.

- **Concorrenza**: I processi operano contemporaneamente.
- **Assenza di Clock Globale**: La nozione di "tempo assoluto" non esiste. La sincronizzazione è solo tramite messaggi.
- **Fallimenti Indipendenti**: Ogni componente può fallire indipendentemente dagli altri, rendendo la tolleranza ai guasti cruciale.

1.2 Modello di Interazione (Tempificazione)

Il modello di interazione definisce le ipotesi sui tempi di esecuzione e comunicazione.

- **Sistema Sincrono**: Si assumono e si conoscono limiti precisi (Upper e Lower Bound) per:
  1. Tempo di esecuzione di ogni passo di elaborazione.
  2. Tempo di consegna dei messaggi.
  3. Deriva degli orologi locali (Clock Drift Rate).
- **Sistema Asincrono**: Non esistono limiti noti o garantiti. Molti sistemi reali (es. Internet, reti basate su TCP/IP) si modellano come asincroni.

1.3 Modello dei Fallimenti

Si definiscono tre stati causali di malfunzionamento:

$$\underbrace{\text{Guasto (Fault)}}_{\text{Causa Originaria}} \longrightarrow \underbrace{\text{Errore (Error)}}_{\text{Stato Intermedio}} \longrightarrow \underbrace{\text{Fallimento (Failure)}}_{\text{Scostamento dal Comportamento Atteso}}$$

2 Il Problema del Consenso Distribuito (C)

Il problema del Consenso (C) richiede che N processi (o nodi) convergano e decidano su un unico valore d.

Tabella 1: Tassonomia dei Tipi di Fallimento

Tipo di Fallimento	Descrizione
Crash	Il processo interrompe le sue azioni e si arresta.
Fail Stop	Il processo si arresta e il suo fallimento è immediatamente rilevabile dagli altri.
Omissione (Omision)	Mancata esecuzione di un'azione attesa (es. perdita di un messaggio).
Temporale (Timing)	Mancato rispetto di una scadenza (solo in sistemi sincroni).
Arbitrario / Bizantino	Il processo si comporta in modo completamente imprevedibile, arbitrario o malizioso, potendo inviare messaggi contraddittori.

2.1 Requisiti di Consenso

Un algoritmo risolve il problema del consenso se i processi corretti soddisfano le seguenti proprietà:

1. **Validità (Validity)**:

$$(\forall p_i \in \text{corretti}, v_i = v) \Rightarrow (d = v) \quad \text{e} \quad d \in D \tag{1}$$

   Se tutti i processi corretti propongono lo stesso valore v, allora il valore di decisione d deve essere v. Inoltre, il valore deciso d deve essere uno dei valori inizialmente proposti D.

2. **Accordo (Agreement)**:

$$\forall p_i, p_j \in \text{corretti}, \quad d_i = d_j \tag{2}$$

   Tutti i processi corretti devono convergere sulla stessa decisione d.

3. **Integrità (Integrity)**:

$$p_i \text{ decide } d_i \Rightarrow d_i \in D \text{ e } p_i \text{ decide al più una volta} \tag{3}$$

   Ogni processo decide al massimo una volta, e il valore deciso è un valore che è stato proposto.

2.2 Consenso in Sistemi Sincroni con Crash Failure

Nei sistemi sincroni, il consenso è **garantito** se il numero di fallimenti è tollerabile.

- **Ipotesi**: Sistema Sincrono; N processi; al più F fallimenti per **crash** ($F < N$).
- **Primitiva**: Si utilizza **B-Multicast**, che garantisce la ricezione da parte dei destinatari corretti se il mittente non fallisce.

Algoritmo e Dimostrazione 1 (Consenso in Sistemi Sincroni (Crash Failure))**.** *Per tollerare F fallimenti per crash, sono necessari $K = F + 1$ round (cicli) di comunicazione.*

Passaggi e Meccanismo: *L'algoritmo si basa sullo scambio iterativo del set di valori proposti, garantendo che l'informazione del fallimento o del valore di un nodo si propaghi attraverso tutta la rete.*

1. ***Inizializzazione (Round 0)**: Ogni processo P_i ha un vettore di valori proposti $V_i^{(0)}$.*
2. ***Iterazioni** ($k = 0$ a $F - 1$):*
   - *P_i invia, tramite $B\_multicast$, l'insieme dei valori $\Delta V_i^{(k)}$ che ha ricevuto e non ancora trasmesso nei round precedenti.*
   - *P_i riceve i messaggi e aggiorna il suo set di valori noti:*

$$V_i^{(k+1)} = V_i^{(k)} \cup \{\text{nuovi valori ricevuti nel round } k\}$$

3. ***Decisione Finale (Dopo $F + 1$ round)**: Al termine dell'ultimo round ($k = F$), tutti i processi corretti P_i avranno ricevuto l'insieme completo di valori noti nonostante i fallimenti. Essi scelgono un valore basandosi su una funzione di scelta deterministica (es. massimo o minimo):*

$$\text{Decidi: } d_i = \max(V_i^{(F+1)})$$

***Perché $F + 1$ round?** La convergenza è garantita perché il processo che fallisce nel round r ha inviato il suo valore nei round precedenti. Se il processo più lento o quello che fallisce per ultimo è cruciale, sono necessari F round per propagare l'informazione della sua assenza e un ulteriore round per raggiungere l'accordo finale tra i restanti. Dopo F round, l'informazione sui fallimenti dei primi F nodi si è propagata.*

3 Varianti e Limiti del Consenso

3.1 Problema dei Generali Bizantini (BG)

Il problema BG è il consenso in presenza di fallimenti **arbitrari (bizantini)**.

- **Scenario**: Un Comandante invia un ordine a $N - 1$ Luogotenenti.
- **Sfida**: I nodi bizantini possono inviare ordini diversi (es. Attacca e Ritirati) ai Luogotenenti corretti.

Teorema 1 (Soluzione BG in Sistemi Sincroni)**.** *Per risolvere il problema dei Generali Bizantini in un sistema sincrono, è necessario che il numero totale di processi N sia **strettamente maggiore di tre volte il numero dei fallimenti bizantini** F:*

$$N > 3F$$

In altre parole, il sistema può tollerare al massimo $F = \lfloor (N - 1)/3 \rfloor$ fallimenti bizantini.

Questo è un requisito molto più stringente del caso Crash, perché la malizia richiede ridondanza molto maggiore.

3.2 Consistenza Interattiva (IC)

Il problema della Consistenza Interattiva è una generalizzazione di C e BG, in cui l'obiettivo non è convergere su un singolo valore, ma su un **vettore di valori proposti** (uno per ogni processo).

- **Relazione con C e BG**: Una soluzione per IC implica una soluzione per BG, e una soluzione per BG è sufficiente per risolvere C.

$$\text{IC} \Rightarrow \text{BG} \Rightarrow \text{C}$$

3.3 Teorema di Impossibilità di FLP

Teorema 2 (Teorema di Impossibilità (FLP), 1985)**.** *Non esiste un algoritmo di consenso* ***deterministico*** *in un sistema distribuito* ***asincrono*** *che possa tollerare il fallimento di* ***anche un solo processo per crash****.*

Spiegazione Discorsiva: In un sistema asincrono, è impossibile distinguere un processo molto lento (ma corretto) da un processo che ha avuto un crash. Un algoritmo deterministico, per poter garantire la Validità e l'Accordo, si troverebbe in un vicolo cieco in cui non può decidere se aspettare il processo "lento" (violando il Progresso/Terminazione) o se procedere senza di esso (rischiando di violare l'Accordo se il processo non ha fallito).

3.4 Paxos e la Sicurezza Asincrona

Il protocollo di **Paxos** (Lamport) è la soluzione più nota per aggirare l'impossibilità di FLP nei sistemi asincroni.

- **Focus**: Paxos garantisce la proprietà di **Safety** (Validità e Accordo) anche in presenza di fallimenti asincroni.
- **Compromesso**: Non può garantire la proprietà di **Liveness** (Progresso o Terminazione) in presenza di fallimenti o di condizioni avverse (come una lunga sequenza di proposte concorrenti, sebbene sia raro). Questo è il prezzo da pagare per operare in un ambiente asincrono.

4 Consenso Sicuro e Autenticazione

Nelle reti reali, gli attacchi possono: far divergere la rete, invertire lo stato o impedire il consenso. Per mitigare i fallimenti bizantini e gli attacchi, si introduce la sicurezza.

4.1 Tecnica di Autenticazione Basata sull'Identità (Identity-Based Cryptography - IBC)

L'autenticazione tramite crittografia è un miglioramento fondamentale per gli algoritmi di consenso in contesti non affidabili.

Algoritmo e Dimostrazione 2 (Schema di Autenticazione con IBC)**.** *Si ipotizza l'uso della crittografia basata sull'identità per garantire l'autenticità dei messaggi e prevenire la manipolazione da parte di nodi bizantini.*

1. ***Firma del Messaggio****: Quando un nodo mittente P_m vuole inviare un valore v agli altri, esso:*
   - *Firma il messaggio con la sua* ***chiave privata*** *(SK_m).*
   - *Crittografa il messaggio utilizzando l'****identificativo del destinatario*** *(ID_r), che funge da chiave pubblica implicita.*
2. ***Invio****: Il mittente P_m invia il messaggio criptato e firmato.*
3. ***Verifica del Ricevente****: Il nodo ricevente P_r riceve il messaggio e compie due passaggi cruciali:*
   - ***Decrittografia****: Utilizza la sua* ***chiave privata*** *(SK_r) per decifrare il messaggio.*

- ***Verifica della Firma**: Utilizza l'**identificativo del mittente** (ID_m), che funge da chiave pubblica, per verificarne l'autenticità.*

4. ***Ingresso nel Consenso**: Se la verifica della firma ha successo, l'autenticità del mittente è garantita e il messaggio può entrare nella procedura di aggiornamento del consenso.*

Questo meccanismo previene che un nodo bizantino si spacci per un altro o che alteri un messaggio in transito.

5 Consenso Sicuro e Autenticazione

Il consenso sicuro affronta il problema degli **attacchi** in ambienti non fidati.

5.1 Effetti degli Attacchi sul Consenso

Gli attacchi (spesso di natura bizantina) agli algoritmi di consenso possono provocare i seguenti fenomeni:

- **Divergenza della Rete**: I nodi convergono su valori diversi, violando l'Agreement.
- **Inversione di Stato**: Alterazione dello stato corretto del sistema.
- **Convergenza al Valore Iniettato**: La rete converge sul valore imposto dall'aggressore.
- **Impedire il Consenso**: La rete non raggiunge mai la decisione finale, violando la Liveness.

5.2 Tecniche di Rilevamento e Autenticazione

5.2.1 Soluzione Basata sulla Devianza

Una prima soluzione per mitigare gli aggressori è calcolare la devianza di ogni nodo dal valore di consenso della maggioranza, e definire affidabili i nodi con una deviazione minima.

$$\text{Affidabilità di } P_i = f(\text{Devianza}(P_i, \text{Consenso}_{\text{Maggioranza}}))$$

Problema: Questo metodo può rimuovere anche utenti non aggressori se colludono, o diventare inefficace quando gli aggressori colludono fra loro, mimando il comportamento della maggioranza.

5.2.2 Crittografia Basata sull'Identità (IBC)

L'autenticazione tramite crittografia è la tecnica fondamentale per prevenire che un nodo bizantino alteri o spacci messaggi.

Algoritmo e Dimostrazione 3 (Schema di Autenticazione con IBC)**.** *L'autenticità si garantisce attraverso la firma del mittente e la crittografia per il destinatario.*

1. ***Firma e Crittografia**: Il mittente P_m firma con la sua chiave privata (SK_m) e crittografa usando l'identificativo del destinatario (ID_r) come chiave pubblica.*
2. ***Verifica del Ricevente** (P_r):*
   - ***Decrittografia**: Usa SK_r.*
   - ***Verifica Firma**: Usa ID_m (chiave pubblica del mittente) per validare l'autenticità.*

   Se la verifica ha successo, il messaggio può entrare nella procedura di consenso, sapendo che proviene sicuramente da P_m.

5.3 Consenso Basato sul Leader (Paxos)

Il protocollo di **Paxos** opera in sistemi asincroni aggirando il Teorema FLP.

- **Ruoli Chiave**: Paxos definisce tre ruoli che possono essere assunti da qualsiasi nodo:
  1. **Proposer**: Propone un valore e guida l'algoritmo.
  2. **Acceptor**: Vota e memorizza il valore accettato.
  3. **Learner**: Decide il valore finale.
- **Consenso Dinamico**: Si basa sull'elezione di un Leader (Proposer) che coordina le fasi del consenso. In caso di fallimento del Proposer, ne viene eletto uno nuovo.
- **Compromesso**: Paxos garantisce la **Safety** (Validità e Accordo) sempre, ma non può garantire la **Liveness** (Progresso/Terminazione) se i Proposer continuano a fallire o a competere tra loro continuamente.

6 Comunicazione di Gruppo e Rilevamento di Fallimenti (Lezione 2)

6.0.1 Gruppi Aperti e Chiusi

Una comunicazione di gruppo può essere classificata in base a chi è autorizzato a inviare messaggi al gruppo:

- **Gruppo Chiuso:** Solo i membri del gruppo possono inviare messaggi in multicast al gruppo stesso. Qualsiasi comunicazione dall'esterno viene bloccata.
- **Gruppo Aperto:** Anche entità non membre del gruppo possono inviare messaggi in multicast a tutti i membri del gruppo.

6.0.2 Strategie per la Consegna Affidabile

L'obiettivo della comunicazione di gruppo è assicurare che i messaggi vengano scambiati correttamente anche in presenza di fallimenti di canali e/o processi. Le strategie principali sono:

- **Error Masking:** Si basa sull'uso della ridondanza per mascherare gli errori di omissione.
  - **Ridondanza Spaziale:** I processi sono connessi tramite percorsi multipli. Per mascherare k omissioni, sono necessari $k + 1$ collegamenti distinti. Il problema principale è la possibile ricezione di messaggi duplicati, che devono essere scartati dal ricevente.
  - **Ridondanza Temporale:** Il messaggio viene inviato più volte sullo stesso canale. Per mascherare k omissioni, il messaggio deve essere inviato $k + 1$ volte. Anche in questo caso, è necessario gestire i duplicati.
- **Error Detection and Recovery:** Si basa sull'uso di messaggi di riscontro (acknowledgements) e timeout per rilevare e recuperare i messaggi persi.
  - **Positive Acknowledgements (ACK):** Il ricevente invia un ACK per ogni messaggio ricevuto. Il mittente ritrasmette il messaggio se non riceve l'ACK entro un timeout predefinito. Questo schema è rapido nell'individuare fallimenti anche con traffico sporadico.

– **Negative Acknowledgements (NACK):** Il ricevente notifica al mittente solo la mancata ricezione di un messaggio atteso. Questo minimizza il traffico di rete ma richiede meccanismi aggiuntivi, come la numerazione sequenziale dei messaggi, per permettere al ricevente di identificare le perdite.

6.0.3 Reliable Multicast

Un protocollo di *Reliable Multicast* (R-Multicast) deve soddisfare le seguenti proprietà per i processi corretti (che non falliscono):

- **Integrity:** Un processo corretto p riceve un messaggio m al più una volta, solo se p appartiene al gruppo destinatario e m è stato inviato dal mittente specificato.
- **Validity:** Se un processo corretto invia in multicast un messaggio m, prima o poi riceverà m esso stesso.
- **Agreement:** Se un processo corretto riceve un messaggio m, allora tutti i processi corretti nel gruppo prima o poi riceveranno m.

Una versione più forte di questa proprietà è la **Uniform Agreement**, che si applica indipendentemente dal fatto che il processo che riceve il messaggio sia corretto o meno: se un processo (corretto o meno) riceve m, allora tutti i processi corretti nel gruppo prima o poi riceveranno m.

6.0.4 Ordinamenti dei Messaggi in Multicast

Per garantire la consistenza, è spesso necessario che i messaggi inviati a un gruppo vengano consegnati a tutti i membri secondo un ordine specifico.

Ordinamento FIFO (First-In, First-Out). Se un processo corretto invia in multicast un messaggio m e successivamente un messaggio m', allora ogni processo corretto che riceve m' deve aver già ricevuto m. Questo ordinamento preserva l'ordine di invio dei messaggi provenienti da un singolo mittente.

Ordinamento Causale. Questo modello estende l'ordinamento FIFO per rispettare la potenziale relazione di causalità tra messaggi inviati da processi diversi. Si basa sulla relazione *happened-before* ($\rightarrow$). Formalmente:

- Se un processo p invia m e un processo p' riceve m prima di inviare m', allora nessun processo corretto riceve m' senza aver prima ricevuto m.
- In altre parole, se $multicast(g, m) \rightarrow multicast(g, m')$, ogni processo corretto che riceve m' deve aver già ricevuto m.

L'ordinamento causale implica quello FIFO.

Ordinamento Totale. Se un processo corretto riceve un messaggio m e successivamente un messaggio m', allora ogni altro processo corretto che riceve entrambi i messaggi deve ricevere m prima di m'.

- L'ordinamento di consegna è lo stesso per tutti i processi membri del gruppo.
- Tuttavia, questo ordine può essere arbitrario e non implica necessariamente l'ordinamento FIFO o quello causale. Ad esempio, un messaggio inviato prima potrebbe essere consegnato dopo, purché l'ordine sia lo stesso per tutti.

6.0.5 Implementazione dell'Ordinamento Totale

Metodo con Sequenziatore. Un approccio si basa su un processo designato come **sequenziatore**, che ha il compito di assegnare un numero di sequenza univoco e progressivo a ogni messaggio inviato al gruppo.

1. Un processo mittente invia un messaggio m sia al gruppo g sia al sequenziatore.
2. Il sequenziatore riceve i messaggi, assegna loro un numero di sequenza progressivo (s_g) e comunica questo numero al gruppo tramite un messaggio di ordinamento via B-multicast.
3. Ogni processo membro del gruppo riceve sia il messaggio originale m sia il messaggio di ordinamento dal sequenziatore. Mantiene i messaggi in una coda di attesa (*hold-back queue*) e li consegna all'applicazione solo quando il loro numero di sequenza corrisponde al prossimo numero atteso.

Metodo Distribuito (Algoritmo ISIS). Un approccio alternativo, valido per gruppi aperti o chiusi, prevede che i processi concordino collettivamente sui numeri di sequenza. L'algoritmo si articola in tre fasi:

1. Un processo invia un messaggio m a tutti i membri del gruppo.
2. Ogni ricevente propone un numero di sequenza per quel messaggio, tipicamente basato sul numero di sequenza più alto che ha visto finora, e lo invia solo al mittente originale. Mantiene il messaggio in una coda locale ordinata in base al numero proposto.
3. Il mittente raccoglie tutte le proposte di sequenza, seleziona il valore più alto come numero di sequenza finale ("agreed") e lo comunica a tutto il gruppo. I riceventi aggiornano il numero di sequenza del messaggio nella loro coda e lo consegnano quando diventa il messaggio in testa alla coda.

6.0.6 Tecniche di Propagazione

Oltre al B-Multicast, esistono tecniche di diffusione (broadcast) meno strutturate:

- **Flooding (Inondazione)**:
  1. Il mittente invia il messaggio a tutti i suoi vicini.
  2. Ogni nodo ricevente (se è la prima volta che vede il messaggio) lo inoltra a tutti i suoi vicini, eccetto quello da cui l'ha ricevuto.

  Vantaggio: Raggiunge rapidamente tutti i nodi. **Svantaggio:** Enorme overhead di messaggi e rischio di cicli.

- **Gossiping (Diffusione a Pettegolezzo)**:
  1. Un nodo mittente P_i invia il messaggio solo a un **sottoinsieme casuale** dei suoi vicini (fan-out).
  2. I nodi riceventi, con una certa probabilità, inoltrano il messaggio ad altri vicini scelti casualmente.

  Vantaggio: Scalabile, basso overhead, robusto ai fallimenti. **Svantaggio:** L'affidabilità (la garanzia di consegna) è **probabilistica**, non deterministica.

6.1 Rilevatori di Fallimenti (Failure Detectors)

I Failure Detectors sono componenti che tentano di identificare i processi falliti. Si classificano per Completezza (identificazione) e Precisione (accuratezza).

7 Classificazione dei Failure Detectors - Riveduta

7.0.1 Classificazione Formale (Chandra e Toueg)

I Failure Detector (FD) vengono classificati in base a due metriche fondamentali:

- **Completezza (Completeness):** La capacità del detector di rilevare effettivamente i fallimenti.
  - *Strong Completeness:* Prima o poi, ogni processo fallito è permanentemente sospettato da **ogni** processo corretto.
  - *Weak Completeness:* Prima o poi, ogni processo fallito è permanentemente sospettato da **almeno un** processo corretto.
- **Accuratezza (Accuracy):** La capacità del detector di non sospettare erroneamente processi corretti (evitare falsi positivi).
  - *Strong Accuracy:* Nessun processo corretto è mai sospettato da alcun altro processo corretto.
  - *Weak Accuracy:* Esiste almeno un processo corretto che non è mai sospettato da alcun altro processo corretto.
  - *Eventual Strong Accuracy:* Esiste un istante di tempo dopo il quale nessun processo corretto è più sospettato da alcun processo corretto.
  - *Eventual Weak Accuracy:* Esiste un istante di tempo dopo il quale esiste almeno un processo corretto che non è più sospettato da alcun processo corretto.

Combinando questi livelli si ottengono otto classi di failure detector. Il più debole in grado di risolvere il consenso in un sistema asincrono è l'**Eventually Weak Failure Detector** ($\Diamond W$).

7.0.2 Implementazione Pratica dei Failure Detector

L'implementazione di un FD inaffidabile in un sistema asincrono si basa tipicamente su meccanismi di *heartbeat*:

1. Ogni processo p_j invia periodicamente (ogni T secondi) un messaggio "I-am-alive" a tutti gli altri processi.
2. Ogni processo p_i monitora gli altri. Se p_i non riceve un messaggio da p_j entro un timeout $T + \Delta$ (dove Δ è una stima del tempo massimo di trasmissione), sospetta che p_j sia fallito e lo marca come `Suspected`.
3. Se successivamente p_i riceve un messaggio da p_j, ritira il sospetto e lo marca come `Unsuspected`.

La scelta di T e Δ è un compromesso: valori brevi aumentano il traffico e i falsi positivi; valori lunghi ritardano il rilevamento. Una soluzione pratica prevede l'uso di valori adattativi.

Esistono tre principali architetture per il monitoraggio:

- **Centralizzata:** Un singolo processo monitora tutti gli altri. Semplice ma introduce un Single Point of Failure (SPOF).
- **Ad Anello:** Ogni processo monitora il suo successore in un anello logico. Non ha SPOF ma è vulnerabile a fallimenti multipli consecutivi.
- **Distribuita (Gossip-based):** Ogni processo mantiene una lista di membri con i relativi contatori di heartbeat e la propaga periodicamente tramite gossiping. Questo approccio è molto robusto, scalabile e tollerante ai fallimenti. Il tempo di propagazione di un aggiornamento è $O(\log N)$.

8 Replicazione e Modelli di Consistenza

La replicazione dei dati nei sistemi distribuiti è fondamentale per migliorare l'affidabilità e le prestazioni. Tuttavia, introduce una sfida cruciale: mantenere la consistenza tra le diverse copie. Un **modello di consistenza** è un contratto tra i processi e l'archivio dati: se i processi rispettano determinate regole, l'archivio si comporterà come previsto. L'obiettivo è definire quali tra le innumerevoli possibili sequenze di esecuzione (interleaving) delle operazioni di lettura e scrittura sono valide.

8.1 Modelli di Consistenza Basati su Architettura (Data-Centric)

Questi modelli mirano a fornire una vista consistente dell'archivio dati a livello di sistema. L'idea è simulare il comportamento di un sistema centralizzato, anche se con diversi gradi di "rilassamento" delle garanzie per migliorare le prestazioni.

8.1.1 Consistenza Stretta (Strict Consistency)

È il modello più forte e intuitivo.

- **Definizione**: Qualsiasi operazione di lettura su un dato x restituisce il valore corrispondente al risultato dell'operazione di scrittura più recente su x, secondo un tempo globale assoluto.
- **Caratteristiche**: Le scritture sono viste come istantanee e atomiche da tutti i processi. Richiede l'esistenza di un orologio globale, rendendolo un modello ideale ma praticamente irrealizzabile nei sistemi distribuiti reali a causa dell'assenza di tale orologio.

8.1.2 Linearizzabilità (Linearizability)

È un leggero indebolimento della consistenza stretta, ma il più forte modello realizzabile in pratica.

- **Definizione**: Il risultato di qualsiasi esecuzione è lo stesso che si otterrebbe se tutte le operazioni di tutti i processi fossero eseguite in un qualche ordine sequenziale. Questo ordine deve essere compatibile con l'ordine temporale reale delle operazioni non sovrapposte. Una lettura deve sempre restituire il valore della scrittura più recente, indipendentemente da chi l'ha invocata[cite: 720].
- **Implementazione**: Si può simulare un tempo globale tramite clock sincronizzati (es. con NTP). È un modello costoso perché richiede un accordo (consenso) sull'ordine totale delle operazioni.

8.1.3 Consistenza Sequenziale (Sequential Consistency)

Questo modello rilassa ulteriormente il vincolo temporale.

- **Definizione (Lamport)**: Il risultato di ogni esecuzione è lo stesso come se le operazioni di tutti i processi fossero eseguite in un qualche ordine sequenziale, e le operazioni di ogni singolo processo appaiono in questa sequenza nell'ordine specificato dal suo programma.
- **Differenza con la Linearizzabilità**: L'ordine globale delle operazioni non deve necessariamente rispettare l'ordine temporale reale. L'unica cosa che conta è che tutti i processi vedano la stessa sequenza di operazioni. Questo dà al sistema più libertà, rendendolo più performante.

8.1.4 Consistenza Causale (Causal Consistency)

Questo modello indebolisce la consistenza sequenziale distinguendo tra operazioni causalmente correlate e operazioni concorrenti.

- **Definizione**: Tutte le scritture che sono potenzialmente correlate causalmente devono essere viste da tutti i processi nello stesso ordine. Le scritture concorrenti possono essere viste in ordini diversi su macchine diverse.
- **Relazione di Causalità**: Una scrittura W1 causa una scrittura W2 se W2 è avvenuta dopo che il processo che la esegue ha letto il valore scritto da W1. Se due scritture non sono legate da questa relazione (direttamente o transitivamente), sono considerate concorrenti.

Esempio 1. *Se il processo P1 scrive 'W(x)a', e successivamente il processo P2 legge 'R(x)a' e poi scrive 'W(x)b', allora 'W(x)a' e 'W(x)b' sono causalmente correlate. Tutti i processi devono vedere 'a' prima di 'b'. Se un terzo processo P3 scrive 'W(y)c' in modo indipendente, questa scrittura è concorrente con le altre e può essere vista in ordine diverso.*

8.2 Teorema CAP (Brewer's Theorem)

Il Teorema CAP stabilisce un limite fondamentale per i sistemi distribuiti.

Teorema 3. *(Teorema CAP). È impossibile per un sistema di archiviazione dati distribuito fornire simultaneamente più di due delle seguenti tre garanzie:*

1. ***Consistenza (Consistency)**: Ogni lettura riceve la scrittura più recente o un errore (tutti i nodi vedono gli stessi dati nello stesso momento).*
2. ***Disponibilità (Availability)**: Ogni richiesta riceve una risposta (non un errore), senza garanzia che contenga la scrittura più recente.*
3. ***Tolleranza al Partizionamento (Partition Tolerance)**: Il sistema continua a funzionare nonostante un numero arbitrario di messaggi vengano persi (o ritardati) dalla rete tra i nodi.*

Poiché le partizioni di rete sono un dato di fatto nei sistemi distribuiti su larga scala, la scelta reale è tra consistenza e disponibilità.

- **Sistemi CP (Consistency + Partition Tolerance)**: In caso di partizione, il sistema sceglie di sacrificare la disponibilità per garantire la consistenza. Bloccherà le richieste che non possono essere soddisfatte in modo consistente.

- **Sistemi AP (Availability + Partition Tolerance)**: In caso di partizione, il sistema sceglie di sacrificare la consistenza (forte) per rimanere disponibile. Risponderà alle richieste, anche se potenzialmente con dati obsoleti. Questo porta al modello di **Eventual Consistency**.

8.3 Filosofie di Design: ACID vs BASE

Queste due filosofie rappresentano gli estremi dello spettro consistenza-disponibilità.

- **ACID (Atomicity, Consistency, Isolation, Durability)**: È l'approccio tradizionale dei database relazionali, che privilegia la consistenza forte (sistema CP). È un approccio pessimistico, meno scalabile per volumi di dati enormi.
- **BASE (Basically Available, Soft state, Eventual consistency)**: È l'approccio dei sistemi NoSQL moderni, che privilegia l'alta disponibilità (sistema AP).
  - **Basically Available**: Il sistema è quasi sempre disponibile.
  - **Soft State**: Lo stato del sistema può cambiare nel tempo anche senza input, a causa della convergenza finale.
  - **Eventual Consistency**: Se non vengono effettuati nuovi aggiornamenti, tutte le repliche alla fine convergeranno sullo stesso stato.

8.4 Risoluzione dei Conflitti (Reconciliation)

Nei sistemi a consistenza finale, possono verificarsi conflitti quando repliche diverse vengono aggiornate in modo concorrente durante una partizione. Le strategie per risolverli includono:

- **Last Writer Wins (LWW)**: Si sceglie la versione con il timestamp più recente. Per gestire la causalità in modo robusto, si usano spesso orologi vettoriali invece di timestamp fisici.
- **Risoluzione a livello Applicativo**: Il datastore presenta le versioni divergenti all'applicazione, che implementa una logica personalizzata per fonderle in un'unica versione corretta.

9 Crittografia e Comunicazione Sicura

La sicurezza dei dati si definisce come l'insieme di misure tecniche, organizzative e normative volte a proteggere le informazioni da accessi non autorizzati, alterazioni, perdite o divulgazioni indebite. La crittografia è la disciplina che si occupa di proteggere le informazioni, garantendo proprietà fondamentali come la confidenzialità, l'integrità e la disponibilità dei dati, note come la triade **C.I.A.** (Confidentiality, Integrity, Availability).

- **Confidenzialità (Confidentiality):** Implica l'applicazione di regole e tecniche per limitare l'accesso non autorizzato a determinate informazioni. Solo gli utenti autorizzati possono accedere ai dati.
- **Integrità (Integrity):** Assicura che i dati siano sempre legittimi, accurati e non manipolati o alterati senza autorizzazione.
- **Disponibilità (Availability):** Garantisce che i dispositivi e i servizi siano sempre disponibili per la raccolta e la visualizzazione dei dati senza interruzioni.

L'applicazione della crittografia è cruciale in contesti insicuri, che emergono principalmente in due scenari:

1. **Memorizzazione dei dati (data-at-rest):** I dati memorizzati su computer, database o dispositivi mobili possono essere vulnerabili a furti, accessi non autorizzati o compromissioni del sistema di archiviazione.
2. **Comunicazione dei dati (data-in-transit):** Durante il trasferimento su reti, come Internet, i dati possono essere intercettati, modificati o dirottati (es. attacchi man-in-the-middle). Protocolli come TCP, pur essendo *affidabili* (garantiscono la consegna), non sono intrinsecamente *sicuri*.

9.1 Principi di Crittografia

Un processo crittografico trasforma un testo in chiaro (plaintext) in un testo cifrato (ciphertext) tramite un'operazione di **cifratura** (encryption). Il processo inverso, che riconverte il testo cifrato in testo in chiaro, è la **decifratura** (decryption). Il processo si basa su due elementi fondamentali:

- **Algoritmo:** È la sequenza di passi e regole, nota e pubblica, utilizzata per eseguire la cifratura e la decifratura. La segretezza dell'algoritmo non è un fondamento della sicurezza di un cifrario moderno.
- **Chiave (Key):** È un'informazione segreta (una sequenza di cifre o caratteri) fornita come input all'algoritmo, che ne guida il processo. La robustezza di un sistema crittografico moderno risiede interamente nella segretezza e nella complessità della chiave.

La qualità di una chiave si misura dalla sua resistenza agli **attacchi a forza bruta (brute-force)**, in cui un attaccante prova tutte le combinazioni possibili. Più la chiave è lunga e complessa, più tempo richiederà un tale attacco, rendendolo computazionalmente proibitivo.

9.2 Crittografia Simmetrica e Asimmetrica

A seconda del tipo di chiave utilizzata, i sistemi crittografici si dividono in due macro-categorie.

9.2.1 Crittografia a Chiave Simmetrica

Utilizza la **stessa chiave** sia per la cifratura che per la decifratura.

- **Vantaggi:** È estremamente veloce ed efficiente dal punto di vista computazionale, rendendola ideale per cifrare grandi volumi di dati.
- **Svantaggi:** Il problema cruciale è la **distribuzione della chiave** (key distribution). Le parti che comunicano devono scambiarsi la chiave segreta in modo sicuro tramite un canale preesistente, poiché se un attaccante la intercetta, l'intera comunicazione è compromessa.

9.2.2 Crittografia a Chiave Asimmetrica (a Chiave Pubblica)

Utilizza una **coppia di chiavi** matematicamente correlate: una **chiave pubblica**, che può essere distribuita a tutti, e una **chiave privata**, che deve rimanere segreta al proprietario. Questa dualità permette due funzionalità principali:

- **Confidenzialità:** Per inviare un messaggio segreto a Bob, Alice lo cifra con la chiave pubblica di Bob. Solo Bob, in possesso della corrispondente chiave privata, potrà decifrare e leggere il messaggio.

- **Autenticazione e Non Ripudio:** Se Alice vuole dimostrare l'autenticità di un messaggio, lo "firma" cifrandolo con la sua chiave privata. Chiunque può usare la chiave pubblica di Alice per decifrare il messaggio e verificare che provenga da lei (autenticazione) e che non sia stato alterato (integrità). Poiché solo Alice possiede la sua chiave privata, non può negare di aver generato quel messaggio (non ripudio).

- **Vantaggi:** Risolve il problema della distribuzione sicura delle chiavi, poiché la chiave pubblica può essere condivisa apertamente.

- **Svantaggi:** È computazionalmente molto più lenta della crittografia simmetrica.

Nei sistemi pratici (es. protocolli SSL/TLS), si usa un **approccio ibrido**: la crittografia asimmetrica viene usata per scambiare in modo sicuro una chiave di sessione, che viene poi utilizzata con un algoritmo simmetrico più veloce per la comunicazione vera e propria.

Public Key Infrastructure (PKI) Per gestire e distribuire le chiavi pubbliche in modo affidabile, si utilizza un'infrastruttura a chiave pubblica (PKI). Un **certificato digitale** è un documento elettronico che lega un'identità a una chiave pubblica. Questi certificati sono emessi da entità fidate chiamate **Certification Authorities (CA)**, che ne garantiscono l'autenticità apponendo la propria firma digitale. Le CA sono organizzate in una gerarchia o "catena di fiducia", che parte da una *Root CA* fidata.

9.3 Firme Elettroniche e Digitali

Una firma elettronica è l'equivalente digitale di una firma autografa e serve a dare valore legale a un documento informatico. Il quadro normativo di riferimento è il regolamento europeo **eIDAS (electronic IDentification, Authentication and trust Services)** e, in Italia, il **Codice dell'Amministrazione Digitale (CAD)**. Un documento viene firmato per diverse ragioni legali: identificare il firmatario, manifestare il consenso, conferire validità legale e fornire una prova in caso di controversie.

9.3.1 Tipologie di Firme Elettroniche

Esistono diversi livelli di firma elettronica, con differente valore probatorio:

- **Firma Elettronica Semplice (FES):** Come definito dal regolamento eIDAS, consiste in "dati in forma elettronica, acclusi oppure connessi tramite associazione logica ad altri dati elettronici e utilizzati dal firmatario per firmare".
  - **Esempi:** Una firma scannerizzata e incollata su un documento, il PIN del bancomat, o il login con username e password (come quando si accetta un voto universitario online).
  - **Valore Probatorio:** Molto debole. In caso di contenzioso, la sua efficacia è liberamente valutabile da un giudice, che ne considererà l'integrità, la sicurezza e l'immodificabilità. Non offre garanzie intrinseche sull'identità del firmatario.

- **Firma Elettronica Avanzata (FEA):** Una firma che soddisfa requisiti più stringenti:
  - È connessa unicamente al firmatario ed è idonea a identificarlo.
  - È creata mediante dati che il firmatario può utilizzare sotto il proprio esclusivo controllo.

  - È collegata ai dati firmati in modo da consentire l'identificazione di ogni successiva modifica.
  - **Esempi:** La **firma grafometrica** su un tablet (es. in banca), dove non viene registrata solo l'immagine della firma ma anche dati biometrici come pressione e velocità. Un altro esempio è l'autenticazione tramite codice **OTP (One-Time Password)** ricevuto via SMS.
  - **Valore Legale:** Ha un valore legale equiparato a quello della firma autografa, ma in caso di disconoscimento, l'onere della prova grava su chi accetta la firma come valida.

- **Firma Elettronica Qualificata (FEQ):** È una FEA creata tramite un dispositivo qualificato (es. smart card, token USB) e basata su un certificato qualificato emesso da un prestatore di servizi fiduciari accreditato.
  - **Caratteristiche:** È la forma di firma elettronica più forte. Il suo principale vantaggio è l'**inversione dell'onere della prova**: in caso di contenzioso, è il firmatario a dover dimostrare che la firma non è sua o che è stata apposta senza il suo consenso. Lo SPID (Sistema Pubblico di Identità Digitale) è un esempio di tecnologia che abilita firme elettroniche qualificate.

9.3.2 La Firma Digitale

La firma digitale **non è un sinonimo** di firma elettronica, ma una specifica implementazione tecnologica di una **Firma Elettronica Qualificata (FEQ)**. Si basa sulla crittografia asimmetrica e sulle funzioni di hash.

Definizione 2 (Processo di Firma Digitale)**.** *Il processo per apporre una firma digitale è il seguente:*

1. *Si calcola un'impronta digitale (o **digest**) del documento tramite una funzione di hash crittografica.*
2. *A questo digest viene concatenata una **marcatura temporale** (timestamp), che certifica in modo legalmente valido la data e l'ora della firma.*
3. *Il risultato (digest + timestamp) viene cifrato utilizzando la **chiave privata** del firmatario. Il risultato di questa operazione è la firma digitale.*
4. *Il documento originale, la firma e il certificato del firmatario vengono inseriti in un contenitore detto **busta crittografica** (es. formati PAdES per PDF, CAdES per file .p7m).*

Definizione 3 (Processo di Verifica della Firma Digitale)**.** *La verifica segue il processo inverso:*

1. *Si decifra la firma utilizzando la **chiave pubblica** del firmatario (contenuta nel suo certificato), ottenendo così il digest originale e la marcatura temporale.*
2. *Si ricalcola autonomamente il digest del documento ricevuto.*
3. *Si confrontano i due digest: se corrispondono, si ha la certezza che il documento non è stato alterato (integrità) e che è stato firmato dal possessore della chiave privata (autenticità).*

9.4 Steganografia vs Crittografia

Entrambe le tecniche mirano a proteggere le informazioni, ma con approcci concettualmente diversi.

- **Crittografia:** Occulta il **contenuto** del messaggio, ma non nasconde la sua esistenza. Un messaggio cifrato è visibile come una sequenza di dati incomprensibili, ma la comunicazione è palese.
- **Steganografia:** Occulta l'**esistenza stessa** del messaggio. Il dato segreto viene nascosto all'interno di un altro file, apparentemente innocuo, detto *cover data* (es. un'immagine, un file audio).

Una tecnica comune di steganografia digitale è la **LSB (Least Significant Bit) insertion**, che consiste nel modificare i bit meno significativi dei pixel di un'immagine per codificare i bit del messaggio segreto. Tali modifiche sono impercettibili all'occhio umano. Spesso, crittografia e steganografia vengono combinate per ottenere una sicurezza a due livelli: si cifra un messaggio e poi si nasconde il testo cifrato all'interno di un altro file.

9.5 Crittografia Classica e Criptoanalisi

La crittografia classica si riferisce ai metodi utilizzati prima dell'avvento dei computer, basati su algoritmi semplici, spesso eseguiti manualmente o con dispositivi meccanici. Si suddivide principalmente in cifrari a sostituzione e a trasposizione.

Cifrari a Sostituzione Sostituiscono le unità del testo in chiaro (lettere o gruppi di lettere) con altre unità.

- **Cifrario di Cesare:** È uno dei più antichi cifrari a sostituzione monoalfabetica, usato da Giulio Cesare. Ogni lettera del messaggio viene spostata di un numero fisso di posizioni (la chiave) nell'alfabeto.

  Esempio 2 (Cifrario di Cesare con chiave k=3)**.** *La cifratura di una lettera P in una lettera C segue la formula:*

  $$C = (P + k) \pmod{26}$$

  dove alle lettere A-Z sono associati i numeri 0-25. La decifratura è $P = (C - k) \pmod{26}$*. Testo in chiaro: 'CESARE' → Cifratura: 'FHVDUH'.*

Criptoanalisi dei Cifrari Classici La criptoanalisi è l'arte di "rompere" i cifrari senza conoscere la chiave. Il metodo più antico ed efficace contro i cifrari a sostituzione semplice è l'**analisi delle frequenze**. Poiché ogni lingua ha una distribuzione di frequenza caratteristica per le sue lettere (ad esempio, in italiano le lettere più comuni sono 'E' e 'A'), un criptoanalista può analizzare la frequenza dei simboli nel testo cifrato. Il simbolo più frequente nel testo cifrato corrisponderà probabilmente alla lettera più frequente della lingua originale, permettendo di dedurre la chiave di spostamento.

Cifrari a Trasposizione Non alterano le lettere del testo in chiaro, ma ne cambiano la posizione secondo uno schema predefinito.

- **Scitala Spartana:** Considerato uno dei primi dispositivi crittografici, era un bastone di un certo diametro. Un messaggio veniva scritto su una striscia di pergamena avvolta a spirale attorno al bastone. Una volta srotolata, la striscia mostrava le lettere in un ordine apparentemente casuale. Solo chi possedeva un bastone dello stesso identico diametro (la chiave) poteva riavvolgere la striscia e leggere il messaggio.

9.6 La Transizione alla Crittografia Moderna

Il passaggio dalla crittografia classica a quella moderna fu graduale, spinto dall'avvento dell'informatica e dalle necessità emerse durante la Seconda Guerra Mondiale. La decifrazione della macchina **Enigma** da parte degli Alleati dimostrò che anche i sistemi meccanici complessi potevano essere violati, mentre l'invenzione del computer rese possibili attacchi a forza bruta su una scala prima inimmaginabile.

I Principi di Shannon Nel 1949, Claude Shannon pubblicò il saggio *Communication Theory of Secrecy Systems*, che trasformò la crittografia da arte a scienza. Egli introdusse due principi fondamentali che guidano ancora oggi la progettazione di cifrari sicuri:

1. **Confusione (Confusion):** Mirare a rendere la relazione tra la chiave e il testo cifrato il più complessa e non lineare possibile. Una piccola modifica alla chiave deve alterare radicalmente il testo cifrato. Questo principio viene tipicamente implementato tramite operazioni di **sostituzione** (es. le S-box).

2. **Diffusione (Diffusion):** Mirare a disperdere l'influenza di ogni singolo bit del testo in chiaro su molti bit del testo cifrato. Questo maschera le regolarità statistiche della lingua originale. Questo principio viene tipicamente implementato tramite operazioni di **permutazione**.

I due principi sono complementari: un cifrario sicuro deve implementarli entrambi per resistere agli attacchi statistici e algebrici.

9.7 Crittografia Simmetrica Moderna

9.7.1 Cifrari a Blocchi (Block Ciphers)

I cifrari a blocchi operano su blocchi di dati di dimensione fissa (es. 64, 128 bit). Per un blocco di n bit, esistono 2^n possibili input e 2^n possibili output. Un cifrario a blocchi ideale sarebbe una permutazione scelta casualmente tra le $(2^n)!$ possibili, ma memorizzare una tale mappatura in una tabella (codebook) è impraticabile per valori realistici di n (es. per $n = 64$, la tabella richiederebbe 64×2^{64} bit). Per questo motivo, i cifrari a blocchi moderni sono costruzioni algoritmiche che approssimano una permutazione pseudo-casuale attraverso la ripetizione di round composti da operazioni più semplici.

La Rete di Feistel Ideata da Horst Feistel in IBM, è una struttura generica per la costruzione di cifrari a blocchi invertibili.

- **Struttura:** Ad ogni round, il blocco di input viene diviso in due metà, Sinistra (L) e Destra (R). La trasformazione è definita come segue:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

  dove F è una funzione di round non necessariamente invertibile e K_i è la sottochiave del round i.

- **Invertibilità:** La genialità della struttura di Feistel è che è intrinsecamente invertibile, indipendentemente dalla funzione F. La decifratura si ottiene semplicemente eseguendo lo stesso algoritmo, ma applicando le sottochiavi K_i in ordine inverso.

DES (Data Encryption Standard) Adottato come standard federale statunitense nel 1977, il DES è stato il cifrario di riferimento per oltre vent'anni.

- **Caratteristiche:** È un cifrario a blocchi basato su una rete di Feistel a 16 round. Opera su blocchi di 64 bit e utilizza una chiave di 56 bit effettivi (più 8 bit di parità).
- **Funzione di Round F:** La funzione F del DES implementa i principi di Shannon:
  1. **Espansione:** La metà destra del blocco (32 bit) viene espansa a 48 bit.
  2. **Mescolamento con la chiave:** Il blocco espanso viene combinato tramite XOR con la sottochiave di round da 48 bit.
  3. **S-Box (Sostituzione):** I 48 bit risultanti vengono suddivisi in 8 blocchi da 6 bit, ciascuno dei quali entra in una diversa S-box, che produce un output di 4 bit. Le S-box sono l'unico componente non lineare del DES e forniscono la **confusione**. Se fossero lineari, l'intero cifrario diventerebbe un sistema di equazioni lineari facilmente risolvibile.
  4. **P-Box (Permutazione):** I 32 bit in uscita dalle S-box vengono permutati da una P-box, che fornisce la **diffusione**.
- **Debolezze e Sostituzione:** La debolezza principale del DES è la sua chiave a 56 bit, che oggi è vulnerabile ad attacchi a forza bruta. Per estenderne la vita, è stato introdotto il **Triple DES (3DES)**, che applica l'algoritmo DES tre volte con due o tre chiavi diverse (modalità Encrypt-Decrypt-Encrypt), offrendo una sicurezza effettiva di circa 112 bit. L'attacco *meet-in-the-middle* dimostra che il Double DES non offre un aumento significativo della sicurezza.

AES (Advanced Encryption Standard) Selezionato nel 2001 per sostituire il DES, AES è l'attuale standard di crittografia simmetrica.

- **Caratteristiche:** Opera su blocchi di 128 bit e supporta chiavi da 128, 192 o 256 bit. Il numero di round varia in base alla lunghezza della chiave (10, 12 o 14).
- **Struttura:** A differenza del DES, l'AES non è una rete di Feistel, ma una **rete di Sostituzione-Permutazione (SPN)**. In una SPN, le operazioni vengono applicate all'intero blocco di dati in ogni round, e ogni operazione deve essere invertibile per permettere la decifratura.
- **Operazioni del Round:** I dati vengono rappresentati come una matrice di byte 4x4 chiamata *state*. Ogni round (eccetto l'ultimo) consiste in quattro trasformazioni:
  1. `SubBytes`: Una sostituzione non lineare in cui ogni byte viene sostituito utilizzando una S-box predefinita.
  2. `ShiftRows`: Una trasposizione in cui le righe della matrice *state* vengono traslate ciclicamente di un offset diverso.
  3. `MixColumns`: Un'operazione di mescolamento che opera sulle colonne della matrice, fornendo diffusione all'interno di ogni colonna.
  4. `AddRoundKey`: La matrice *state* viene combinata tramite XOR con la sottochiave del round corrente.

Le modalità operative (ECB, CBC, CFB, ecc.), utilizzate per cifrare messaggi più grandi di un singolo blocco, si applicano anche ad AES.

9.7.2 Cifrari a Flusso (Stream Ciphers)

A differenza dei cifrari a blocchi, i cifrari a flusso operano su singoli bit o byte di dati alla volta.

- **Meccanismo:** Un generatore di numeri pseudocasuali produce una sequenza di bit chiamata **keystream**, a partire da una chiave segreta. Il keystream viene quindi combinato con il flusso del testo in chiaro tramite un'operazione XOR bit a bit per produrre il testo cifrato.
$$C_i = P_i \oplus K_i$$
La decifratura è la stessa operazione, poiché l'XOR è la propria inversa:
$$P_i = C_i \oplus K_i$$

- **Vantaggi:** Sono generalmente molto più veloci dei cifrari a blocchi e non richiedono padding. Sono ideali per applicazioni in tempo reale dove i dati arrivano come un flusso continuo (es. comunicazioni wireless).

- **Sicurezza:** La sicurezza di un cifrario a flusso dipende interamente dall'impredicibilità del keystream. Il keystream deve avere un periodo molto lungo e buone proprietà statistiche, simili a una sequenza veramente casuale. È fondamentale **non riutilizzare mai la stessa chiave** (o la stessa coppia chiave/vettore di inizializzazione), altrimenti un attaccante può calcolare l'XOR dei due testi in chiaro e compromettere la confidenzialità.

- **Costruzioni:**
  - **LFSR (Linear Feedback Shift Register):** Una costruzione hardware classica per generare sequenze. Da soli sono insicuri a causa della loro linearità e vulnerabili all'algoritmo di Berlekamp-Massey. I cifrari moderni li usano in combinazioni non lineari.
  - **RC4:** Storicamente molto popolare (usato in WEP, SSL), ma oggi considerato insicuro a causa di diversi bias statistici nel suo keystream.
  - **Salsa20 e ChaCha20:** Cifrari di flusso moderni e sicuri, basati su operazioni **ARX** (Add-Rotate-XOR) che forniscono eccellente diffusione e non linearità. ChaCha20 è una variante di Salsa20 ed è ampiamente utilizzato oggi, ad esempio in TLS 1.3.

9.8 Scambio di Chiavi e Crittografia Asimmetrica

9.8.1 Accordo di Chiavi Diffie-Hellman

È un protocollo che permette a due parti, Alice e Bob, di stabilire una chiave segreta condivisa comunicando su un canale insicuro, senza mai trasmettere la chiave stessa. La sua sicurezza si basa sulla difficoltà computazionale del **problema del logaritmo discreto**.

Definizione 4 (Protocollo Diffie-Hellman)**.** *1. **Parametri Pubblici:** Alice e Bob si accordano pubblicamente su un grande numero primo p e un generatore g del gruppo moltiplicativo $\mathbb{Z}_p^*$.*

*2. **Segreti Privati:***

- *Alice sceglie un numero segreto privato a e calcola il suo valore pubblico $A = g^a \pmod p$.*
- *Bob sceglie un numero segreto privato b e calcola il suo valore pubblico $B = g^b \pmod p$.*

3. ***Scambio Pubblico:*** *Alice invia A a Bob, e Bob invia B ad Alice. Un eventuale intercettatore (Eve) conosce p, g, A, B.*

4. ***Calcolo della Chiave Condivisa:***

   - *Alice calcola il segreto condiviso $S = B^a \pmod p = (g^b)^a \pmod p = g^{ab} \pmod p$.*
   - *Bob calcola il segreto condiviso $S = A^b \pmod p = (g^a)^b \pmod p = g^{ab} \pmod p$.*

   Entrambi ottengono lo stesso valore segreto S, che può essere usato come chiave per una successiva comunicazione simmetrica. Per Eve, ricavare a da A (o b da B) è computazionalmente proibitivo.

Vulnerabilità: Il protocollo Diffie-Hellman base è vulnerabile ad attacchi **Man-in-the-Middle (MitM)** perché non fornisce autenticazione delle parti. Un attaccante può interporsi tra Alice e Bob, stabilendo una chiave segreta con ciascuno di essi e agendo da intermediario, inoltrando e leggendo tutti i messaggi. Per questo motivo, nella pratica, viene sempre utilizzato in combinazione con meccanismi di autenticazione, come le firme digitali.

9.8.2 Funzioni One-Way e RSA

La crittografia a chiave pubblica si basa sull'esistenza di **funzioni unidirezionali (one-way function)**: funzioni facili da calcolare in una direzione, ma computazionalmente difficili da invertire.

- **Funzione One-Way con Trapdoor:** È una funzione one-way in cui l'inversione diventa facile se si possiede un'informazione segreta aggiuntiva, la "trapdoor".

Algoritmo RSA L'algoritmo RSA (Rivest-Shamir-Adleman) è l'algoritmo a chiave pubblica più noto. La sua sicurezza si basa sulla difficoltà del **problema della fattorizzazione di grandi numeri interi**.

Definizione 5 (Generazione delle chiavi RSA)**.** 1. *Si scelgono due numeri primi molto grandi, p e q, tenuti segreti.*

2. *Si calcola il modulo $n = p \cdot q$. La lunghezza di n (es. 2048 bit) definisce la dimensione della chiave. n è pubblico.*

3. *Si calcola la funzione totiente di Eulero: $\phi(n) = (p-1)(q-1)$. Questo valore è segreto.*

4. *Si sceglie un esponente pubblico e (comunemente 65537), tale che $1 < e < \phi(n)$ e $gcd(e, \phi(n)) = 1$.*

5. *Si calcola l'esponente privato d come l'inverso moltiplicativo di e modulo $\phi(n)$:*

$$d \cdot e \equiv 1 \pmod{\phi(n)}$$

   d è la chiave privata segreta.

*La **chiave pubblica** è la coppia (n, e), mentre la **chiave privata** è (n, d). La "trapdoor" è la conoscenza dei fattori p e q, che permette di calcolare $\phi(n)$ e quindi d.*

Definizione 6 (Cifratura e Decifratura RSA)**.** *Dato un messaggio M rappresentato come un numero intero ($0 \leq M < n$):*

- ***Cifratura:*** $C = M^e \pmod n$
- ***Decifratura:*** $M = C^d \pmod n$

La correttezza è garantita dal Teorema di Eulero.

9.9 Funzioni di Hash Crittografiche

Le funzioni di hash sono trasformazioni matematiche che prendono un input di lunghezza arbitraria e producono un output a lunghezza fissa, detto **digest**, **hash** o impronta digitale.

Proprietà Fondamentali Per essere crittograficamente sicura, una funzione di hash $h(M)$ deve possedere le seguenti proprietà:

1. **Resistenza alla Pre-immagine (Preimage Resistance):** Dato un digest y, deve essere computazionalmente infattibile trovare un messaggio M tale che $h(M) = y$. Questa è la proprietà di essere *one-way*.
2. **Resistenza alla Seconda Pre-immagine (Second Preimage Resistance):** Dato un messaggio M_1, deve essere infattibile trovare un altro messaggio $M_2 \neq M_1$ tale che $h(M_1) = h(M_2)$.
3. **Resistenza alle Collisioni (Collision Resistance):** Deve essere infattibile trovare una qualsiasi coppia di messaggi distinti M_1, M_2 tali che $h(M_1) = h(M_2)$.

Inoltre, deve esibire l'**effetto valanga**: un cambiamento minimo nell'input (anche un solo bit) deve produrre un cambiamento radicale e imprevedibile nell'output.

Costruzione di Merkle-Damgård Molte funzioni di hash classiche (come MD5, SHA-1, SHA-2) si basano su questa struttura iterativa:

1. Il messaggio di input viene suddiviso in blocchi di dimensione fissa. All'ultimo blocco viene aggiunto un **padding** che include la lunghezza del messaggio originale.
2. Una **funzione di compressione** f viene applicata iterativamente. Per ogni blocco M_i, calcola un nuovo stato interno $H_i = f(H_{i-1}, M_i)$, dove H_{i-1} è lo stato del blocco precedente.
3. Il digest finale è il risultato dell'ultima applicazione della funzione di compressione.

Il teorema di Merkle–Damgård afferma che se la funzione di compressione f è resistente alle collisioni, allora anche la funzione di hash risultante lo sarà.

10 Introduzione alla Blockchain

La tecnologia Blockchain è un registro digitale (*ledger*) distribuito, immutabile e condiviso tra i partecipanti di una rete. La sua funzione principale è quella di permettere la registrazione di transazioni in modo sicuro e trasparente, senza la necessità di un'autorità centrale o di un intermediario fidato (es. una banca).

- **Distributed Ledger:** A differenza di un registro centralizzato, un registro distribuito è replicato e memorizzato su più nodi della rete. Ogni partecipante possiede una copia del registro.
- **Immutabilità:** Una volta che una transazione è stata registrata su un blocco e aggiunta alla catena, non può essere alterata o cancellata. Questo è garantito da meccanismi crittografici.
- **Consenso:** Affinché un nuovo blocco di transazioni sia aggiunto al registro, i partecipanti della rete devono raggiungere un accordo collettivo, noto come **consenso**.

Una problematica fondamentale che la blockchain risolve è il **problema del double spending** (doppia spesa), ovvero la possibilità di spendere la stessa moneta digitale più di una volta. La blockchain previene questo assegnando un ordine temporale univoco e verificabile a tutte le transazioni.

11 Struttura della Blockchain

Una blockchain è una catena di blocchi, dove ogni blocco contiene un insieme di transazioni, un timestamp e un riferimento crittografico al blocco precedente.

11.1 Blocchi e Funzioni di Hash

Ogni blocco è legato a quello che lo precede tramite un **hash crittografico**. L'hash dell'header del blocco precedente è incluso nel blocco corrente, creando una catena sequenziale e cronologica.

$$\text{Blocco}_i = (\text{Transazioni}_i, \text{Timestamp}_i, \text{Nonce}_i, H(\text{Header}_{i-1}))$$

Questa struttura garantisce l'immutabilità: la modifica di una transazione in un blocco passato altererebbe il suo hash, invalidando a cascata tutti i blocchi successivi.

Teorema 4 (Attacco del Compleanno)**.** *Per una funzione di hash che produce un output di N bit, un attaccante necessita di calcolare solo circa $2^{N/2}$ hash per avere una probabilità superiore al 50% di trovare una collisione (due input diversi con lo stesso hash). Per questo, le funzioni di hash crittografiche come SHA-256 (usata da Bitcoin) utilizzano output sufficientemente lunghi (256 bit) per rendere tali attacchi impraticabili.*

11.2 Alberi di Merkle (Merkle Trees)

All'interno di un blocco, le transazioni sono organizzate in un **Albero di Merkle**.

- **Costruzione:** Ogni transazione viene "hashata" (foglie dell'albero). Le foglie vengono raggruppate a coppie e l'hash della loro concatenazione viene calcolato, ripetendo il processo fino a ottenere un singolo hash, la **Merkle Root**.

$$H_{AB} = H(H_A || H_B)$$

- **Efficienza e Verifica Semplificata (SPV):** La Merkle Root, inclusa nell'header del blocco, riassume tutte le transazioni. Ciò permette di verificare l'inclusione di una transazione scaricando solo l'header del blocco e una piccola prova crittografica (*Merkle path*), con una dimensione logaritmica rispetto al numero di transazioni: $|\pi| = \Theta(\log |D|)$.

12 Meccanismi di Consenso

Il protocollo di consenso è il meccanismo con cui i nodi della rete si accordano sullo stato del registro.

12.1 Proof-of-Work (PoW)

È il meccanismo di consenso introdotto da Bitcoin.

- **Funzionamento:** Per aggiungere un blocco, i nodi (*miners*) devono risolvere un complesso puzzle crittografico che richiede un'intensa attività computazionale. Il puzzle consiste nel trovare un valore, chiamato **nonce**, tale che l'hash dell'header del blocco sia inferiore a un certo valore target.

$$H(\text{nonce}||\text{dati del blocco}) \leq \text{Target}$$

- **Sicurezza e Attacco del 51%:** La sicurezza si basa sulla **regola della catena più lunga**, considerata quella con il maggior lavoro computazionale accumulato. Per alterare la cronologia, un attaccante dovrebbe ricalcolare la PoW per tutti i blocchi successivi più velocemente del resto della rete, un'impresa che richiederebbe più del 50% della potenza di calcolo totale (**attacco del 51%**).

- **Svantaggi:** L'enorme consumo di energia è il suo principale svantaggio.

12.2 Consenso di Nakamoto e Blockchain Forking

Il protocollo di Bitcoin, noto come Consenso di Nakamoto, gestisce i conflitti tramite la regola della catena più lunga.

- **Processo di Consenso:** Le transazioni vengono trasmesse, raccolte in blocchi e i miners competono per trovare la PoW. Il primo che ci riesce trasmette il suo blocco, che viene verificato e aggiunto alla catena dagli altri nodi.

- **Blockchain Forking (Biforcazione):** Se due miners trovano una PoW quasi contemporaneamente, la rete si divide temporaneamente in due rami. Il conflitto si risolve non appena viene minato il blocco successivo: la catena che si allunga per prima diventa quella valida, mentre il ramo più corto viene scartato.

12.3 Proof-of-Stake (PoS)

È un'alternativa a PoW più efficiente dal punto di vista energetico.

- **Funzionamento:** Il diritto di creare un nuovo blocco è proporzionale alla quantità di criptovaluta che un nodo possiede e "mette in gioco" (*stake*). I nodi che convalidano i blocchi sono chiamati *validators*.

- **Svantaggi:** Rischio di centralizzazione della ricchezza ("rich get richer") e il problema del "nothing-at-stake", dove i validatori non hanno disincentivi a votare su rami multipli in caso di fork.

12.4 Practical Byzantine Fault Tolerance (PBFT)

Algoritmo di consenso per blockchain private o *permissioned*, progettato per tollerare guasti bizantini.

- **Requisito di Sicurezza:** Raggiunge il consenso se il numero di nodi bizantini F è inferiore a un terzo del totale: $N \geq 3F + 1$.

- **Processo:** Un leader (*primary*) propone un blocco e i nodi eseguono un processo di voto in tre fasi per accordarsi. Garantisce una finalità immediata e rende i fork quasi impossibili (*Safety-focused*).

- **Teorema CAP:** PBFT privilegia la **Consistenza** (CP), mentre il consenso di Nakamoto privilegia la **Disponibilità** (AP), raggiungendo una consistenza eventuale.

13 Classificazione delle Blockchain

Tipo	**Accesso in Lettura**	**Diritto di Scrittura/Consenso**	**Esempi**
Public Permissionless	Pubblico	Chiunque può partecipare	Bitcoin, Ethereum
Private Permissioned	Ristretto	Solo nodi autorizzati da un'organizzazione	Hyperledger Fabric
Consortium	Ristretto	Un gruppo preselezionato di nodi (consorzio)	Corda, Quorum