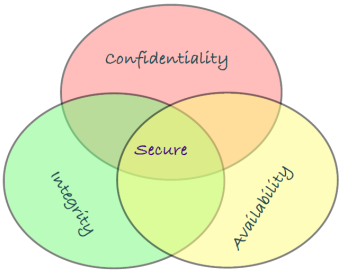
Security Policies

La sicurezza deve godere di tre proprietà principali, rispetto a quelli che sono gli assets (ovvero gli oggetti di valore) rispetto al sistema, ovvero:

1. **Confidenzialità** → garantisce che solo chi è autorizzato possa accedere alla risorsa. Quindi la confidenzialità va a controllare gli accessi;
2. **Integrità** → mi assicura quanto mi possa fidare, ovvero quant'è autentico, il dato e/o il codice;
3. **Disponibilità** → mi assicura che i dati/i servizi siano presenti ed accessibili.

Nell'intersezione di queste tre proprietà vi è quindi la sicurezza, la quale punta a trovare il giusto bilanciamento tra questi tre aspetti.



Quindi, **le security policies cosa fanno?** Vanno a definire **cosa vuol dire essere sicuro rispetto ad uno dei tre aspetti** visti sopra (Confidenzialità, Integrità e Disponibilità) → vi saranno, quindi, policies che mettono l'accento sulla confidenzialità, altre invece sull'integrità ed altre ancora sulla disponibilità. La policy, quindi, va a definire che cosa si intende per sicurezza sul sistema in considerazione e di conseguenza, **le policies sono un insieme di regole e requisiti di alto livello, che devono partizionare gli stati del sistema in sicuri e non sicuri** → naturalmente, più le policies sono formali, meno saranno ambigue e di conseguenza sarà più facile capire cosa dicono ed implementarle.

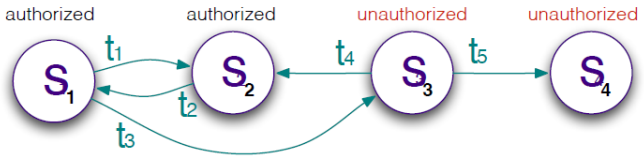
Tipicamente, le security policies sono definite su un **modello**, ovvero: abbiamo detto sopra, che lo scopo delle security policies è quello di partizionare gli stati del sistema in sicuri e non sicuri. Questo significa, andare a prendere il sistema e se ne creerà un modello, il quale sarà un'astrazione più o meno formale del sistema, e su tale modello andremo a definire la policy di sicurezza. Il modello, in generale, descriverà tutti i possibili comportamenti del sistema, ovvero tutte le azioni che un utente può

fare nel sistema → il modello di un sistema, quindi, non ha nulla a che fare con la sicurezza, in quanto descrive tutti i possibili comportamenti del sistema ed è successivamente la policy che etichetta gli stati o i possibili comportamenti come sicuri e non sicuri → se il modello è sufficientemente formale e la policy può essere definita su tale modello, è possibile che riusciamo a dimostrare la soddisfacibilità della policy sul modello.

Per esempio, se noi ci immaginiamo un modello, in cui utilizziamo un automa e quindi un grafo con:

- dei **nodi** → i quali possono rappresentare i possibili stati in cui il sistema può trovarsi;
- delle **transizioni** → ovverosia i cambi di stati, che vengono modellati come degli archi fra gli stati, che ci indicano come può evolvere uno stato.

possiamo trovarci in una situazione simile alla figura sotto:



il seguente modello ha quattro stati e delle transizioni, che portano da uno stato all'altro. La policy di sicurezza potrebbe etichettare questi stati come sicuri e non sicuri e in questo caso:

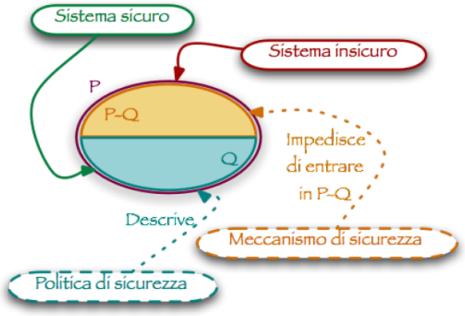
- S1 e S2 → etichettati come sicuri e quindi autorizzati;
- S3 e S4 → etichettati come non sicuri e quindi non autorizzati.

Il meccanismo di sicurezza, quindi, deve impedire la transizione da S1 ad S3, ovvero la transizione che collega gli stati sicuri da quelli non sicuri. Impedendo questa transizione, riusciamo ad assicurare che il sistema permanga in uno stato sicuro e quindi autorizzato. Il concetto, quindi è:

- ho un sistema e lo modello → il modello può essere un automa;
- su questo automa posso andare a definire, con dei criteri, gli stati sicuri e gli stati non sicuri;
- capire quali sono le transizioni da bloccare, al fine di non permettere di raggiungere stati non autorizzati.

- Lo stato di un sistema raccoglie i valori correnti memorizzati in memoria, i registri di memoria, chi ha accesso a cosa, quali programmi sono in esecuzione, ecc. Lo stato, quindi, fornisce **un'istantanea** del sistema in un determinato istante di tempo.
- Il modello è sempre **un'astrazione** del sistema reale.

Lo scopo della policy di sicurezza, quindi, è creare una linea/separazione netta tra ciò che è permesso da ciò che non è permesso in modo **non ambiguo** → deve, quindi, essere chiaro se uno stato è autorizzato o non autorizzato e quindi per non avere ambiguità, le policy devono essere espresse in modo formale. I meccanismi di sicurezza, poi, devono garantire che il sistema si muovi solo negli stati sicuri, ovvero devono impedire al sistema, in un qualche modo, il raggiungimento di uno stato non sicuro.



→ è chiaro che certe policy le possiamo esprimere sulla **storia del sistema** (detta anche **traccia**) **e che quindi riguardano proprietà temporali**. Per esempio, possiamo dire che non vorremmo mai che venga aggiunto un file, che precedentemente era stato cancellato → avremo, quindi, un modello che mi va a definire l'insieme delle tracce e poi saranno le tracce ad essere classificate come sicure o non sicure, in base alle proprietà temporali che siamo andati a definire.

Vi sono diverse tipologie di security policies:

- **Military security policy** → politica di sicurezza sviluppata principalmente per garantire la riservatezza quando le informazioni sono classificate come:
  - non classificate;
  - limitate;
  - classificate;
  - segrete;

    - top-secret.
- **Commercial security policy** → politica di sicurezza sviluppata primariamente per fornire integrità;
- **Confidentiality policy** → politica di sicurezza che si occupa solo di riservatezza;
- **Integrity policy** → politica di sicurezza che si occupa esclusivamente di integrità.

Access Control

Tanti esempi che abbiamo fatto, quando abbiamo parlato di integrità, confidenzialità e disponibilità, di fatto si possono ricondurre ad un problema di chi fa cosa sugli oggetti di un sistema, ovvero chi può:

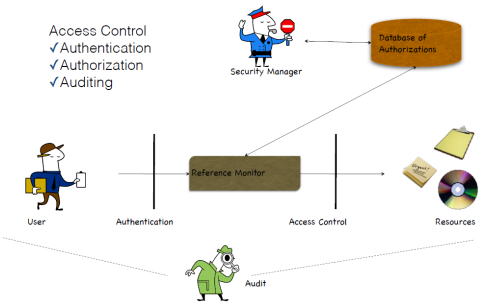
- leggere;
- scrivere;
- modificare;
- eseguire;
- cancellare.

Quindi, i meccanismi di controllo degli accessi sono alla base del controllo della sicurezza, quindi sono alla base della confidenzialità, dell'integrità e della disponibilità degli assets di un sistema → da porre attenzione. al fatto che i meccanismi di controllo degli accessi prevedono una frase di autenticazione prima, ovvero parliamo di controllo degli accessi nel momento in cui vado a regolamentare chi può fare cosa sull'oggetto del sistema, ma quel chi deve essere innanzitutto identificato → quindi, quando si parla di controllo degli accessi, si assume che la fase di autenticazione sia stata gestita dai meccanismi di autenticazione.

I meccanismi di controllo degli accessi vengono utilizzati quotidianamente e sono formati da tre "passaggi":

1. L'**access control policy** → andrà a definire chi può accedere a cosa. La policy, in base alla sicurezza che si vuole ottenere, viene definita sul modello;
2. L'**access control model** → modello su cui vado a definire la policy di sicurezza;
3. L'**access control mechanism** → meccanismo che va ad implementare la policy di sicurezza.

Per vedere meglio questi concetti, vediamo l'immagine sotto:



Abbiamo l'utente che vuole accedere ad una qualche risorsa del sistema e quindi per prima cosa si deve autenticare nel sistema. Una volta che l'utente è stato autenticato dal sistema, quest'ultimo andrà a controllare se l'utente ha o meno i diritti per accedere alla risorsa d'interesse. Quindi vi sarà un meccanismo di controllo degli accessi, che consentirà o meno l'azione all'utente. Vediamo, inoltre, che in basso nella figura abbiamo la fase di **audit**, che va tenere traccia di tutto quello che succede → si ha questa fase, perchè nel caso avvengano degli accessi non autorizzati, si riesce a capire da chi sono stati commessi e dov'è avvenuto → questo ci fa capire, che nella fase di audit bisogna fare due cose fondamentali:

1. **collezionare** informazioni → e quindi capire quali informazioni collezionare rispetto a tutto quello che avviene nel sistema ed organizzare tale informazioni per verificarle;
2. **analizzare** i dati raccolti → al fine di individuare delle violazioni di sicurezza → quando si analizzano i dati raccolti, vi sono due approcci:
    a. anomaly detection → consiste nel riconoscere i comportamenti anomali rispetto a qualcosa che viene definito normale e quindi allenando il sistema a riconoscere delle anomalie/singolarità;
    b. andare a definire cosa non deve succedere e quindi riconoscendo che è successo qualcosa, che è stato definito come pericoloso.

Come dicevamo sopra, i meccanismi di controllo degli accessi assumono che l'utente sia già stato autenticato nel sistema (quindi non si preoccupano della fase di autenticazione) ed è bene che i meccanismi di controllo degli accessi siano in grado di:

- controllare l'accesso a tutti gli elementi presenti nel sistema;
- implementare il principio del need-to-know, ovvero dovrebbero ad una risorsa/oggetto del sistema, solo quando l'utente ha effettivamente bisogno di accedere a quella risorsa/oggetto per compiere le proprie azioni nel sistema;

- definire cosa si può e cosa non si può fare, quindi la policy di controllo degli accessi dovrà dire, per ogni elemento, quale soggetto può e chi non può accedere → se ci sono più policy che si combinano, allora si dovrà capire se ci sono dei punti di conflitto ed eventualmente gestirli e risolverli;
- regolamentare i soggetti che possono modificare i permessi di accesso, dato che questi permessi possono variare nel tempo.

Cosa intendiamo con il termine soggetto? **Il soggetto è un'entità in grado di accedere a un oggetto. Generalmente il concetto di soggetto equivale a quello di processo, che accede effettivamente all'oggetto** → di fatto, quindi, i soggetti di un sistema possono essere utenti oppure processi. Tipicamente, i sistemi di controllo degli accessi gestiscono tre classi di soggetti:

- **proprietario** → può essere il creatore di una risorsa/oggetto di cui si deve regolamentare l'accesso;
- **gruppo** → insieme di utenti, che possono godere di privilegi particolari;
- **mondo** → classe di utenti che hanno il più basso livello di privilegi.

Cosa intendiamo, invece, con il termine oggetto? L'oggetto è una risorsa il cui accesso è controllato. È un'entità utilizzata per contenere e/o ricevere informazioni (come per esempio: record, pagine, file, directory, caselle di posta, messaggi).

> 💡 A volte anche un soggetto è un oggetto, nel senso che ad esempio vorrei cancellare un soggetto dal sistema e quindi in un qualche modo anche il soggetto potrebbe essere un oggetto a cui vogliamo avere accesso.

Vi sono poi i diritti di accesso all'oggetto: lettura, scrittura, esecuzione, cancellazione, creazione, concatenazione e ricerca.

Access Control Policies

Andiamo a vedere i quattro approcci ai meccanismi di controllo degli accessi, che sono:

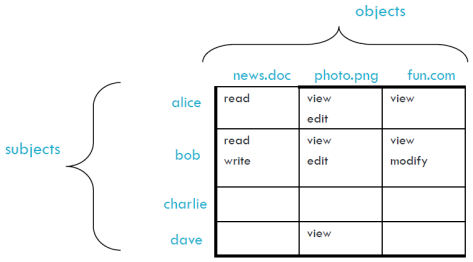
- **Discretionary Access Control (DAC)** → la caratteristica di tale approccio, è che gli utenti o i soggetti sono i **proprietari** delle risorse, ovvero sono loro che stabiliscono chi può e chi non può accedervi → vi è, quindi, una **discrezionalità** dell'utente o del soggetto su chi può e chi non può fare qualcosa. L'accesso ai

dati viene fornito in base all'identità dell'utente, quindi ogni utente è identificato nel sistema → il DAC è un meccanismo **flessibile**, in quanto:

- il proprietario della risorsa/oggetto ha massimi privilegi;
- il proprietario può dare il permesso di accedere alla risorsa ad un altro utente e insieme al permessi d'accesso, può dare anche il permesso di dare ad altri il permesso di accedere alla risorsa → questa flessibilità deve essere gestita e controllata, altrimenti si potrebbe perdere il controllo di chi può accedere alla risorsa.

Tipicamente il DAC viene implementato attraverso una **matrice di controllo degli accessi**, in cui si ha:

- sulle righe si hanno gli utenti;
- una colonna per ogni oggetti;
- nella cella corrispondente si va a dire cosa l'utente può fare sull'oggetto.



Da notare, il fatto che la matrice di controllo degli accessi è essa stessa un oggetto su cui va verificato gli utenti che vi possono accedervi → quindi, devono essere regolamentati gli utenti che vi possono accedervi e scriverci.

Spesso queste matrice sono **molto sparse**, ovvero hanno tante celle vuote (in quanto possono esserci utenti che non possono compiere alcuna azione sull'oggetto) e quindi, in generale, tale matrici **non scalano**. Per cercare, quindi, di limitare questo problema di scalabilità, nel tempo si è deciso di fare due **proiezioni**:

- una proiezione sulle **righe** → e in questo caso si parla di **Access Control List** → esse tengono per ogni oggetto, la lista degli utenti che hanno accesso a quell'oggetto;

- oppure una proiezione sulle **colonne** → e in questo caso si parla di **Capability List** → esse hanno una riga per ogni utente e di conseguenza semplificano molto, ad esempio, l'eliminazione di un utente dal sistema.

Alcuni esempi delle politiche di DAC sono:

- **Graham-Denning Model** → meccanismo di controllo degli accessi con approccio DAC, basato su regole di protezione. In questo modello abbiamo:
  - i soggetti S;
  - gli oggetti O;
  - i permessi R;
  - la matrice M → ha una riga per ogni soggetto e una colonna per ogni oggetto e all'interno delle celle vi saranno i rispettivi diritti.

  Ogni oggetto ha un proprietario, che stabilisce cosa si può fare con quell'oggetto → quindi il proprietario ha dei privilegi speciali sull'oggetto di cui è proprietario. Ogni soggetto, poi, ha un controllore, il quale (= il controllore) ha dei privilegi speciali sull'oggetto e sul soggetto. Vi sono poi 8 privilegi di protezione:
  - creazione/cancellazione dell'oggetto;
  - creazione/cancellazione del soggetto;
  - diritto di lettura dei permessi;
  - diritto di garantire i permessi;
  - diritto di trasferire i permessi;
  - diritto di cancellare i permessi.

  La matrice di questo modello è la seguente:

Rule	Command (from s_0)	Condition	Operation
R1	create o	--	add column for o in M; insert 'owner' in $M[s_0,o]$
R2	delete o	'owner' in $M[s_0,o]$	deletes column for o in M
R3	create s	--	add a row for s in M; insert 'control' in $M[s_0,s]$
R4	delete s	'control' in $M[s_0,s]$	delete row for s in M
R5	read the access right of s on o	'control' in $M[s_0,s]$ or 'owner' in $M[s_0,o]$	copy M[s,o] in s_0
R6	grant r [r*] for s on o	'owner' in $M[s_0,o]$	add r [r*] in M[s,o]
R7	delete r from s on o	'control' in $M[s_0,s]$ or 'owner' in $M[s_0,o]$	remove r in M[s,o]
R8	transfer r[r*] for s on o	'r*' in $M[s_0,o]$	add r[r*] in M[s,o]

La prima regola dice: Il soggetto S0 può creare un oggetto senza alcuna condizione, quindi ogni soggetto può creare liberamente un oggetto e questo corrisponde nell'aggiungere una colonna nella matrice e in corrispondenza del soggetto che l'ha creata, esso verrà indicato come owner.

- **Harrison-Ruzzo-Ullman Model (HRU)** → meccanismo di controllo degli accessi, che permette di costruire dei comandi in modo più flessibile rispetto al Graham-Denning Model, in modo da adattare la policy a diverse situazioni. Attraverso questo meccanismo, quindi, si vanno a creare dei comandi, i quali sono fatti dalla composizione di una o più operazioni primitive → vengono quindi definite delle operazioni primitive, che alterano il controllo degli accessi sulla matrice, che sono:
  - creare un soggetto;
  - creare un oggetto;
  - distruggere un soggetto;
  - distruggere un oggetto;
  - aggiungere un diritto;
  - togliere un diritto.

  Gli stati, poi, vengono definiti come: l'insieme dei soggetti esistenti, l'insieme degli oggetti esistenti e la matrice degli oggetti in quel momento. Le operazioni, allora, si traducono nel seguente modo:

Operation	Conditions	New State
enter r into M[s,o]	s∈S, o∈O	S'=S O'=O M'[s,o]=M[s,o]∪{r} M'[s_1,o_1]=M[s_1,o_1] ∀(s_1,o_1)≠ (s,o)
delete r from M[s,o]	s∈S, o∈O	S'=S O'=O M'[s,o]=M[s,o] ∖ {r} M'[s_1,o_1]=M[s_1,o_1] ∀(s_1,o_1)≠ (s,o)
if s ∉ S or o ∉ O		then (S',O',P')=(S,O,P) and the request fails

Aggiungere un diritto "r" sull'oggetto "o" da parte di un soggetto "s" lo posso fare solamente se "s" ed "o" fanno entrambi parte della matrice e la matrice verrà aggiornata solamente nella cella in cui viene aggiunto il diritto "r" → il resto della matrice rimane invariata. Analogo se vogliamo cancellare un diritto "r".

I comandi, invece, vengono strutturati nel seguente modo:

```
command name (o1,o2,...ok)
    if    r1 in M[s1,o1]
          r2 in M[s2,o2]
          ...
          rm in M[sm,om]
    then
          op1
          op2
          ...
          opn
end
```

r = right
op = primitive operation

ovvero:

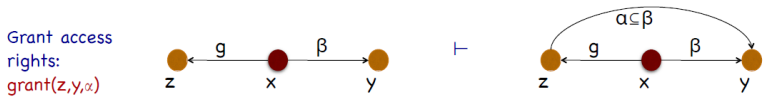
- hanno un nome;
- va ad agire su una serie di oggetti attraverso delle condizioni (ovvero degli "if"), che se soddisfatte comportano una serie di operazioni di base → chiaramente è un **meccanismo più flessibile**, che mi permette di implementare diverse policy (permette di implementare le 8 policy viste sopra e poi sicuramente altre).

A questo punto, il sistema di controllo degli accessi verrà definito attraverso un insieme di comandi, che specificano come si può agire sulla matrice di controllo degli accessi.

- **Take-Grant Model** → approccio più grafico, che prevede solamente quattro operazioni primitive:
  - creazione;
  - cancellazione;
  - take → nel senso di prendere da un altro soggetto i diritti;

- grant → nel senso di dare ad altri soggetti i diritti.

Con questo approccio i soggetti e gli oggetti vengono rappresentati graficamente con dei nodi, mentre i diritti attraverso degli archi che collegano i nodi, che dicono il diritto che la sorgente ha sulla destinazione dell'arco. Per esempio:



Se "x" ha diritti beta su "y", "x" può garantire ad un altro soggetto "z" **parte** (quindi alcuni) **dei diritti** che ha su "y".

Il concetto fondamentali di questo approccio, è che se le regole vengono definite in maniera formale, può essere possibile rispondere e dire chi in futuro potrà avere accesso agli oggetti.

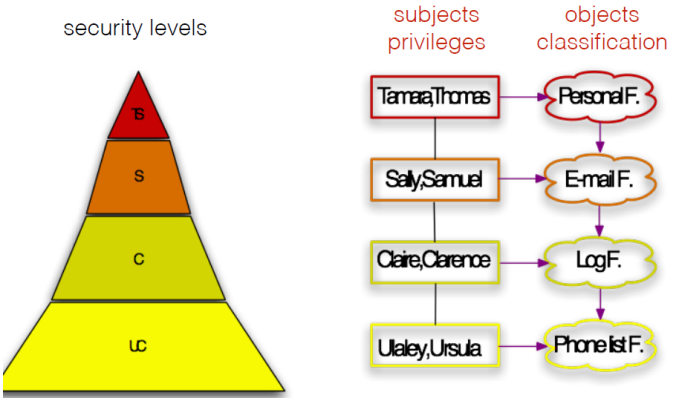
Sorge a questo punto spontanea la domanda: "**Quali sono i limiti del DAC?**" I limiti del DAC sono:

- La gestione di una politica è un compito complesso in un sistema di grandi dimensioni;
- L'insieme dei soggetti o degli oggetti è grande e non scala;
- L'insieme di soggetti o oggetti cambia frequentemente;
- Un meccanismo flessibile è soggetto a errori, in quanto è necessario che tutti gli utenti comprendano il meccanismo e rispettino la politica di sicurezza. Inoltre, non vi è controllo sul flusso di informazioni.

- **Mandatory Access Control (MAC)** → in questo approccio, a differenza del DAC in cui l'utente ha tanto potere, dato che è lui che decide chi ha l'acceso all'oggetto, nel MAC l'utente non ha alcun potere, bensì il potere è tutto nelle mani del sistema → nel MAC vengono classificati gli oggetti a diversi livelli di segretezza:
  - un-clussified;
  - clussified;
  - secret;
  - top-secret:

  e anche agli utenti di sistema viene associato un livello di segretezza, quado essi si autenticano nel sistema. Quindi il sistema agli utenti associa un livello di

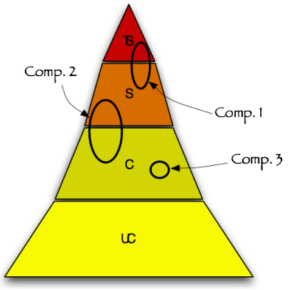
autorizzazione e in base al livello dell'utente e al livello dell'oggetto viene deciso se l'utente può avere accesso o meno all'oggetto → quindi, nel momento in cui il sistema dice che l'utente può accedere, ad esempio al livello di segretezza secret, allora l'utente potrà vedere solamente da quel livello in giù → quindi non vi è alcuna discrezionalità e questo comporta il fatto che è sicuramente **meno flessibile rispetto al DAC, ma è più sicuro**.



Vediamo che Tamara e Thomas possono accedere alle informazioni personali e tutto quello che ci stà sotto, mentre Claire e Clarence non possono accedere alle informazioni personali, ma possono accedere ai file di log e a quello che stà sotto.

Per implementare il principio del need-to-know, oltre ai livelli di segretezza, vengono definiti i **compartimenti** → questo significa, che se io ho accesso alle informazioni top-secret, può darsi che non sia necessario che abbia accesso a tutte le informazioni top-secret, ma solo a quell'informazione top-secret che riguarda un mio progetto.

I compartimenti possono prendere oggetti classificati a livelli diversi di segretezza, ma limitano quello che l'utente può fare → quindi l'utente può accedere alle informazioni presenti nel compartimento di cui fa parte, ma solamente a quelle del suo livello di segretezza.



Quindi, nel momento in cui abbiamo solamente i livelli di segretezza, abbiamo una **catena**, ovvero il soggetto che sta sopra può leggere le informazioni al proprio livello e ai livelli sottostanti, mentre quando combino i livelli di segretezza con i compartimenti, non ho più una catena, bensì ho un **reticolo**.

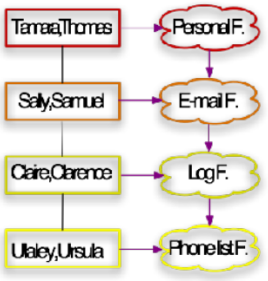
Vediamo adesso due modelli di controllo degli accessi molto importanti, ovvero:

- **Bell-LaPadula** → mette l'accento sulla confidenzialità e combina DAC e MAC;
- **Biba** → mette l'accento sull'integrità.

Il focus del Bell-LaPadula è sulla confidenzialità, ovvero vogliamo controllare chi può leggere le informazioni → per fare questo, il Bell-LaPadula va a definire dei livelli di sicurezza e quindi implementando una MAC, combinandola però con una matrice di controllo degli accessi, quindi con un DAC → quindi, quando si parla di confidenzialità, vogliamo impedire che un soggetto possa accedere ad informazioni che hanno un livello di segretezza più alto del suo, ovvero se il soggetto ha un livello di segretezza basso, non può accedere ad informazioni con un livello di segretezza alto → **vogliamo, quindi, impedire letture verso l'alto**.

Abbiamo detto, che Bell-LaPadula combina MAC e DAC, in quanto con il MAC, il sistema mi dice chi può accedere a cosa e poi possiamo dare accesso ad altri soggetti nel rispetto del MAC e del DAC. Il Bell-LaPadula gode di diverse proprietà:

- **Non si può leggere dal basso verso l'alto**



Ad esempio, Claire non può leggere i dati personali → la prima regola, quindi, dice che un soggetto può leggere un oggetto, solo se il soggetto ha un livello di segretezza più alto. Quindi, un soggetto può leggere ciò che è al suo livello e tutto quello che gli sta sotto → sono quindi proibite le letture verso l'alto.

Può, però, succedere che un soggetto alto (nell'immagine Tamara) potrebbe leggere le informazioni personali, copiarle a livello dei file di log e in questo modo le rende leggibili a chi sta sotto. Questo ci porta alla seconda proprietà

- **Non si può scrivere verso il basso** → un soggetto, che ha accesso ad informazioni personali, non può scrivere ad un livello più basso del suo, perchè altrimenti potrebbe ricopiare le informazioni personali in un file con un basso livello di segretezza e quindi renderle leggibili da chiunque → impedendo di leggere verso l'alto e scrivere verso il basso **si garantisce la confidenzialità delle informazioni**.

> 💡 Se un sistema si trova inizialmente in uno stato sicuro e tutte le transizioni di stato mantengono le due proprietà descritte sopra, allora il sistema si troverà ancora in uno stato sicuro → quindi il sistema può permettere tutte le transizioni che mantengono le due proprietà.

- La terza proprietà del Bell-LaPadula è che un soggetto, che ha un determinato tipo di accesso su un oggetto, lo può trasferire ad un altro soggetto → quindi, **un soggetto può dare ad un altro soggetto l'accesso ad un oggetto, se questo soddisfa (non viola) le MAC rules**.

Un esempio di queste proprietà viene riportato nella seguente immagine:

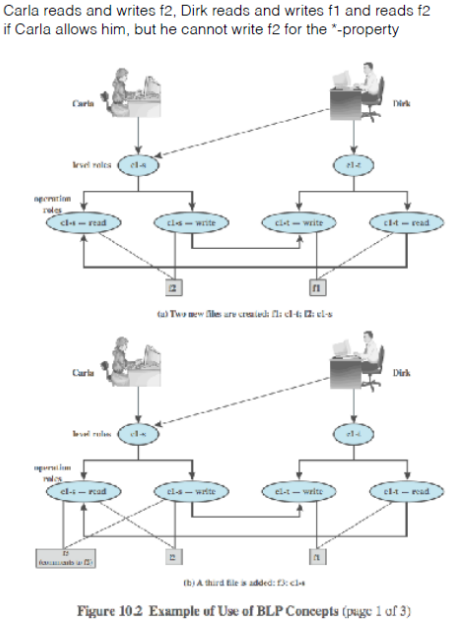


Figure 10.2 Example of Use of BLP Concepts (page 1 of 3)

Dirk reads f2 and wants to create a file with the comments for Carla
Dirk logs as a student and creates f3 so that Carla can read it

I **vantaggi** del Bell-LaPadula (BLP) sono:

- Buone capacità descrittive → il BLP descrive le operazioni di accesso correnti e tutti i permessi di accesso correnti;

- Le politiche di sicurezza si basano su livelli di sicurezza e matrice di controllo degli accessi;
- È facile modificare le strutture per nuove politiche e modificando le proprietà si può gestire l'integrità.

Gli **svantaggi**, invece, sono:

- Il BLP non copre tutti gli aspetti della sicurezza, in quanto tratta solo la riservatezza (non l'integrità);
- Non affronta la gestione del controllo degli accessi;
- Contiene canali nascosti, ovvero un soggetto di basso livello crea un oggetto dummy.obj al proprio livello. Il suo complice di alto livello (detto anche cavallo di Troia) aumenta il livello di sicurezza di dummy.obj ad alto o lo lascia invariato. In seguito, il soggetto di basso livello tenta di leggere dummy.obj. Il successo o il fallimento di questa richiesta rivela l'azione del soggetto di alto livello. Un bit di informazione è stato trasmesso dall'alto al basso.

Giriamo il Bell-LaPadula e creiamo il modello **Biba (quindi il Bell-LaPadula e il Biba sono uno il duale dell'altro)**, che è un modello che ha che fare con l'integrità → l'integrità riguarda l'affidabilità degli oggetti a cui si ha accesso, quindi ha più a che fare con chi può modificare gli oggetti → si ottiene andando a girare le due proprietà del Bell-LaPadula, ovvero le due proprietà del modello Biba sono:

- Un soggetto non può scrivere su un oggetto, che è più in alto come livello di integrità, perchè se il soggetto sta più in basso, significa che il soggetto è meno affidabile dell'oggetto e quindi non vogliamo che il soggetto vada a scrivere su un oggetto con un più alto livello di integrità, in quanto potrebbe compromettere l'affidabilità dell'oggetto → proprietà del **no write-up**.
- Un soggetto che ha un'alta affidabilità non voglio che legga dati poco affidabili, perchè potrebbe diminuire l'affidabilità delle azioni che compie successivamente, in quanto quest'ultime si basano su informazioni poco affidabili → proprietà del **no read-down**.

Una variante del Biba permetteva ad un soggetto di leggere qualsiasi cosa, però se un soggetto legge qualcosa di meno affidabile di lui, allora il suo livello di affidabilità viene diminuito.

- **Role-based Access Control (RBAC)** → il DAC è un'ottimo approccio, se non fosse per il fatto che la matrice non scala. Un'idea per risolvere questo problema è quella di utilizzare i **ruoli** → ovverosia, invece di assegnare i permessi ai

singoli utenti, identifico all’interno del sistema dei ruoli e definisco cosa l’utente può fare, in base al ruolo che è stato assegnato all’utente → il caso ideale di questo approccio è quello di avere tanti soggetti e pochi ruoli.

Nel RBAC un utente può avere più di un ruolo e sicuramente un ruolo può essere assunto da più utenti. Vi sono tre tipologie di RBAC:

1. **RBAC zero** → in cui abbiamo: Utenti, Ruoli e Permessi e i permessi vengono dati ai ruoli → quindi un utente si identifica nel sistema, esso viene identificato con un ruolo e in base al ruolo si hanno determinati permessi;
2. **RBAC uno** → in cui, oltre agli Utenti, Ruoli e Permessi, si hanno anche le sessioni → la sessione è un’associazione temporanea tra utenti e ruoli → ovvero in una sessione mi posso autenticare con un ruolo ed in un’altra sessione con un altro ruolo.
3. **RBAC due** → in cui si aggiungono i Vincoli sulla numerosità o sulla compatibilità.

- **Attribute-based Access Control (ABAC)** → esso guarda, sono solo gli attributi dell’utente, ma anche gli attributi dell’ambiente. Per esempio, può dire che un determinato utente può accedere al sistema solamente in determinate fasce orarie oppure da solamente un determinato dispositivo, come per esempio un computer aziendale piuttosto che il proprio smartphone. Quindi, con questo modello, oltre ad avere il soggetto (che ha dei suoi attributi), l’oggetto (che ha dei suoi attributi) abbiamo anche l’ambiente.