Complementi di sicurezza e privatezza

Sicurezza e privatezza (Università degli Studi di Milano)

Complementi di sicurezza e privatezza

Ilaria Salbe

Appunti per il CdL Magistrale:
Sicurezza Informatica

Università degli Studi di Milano
Anno Accademico 2019/2020

Indice

1 Controllo dell'accesso

Il controllo dell'accesso è un meccanismo di sicurezza che ha come obbiettivo quello di stabilire, data una richiesta di accesso ad una specifica risorsa del mio sistema, se tale accesso può essere concesso o meno. Questo meccanismo si basa sull'identità dell'utente che richiede l'accesso e sulla presenza di determinate regole di accesso presenti nel sistema. Può essere limitato al solo accesso diretto e può essere arricchito con controlli di inferenza, flusso di informazioni e controlli di non interferenza. La correttezza del meccanismo di controllo si basa su:

- Identificazione e autenticazione propria dell'utente: nessuno deve essere in grado di acquisire privilegi diversi dai propri.
- Correttezza delle autorizzazioni rispetto alle quali viene valutato l'accesso (che deve essere protetto da modifiche improprie).

L'autenticazione del richiedente è importante per mantenere il concetto di accountability (responsabilità). Attraverso il concetto di accountability è possibile stabilire con certezza chi ha causato un determinato evento in un sistema. Ciò è fondamentale per stabilire le responsabilità in caso di un'azione impropria o dannosa. Quindi, ogni soggetto autenticato nel sistema deve corrispondere ad un singolo utente: non è possibile utilizzare account condivisi, altrimenti non avrei assicurata la proprietà di accountability. Nei sistemi aperti, il controllo degli accessi dovrebbe far affidamento sull'autenticità delle informazioni, in contrasto con l'autenticità dell'identità (autenticazione). Nello studio del controllo degli accessi, è utile separare tre concetti:

- Politica: definisce linee guida (ad alto livello) e regole che descrivono gli accessi che devono essere autorizzati dal sistema (ad esempio, politiche chiuse o aperte). Le politiche aperte permettono qualunque operazione, a meno che non sia presente una politica esplicita che stabilisca il contrario. Mentre, nelle politiche chiuse le richieste vengono valutate attraverso le regole esplicite: se c'è una regola che permette esplicitamente un'operazione, la richiesta viene accettata; altrimenti viene respinta. Spesso il termine politica viene abusato e utilizzato per fare riferimento alle autorizzazioni effettive (ad esempio, i dipendenti possono leggere la bacheca).
- Modello: definisce formalmente le specifiche e l'applicazione del controllo di accesso. É importante perchè permette di dimostrare che un sistema è sicuro. Con sicurezza si intende un livello che viene stabilito.
- Meccanismo: implementa le politiche tramite funzioni di basso livello (software e hardware). Quindi, si tratta di funzioni per implementare le politiche di accesso.

La separazione di questi tre concetti, in particolare, quella tra meccanisimi e politiche ci permette di:

- Discutere i requisiti di accesso indipendentemente dalla loro implementazione
- Confrontare diverse politiche di controllo degli accessi, così come diversi meccanismi, che applicano la stessa policy.
- Progettare meccanismi in grado di applicare più politiche. É importante, perchè se un meccanismo è strettamente legato ad una politica, fa sì che se la politica subisse una modifica bisognerebbe modificare il meccanismo corrispondente.

Il meccanismo di controllo dell'accesso per poter funzionare all'interno di un sistema tipicamente fa riferimento al reference monitor, un componente trusted attraverso il quale passa qualunque richiesta di accesso diretta al sistema. Il reference monitor deve soddisfare le seguenti proprietà:

- Tamper-proof: non dovrebbe essere possibile modificarlo. Inoltre, i tentativi di modifica vengono identificati.
- Non-bypassabile: tutte le richieste devono passare dal reference monitor. Quindi, media tutti gli accessi al sistema e alle sue risorse.
- Security kernel: deve essere confinato in una parte limitata del sistema (la diffusione delle funzioni di sicurezza in tutto il sistema implica la verifica di tutto il codice). In parole semplice, se io avessi diverse funzioni di controllo sparse nel sistema, significa che sono presenti pezzi di codice sparpagliati a cui devo sottoporre la richiesta e, di conseguenza sarebbe necessario estendere i controlli di sicurezza ad una porzione più grande del sistema.
- Abbastanza piccolo da essere suscettibile di rigorosi metodi di verifica.

L'implementazione di un meccanismo corretto è lungi dall'essere banale ed è complicata dalla necessità di far fronte a due problematiche:

- Canali di archiviazione (Storage channel): Gli elementi di archiviazione come pagine di memoria e settori del disco devono essere cancellati prima di essere rilasciati a un nuovo soggetto, per evitare la raccolta di dati. Ci sono elementi memorizzati nel sistema che devono essere ripuliti prima che vengano utilizzati, attraverso qualunque azione, da un altro soggetto.
- Canali nascosti (Covert channel): Canali non destinati al trasferimento di informazioni (ad es. Effetto del programma sul carico del sistema) che possono essere sfruttati per inferire informazioni. Si tratta di canali di comunicazione illegittimi, utilizzati per far fluire informazioni sensibili verso l'esterno

Come è possibile essere sicuri di aver realizzato un meccanismo in grado di far fronte a queste problematiche? Sono stati definiti dei principi che devono essere presi in considerazione ogni qualvolta si realizza un meccanismo di sicurezza basati sul concetto di minimizzare le interazioni tra le componenti presenti nel sistema, sulla semplicità dell'implementazione e sul fatto che questi meccanismi dovrebbero essere facilmente comprensibili e analizzabili:

- Economy of mechanism: il meccanismo di protezione dovrebbe avere un design semplice e piccolo. Ridurre al minimo la complessità del meccanismo permette di ridurre la possibilità di errore.
- Fail-safe defaults: il meccanismo di protezione dovrebbe negare l'accesso per impostazione predefinita e concedere accesso solo quando esiste un'autorizzazione esplicita. Rappresenta il concetto di politca chiusa.
- Mediazione completa: il meccanismo di protezione dovrebbe controllare ogni accesso a ciascun oggetto. Tutte le richieste di accesso, nel caso specifico, devono essere controllate, per stabilire che siano effettivamente permesse, anche quando le richieste sono ripetute.
- Open design: il meccanismo di protezione non dovrebbe dipendere dal fatto che gli attaccanti ignorino il suo progetto per avere successo; può tuttavia essere basato sull'ignoranza dell'attaccante di informazioni specifiche come password o chiavi di cifratura.
- Separazione dei privilegi: il meccanismo di protezione dovrebbe consentire l'accesso in base a più di un'informazione o condizione.
- Least privilege : il meccanismo di protezione dovrebbe forzare ogni processo a funzionare con i privilegi minimi necessari per svolgere il suo compito.

- Least common mechanism: il meccanismo di protezione dovrebbe essere condiviso il meno possibile tra gli utenti, perché se viene condiviso può essere usato per trasferire informazioni tra utenti che lo condividono.
- Psychological acceptability: il meccanismo di protezione dovrebbe essere facile da usare (almeno altrettanto facile come non usarlo).

Il processo di sviluppo di un meccanismo di controllo degli accessi è un approccio multifase, che parte dalla definizione di politiche fino ad arrivare allo sviluppo del meccanismo vero e proprio. Durante questo processo si passa da una fase di modellizzazione. Il modello di controllo degli accessi definisce formalmente le specifiche e l'applicazione del controllo degli accessi. Il modello deve essere:

- completo: dovrebbe comprendere tutti i requisiti di sicurezza che devono essere rappresentati.
- coerente: privo di contraddizioni; ad esempio, non può negare e concedere contemporaneamente un accesso.

La definizione di un modello formale consente la prova delle proprietà sul sistema. Dimostrando che il modello è "sicuro" e che il meccanismo implementa correttamente il modello, possiamo sostenere che il sistema è "sicuro" (secondo la definizione data di sicuro).

Le politiche di sicurezza possono essere distinte in:

- Criteri di controllo dell'accesso (Access Control Policies), che permettono di definire chi può (o non può) accedere alle risorse. Sono formati da tre classi principali:
  - Politiche discrezionali (DAC)
  - Politiche obbligatorie (MAC)
  - Politiche basate sul ruolo (RBAC)
- Politiche amministrative: definire chi può specificare autorizzazioni (o regole di accesso) che regolano il controllo degli accessi. Possono essere accoppiate con politiche DAC e RBAC.

1.1 Controllo dell'accesso: DAC, MAC, RBAC

1.1.1 Politiche discrezionali (DAC)

Le politiche discrezionali sono politiche che applicano il controllo dell'accesso in base a:

- l'identità dei richiedenti (o sulle proprietà che hanno).
- regole di accesso esplicite che stabiliscono chi può o non può eseguire quali azioni su quali risorse.

Sono chiamate discrezionali in quanto agli utenti può essere data la possibilità di trasferire i loro diritti ad altri utenti (concessione e revoca dei diritti regolati da una politica amministrativa). Uno dei primi modelli per le politiche discrezionali conste nella matrice di accesso, la quale fornisce un framework per la descrizione dei sistemi di protezione. É spesso riportato come modello HRU (dalla successiva formalizzazione da parte di Harrison, Ruzzo e Ullmann). Viene chiamata matrice di accesso poiché lo stato di autorizzazione (o sistema di protezione) è rappresentato come una matrice. La matrice di accesso è una rappresentazione astratta del sistema di protezione presente nei sistemi reali (molti sistemi successivi possono essere classificati in base alla matrice di accesso). Lo stato di un sistema può essere definito mediante una tripla (S, O, A), dove:

- S: insieme dei soggetti (chi può esercitare privilegi)
- O: insieme degli oggetti (sui quali, i privilegi possono essere esercitati). I soggetti possono essere considerati come oggetti, nel qual caso $S \subseteq O$.
- A: matrice di accesso, dove:
  - le righe corrispondono ai soggetti
  - le colonne corrispondono agli oggetti
  - $A[s, o]$ riporta i privilegi di s su o.

Sono possibili cambiamenti di stati tramite comandi che chiamano operazioni primitive:

- **enter** r in $A[s, o]$, che concede il permesso r al soggetto s sull'oggetto o (grant).
- **delete** r da $A[s, o]$, che non permette il permesso r al soggetto s sull'oggetto o.
- **create subject** s', la quale inserisce una nuova riga.
- **destroy subject** s', che cancella una riga esistente.
- **create object** o', che aggiunge una colonna
- **destroy object** o', che cancella una colonna.

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Figura 1: Esempio di Matrice di accesso

La combinazione di comandi primitivi permette la crazione di comandi più generici. Questi comandi sono modifiche allo stato di un sistema modellato da una serie di comandi primitivi del modulo:

command $c(x_1, ..., x_k)$

if

r_1 in $A[x_{s_1}, x_{o_1}]$ and
r_2 in $A[x_{s_2}, x_{o_2}]$ and
...
r_m in $A[x_{s_m}, x_{o_m}]$

then

op_1
op_2
...

op_n

end

$r_1, ..., r_m$ sono le modalità di accesso. $s_1, ..., s_m$ e $o_1, ..., o_m$ sono numeri interi tra 1 e k. Se $m = 0$, il comando non ha una parte condizionale. Alcuni esempi di comandi sono riportati nella Figura 2.

```
command CREATE(subj,file)
        create object file
        enter Own into A[subj,file] end.

command CONFER_read(owner,friend,file)
        if Own in A[owner,file]
          then enter Read into A[friend,file] end.

command REVOKE_read(subj,exfriend,file)
        if Own in A[subj,file]
          then delete Read from A[exfriend,file] end.
```

Figura 2: Esempi di comandi

La delega di autorità può essere ottenuta assegnando flag ai privilegi:

- copy flag (*): il soggetto può trasferire il privilegio ad altri

```
command TRANSFER_read(subj,friend,file)
        if Read* in A[subj,file]
          then enter Read into A[friend,file] end.
```

Figura 3: Esempio di copy flag

- flag di solo trasferimento (+): il soggetto può trasferire ad altri il privilegio (e il flag su di esso); ma così facendo perde l'autorizzazione, cioè nega il privilegio a sé, per darlo a qualcun altro.

```
command TRANSFER-ONLY_read(subj,friend,file)
        if Read+ in A[subj,file]
          then delete Read+ from A[subj,file]
               enter Read+ from A[friend,file] end.
```

Figura 4: Esempio di transfer only flag

L'esecuzione di un comando $c(x_1, ..., x_k)$ su uno stato di sistema $Q = (S, O, A)$ provoca la transizione verso uno stato Q' tale che: $Q = Q_0 \models op_1^* Q_1 \models op_2^* ... \models op_n^* Q_n = Q'$, dove:

- $op_1, .., op_n$ sono operazioni primitive in c.
- i parametri formali $(x_1, ..., x_k)$ nella definizione sono sostituiti dai parametri effettivi forniti al comando.

L'esecuzione di un comando causa la transizione in un nuovo stato, in particolare se il comando è composto da n operazioni primitive l'esecuzione di ognuna di queste mi porta ad una evoluzione dello stato del mio sistema. Se la parte condizionale del comando non è verificata, il comando non ha alcun effetto e $Q = Q'$, cioè se la condizione non è soddisfatta lo stato rimane invariato.

Il safety problem consiste nel dare una risposta alla domanda: Dato un sistema con configurazione iniziale Q_0 esiste una sequenza di richieste eseguite su Q_0 che produce uno stato Q' dove a appare in una cella $A[s, o]$ che non lo aveva in Q_0? La risposta mi permette di stabilire se ci saranno mai degli stati in cui il sistema non è sicuro, perchè s non doveva mai avere il permesso di compiere $A[s, o]$. Non tutte le perdite di diritti sono cattive: i soggetti affidabili sono ignorati nell'analisi. Il safety problem è:

- Indecidibile in generale (ridotto al problema di arresto di una macchina di Turing).
- Decidibile per comando mono-operativo (cioè contenente un'unica operazione primitiva).
- Decidibile quando soggetti e oggetti sono finiti.

Le matrici di accesso spesso non vengono implementate effettivamente come matrici, perchè sono grandi e scarne. Quindi utilizzare le matrici come modelli rappresenta uno spreco di spazio di memoria. Le matrici di accesso sono implementate in 3 modi, che rappresentano approcci diversi al problema:

- Authorization table: dove vengono memorizzate tutte le triple non nulle della forma soggetto, oggetto, azione. É l'approccio tipicamente usato nella basi di dati.

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 2
Bob	write	File 2
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

Figura 5: Esempio di Authorization table

- Access control lists (ACLs): vengono memorizzate le matrici per colonne, quindi sulla base degli oggetti.

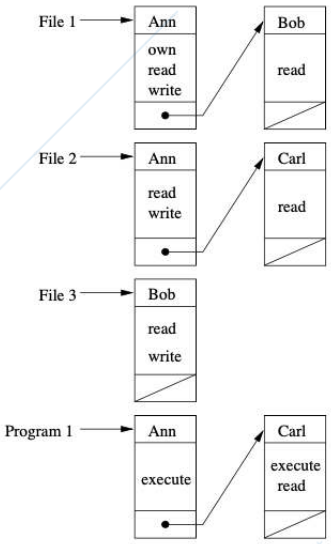


Figura 6: Esempio di ACLs

- Capability lists: vengono memorizzate le matrici per righe, quindi sulla base degli soggetti.

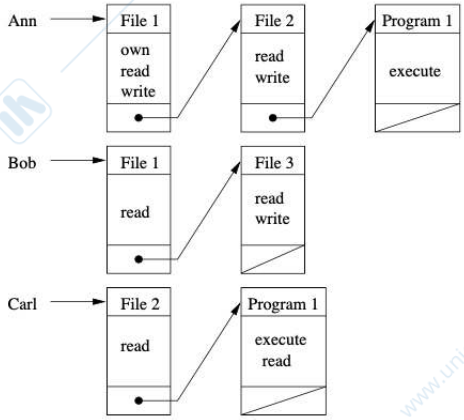


Figura 7: Esempio di Capability lists

Le ACL richiedono che l'utente sia autenticato all'interno del sistema, mentre questo non succede per quanto riguarda le Capability lists, perché è il soggetto che si presenta al sistema porta le sue capability. Questo richiede che le capability siano corrette e che non siano state modificate in modo non corretto. L'approccio tramite ACL funziona meglio quando viene effettuata una revoca per oggetto, mentre se la revoca viene effettuata per soggetto conviene usare l'approccio tramite Capability lists. Mediamente l'approccio tramite ACL è quello dominante in quanto quello tramite Capability lists ha il grande problema di dover avere un meccanismo di controllo della correttezza delle capability presentate dagli utenti. Ad esempio, Unix usa approccio tramite ACL (i 9 bit che permettono di stabilire i privilegi per un determinato file).

Le politiche di accesso discrezionali hanno il limite di poter controllare solamente gli accessi diretti. Non è possibile effettuare alcun controllo su ciò che accade alle informazioni una volta rilasciate. Da ciò consegue che DAC è vulnerabile ai cavalli di Troia che esplorano i privilegi di accesso dell'oggetto chiamante. Il Cavallo di Troia (Trojan horse) è un software Rogue, che

contiene un codice nascosto che esegue funzioni (non legittime) non note al chiamante. I virus e le bombe logiche vengono solitamente trasmessi sotto forma di cavallo di Troia. La Figura 8 rappresenta un esempio di questa problematica. Il sistema presenta un file Market che contiene informazioni sensibili sui prodotti di un'azienda, di cui Jane è proprietaria. John, che non ha accesso a questo file, desidera accedere ad esso. Per fare questo, John crea un file chiamato Stolen, di cui è proprietario, e da a Jane il permesso di accedere in scrittura. Successivamente, John inserisce in maniera nascosta due operazioni in un'applicazione che sa essere necessaria a Jane. Le operazioni sono:

- read Market
- write Stolen

Quando Jane invocherà Application, eseguirà anche questi due comandi, che sono entrambi possibili sulla base dei suoi privilegi. Quindi, Jane leggerà il contenuto di Market e copierà ciò che legge sul file Stolen. A questo punto, John potrà leggere il contenuto di Market, senza aver mai avuto il permesso di leggerlo. Quindi, John acquisisce indirettamente il privilegio in lettura del file Market.

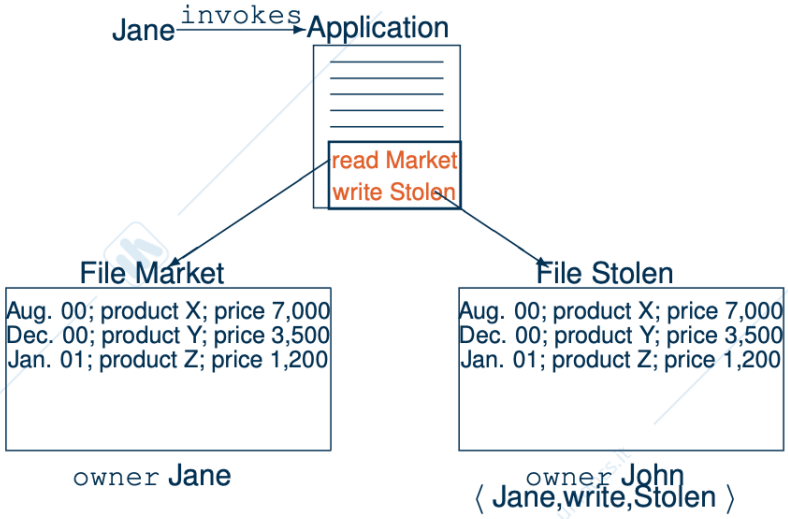


Figura 8: Trojan Horse

1.1.2 Politiche obbligatorie (MAC)

Con il controllo dell'accesso obbligatorio si impongono restrizioni sul flusso di informazioni che non possono essere eluse da Trojan Horses. Il controllo MAC distingue tra utenti e soggetti che operano per loro conto:

- Utente. Essere umano
- Soggetto. Processo nel sistema (programma in esecuzione), che funziona per conto di un utente.

Mentre gli utenti possono trusted, per quanto riguarda comportamenti impropri, i programmi che eseguono non lo sono.

La politica mandatoria più diffusa è la cosiddetta politica multilivello che si basa su classificazioni effettuate sia sui soggetti che sugli oggetti. Si distinguono due classi di politiche a seconda della proprietà su cui la politica stessa si focalizza:

- Secrecy-based: basata sulla confidenzialità (es. modello Bell La Padula)
- Integrity-based: basata sull'integrità (es. modello Biba)

Una classe di sicurezza è generalmente formata da due componenti:

- Livello di sicurezza: elemento di un insieme gerarchico di elementi, cioè elementi su cui è definita una relazione d'ordine totale. Ad esempio: TopSecret (TS), Secret (S), Confidential (C), Non classificato (U), dove $TS > S > C > U$. Un'altra modalità di classificazione di livelli di sicurezza, basata sull'integrità è: Crucial (C), Very Important (VI), Important (I); dove $C > VI > I$.
- Categorie: insiemi di elementi non gerarchici, hanno come obiettivo quello di indicare le diverse aree di competenza delle risorse presenti all'interno del sistema. Permettono di realizzare il principio di "need-to-know", ossia è possibile accedere esclusivamente a risorse strettamente necessarie per svolgere il proprio lavoro.

La combinazione tra livelli di sicurezza e categorie introduce un insieme di classi di sicurezza su cui è definita una relazione di ordine parziale di dominanza: $(L_1, C_1) \succeq (L_2, C_2) \iff L_1 \succeq L_2 \wedge C_1 \supseteq C_2$. Le classi di sicurezza assieme alla relazione di dominanza forma un reticolo, soddisfa cioè tutte queste proprietà:

- Riflessività (Reflexity): $\forall x \in SC : x \succeq x$
- Transitività: $\forall x, y, z \in SC : x \succeq y, y \succeq z \implies x \succeq z$
- Antisimmetria: $\forall x, y \in SC : x \succeq y, y \succeq x \implies x = y$
- Least upper bound (LUB): $\forall x, y \in SC : \exists! z \in SC$
  - $z \succeq x$ and $z \succeq y$
  - $\forall t \in SC : t \succeq x$ and $t \succeq y \implies t \succeq z$.

  z rappresenta la più bassa classe di sicurezza che domina sia x e y.
- Greatest lower bound (GLB): $\forall x, y \in SC : \exists! z \in SC$
  - $x \succeq z$ and $y \succeq z$
  - $\forall t \in SC : x \succeq t$ and $y \succeq t \implies z \succeq t$.

  z rappresenta la più grande classe di sicurezza che è dominata sia da x che da y.

Un esempio di reticolo è rappresentato nella Figura 9. Ogni voce rappresenta una classe di sicurezza, composta dall'unione di livelli di sicurezza e categorie. In questo caso il $LUB(< TS, \{\text{Nuclear}\}, < S, \{\text{Army, Nuclear}\} >) =< TS, \{\text{Army, Nuclear}\} >$; menrtre il $GLB(< TS, \{\text{Nuclear}\}, < S, \{\text{Army, Nuclear}\} >) =< S, \{\text{Nuclear}\} >$.

A ciascun utente viene assegnata una classe di sicurezza (autorizzazione - clearence). Per quanto riguarda la confidenzialità, un utente può connettersi al sistema in qualsiasi classe dominata dalla sua autorizzazione. I soggetti attivati in una sessione assumono la classe di sicurezza con cui l'utente si è connesso. Nel caso di MAC si differenzia il concetto di utente e soggetto. Le classi di segretezza assegnate agli utenti riflette l'affidabilità (il grado di fiducia) dell'utente a non divulgare informazioni sensibili a persone che non dispongono di un'adeguata autorizzazione. Le classi di sicurezza assegnate agli oggetti riflettono la sensibilità delle informazioni contenute negli oggetti e il potenziale danno che potrebbe derivare dalla loro perdita impropria. Le categorie definiscono l'area di competenza degli utenti e dei dati. Anche le categorie vengono assegnate a utenti e dati.

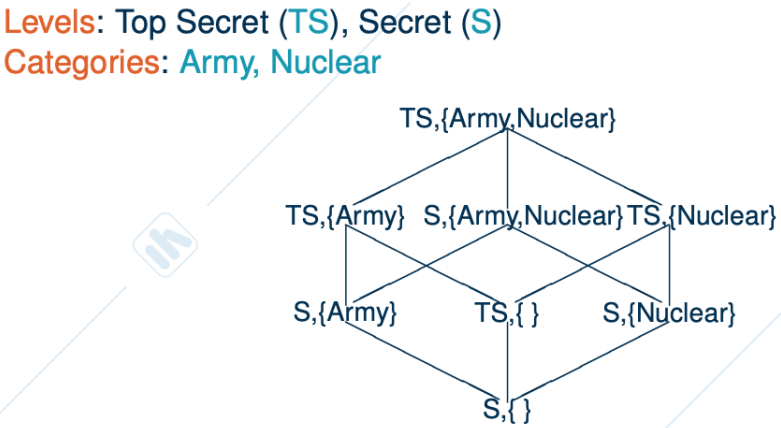


Figura 9: Esempio di reticolo

Il modello Bell La Padula definisce la politica obbligatoria per la segretezza. Sono state proposte diverse versioni del modello (con piccole differenze o correlate a specifici ambienti applicativi); ma i principi di base rimangono gli stessi. L'obiettivo del modello è impedire il flusso di informazioni verso classi di sicurezza inferiori o ineguagliabili. Per evitare il flusso di informazioni discendente si utilizzano due principi:

- NO READ UP, cioè non è possibile leggere al di sopra della classe di sicurezza dell'utente.
- NO WRITE DOWN, cioè non è possibile scrivere al di sotto della classe di sicurezza dell'utente.

Questi due principi garantiscono il fatto che non sussistano flussi di informazioni verso il basso. Questi due principi vengono modellati mediante le seguenti proprietà:

- Simple property: un soggetto s può leggere l'oggetto o solo se $\lambda(s) \succeq \lambda(o)$.
- *-property: un soggetto s può scrivere l'oggetto o solo se $\lambda(o) \succeq \lambda(s)$.

É facile vedere che i Trojan Horses che perdono informazioni attraverso canali legittimi sono bloccati.

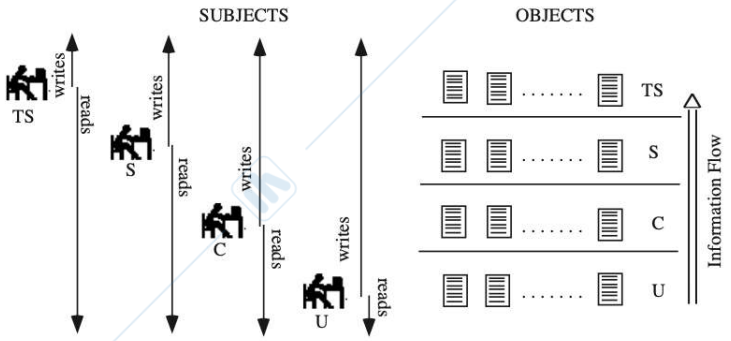


Figura 10: Flusso di informazioni per la segretezza

Un sistema è modellato come uno stato e transizioni ad altri stati. Uno stato $v \in V$ è rappresentato da una tripla ordinata (b, M, λ) dove:

- $b \in \wp(S \times O \times A)$: insieme degli accessi effettuati in un determinato stato, cioè un insieme di triple (soggetto, oggetto, azione) in un determinato stato.
- M: matrice di accesso con S righe, colonne O, voci A.
- λ: $S \cup O \to L$: funzione di classificazione che restituisce la classe di sicurezza di oggetti o soggetti.

Le proprietà simple e * si modellano nel seguente modo:

- simple security: lo stato (b, M, λ) è sicuro iff $\forall (s, o, a) \in b, a = \text{read} \implies \lambda(s) \succeq \lambda(o)$
- *-security: lo stato (b, M, λ) è sicuro iff $\forall (s, o, a) \in b, a = \text{write} \implies \lambda(o) \succeq \lambda(s)$.

Uno stato è sicuro se soddisfa la proprietà di sicurezza semplice e la proprietà *. Una funzione di transizione di stato $T : V \times R \to V$ trasforma lo stato in un altro che soddisfa le due proprietà.

Un sistema (v_0, R, T) è sicuro se v_0 è sicuro e ogni stato raggiungibile da v_0 eseguendo una sequenza finita di richieste da R è sicuro. Un sistema (v_o, R, T) è sicuro iff v_0 è uno stato sicuro e se T è tale che $\forall v$ raggiungibile da v_0 eseguendo una o più richieste da R, se $T(v, c) = v'$, dove $v = (b, M, \lambda)$ e $v' = (b', M', \lambda')$, quindi $\forall s \in S, o \in O$:

- $(s, o, \texttt{read}) \in b' \land (s, o, \texttt{read}) \notin b \Rightarrow \lambda'(s) \succeq \lambda'(o)$
- $(s, o, \texttt{read}) \in b \land \lambda'(s) \not\succeq \lambda'(o) \Rightarrow (s, o, \texttt{read}) \notin b'$
- $(s, o, \texttt{write}) \in b' \land (s, o, \texttt{write}) \notin b \Rightarrow \lambda'(o) \succeq \lambda'(s)$
- $(s, o, \texttt{write}) \in b \land \lambda'(o) \not\succeq \lambda'(s) \Rightarrow (s, o, \texttt{write}) \notin b'$

Figura 11: Proprietà

Il problema è che nessuna restrizione è posta su T, che può essere sfruttato per perdere informazioni (Sistema Z di Mc Lean). Le limitazioni di sicurezza non sono sufficienti. É necessario controllare T. Assumere T come segue: quando un soggetto richiede l'accesso a un oggetto, il livello di sicurezza di tutti i soggetti e di tutti gli oggetti viene ridotto al livello più basso e l'accesso viene concesso. É sicuro secondo Bell La Padula, ma non è sicuro in un senso significativo. BLP è reso sicuro dall'aggiunta della proprietà di tranquillità. Per evitare questo tipo di problema viene introdotta la proprietà di tranquillità che impedisce qualsiasi tipo di cambio di livello di sicurezza, un rilassamento di questa proprietà prevede che:

- Le classi di sicurezza posso essere cambiate, ma non in modo indiscriminato, magari soltanto per i soggetti fidati
- Non tutti i cambiamenti portano al leakage, per esempio l'innalzamento del livello di sicurezza degli oggetti. Oppure, è possibile autorizzare materie affidabili per il downgrade (ad es. Sanificazione).

Altre eccezioni alla proprietà di tranquillità sono:

- Sanitization and downgrading: alcuni downgrade possono essere necessari dopo alcuni anni (embargo), oppure ci potrebbe essere un processo fidato che prende in pasto una informazione sensibile e sputa un risultato non sensibile.
- Data association: un insieme di valori può avere una classificazione più alta di ogni valore preso singolarmente.
- Aggregation: una aggregazione di valori può avere una classificazione più alta dei singoli elementi

- Un soggetto di fiducia è autorizzato a bypassare (in modo controllato) alcune restrizioni imposte dalla politica obbligatoria.

DAC e MAC non si escludono a vicenda. Ad esempio, BLP applica anche DAC: DAC property $b \subseteq \{(s, o, a) s.t. a \in M[S, O]\}$, dove b rappresenta il sottoinsieme delle triple nella matrice di accesso. Se vengono applicati sia DAC che MAC, sono consentiti solo accessi che soddisfano entrambi. DAC offre discrezionalità entro i limiti del MAC.

Le limitazione delle politiche mandatorie (MAC), utilizzate per garantire la confidenzialità è che queste politiche (anche con l'aggiunta della proprietà di tranquillità) controllano solo i canali palesi di informazioni (flusso attraverso canali legittimi), ma rimane vulnerabile ai canali nascosti, ossia canali nati per altri scopi, ma utilizzati per scambiarsi informazioni in modo illegittimo. Ogni risorsa del sistema, condivisa da processi di diversi livelli, può essere sfruttato per creare un canale nascosto. Alcuni esempi di questa casistica sono:

- Un soggetto di basso livello chiede di scrivere un file di alto livello. Il sistema restituisce che il file non esiste (se il sistema crea il file l'utente potrebbe non essere a conoscenza quando necessario).

- Un soggetto di basso livello richiede una risorsa (ad es. CPU o blocco) occupata da un soggetto di alto livello. Può essere sfruttato da soggetti di alto livello per divulgare informazioni a soggetti di livello inferiore.

- Un processo di alto livello può bloccare le risorse condivise e modificare i tempi di risposta del processo a livelli inferiori (canale di temporizzazione). Con il canale di temporizzazione la risposta restituita a un processo di basso livello è la stessa, è il momento di restituirla che cambia.

L'analisi dei canali nascosti è l'unica soluzione per evitare lo sfruttamento di covert channels da parte dei ssoggetti. L'analisi dei canali nascosti, di solito, viene eseguita nella fase di implementazione, per assicurare che l'implementazione di un sistema della primitiva del modello non sia troppo debole. I modelli di interfaccia tentano di escludere tali canali nella fase di modellazione. L'analisi ha quindi lo scopo di garantire la proprietà di non interferenza: l'attività dei processi di alto livello non deve avere alcun effetto sui processi a livelli inferiori o incomparabili.

Fino ad ora si è parlato solamente di politiche MAC per la segretezza dei dati, menre adesso iniziamo ad occuparci di politiche MAC per l'integrità dei dati. Le politiche obbligatorie di segretezza controllano solo la perdita impropria di informazioni. Non salvaguardare l'integrità comporta che le informazioni possono essere manomesse. É possibile applicare una politica duale per l'integrità, in base all'assegnazione di classificazioni (integrità). Le classi di integrità assumono significati diversi a seconda che si parli di utenti o oggetti:

- se assegnata agli utenti riflette l'affidabilità degli utenti di non modificare in modo improprio le informazioni.

- se assegnata agli oggetti riflette il grado di fiducia nelle informazioni contenute negli oggetti e il potenziale danno che potrebbe derivare dalla sua modifica/cancellazione impropria.

Le categorie definiscono l'area di competenza degli utenti e dei dati.

Uno dei modelli utilizzati per rappresentare politiche MAC per l'integrità dei dati è il modello Biba. Il modello Biba definisce la politica obbligatoria per l'integrità. L'obiettivo che si prefigge è impedire il flusso di informazioni verso classi di sicurezza superiori o ineguagliabili. Per evitare il flusso di informazioni ascendenti si utilizzano due principi, che non sono altri che i duali dei principi del modello di Bell La Padula:

- NO READ DOWN, cioè non è possibile leggere al di sotto della classe d'integrità dell'utente.

- NO WRITE UP, cioè non è possibile scrivere al di sopra della classe d'integrità dell'utente.

Questi due principi garantiscono il fatto che non sussistano flussi di informazioni verso l'alto. Questi due principi vengono modellati mediante le seguenti proprietà:

- Simple property: un soggetto s può leggere l'oggetto o solo se $\lambda(o) \succeq \lambda(s)$.
- *-property: un soggetto s può scrivere l'oggetto o solo se $\lambda(s) \succeq \lambda(o)$.

Lo svantaggio di questo modello è che non evita i problemi di integrità, ma semplicemente segnala se ci sono state compromissioni, dal punto di vista dell'integrità.

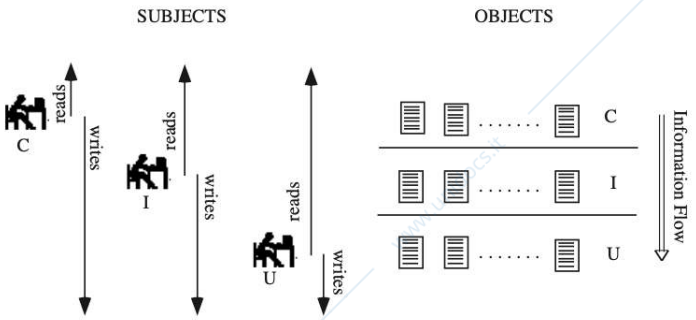


Figura 12: Flusso di informazioni per l'integrità

Oltre al modello di base (rigorosa politica di integrità) esistono politiche alternative più lasche in lettura e scrittura:

- Low-water mark per i soggetti: il principio no write-up continua a valere, le operazioni di lettura, invece, sono sempre permesse e quindi a seguito di ogni lettura la classe di integrità di ogni soggetto deve cambiare prendendo il GLB tra la classe di integrità associata al soggetto e la classe di integrità associata all'oggetto, perché la lettura di informazioni non molto fidate da parte di un soggetto fidato inquina le sue conoscenze e quindi diminuisce il suo grado di fiducia. Quindi dopo l'accesso la classe di integrità del soggetto sarà uguale a: $\lambda(s) := glb(\lambda(s), \lambda(o))$. Il problema che sorge dall'utilizzo di queste politiche è che l'ordine delle operazioni effettuate ha un impatto sui privilegi del soggetto.
- Low-water mark per gli oggetti: il principio no read-down continua a valere, le operazioni di scrittura, invece, sono sempre permesse e quindi a seguito di ogni scrittura la classe di integrità di ogni oggetto deve cambiare prendendo il GLB tra la classe di integrità associata al soggetto e la classe di integrità associata all'oggetto, perché la scrittura di informazioni da parte di un soggetto non molto fidato inquina il grado di fiducia dell'oggetto. Quindi dopo l'accesso la classe di integrità dell'oggetto sarà uguale a: $\lambda(o) := glb(\lambda(s), \lambda(o))$. Un problema che sorge dall'utilizzo di queste politiche è che non salvaguardano l'integrità, ma semplicemente notificano la compromissione.

Il modello di Biba per la protezione dell'integrità presenta carenze:

- Le restrizioni al flusso possono risultare troppo restrittive
- Impone l'integrità solo impedendo flussi di informazioni da classificazioni di accesso inferiori a superiori, quindi acquisisce solo una minima parte del problema di integrità

L'applicazione dei modelli all'interno del contesto della basi di dati, in cui è necessario garantire la confidenzialità, è possibile utilizzando il modello Bell La Padula. Il modello Bell-La

Padula è stato proposto per la protezione a livello di sistema operativo, ma approcci successivi hanno studiato l'applicazione di politiche multilivello a modelli di dati (DBMS, sistemi orientati agli oggetti, ...). Mentre nel contesto del sistema operativo il livello di sicurezza è assegnato a un file, i DBMS possono permettersi una classificazione più fine, per quanto riguarda la granularità:

- relazione
- attributo
- tupla
- elemento

Le prime implementazioni presero in considerazione le relazioni; nel modello relazionale classico ogni relazione è caratterizzata da:

- Lo schema della relazione $R(A_1, ..., A_n)$, il quale è indipendente dallo stato. R rappresenta il nome della relazione e $A_1, ..., A_n$ rappresentano gli attributi.
- L'istanza della relazione, dipendente dallo stato, composta da tuple $(a_1, ..., a_n)$.

Una relazione è caratterizzata da attributi chiave (Key Attributes), i quali identificano univocamente le tuple: nessuna coppia di tuple può avere la stessa chiave e gli attributi chiave non possono avere valori nulli. Nei DBMS che supportano la classificazione a livello di elemento, ogni relazione è caratterizzata da:

- Schema della relazione $R(A_1, C_1, ..., A_n, C_n)$, sempre indipendente dallo stato, dove $C_i, i = 1, ..., n$ rappresenta l'intervallo di classificazioni di sicurezza. C rappresenta la classe di sicurezza che viene associata ai singoli elementi nella relazione.
- Set di istanze della relazione R_c dipendente dallo stato; un'istanza per ogni classe di sicurezza c. Ogni istanza è composta da tuple $(a_1, c_1..., a_n, c_n)$. L'istanza al livello c contiene solo elementi la cui classificazione è dominata da c.

Il controllo degli accessi obbedisce ai principi del modello Bell La Padula, per cui:

- no read up (la vista di un soggetto in una determinata classe di accesso c contiene solo gli elementi la cui classificazione è dominata da c)
- no write down ulteriormente limitata. Ciò significa che ogni soggetto scrive solo al suo livello. Quindi, è presente un'ulteriore restrizione rispetto al modello originale, ogni soggetto può scrivere solo al suo stesso livello in quanto dato che la classificazione è di livello molto più fine, a differenza dei SO non sarà mai necessario che un soggetto di classificazione bassa debba aggiungere informazioni a livelli più alti.

Nella figura 13 viene presentato un esempio di relazione multilivello, in cui viene presentata un'istanza del DB con livello U. Si può notare che le tuple con livello superiore ad U vengono nascoste, così come i singoli elementi del DB.

Per ogni tupla in una relazione multilivello è necessario specificare delle raccomandazioni:

- Gli attributi chiave devono essere classificati uniformemente, cioè tutti gli attributi chiave facenti parte della relazione devono essere classificati allo stesso livello: $\forall t \in T, \forall A_i, A_j \in AK : \lambda(t[A_i]) = \lambda(t[A_j])$
- Le classificazioni degli attributi non chiave devono dominare quella degli attributi chiave: $\forall t \in T, \forall A_i \in AK, A_j \notin AK : \lambda(t[A_j]) \geq \lambda(t[A_i])$.

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S

Instance U

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Sam	U	Dept1	U	-	U

S-instance is the whole relation

Figura 13: Esempio relazione multilivello

La classificazione a grana fine deve prendere in considerazione la semantica dei dati e l'eventuale perdita di informazioni. La complicanza che nascono dalla granularità più fine è la Polyinstantiation, ossia la presenza di più oggetti con la stessa chiave, ma diverse classificazioni. In parole semplici, si tratta di tuple diverse con lo stesso chiave ma:

- diversa classificazione per la chiave (tuple polinistanziate)
- valori e classificazioni diverse per uno o più attributi (elementi polinistanziati)

Perché vengono generate queste polinstanziazioni? Spesso, non è possibile evitarlo, perché avviene a seguito di operazioni di inserimento o modifica delle tuple. La polinstanziazione si suddivide in:

- Invisibile: Un soggetto di basso livello inserisce i dati in un campo che contiene già dati di livello superiore o incomparabile
- Visibile: Un soggetto di alto livello inserisce i dati in un campo che contiene dati di livello inferiore

Per quanto riguarda la polinstanziazione invisibile, un soggetto di basso livello richiede l'inserimento di una tupla. La relazione contiene già una tupla con la stessa chiave primaria, ma con una classificazione più elevata, e quindi non visibile al soggetto che intende inserirla. Le soluioni possibili sono:

- Indicare al soggetto l'impossibilità di inserire la tupla, ma ciò provoca una perdita di informazioni, perché ricevuta la comunicazione dell'impossibilità, il soggetto è a conoscenza del fatto che esista una tupla con quella chiave.
- Sostituire la vecchia tupla con quella nuova, ma ciò provoca una perdita di integrità, perché un'informazione con livello di classificazione più alto iene modificata da un utente con livello di classificazione più basso.
- Inserire una nuova tupla. Quest'operazione genera una tupla polinistanziata.

La Figura 14 riporta un esempio di tupla polinstanziata. L'utente, con livello di segretezza U, a cui non è visibile la tupla Ann con livello di segretezza S, vorrebbe inserire una tupla Ann. Quindi, per via dei motivi precedenti, viene inserita una nuova tupla con chiave Ann e valori differenti.

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S

Request by U-subject
INSERT INTO Employee VALUES Ann,Dept1,100K

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Ann	U	Dept1	U	100K	U

Figura 14: Esempio tupla polinstanziata

La Figura 15 riporta un esempio di elemento polinstanziato. La classe di segretezza dell'elemento Salary della tupla Sam è superiore alla classe di segretezza dell'utente che vuole modificare il dato. L'utente vede il valore NULL, quindi potrebbe pensare che il dato manchi. Per via dei motivi precedentemente esposti, viene creata una nuova tupla Sam con il valore aggiornato.

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S

Request by U-subject
UPDATE Employee SET Salary="100K" WHERE Name="Sam"

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Sam	U	Dept1	U	100K	U

Figura 15: Esempio elemento polinstanziato

Per quanto riguarda la polinstanziazione visibile, un soggetto di alto livello richiede l'inserimento di una nuova tupla. La relazione contiene già una tupla con la stessa chiave primaria ma con una classificazione inferiore. Le possibili azioni sono:

- Indicare al soggetto l'impossibilità di inserimento, ma ciò provoca un denial of service.
- Sostituire la vecchia tupla con la nuova. Ciò provoca una perdita di informazioni, perchè gli utenti con livelli di segretezza più bassi smetterebbero di avere accesso ad una tupla a cui avevano accesso.
- Inserire una nuova tupla, che porta alla polinstanziazione di tuple

La Figura 16 riporta un esempio di tupla polinstanziata. L'utente, con livello di segretezza S vorrebbe inserire una tupla Ann. Esiste già una tupla Ann con livello di segretezza U. Per via dei motivi precedenti, viene inserita una nuova tupla con chiave Ann e valori differenti.

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	U	Dept1	U	100K	U
Sam	U	Dept1	U	150K	S

Request by S-subject
INSERT INTO Employee VALUES Ann,Dept2,200K

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	U	Dept1	U	100K	U
Sam	U	Dept1	U	150K	S
Ann	S	Dept2	S	200K	S

Figura 16: Esempio tupla polinstanziata

La Figura 17 riporta un esempio di elemento polinstanziato. La classe di segretezza dell'elemento Salary della tupla Sam è inferiore alla classe di segretezza dell'utente che vuole modificare il dato. Per via dei motivi precedentemente esposti, viene creata una nuova tupla Sam con il valore aggiornato.

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	100K	U

Request by S-subject
UPDATE Employee SET Salary="150K" WHERE Name="Sam"

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	100K	U
Sam	U	Dept1	U	150K	S

Figura 17: Esempio elemento polinstanziato

La semantica possibile per tuple ed elementi polinistanziati prevede:

- tuple polinistanziate, che consistono in diverse entità del mondo reale
- elementi polinistanziati, che consistono nella stessa entità del mondo reale

La polinstanziazione deve essere controllata (non tutte le istanze del database possono avere senso). Al massimo dovrebbe esistere una tupla per entità per ogni livello. Il problema è che la polinstanziazione può rapidamente sfuggire di mano. Un alternativa è l'uso di valori "restricted" (anziché "null"). La comunità ha discusso a lungo riguardo a qual è il livello di granularità di classificazione corretto e tra il rapporto tra polinstanziazione e utilizzo di valori limitati.

M-DBMS commerciali (ad es. Oracle) supportano la classificazione a livello di tuple. La polinstanziazione è considerata uno dei motivi principali per cui i DBMS multilivello non hanno

successo, però la polinstanziazione non è sempre negativa. Infatti, può essere utile per supportare le storie di copertina (cover stories). Le cover stories sono valori errati restituiti a soggetti di basso livello per proteggere il valore reale. Ciò è utile quando la restituzione del valore null o restricted produrrebbe leakage di informazioni. Il supporto della classificazione dettagliata (fine-grained) ha anche altri problemi:

- il supporto dei vincoli di integrità diventa complesso
- la necessità di controllare i canali di inferenza

Altro aspetto importante quando si ha a che fare con una base di dati multilivello è la modalità di memorizzazione dei dati all'interno del sistema. Ci sono due possibili soluzioni corrispondenti a due architetture multi livello:

- Soggetto trusted: i dati a diversi livelli sono memorizzati in un unico database. Il DBMS deve essere attendibile per garantire l'obbedienza alla politica obbligatoria (MAC).
- Base di calcolo trusted: i dati sono partizionati in database diversi, uno per ogni livello. Solo il sistema operativo deve essere considerato attendibile. Ogni DBMS è limitato per accedere ai dati che possono essere letti al suo livello (senza lettura). Gli algoritmi di decomposizione e ripristino devono essere attentamente costruiti per essere corretti ed efficienti.

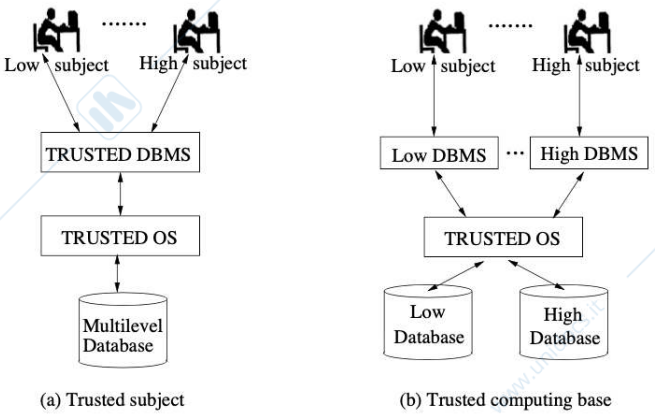


Figura 18: Architetture dei Database con politiche MAC

1.1.3 Politiche basate sul ruolo (RBAC)

Il ruolo rappresenta un denominato set di privilegi relativi all'esecuzione di una particolare attività. Secondo queste politiche, l'accesso degli utenti agli oggetti è mediato sulla base dei ruoli. Ai ruoli vengono concesse le autorizzazioni per accedere agli oggetti. Gli utenti hanno le autorizzazioni per l'attivazione dei ruoli. Attivando un ruolo r un utente può eseguire tutto l'accesso concesso a r. I privilegi associati a un ruolo non sono validi quando il ruolo non è attivo. Sussiste una differenza netta tra il concetto di gruppo, che rappresenta un set di utenti, e il concetto di ruolo, che invece rappresenta un insieme di privilegi. Quindi, le autorizzazioni vengono concesse sulla base dei privilegi associati ai ruoli; mentre agli utenti è concesso il privilegio di attivare uno o più ruoli.

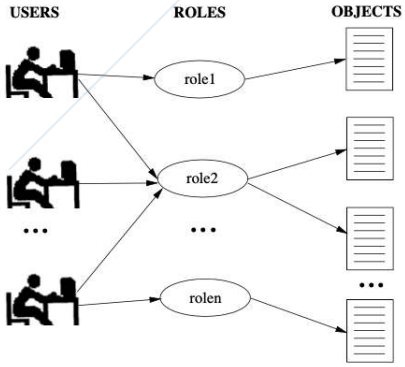


Figura 19: Modello di controllo degli accessi basato sui ruoli

La gerarchia dei ruoli definisce le relazioni di specializzazione, come mostrato nella Figura 20

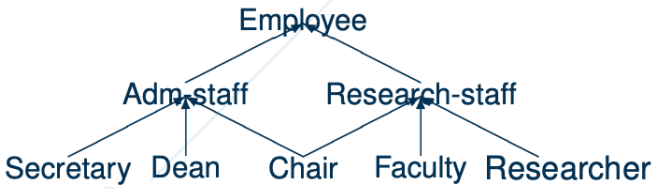


Figura 20: Modello di controllo degli accessi basato sui ruoli

Una relazione di tipo gerarchico prevede il concetto di propagazione dell'autorizzazione, cioè se a un ruolo r viene concessa l'autorizzazione all'esecuzione (azione, oggetto), tutti i ruoli che sono specializzazioni di r possono eseguire quella determinata (azione, oggetto). Inoltre, Se viene concessa l'autorizzazione ad un utente u di attivare il ruolo r, l'utente u può attivare tutte le generalizzazioni di r.

Le politiche basate suoi ruoli permettono di modellare meglio le autorizzazioni sulla base delle gerarchi. Inoltre, forniscono i seguenti vantaggi:

- Gestione semplice: è facile specificare le autorizzazioni. Ad esempio, è sufficiente assegnare o rimuovere un ruolo per un utente per consentire all'utente di eseguire un intero set di attività. Se venisse assunto un nuovo impiegato in un'azienda non è più necessario specificare ogni singola autorizzazione, ma è sufficiente specificare un ruolo per il nuovo impiegato.
- La gerarchia dei ruoli può essere sfruttata per supportare le implicazioni. Semplifica la gestione delle autorizzazioni.
- Restrizioni: ulteriori restrizioni possono essere associate a ruoli, come cardinalità o mutua esclusione. La cardinalità esprime un numero massimo di ruoli che possono essere attivate da un utente; mentre la mutua esclusione rappresenta l'impossibilità di attivare due ruoli contemporaneamente.
- Minimo privilegio: permette di associare ad ogni soggetto il minor set di privilegi di cui il soggetto ha bisogno per eseguire il suo lavoro. Questa proprietà limita abusi e danni dovuti a violazioni ed errori.

- Separazione del dovere: i ruoli consentono l'applicazione della separazione del dovere (ripartizione dei privilegi tra soggetti diversi).

Il lavoro sui modelli basati sui ruoli si è anche occupato di:

- relazioni oltre le gerarchie (ad es. Il segretario può operare a nome del suo manager)
- propagazione basata sulla gerarchia non sempre desiderata (alcuni privilegi potrebbero non propagarsi alle sottoregole)
- politiche amministrative arricchite (confinamento dell'autorità)
- relazioni con identificativi utente (necessarie per le relazioni individuali, ad esempio "Il mio segretario")
- ulteriori vincoli. Ad esempio, separazione dinamica del servizio: il completamento di un'attività richiede la partecipazione di almeno n individui.

In SQL i privilegi possono essere raggruppati in ruoli che possono essere assegnati agli utenti o ad altri ruoli (nidificati).

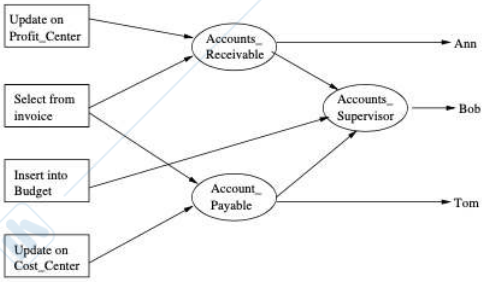


Figura 21: Ruoli in SQL

Attivando un ruolo, un utente è abilitato per tutti i privilegi in un sottoinsieme radicato in quel ruolo. É presente sempre, al massimo un ruolo attivo per utente. I ruoli possono essere concessi agli utenti con l'opzione di concessione, cioè un utente può concederlo ad altri.

1.1.4 Politiche amministrative

L'obiettivo di queste politiche è definire chi può concedere e revocare le autorizzazioni di accesso. Si suddividono in due macro aree:

- Centralizzato: un'autorità per i privilegi (responsabile della sicurezza del sistema) è responsabile della specifica dell'autorizzazione.
- Ownership: il creatore di un oggetto è il suo proprietario e come tale può amministrare l'autorizzazione all'accesso sull'oggetto. Questa proprietà non sempre chiara in modelli gerarchici di dati (ad es. Orientati agli oggetti) o per quanto riguarda framework con politiche basate sui ruoli. Questa macro area si adatta maggiormente a politiche di tipo DAC.

L'autorità per specificare le autorizzazioni può essere delegata. La delega è spesso associata al concetto di ownership: il proprietario di un oggetto delega i privilegi amministrativi ad altri. L'amministrazione decentralizzata introduce flessibilità, ma complica lo scenario.

Diverse politiche amministrative possono differire per il modo in cui rispondono alle seguenti domande:

- A quale granularità dovrebbero essere supportate le autorizzazioni amministrative? Può andare alla grana fine di ogni singolo accesso (azione, oggetto).
- É possibile limitare ulteriori deleghe? Di solito no; ma può essere utile.
- Chi può revocare le autorizzazioni? Cambia a seconda della politica amministrativa: chi li ha concessi; il proprietario; ogni amministratore.
- Cosa succede alle autorizzazioni concesse da qualcuno i cui privilegi amministrativi sono stati revocati? Anche questo cambia a seconda della politica: dovremmo eliminarli? dovremmo tenerli?

Per quanto riguarda le autorizzazioni amministrative in SQl, ossia in un contesto di basi di dati relazionali, l'utente che crea una tabella è il suo proprietario e può concedere autorizzazioni sulla tabella ad altri utenti. Le autorizzazioni possono essere concesse con l'opzione di concessione (grant option). L'opzione di grant su un accesso consente a un utente di concedere ulteriormente tale accesso (e concedere l'opzione) ad altri, dando così vita ad una catena di autorizzazioni. Gli utenti possono revocare solo le autorizzazioni concesse. La Figura 22 riporta un esempio di concessione di autorizzazioni: Ann è l'owner della tabella Department. Ann concede l'accesso all'operazione SELECT sulla tabella a Bob (con grant option) e a Chris (senza grant option). Bob, a sua volta, concede l'accesso a Frank (senza grant option). Chris e Frank possono svolgere l'azione SELECT, ma non possono concedere l'accesso ad altri utenti.

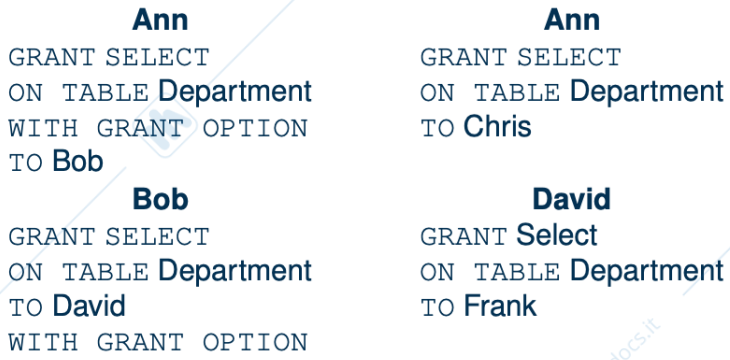


Figura 22: Esempio di Grant

Quando ad un utente viene revocata l'opzione di concessione per un privilegio cosa dovrebbe accadere alle autorizzazioni per il privilegio che ha concesso? La revoca può essere richiesta:

- con cascata (ricorsivo): Se il revocatore non detiene più il privilegio con l'opzione di concessione, le autorizzazioni concesse vengono eliminate in modo ricorsivo. É necessario prestare attenzione ai cicli.
- senza cascata: Se la revoca di un'autorizzazione implicherebbe la cancellazione ricorsiva, l'operazione di revoca non viene eseguita.

La politica di revoca originale (a cascata) era basata sul tempo: tutte le autorizzazioni concesse in virtù di un'autorizzazione che era stata revocata sono state eliminate, indipendentemente dalle altre autorizzazioni successive che l'utente aveva ricevuto. La Figura 23 presenta un esempio dell'operazione di revoca con cascata: Bob revoca l'autorizzazione di grant a David con cascade. La Figura mostra la situazione prima e dopo l'operazione di revoca.

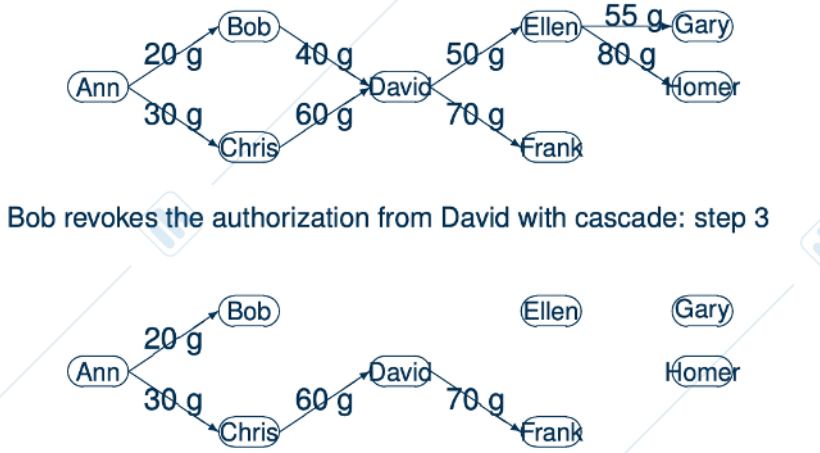


Figura 23: Esempio di revoca con cascata

Inizialmente, quando ad un utente viene revocata l'opzione di grant, tutte le autorizzazioni, che non sarebbero esistite se l'utente non avesse mai ricevuto l'autorizzazione revocata, devono essere eliminata in modo ricorsivo:

- Stato autorizzazione iniziale $AUTH$
- Sequenza $G1, ..., G_n$ (cronologia) delle operazioni di concessione
- $AUTH'$ risultante dopo le operazioni grant $G_1, ..., G_n$
- La revoca di una sovvenzione G_k deve comportare lo stato di autorizzazione $AUTH''$ che sarebbe derivato dall'esecuzione su AUTH della sequenza di sovvenzione $G_1, ..., G_{k-1}, G_{k+1}$, $..., G_n$, ossia uno stato in cui non viene effettuato l'operazione di grant G_k.

Per far rispettare la semantica della revoca dobbiamo tenere traccia, per ogni autorizzazione:

- se ha l'opzione di concessione
- chi l'ha concessa
- il momento in cui è stato concessa

Sia $(u, a, T, go, grantor, time)$ un'autorizzazione revocata:

1. Determina il tempo minimo t_m di autorizzazioni per l'azione a sulla tabella T con l'opzione di concessione
2. Revoca in modo ricorsivo tutte le autorizzazioni per a su T concesse da u prima del tempo t_m

La Figura 24 mostra un esempio dell'approccio, inizialmente utilizzato, per la revoca con cascata. I numeri sulle frecce rappresentano gli istanti di tempo in cui sono state concesse le grant option. Quindi, nel momento in cui Bob revoca l'autorizzazione di grant a David, vengono cancellate tutte le grant option concesse da David tra l'istante in cui Bob ha concesso la grant option (40) e l'istante in cui David ha ricevuto la grant option anche da Chris (60), ossia il periodo in cui David non possedeva l'opzione di concessione. Quindi, viene revocata l'opzione di grant ad Ellen, effettuata all'istante 50, e, di conseguenza, anche tutte le grant option concesse da Ellen (Gary e Homer).

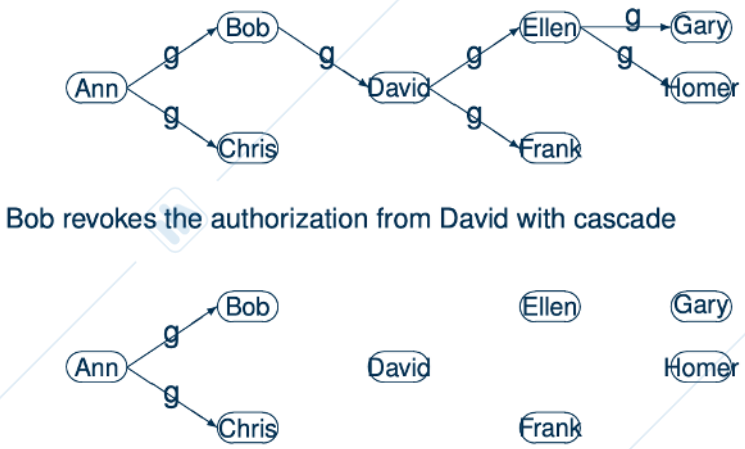


Figura 24: Esempio di revoca con cascata

La revoca ricorsiva non è sempre desiderata: non necessariamente tutto ciò che un utente ha concesso deve essere eliminato (ad esempio, l'utente cambia lavoro a causa di una promozione). Inoltre, tutti i programmi e le viste dipendenti dalle autorizzazioni revocate verranno invalidati. Un'alternativa possibile è, invece di revocare le autorizzazioni, specificarle con colui che revoca al diretto interessato come concedente. La Figura 25 mostra un esempio di questa soluzione alternativa. Possiamo vedere nella Figura che Bob revoca l'opzione di concessione a David e diventa il diretto concessore di Ellen, a cui David aveva fornito la grant option.

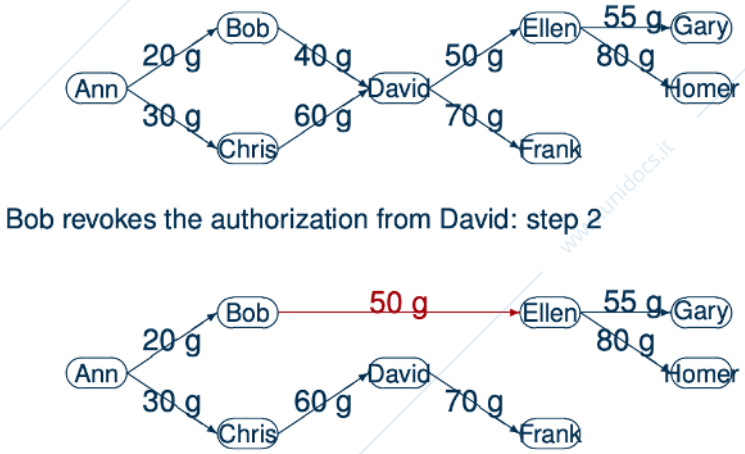


Figura 25: Esempio di revoca rivisitata

Nelle operazioni di revoca in SQL odierne, il tempo non è più considerato. La revoca può essere ricorsiva a cascata o non ricorsiva (nessuna cascata):

- Ricorsiva: le autorizzazioni concesse da un utente vengono eliminate solo se gli utenti non detengono più l'opzione di concessione sull'accesso (indipendentemente dal tempo). É necessario prestare attenzione ai cicli.
- Non ricorsiva: se la cancellazione di un'autorizzazione revocata implicherebbe la necessità di una cancellazione ricorsiva, la revoca non viene eseguita.

1.1.5 Modello di Clark e Wilson per l'integrità

Il modello di Biba è un modello utilizzato nello sviluppo di politiche mandatorie. Il modello di Biba per la protezione dell'integrità, però, presenta carenze, dal momento che le restrizioni al flusso possono risultare troppo restrittive ed, inoltre, impone l'integrità solo impedendo flussi di informazioni da classificazioni di accesso inferiori a superiori, ma acquisisce solo una minima parte del problema di integrità. I meccanismi MAC vengono utilizzati in contesti molto eterogenei tra loro. La Figura 26 riporta un confronto tra contesti eterogenei, ma che comunque mantengono il problema dell'integrità. Quindi, è necessario trovare soluzioni adatte ad entrambi i contesti.

Military	Commercial
Data item associated with a particular level	Data item associated with a set of programs permitted to manipulate it
Users constrained by what they can read and write	Users constrained by which programs they are allowed to execute

Figura 26: Meccanismi militari vs commerciali

L'integrità è un concetto più complesso: garantire che nessuna risorsa sia stata modificata in modo non autorizzato o improprio e che i dati memorizzati nel sistema riflettano correttamente la parola reale che intendono rappresentare. Sorge la necessità di prevenire difetti ed errori, piuttosto che segnalare eventuali modifiche a posteriori. Qualsiasi sistema di gestione dei dati ha funzionalità per garantire l'integrità, attraverso:

- tecniche di controllo e recupero della concorrenza: per garantire che nessun accesso simultaneo possa portare alla perdita o all'incoerenza dei dati
- tecniche di recupero: per ripristinare lo stato del sistema in caso di errori o violazioni
- vincoli di integrità: che impongono la limitazione dei valori che possono essere dati ai dati

Tutte le transizioni di un sistema devono soddisfare le proprietà ACID:

- Atomicità: vengono eseguite tutte le azioni di una transazione o nessuna di esse.
- Coerenza: una transazione deve preservare la soddisfazione dei vincoli di integrità sui dati.
- Isolamento: l'esecuzione parallela di una serie di transazioni deve avere lo stesso effetto di quella di una certa esecuzione sequenziale di quella serie.
- Durabilità: i risultati delle transazioni impegnate sono permanenti.

Queste proprietà non sono abbastanza, dal momento che non prendono in considerazione il soggetto che esegue l'azione. Non proteggono dagli abusi degli utenti legittimi.

Un esempio di modello di integrità è il modello di Clark e Wilson, che definisce quattro criteri base per salvaguardare l'integrità:

- Autenticazione: il sistema deve autenticare e identificare separatamente ogni utente, in modo che le sue azioni possano essere controllate e verificate.

- Controllo (Audit): il sistema deve mantenere un registro di controllo che registra tutti i programmi eseguiti e il nome dell'utente che lo ha autorizzato. É utile per capire cosa è successo all'interno di un sistema.
- Transazioni ben formate: il sistema deve garantire che gli elementi di dati specifici possano essere manipolati solo da un insieme ristretto di programmi e i controlli del data center devono garantire che tali programmi soddisfino la regola di transazione ben formata. Si tratta di una certificazione del fatto che i programmi eseguono modifiche corrette dal punto di vista dell'integrità
- Separazione del dovere (Separation of duty): il sistema deve associare a ciascun utente un set valido di programmi da eseguire e i controlli del data center devono garantire che questi insiemi soddisfino la regola di separazione del dovere.

Il modello è, inoltre, basato sui seguenti concetti:

- Elementi dati vincolati (Constrained Data Items - CDI): tutti quegli elementi dati all'interno del sistema a cui deve essere applicato il modello di integrità.
- Dati non vincolati (Unconstrained Data Items): articoli non soggetti a vincoli di integrità
- Procedure di verifica dell'integrità (Integrity Verification Procedures IVP). Le procedure IVP consentono di verificare l'integrità, ovvero il loro obiettivo è confermare che tutti i CDI siano conformi alla specifica di integrità al momento dell'esecuzione dell'IVP.
- Procedure di trasformazione (Transformation Procedures - TP). Le TP sono tutte le procedure che possono modificare i CDI o ricevere input arbitrari dagli utenti e creare CDI. Corrispondono a transazioni ben formate: cambiano l'insieme di CDI da uno stato valido a un altro, ossia da uno stato che soddisfa i criteri di integrità ad un altro che li soddisfa a sua volta.

La verifica dell'integrità è un processo composto da due tipologie di regole:

- Regole di Certificazione: rilasciata dal responsabile della sicurezza del sistema, dal proprietario del sistema e dal custode del sistema in relazione a una politica di integrità.
- Regole di Applicazione: eseguite dal sistema.

Alcuni esempi di regole inerenti al modello di Clark e Wilson sono:

- Framework di base (coerenza interna)
  - C1: Tutti gli IVP devono garantire correttamente che tutti i CDI siano in uno stato valido al momento dell'esecuzione dell'IVP.
  - C2: Tutti i TP devono essere certificati per essere validi (preservare la validità dello stato). Per ogni TP e ogni set di CDI, il responsabile della sicurezza deve specificare una "relazione", che definisce tale esecuzione; le relazioni sono nella forma (TPi, (CDIa, CDIb, ...)). Ciò significa che possono essere effettuate modifiche, mediante procedure di trasformazione, ma devono comunque soddisfare i vincoli di integrità.
  - E1: Il sistema deve mantenere l'elenco delle relazioni specificate nella regola C2 e deve garantire che l'unica manipolazione di qualsiasi CDI sia da parte di un TP, in cui TP sta operando sul CDI come specificato in alcune relazioni.
- Coerenza esterna (separazione dei compiti)

  - E2: Il sistema deve mantenere un elenco di relazioni del modulo (UserID, TPi, (CDIa, CDIb, ...)), che si riferisce a un utente, a un TP e agli oggetti dati a cui TP può fare riferimento per conto dell'utente. Deve garantire che vengano eseguite solo le esecuzioni descritte in una delle relazioni.
  - C3: L'elenco delle relazioni in E2 deve essere certificato per soddisfare il requisito della separazione dei compiti.
  - E3: Il sistema deve autenticare l'identità di ciascun utente che tenta di eseguire un TP.

- Audit
  - C4: Tutti i TP devono essere certificati per scrivere su un CDI append-only (il registro) tutte le informazioni necessarie per consentire la ricostruzione della natura dell'operazione.
  - Non è necessaria nessuna regole di enforcment (applicazione)

- Unconstrained Data Items (EDI), ad esempio: informazioni digitate dall'utente sulla tastiera
  - C5: Qualsiasi TP che accetta un UDI come valore di input deve essere certificato per eseguire solo una trasformazione valida, oppure nessuna trasformazione, per qualsiasi possibile valore dell'UDI. La trasformazione dovrebbe portare l'input da un UDI a un CDI, altrimenti l'UDI verrà rifiutato. In genere si tratta di un programma di modifica.
  - E4: Solo l'agente autorizzato a certificare le entità può modificare l'elenco delle entità associate ad altre entità: specificamente associate a un TP. Un agente che può certificare un'entità non può avere alcun diritto di esecuzione rispetto a tale entità.

L'insieme di tutte queste regole ha come obiettivo quello di garantire l'integrità del sistema. I vantaggi di questo modello sono che affronta in modo più completo il problema dell'integrità e che modella le carenze degli ambienti. Per gli ambienti commerciali non è ben formalizzato, dal momento che è difficile ragionare sulle proprietà di sicurezza.

1.1.6 Chinese Wall

Il modello Chinese Wall è un tipo di separazione dinamica dei compiti MAC per proteggere la confidenzialità. L'obiettivo è quello di impedire flussi informativi che causano conflitti di interesse per singoli consulenti (ad esempio, un singolo consulente non dovrebbe disporre di informazioni su due banche o due compagnie petrolifere). Gli oggetti sono organizzati gerarchicamente su tre livelli:

- oggetti di base (ad es. File)
- company dataset: raggruppano oggetti che fanno riferimento a una stessa società
- classi di conflitto di interessi: raggruppano tutti i set di dati della società in concorrenza fra loro.

La regola alla base di questo modello, chiamata simple security rule, stabilisce che: un soggetto s può accedere a un oggetto o solo se:

- o si trova nello stesso set di dati dell'azienda di tutti gli oggetti a cui l'utente s ha già avuto accesso, ovvero all'interno del wall.
- o appartiene a una classe di conflitto di interessi completamente diversa.

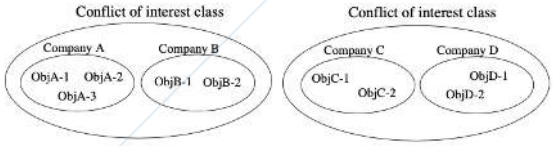


Figura 27: Suddivisione degli oggetti nel modello Chinese Wall

Quindi, prendendo come esempio la Figura 27 se l'utente s ha avuto accesso ai dati della compagnia a in un futuro non potrà accedere ai dati della compagnia b, facente parte dello stessa classe di conflitto. Gli utenti potrebbero dover confrontare le informazioni di diverse società, quindi una semplice regola di sicurezza potrebbe essere troppo restrittiva, quindi si può consentire la "sanificazione" delle informazioni. La sanificazione assume la forma di mascherare le informazioni aziendali, in particolare per impedire la scoperta dell'identità di una società.

Inoltre, si utilizza la *-property per evitare flussi di informazioni derivati dalla collisione di più utenti. Un soggetto s può scrivere un oggetto solo se l'accesso è consentito dalla regola semplice di sicurezza e nessun oggetto che si trova in un set di dati dell'azienda diverso da o e che contiene informazioni non autorizzate può essere letto da s (secondo le autorizzazioni). Prendiamo come esempio la Figura 27, supponiamo che Alice possa leggere l'oggetto ObjA-1 e scrivere l'oggetto OBjC-1 e che Bob potesse leggere l'oggetto OBjC-1 e scrivere l'ObjB-1. Se questo fosse permesso, Alice potrebbe scrivere informazioni su A nel file C, al quale Bob ha accesso. Il problema è che Bob, a sua volta, ha accesso a B in scrittura, la quale è una compagnia in conflitto con la compagnia A.

La semplice regola di sicurezza impedisce il flusso da parte di un singolo utente. La regola * -property impedisce flussi indiretti che possono essere emanati da collusioni tra utenti. La sanificazione offre maggiore flessibilità rispetto all'applicazione della politica. In questo modello si può assumere che sia applicato una politica di accesso di tipo discrezionale (DAC). Il modello Chinese Wall non è completamente formalizzato e lascia aperti problemi, come:

- conservare e gestire la cronologia degli accessi
- assicurare l'accessibilità (ad esempio: Se tutti gli utenti leggono lo stesso set di dati, il sistema non sarà utilizzabile)
- la sanificazione dei dati non viene risolta (ed è un problema complesso)

Questo modello rimane un contributo importante, introducendo il concetto di separazione dinamica del dovere e la definizione di conflitti di interessi.

Il principio di separazione del dovere consiste nel fatto che nessun utente (o insieme limitato di utenti) dovrebbe avere privilegi sufficienti per poter abusare del sistema. Essa si può suddividere in:

- statica, ossia chi specifica le autorizzazioni deve assicurarsi di non dare "troppi privilegi" a un singolo utente
- dinamica, in cui il controllo sulla limitazione dei privilegi viene applicato in fase di esecuzione: un utente non può utilizzare "troppi" privilegi, ma può scegliere quale utilizzare. Di conseguenza, il sistema negherà altri accessi. Questa modalità è più flessibile.

Ad esempio, le operazioni da svolgere nel sistema sono: ordinare beni, spedire ordine, ricevere ricevuta e pagare. In questo sistema ci sono quattro dipendenti. Un requisito di protenzione consiste nel fatto che almeno due persone devono essere incluse nel processo, ossia l'intero processo non può essere svolto da un unico dipendente. Nel caso di separazione del dovere statica,

l'amministratore assegna compiti agli utenti in modo che nessuno possa eseguire tutte e quattro le operazioni, ossia stabilisce che cosa possono fare gli utenti, in modo che non ci sia un unico utente in grado di fare tutte le operazioni. Nel caso di separazione dinamica, ogni utente può eseguire qualsiasi operazione, ma non può completare il processo ed eseguire tutte e quattro le operazioni. All'inizio i quattro dipendenti possono effettuare tutte e quattro le operazioni, senza alcun vincolo, ma se l'utente Alice eseguisse le prime tre operazioni e tentasse di effettuare l'ultima, l'operazione verrebbe bloccata.

1.1.7 Autorizzazioni in espansione

Nel caso di modelli DAC, i sistemi sono basati sulla definizione di autorizzazioni che nella forma base è composta da: *(azione, soggetto, oggetto)*. É possibile espandere il modello base delle politiche DAC, per renderle maggiormente flessibili. Vengono espanse le regole, supportando:

- Regole per gruppi di utenti: Utenti raccolti in gruppi e autorizzazioni specificate per i gruppi
- Regole Condizionali: validità delle autorizzazioni dipendente dal soddisfacimento di alcune condizioni. Le condizioni hanno un impatto sulle autorizzzioni. Le condizioni possono essere:
  - dipendenti dal sistema, per valutare la soddisfazione dei predicati del sistema: posizione, tempo.
  - dipendenti dal contenuto in base al valore dei dati (DBMS).
  - dipendenti dalla storia in base alla cronologia delle richieste.

L'aggiunta dei supporti è relativamente facile da implementare in sistemi semplici, ma può introdurre complicazioni nei modelli più ricchi.

Un aspetto che quasi tutti i modelli DAC supportano sono i gruppi. Infatti, le specifiche per singole entità (utenti, file, ...) sono troppo pesanti, quindi si supportano le astrazioni (raggruppandole). Di solito, si utilizzano le relazioni gerarchiche: utenti/gruppi; oggetti/classi; file/directory; etc. Le autorizzazioni possono propagarsi lungo le gerarchie. Il supporto delle gerarchie può essere applicato a tutte le dimensioni delle autorizzazioni:

- Soggetti (ad es. Utenti vs gruppi)

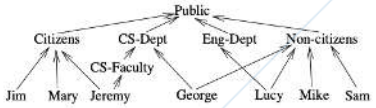


Figura 28: Esempi di gruppi a livello di soggetto

- Oggetti (ad es. File vs directory, oggetti vs classi)

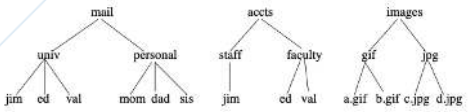


Figura 29: Esempi di gruppi a livello di oggetto

- Raggruppamento a livello di azione, ad esempio, Modalità di scrittura. Prendiamo come esempio, la seguente relazione di raggruppamento: Scrittura $\succeq$ Lettura, significa che ogni qual volta un utente ottiene un'autorizzazione in scrittura ottiene automaticamente i privilegi in lettura.

L'utilità delle astrazioni è limitata, se non sono possibili eccezioni. Infatti, l'utilità delle astrazioni è rafforzata dall'introduzione delle autorizzazioni negative. Ad esempio, tutti i dipendenti tranne Sam possono leggere un file, supportando le autorizzazioni negative possiamo utilizzare le seguenti autorizzazioni:

- (Dipendenti, leggi, file, +): tutti gli employees possono leggere il file
- (Sam, leggi, file, -): Sam non può leggere il file.

La presenza di autorizzazioni e smentite può portare incoerenze, ossia per una determinata richiesta c'è un'autorizzazione positiva e una negativa. Come dovrebbe gestire il sistema queste incongruenze? Bisogna trovare un modo semplice per supportare le eccezioni tramite autorizzazioni negative. Le autorizzazioni negative sono state inizialmente introdotte da sole come: politica aperta: tutto ciò che non viene esplicitamente negato può essere eseguito; al contrario di politica chiusa: possono essere eseguiti solo accessi esplicitamente autorizzati. Le recenti politiche ibride supportano entrambi, ma cosa succede se per un accesso abbiamo sia una richiesta positiva che una richiesta negativa? Si ha una situazione di Inconsistenza. Cosa succede se per un accesso non abbiamo né una richiesta positiva né una richiesta negativa? Si ha una situazione di incompletezza.

L'incompletezza può essere risolta da entrambe le soluzioni:

- presupponendo completezza: per ogni accesso deve esistere una negazione o un'autorizzazione, ma la soluzione è troppo pesante
- assumendo chiuso o aperto come decisione predefinita di base. Se manca un'autorizzazione esplicita, si usa la politica di default.

Di seguito, elenco tre esempi di politiche per la risoluzione dei conflitti (inconsistenza):

- denials-take-precedence: l'autorizzazione negativa vince (principio di sicurezza). Nel dubbio, nego.
- most-specific-takes-precedence: l'autorizzazione più specifica vince.
- most-specific-along-a-path-takes-precedence: l'autorizzazione che è "più specifica" vince solo sui percorsi che la attraversano. Le autorizzazioni si propagano fino a quando non vengono sostituite da autorizzazioni più specifiche

Ne esistono altre di politiche, ad esempio il duale di denials-take-precedence.

A partire dalla Figura 30 che rappresenta le autorizzazioni esplicite (negative e positive), vediamo come vengono applicate le politiche di most-specific-takes-precedence e most-specific-along-a-path-takes-precedence. Nella Figura 31 vediamo l'applicazione delle due politiche. Per quanto riguarda most-specific, G_2 e G_4 sono più specifiche di G_1, quindi su i nodi inferiori vengono propagate le loro autorizzazioni piuttosto che quella di G_1. u_1 prende la regola più specifica tra G_5 e G_6, ossia l'autorizzazione relativa a G_5. Invece, quando viene applicata la politica most-specific-along-a-path, u_1 entra in una situazione di conflitto, perché sul percorso G_5 e G_6 risultano essere allo stesso livello.

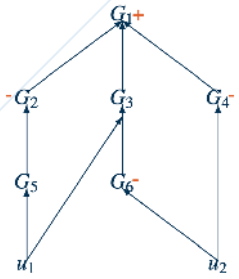


Figura 30: Autorizzazioni esplicite

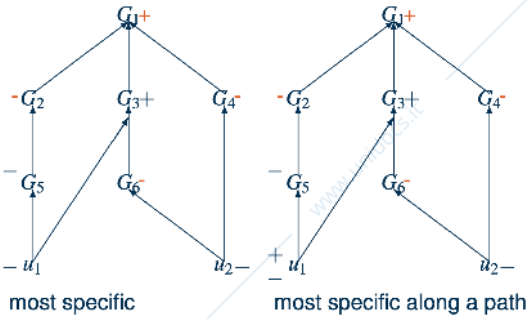


Figura 31: Esempi di applicazioni di politiche

La politica most-specific è più naturale e intuitiva, ma cosa è più specifico se si hanno più gerarchie?

Regola 1: (Employees, leggi, file1, +)

Regola 2: (Sam, leggi, directory1, -)

File1 appartiene a directory1, quindi esiste una doppia dichiarazione. Sam può leggere file1, per via della Regola 1, ma non può leggere directory1, per via della Regola 2. Se considero la specificità per soggetto Sam è più specifico di Employees, ma se considero gli oggetti File è più specifico di directory1. In alcuni casi, l'inconsistenza non è voluta. Ad esempio, nel caso di autorizzazioni che non consentono eccezioni: Non voglio che nessuno sia in grado di proibire questa regola: (Dipendenti, leggi, bacheca, +). Non voglio che nel caso sussistano queste due regole:

(Dipendenti, lettura, budget, +)

(Temporary_employees, lettura, budget, -)

la mia limitazione sui dipendenti temporanei venga ignorata.

Esistono altri modelli per la risoluzione di conflitti:

- Per risolvere i conflitti erano stati introdotti due tipi di autorizzazioni, per stabilire una priorità tra autorizzazioni (livelli di priorità):
  - Strong, vanno sempre a sovrascrivere le autorizzazioni weak
  - Weak, si sovrascrivono tra di loro usando la politica del più specifico

  Alcune limitazioni di questa tipologia di risoluzione dei conflitti sono:

  - Supporta solo due livelli. Sono sufficienti? Sulla base di cosa stabilisco il numero di livelli?
  - Dal momento che le autorizzazioni di tipo strong non si possono sovrascrivere, significa che devono essere consistenti tra loro. Ciò non è semplice quando i gruppi sono dinamici.

- Explicit priority: le autorizzazioni hanno associato priorità esplicite. Sono difficili da gestire.
- Positional: la forza delle autorizzazioni dipende dall'ordine nell'elenco delle autorizzazioni. Attribuisce la responsabilità della risoluzione esplicita dei conflitti all'amministratore della sicurezza. L'amministrazione controllata è difficile da applicare.
- Grantor-dependent: la forza delle autorizzazioni dipende da chi le ha concesse. Devono essere accoppiati con altri per supportare le eccezioni tra le autorizzazioni dichiarate da un singolo amministratore.
- Time-dependent: la forza delle autorizzazioni dipende dal tempo in cui sono state concesse (ad es. le autorizzazioni più recenti prevaricano). Ha una limitata applicabilità.

Diverse politiche di risoluzione dei conflitti non si escludono a vicenda. Ad esempio, posso prima applicare "most specific" e poi "denials-take-precedence" sui conflitti rimanenti. Non esiste una politica migliore delle altre: politiche diverse corrispondono a scelte diverse che possiamo applicare per risolvere i conflitti. Cercare di supportare tutte le diverse semantiche che la negazione può avere (forte negazione, eccezione, ...) può portare a modelli non gestibili. Per questo motivo, spesso le autorizzazioni negative non vengono utilizzate, tuttavia, possono essere utili. I sistemi che supportano le autorizzazioni negative di solito adottano una specifica politica di risoluzione dei conflitti.

Le autorizzazioni possono essere positive o negative. Gli utenti possono specificare un ordine che definisce come interpretare le autorizzazioni positive/negative. Si possono attuare due scelte:

- deny, allow: le autorizzazioni negative vengono valutate per prime e l'accesso è consentito per impostazione predefinita. Ad un richiedente è concesso l'accesso se non ha autorizzazioni negative o ha un'autorizzazione positiva.
- allow, deny: le autorizzazioni positive vengono valutate per prime e l'accesso è negato per impostazione predefinita. A un richiedente viene negato l'accesso se non ha alcuna autorizzazione positiva o se ha un'autorizzazione negativa.

Ad esempio, prendiamo come esempio un ordine deny allow, in cui si hanno le seguenti regole:

- Deny from all
- Allow from .crema.unimi.it

In questo caso valuto prima le autorizzazioni negative e poi positive. I modelli DAC recenti cercano di includere almeno:

- autorizzazioni positive e negative
- propagazione delle autorizzazioni basata su gerarchie
- risoluzione dei conflitti e strategie decisionali
- relazioni di implicazione aggiuntive

L'obiettivo è quello di rendere il modello flessibile e orientarsi verso il supporto di più politiche:

- Diversi amministratori possono avere requisiti di protezione diversi
- Stesso amministratore, ma oggetti da proteggere in modo diverso
- I requisiti di protezione possono cambiare nel tempo

Gli approcci recenti sono basati sull'uso di alcune logiche, ossia su principi logici. Questo garantisce una maggiore espressività e flessibilità, ma dobbiamo stare attenti a bilanciare flessibilità/espressività rispetto alle prestazioni, a non perdere il controllo sulle specifiche e a garantire il comportamento delle specifiche (non dimenticare che stiamo parlando di sicurezza). Alcune proposte consentono interpretazioni multiple, che rendono la semantica delle specifiche di sicurezza ambigua. Le regole logiche sono basate su predicati, alcuni esempi:

- $cando(o, s, < sign > a)$: autorizzazione esplicita.
- $dercando(o, s, < sign > a)$: definisce autorizzazioni implicite. Data un autorizzazione di un gruppo, posso derivare attraverso questo predicato l'autorizzazione per i singoli.
- $do(o, s, < sign > a)$: indica gli accessi che devono essere consentiti o negati.
- $done(o, s, r, a, t)$: Storia degli accessi. Modella la storia degli accessi.
- $error$: vincoli di integrità. Modella eventuali errori.
- $hie - predicates$: predicati gerarchici.
- $rel - predicates$: predicati specifici dell'applicazione (es., proprietario (utente, oggetto), supervisore (utente1, utente2)).

La stratificazione delle regole FAF impone restrizioni senza definizioni dei predicati. Il formato della regola è limitato per garantire la stratificazione delle regole. La regola di default consiste in: $do(o, s, -a) \leftarrow \neg do(o, s, +a)$. La Figura 32 mostra un esempio di stratificazione in cui le regole a livello 1, sono definite considerando le regole a livello 0.

Level	Predicate	Rules defining predicate

0	`hie-predicates`
	`rel-predicates`
	`done`
		base relations.
		base relations.
		base relation.

1	`cando`	body may contain `done`, `hie-` and `rel-`literals.
2	`dercando`	body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals. Occurrences of `dercando` literals must be positive.
3	`do`	in the case when head is of the form `do(_,_,+a)` body may contain `cando`, `dercando`, `done`, `hie-` and `rel-` literals.
4	`do`	in the case when head is of the form $do(o, s, -a)$ body contains just one literal $\neg do(o, s, +a)$.
5	`error`	body may contain `do`, `cando`, `dercando`, `done`, `hie-`, and `rel-` literals.

Figura 32: FAF rule stratification

Alcuni esempi di regole FAF sono mostrate nell'elenco sottostante:

- $cando(file1, s, +read) \leftarrow in(s, Employees) \& \neg in(s, Soft-Developers)$, ossia il soggetto s può leggere il file1 se s è un impiegato e s non appartiene alla categoria dei soft-developers.

- $cando(file2, s, +read) \leftarrow in(s, Employees) \& in(s, Non - citizens)$.
- $dercando(file1, s, -write) \leftarrow dercando(file2, s, +read)$, ossia se s può leggere file2 non può scrivere il file1.
- $dercando(o, s, -grade) \leftarrow done(o, s, r, write, t) \& in(o, Exams)$, ossia s non può autovalutarsi un esame.
- $dercando(file1, s, -read) \leftarrow dercando(file2, s', +read) \& in(s, g) \& in(s', g)$, che consiste nel dire che se s e s' sono in g e s' può leggere il file2, allora $s1$ non può leggere il file1.
- $error \leftarrow dercando(file2, s', +read) \& in(s, g) \& in(s', g)$.
- $error \leftarrow do(o, s, +write) \& do(o, s, +evaluate) \& in(o, Tech - reports)$

La capacità di eseguire attività richiede diversi privilegi. Concedere e revocare tali privilegi può diventare una seccatura, per questo motivo le autorizzazioni del tipo (utente, oggetto) sono autorizzazioni troppo complesse per essere gestite manualmente in database di grandi dimensioni. Il modulo di autorizzazione in espansione guarda le applicazioni e fornisce supporto per:

- Concetti relativi ad applicazioni/attività
- Il privilegio minimo
- Separazione del dovere (statico e dinamico)

1.1.8 Indicazioni recenti nel controllo degli accessi

Alcuni esempi di modelli si applicano in contesti più moderni. Fino ad ora, abbiamo considerato unicamente sistemi centralizzati. Ora consideriamo ambienti più aperti e distribuiti, introducendo nuove funzionalità. Il controllo degli accessi nell'infrastruttura globale (un sistema aperto, come può essere il Cloud) necessita di interagire con parti remote e accedere a risorse remote. Gli accessi visti fino ad ora: (azione, oggetto) possono essere molto limitanti. Ad esempio, nel caso di accesso ad un servizio. I rapporti con l'autenticazione possono cambiare: in alcuni casi non è nemmeno richiesta l'autenticazione (transazioni anonime). In un sistema aperto come Internet, i nuovi utenti (non conosciuti sul server) possono presentare richieste. L'amministrazione di gruppi e ruoli potrebbe non essere centralizzata e il sistema di protezione delle risorse potrebbe non conoscere in anticipo i propri utenti, per questi motivi si utilizza un controllo degli accessi basato sull'uso di certificati digitali (credenziali). Quindi, il controllo degli accessi non si basa più sull'identità, ma sul possesso di certificati digitali (credenziali).

Un approccio più generale a supporto dei certificati consente agli utenti di presentare certificati digitali, firmati da un'autorità di fiducia per fare una dichiarazione, e possono:

- associare una chiave pubblica a un'identità (identità)
- associare una chiave pubblica o un'identità ad alcune proprietà (ad es. Appartenenza a gruppi)
- associare una chiave pubblica o un'identità alla capacità di godere di alcuni privilegi (autorizzazione)

Il server può utilizzare i certificati per imporre il controllo dell'accesso. La gestione dei certificati si riferisce al contesto di:

- Autorità di certificazione
- Infrastruttura a chiave pubblica

- Gestione della fiducia

Il modo in cui viene applicato il controllo dell'accesso può cambiare, dal momento che ciò che gli utenti possono fare dipende da asserzioni (attributi) che possono dimostrare di avere, presentando certificati. Il controllo dell'accesso non restituisce più "sì / no", ma risponde ai requisiti che il richiedente deve soddisfare per ottenere l'accesso. Non si tratta più di un quesito boolano, ma l'accesso è garantito presentando dei certificati per soddisfare i requisiti di sistema. Non solo il server deve essere protetto, anche i clienti vogliono garanzie (ad esempio, privacy). Non si tratta di una presentazione univoca di certificati, ma per l'accesso è possbile utilizzare una qualche forma di negoziazione. Lo scenario di base prevede che la rete sia composta da parti diverse che interagiscono tra loro per offrire servizi (server) e richiedere servizi (clienti). Bisogna tener presente che una parte può agire sia come server che come client (servent nelle reti P2P). Ciascuna parte può avere:

- Un set di servizi che fornisce
- Un portfolio di proprietà di cui la parte gode, ossia un insieme di:
  - dichiarazioni: rilasciate dalla parte e non certificate da alcuna autorità
  - credenziali: certificati digitali (c, K), dove:
    - c: contenuto firmato
    - K: chiave di verifica della firma digitale delle cifre

L'interazione tra client e server si basa sul principio della negoziazione (Figura 33). L'output della negoziazione è l'accesso oppure il rifiuto.

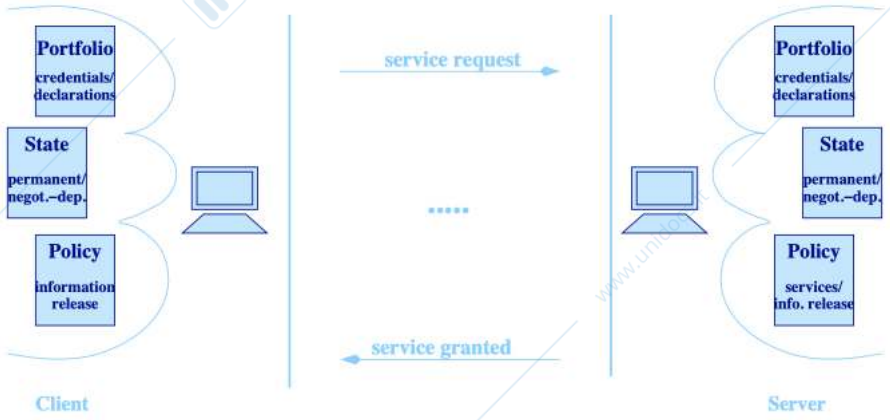


Figura 33: Interazione client-server

I problemi da affrontare in questo scenario sono:

- Espressione delle restrizioni sul controllo degli accessi: linguaggio. Un linguaggio deve avere le seguenti proprietà:
  - supportare il riferimento alle credenziali e alle loro autorità di firma
  - supportare riferimenti dettagliati a proprietà/attributi all'interno di una credenziale
  - supportare tipiche relazioni gerarchiche e astrazioni su servizi e portfolio: fare riferimento a insiemi di servizi/proprietà con un solo nome e utilizzare un modello di accesso graduale ai servizi.
  - solita espressività e flessibilità

  - fornire metapolicies per proteggere la politica in caso di requisiti di comunicazione

- Comunicazione delle restrizioni al controllo dell'accesso da soddisfare, ossia il meccanismo di comunicazione che preserva la privacy sia del client che del server. Questa comunicazione delle restrizioni su cui si basa il controllo dell'accesso si pone l'obiettivo di salvaguardare la privacy delle parti interessate, evitando la necessità di rilasciare certificati o informazioni, se non sufficienti, ma anche evitando perdite di politiche di controllo e informazioni, cioè evitando di rilasciare per intero il meccanismo di controllo. Se un server espone la sua politica di accesso interamente, anche un utente che non può aver accesso, può intuire il contenuto del servizio. É necessario introdurre un filtro e una ridenominazione delle politiche. Quindi, in che modo bisogna comunicare le policy per il controllo degli accessi:

  - In che modo il server deve comunicare al client i requisiti che deve soddisfare per ottenere l'accesso? Come dicevamo prima, non può inviare le sue regole. Potrebbe essere necessario eseguire prima una valutazione parziale (alcune condizioni potrebbero richiedere le informazioni private del server). Inoltre, potrebbe essere necessario eseguire un processo di ridenominazione (per non rivelare la struttura dei suoi controlli)
  - Cosa fa il client alla ricezione dei requisiti? Esistono diverse strategie rispetto alle credenziali/dichiarazioni presentate e, inoltre, può presentare contro-richieste, per assicurarsi che il server sia sicuro. Potrebbe essere necessario garantire condizioni sufficienti.

  É necessario garantire la correttezza dell'intero processo.

L'interscambio di politiche e certificati può essere eseguito in due modalità differenti:

- l'intera politica viene comunicata contemporaneamente:

  - il server comunica la politica (possibilmente una disgiunzione di condizioni)
  - il client può presentare una contro-richiesta (per credenziali/dichiarazioni)
  - se la contro-richiesta è soddisfatta e il client accetta la politica, decide le credenziali/dichiarazioni da presentare

  Questa modalità presenta svantaggi e vantaggi:

  - Vantaggi: Consente di stabilire condizioni sufficienti
  - Svantaggi:
    * Non è sempre possibile senza compromettere le informazioni sensibili nello stato del server (ad es. Controllo di accesso/password)
    * Può divulgare una parte eccessiva di una politica (ad es. 1. registrato, 2. cittadini dell'UE, 3. membro di un'associazione partner)

  Una possibile soluzione è quella di introdurre una fase di negoziazione preliminare.

- politica comunicata passo dopo passo: ad ogni passo ciascuna parte può soddisfare la richiesta della controparte o effettuare richieste aggiuntive

Ci sono ancora molti progressi da fare:

- Strategie per la selezione delle credenziali / dichiarazioni da rilasciare
- Ontologie per il riferimento ad attributi e certificati
- Regolamentazione dell'uso dei certificati

- Valuta la privacy della politica del server
- Sviluppare il modello di negoziazione e la finestra di dialogo
- Sviluppare metapolicies per i requisiti di comunicazione
- Determinazione del recupero di certificati digitali non archiviati in remoto
- Provare proprietà senza rilasciare certificati

I recenti modelli di controllo degli accessi cercano di includere almeno:

- autorizzazioni positive e negative
- propagazione dell'autorizzazione basata su gerarchie di astrazione (ad es. utenti / gruppi, oggetti / classi, etc.)
- risoluzione dei conflitti e strategie decisionali
- relazioni di implicazione aggiuntive
- valutazione aggiuntiva degli attributi
- utilizzo di lingue di alto livello per l'interscambio di politiche

L'obiettivo è rendere flessbile il modello e orientarsi verso il supporto di più politiche. Anche i recenti approcci ricchi risultano limitanti:

- potrebbe essere necessario mantenere e applicare in combinazione diverse politiche indipendenti di controllo degli accessi. Le restrizioni sul controllo degli accessi potrebbero dover "seguire" i dati mentre si muovono nell'infrastruttura, quindi sorge la necessità di composizione di politiche differenti.
- l'autenticazione potrebbe non essere sempre possibile o desiderata, quindi sorge la necessità di controllo degli accessi in base a proprietà/certificati
- potrebbe essere necessario aumentare l'espressività del controllo di accesso (ad es. restrizioni basate sullo scopo) e il supporto di condizioni dinamiche (ad es. pagamento), quindi sorge la necessità di sistemi di controllo degli accessi interattivi
- gli utenti desiderano espressività ma non amano la complessità, quindi sorge la necessità di valutare il trade-off e fornire linguaggi intuitivi per la specifica del controllo di accesso

Le politiche di sicurezza derivano da più requisiti di input:

- Database federati / legacy, datawarehouse. Le politiche di sicurezza di tutte le singole fonti di dati devono essere armonizzate.
- Archivi statistici e Data Broker. I requisiti dei proprietari dei dati devono essere combinati con la politica del repository
- Coalizioni dinamiche. Le organizzazioni si riuniscono per un periodo di tempo limitato e devono stabilire politiche coordinate.
- Leggi sulla protezione della privacy. Deve essere riconciliato con la politica dell'organizzazione.
- Controllo proprietario. Le restrizioni sul controllo degli accessi potrebbero dover "seguire" i dati mentre si muovono nell'infrastruttura.

É necessario combinare le politiche e mantenere l'indipendenza e il controllo sulla composizione. La composizione desiderata si basa sulle seguenti proprietà:

- Supporto politico eterogeneo: linguaggi arbitrari, meccanismi diversi, aperto/chiuso
- Supporto di politiche sconosciute: politiche parzialmente / totalmente sconosciute (scatole nere); alcuni possono solo essere interrogati.
- Interferenza controllata: assicurarsi che la combinazione rispetti il comportamento previsto dei componenti
- Espressività: delle combinazioni (ad es. Privilegio massimo / minimo, livelli di priorità, sostituzione, confinamento, perfezionamento)
- Supporto di diversi livelli di astrazione: facilitare l'analisi e la progettazione delle specifiche e l'amministrazione cooperativa
- Supporto per espressioni dinamiche e modifiche controllate: criteri mobili che seguono (rispettano) i dati
- Linguaggio di specifica di alto livello con approccio formale sottostante

In uno scenario aperto con diverse parti che necessitano di collaborare tra loro, condividendo risorse, può essere necessario comporre politiche delle due parti. Alcuni esempi di operatori di composizione sono:

- Addizione: Unisce due politiche restituendo la loro unione (ad esempio, il privilegio massimo)
- Congiunzione: Unisce due politiche restituendo la loro intersezione (ad es. Privilegio minimo)
- Sottrazione: Limita una politica eliminando tutti gli accessi in una seconda politica (ad es. Eccezioni, divieti espliciti)
- Chiusura: Chiude una politica in base a una serie di regole di derivazione
- Limitazione di scoping: Limita l'applicazione di una politica a un dato insieme di soggetti, oggetti e azioni (ad es. confinamento di autorità)
- Riscrittura: Sostituisce parte di una politica con un frammento corrispondente di una seconda politica (ad es. Sostituzione della politica)
- Sequenziamento: Applica le politiche una dopo l'altra
- Modello: Definisce una politica parzialmente specificata (con parametri)

La Figura 34 presenta la semantica delle espressioni che rappresentano la composizione di policy.

- $[\![P_1 + P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \cup [\![P_2]\!]_e$ e.g., maximum privilege
- $[\![P_1 \,\&\, P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \cap [\![P_2]\!]_e$ e.g., minimum privilege
- $[\![P_1 - P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \setminus [\![P_2]\!]_e$ e.g., exceptions, explicit prohibitions
- $[\![P * R]\!]_e \stackrel{\text{def}}{=} \text{closure}(R, [\![P]\!]_e)$ e.g., for inheritance
- $[\![P\hat{}C]\!]_e \stackrel{\text{def}}{=} \{(s,o,a) \in [\![P]\!]_e \mid (s,o,a) \text{ satisfy } C\}$ authority conf.
- $[\![o(P_1,P_2,P_3)]\!]_e \stackrel{\text{def}}{=} [\![(P_1 - P_3) + (P_2 \& P_3)]\!]_e$ partial overriding
  $o(P_1,P_2,\hat{}C) = o(P_1,P_2,P_1\hat{}C)$
- $[\![\tau X.P]\!]_e(S) \stackrel{\text{def}}{=} [\![P]\!]_{e[X/S]}$ $S \subseteq \mathsf{S} \times \mathsf{O} \times \mathsf{A}$ a function over policies
  $[\![(\tau X.P)(P_1)]\!]_e \stackrel{\text{def}}{=} [\![\tau X.P]\!]_e([\![P_1]\!]_e) = [\![P]\!]_{e[X/[\![P_1]\!]_e]}$

Figura 34: Espressioni policy

La Figura 35 presenta la rappresentazione grafica delle espressioni che compongono più policy.

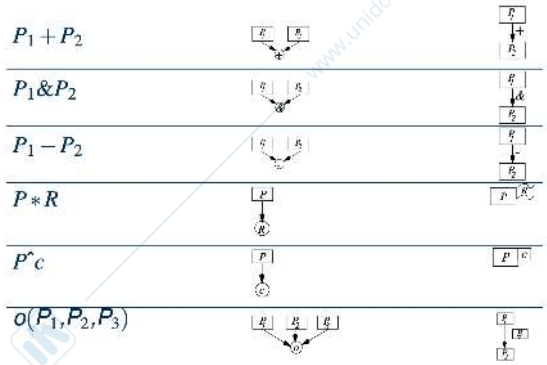


Figura 35: Espressioni policy - Rappresentazione grafica

Un esempio è dato da un dominio di un ospedale. L'ospedale H è composto da tre dipartimenti: Surgery, Radiology, Medicine. Ogni dipartimento ha una sua politica di accesso. Inoltre H non permette nessun accesso al lab_test a meno che il paziene non sia consenziente. Nella Figura 36 è riportata la rappresentazione grafica della politica di accesso dell'ospedale, che è composta dalle politiche dei vari dipartimenti. Infatti, le politiche dei vari dipartimenti vengono composte in unione. Vengono composti con una composizione in and con la politica di consenso del lab_test.

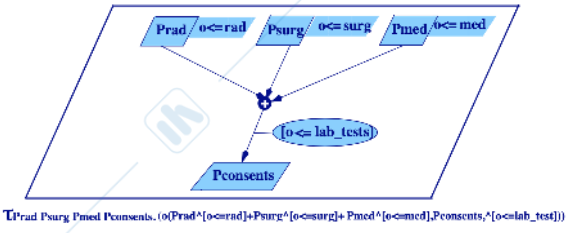


Figura 36: Politiche ospedaliere - Rappresentazione grafica

Inoltre, possiamo aggiungere che il consenso del paziente viene raccolto per mezzo di moduli e propagato dalle regole RH. La Figura 37 riporta l'aggiunta di questa nuova regola.

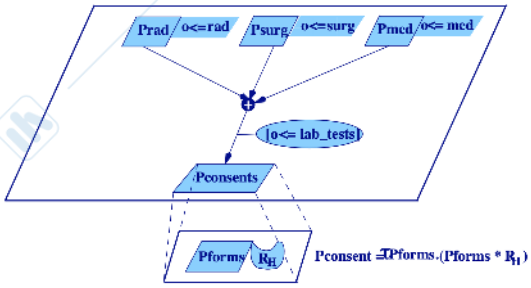


Figura 37: Politiche ospedaliere - Rappresentazione grafica

Le espressioni delle policy vengono applicate traducendole in programmi logici: pe2lp crea un simbolo predicato distinto per ciascun identificatore di policy e per ogni occorrenza dell'operatore. Ogni occorrenza dell'operatore è etichettata con un intero distinto (espressione della politica etichettata). pe2lp accetta un'espressione etichettata e un ambiente come input e restituisce un programma logico per ciascun identificatore di politica $P \rightarrow auth_P$ e per ogni operatore $op_i \rightarrow auth_i$. La Figura 38 riporta il procedimento di conversione.

E	*pe2lp*(E,e)
P	$\{\mathtt{auth}_P(s,o,a) \mid (s,o,a) \in e(P)\}$ if $e(P)$ is defined, $\emptyset$ otherwise
$F +_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z),\ \mathtt{auth}_i(x,y,z) \leftarrow mainp_G(x,y,z)\}$ $\cup$*pe2lp*$(F,e) \cup$*pe2lp*(G,e)
$F \&_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge mainp_G(x,y,z)\}$ $\cup$*pe2lp*$(F,e) \cup$*pe2lp*(G,e)
$F -_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge \neg mainp_G(x,y,z)\}$ $\cup$*pe2lp*$(F,e) \cup$*pe2lp*(G,e)
$F \hat{}_i C$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge C\}$ $\cup$*pe2lp*(F,e)
$o_i(F,G,R)$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge \neg mainp_R(x,y,z),$ $\mathtt{auth}_i(x,y,z) \leftarrow mainp_G(x,y,z) \wedge mainp_R(x,y,z)\}$ $\cup$*pe2lp*$(F,e) \cup$*pe2lp*$(G,e) \cup$*pe2lp*(R,e)
$F *_i R$	$\{\mathtt{auth}_i(s,o,a) \leftarrow \mathtt{auth}_i(s_1,o_1,a_1) \wedge .. \wedge \mathtt{auth}_i(s_n,o_n,a_n) \mid$ $((s,o,a) \leftarrow (s_1,o_1,a_1) \wedge \ldots \wedge (s_n,o_n,a_n)) \in R\}$ $\cup\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z)\} \cup$*pe2lp*$(F,e)$
$\tau_i X.F(G)$	$\{\mathtt{auth}_X(x,y,z) \leftarrow mainp_G(x,y,z)\}$ $\cup$*pe2lp*$(F,e) \cup$*pe2lp*(G,e)

Figura 38: Traduzione in un programma logico

Nonostante i progressi nei modelli, c'è ancora un sacco di livelli da fare:

- Studiare diversi operatori di algebra e linguaggi formali per far rispettare l'algebra e dimostrare le proprietà
- Politiche amministrative e lingua con supporto per più autorità
- Approcci incrementali per applicare le modifiche alle politiche dei componenti
- Convalida della politica meccanizzata
- Applicazione delle politiche (ad es. Materializzazione e valutazione parziale)
- Garantire l'obbedienza delle politiche (quando gli oggetti si muovono)

Le indicazioni attuali nel controllo degli accessi prevedono:

- Supporto delle politiche sticky (e controllo dell'uso secondario, P3P)

- Controllo dell'accesso basato su certificati e attributi
- Controllo degli accessi "interattivo"
- Supporto per condizioni dinamiche (scopo, pagamento, ...)
- Consentire il supporto per l'accesso anonimo e la gestione di identità multiple (ad es. Sistemi peer-to-peer)
- Indirizzare, proteggere e sfruttare specifiche semanticamente complesse

Ci si sta muovendo alla ricerca di flessibilità ed espressività, ma anche di semplicità e gestibilità.

1.2 Esempio di sistema che usa politica MAC: Oracle

Oracle Database è uno tra i più famosi software di database management system sviluppato da Oracle Corporation. Scritto in linguaggio C, che fa parte dei cosiddetti RDBMS (Relational DataBase Management System) ovvero di sistemi di database basati sul modello relazionale affermatosi come standard di riferimento dei database dell'ultimo decennio. Oracle fornisce già un controllo di accesso discrezionale (DAC). Oracle label security (OLS) è un'opzione del database Oracle Edizione Enterprise che lo migliora con un certo supporto di MAC. Quindi, di default viene selezionata la politica discrezionale, ma attivando l'opzione OLS si può decidere di attuare la politica mandatoria. Vengono applicati criteri di sicurezza a livello di riga: per ogni tabella che deve applicare la sicurezza a livello di riga, viene aggiunta una colonna speciale, denominata colonna etichetta. Il processo di mediazione delle etichette funziona confrontando l'etichetta di classificazione associata ai dati con l'etichetta di sicurezza dell'utente. L'accesso ai dati è mediato sulla base di questi fattori:

- l'etichetta associata a una riga di dati
- l'etichetta associata a una sessione utente
- i privilegi dei criteri associati a una sessione utente
- le opzioni di applicazione della politica associate a una tabella

Ogni riga di una tabella può essere etichettata in base al suo livello di riservatezza. Le etichette si compongono di tre elementi:

- Livello: il livello è una componente gerarchica che indica la sensibilità dei dati (ad esempio, pubblico, segreto)
- Compartimenti (opzionale): un insieme di scomparti non gerarchici. In genere, uno o più scomparti sono definiti per separare i dati (ad esempio, i dati possono avere lo stesso livello ma possono appartenere a progetti diversi all'interno di un'azienda)
- Gruppi (opzionale): un insieme di gruppi, l'elemento gruppi viene utilizzato per registrare la proprietà; i gruppi possono essere organizzati gerarchicamente.

Un esempio è dato dall'etichetta: Reserved : Lab : Patient, la quale è composta da:

- Livello = Riservato
- Compartimento = Laboratorio
- Gruppo = Paziente

Questa etichetta associata ad una tupla stabilisce che l'informazione presente all'interno della tupla è confidenziale, appartiene al lab e l'utente a cui fa riferimento è un paziente.

Con una politica Oracle Label Security, l'accesso ai dati è controllato in tre dimensioni:

- Etichette dati. Le tuple sono etichettate per indicare il livello e la natura della loro sensibilità. Rappresenta la sensibilità della tupla.
- Etichette utente. Agli utenti viene assegnata una gamma di livelli, scomparti e gruppi che indicano le loro autorizzazioni per le etichette. Viene fissata quando l'utente accede al sistema, ma si tratta di un range di livelli.
- Privilegi policy. Alcuni utenti possono avere il diritto di eseguire operazioni speciali e di accedere ai dati oltre alle loro autorizzazioni di etichetta.

Un utente può accedere ai dati solo nell'ambito delle proprie autorizzazioni di etichetta. Un utente ha:

- livelli massimi e minimi
- una serie di scomparti autorizzati
- una serie di gruppi autorizzati (e, implicitamente, l'autorizzazione per eventuali sottogruppi)

Ciascun utente ha associato un valore minimo e un valore massimo, che rappresenta la possibilità dell'utente di leggere dati che ricadono in quel range.

Oracle Label Security fornisce interfacce amministrative per definire e gestire le etichette utilizzate in un database. Un amministratore deve definire i livelli, i compartimenti e i gruppi che compongono le etichette, quindi può definire l'insieme di etichette di dati valide per i contenuti del database. L'amministratore può applicare una politica a singole tabelle nel database o a interi schemi di applicazione. L'amministratore assegna a ciascun utente del database i componenti dell'etichetta (e i privilegi, se necessario) appropriati per la funzione lavorativa della persona. Quindi, le politiche MAC vengono stabilite dall'amministratore di sistema. Quando un utente si collega al sistema gli viene assegnata un'etichetta di sessione dall'amministratore. OLS regola l'accesso ai dati in base all'etichetta della sessione assegnata all'utente:

- L'amministratore della politica di sicurezza definisce l'etichetta della sessione iniziale dell'utente (impostazione predefinita).
- L'etichetta della sessione può essere modificata dall'utente in qualsiasi combinazione dei suoi componenti autorizzati.
- L'etichetta della sessione è definita da:
  - livelli minimi e massimi di sicurezza
  - zero, uno o più scomparti autorizzati
  - zero, uno o più gruppi autorizzati
- L'accesso in scrittura deve essere esplicitamente fornito per ogni compartimento e gruppo. Quindi, è sempre l'amministratore che deve dichiarare esplicitamente il privilegio di scrittura.

L'etichetta riga è l'etichetta particolare assegnata per impostazione predefinita ai dati immessi da un utente durante una sessione di lavoro. Può essere impostato a qualsiasi livello, da quello specificato nell'etichetta della sessione corrente dell'utente, fino al livello minimo dell'utente. Può includere solo scomparti e gruppi contenuti nell'etichetta della sessione corrente e per

i quali l'utente ha accesso in scrittura. L'utente può cambiare l'etichetta della riga con qualsiasi etichetta per la quale è autorizzato. Quando l'amministratore configura le autorizzazioni utente, specifica anche un'etichetta di riga predefinita iniziale. La Figura 39 presenta un esempio di etichetta di sessione per un utente. L'etichetta di sessione è composto da:

- Livello = Classified
- Dipartimenti: Fin & Op
- Group = WR

L'utente, sulla base della sua etichetta di sessione, non può avere accesso alla terza e all'ultima tupla. Nel caso della terza tupla, non può accedere perchè la tupla fa parte di un dipartimento, di cui l'utente non fa parte. Mentre, per quanto riguarda l'ultima tupla, il livello di sicurezza assegnato alla tupla è superiore a quello assegnato all'utente.

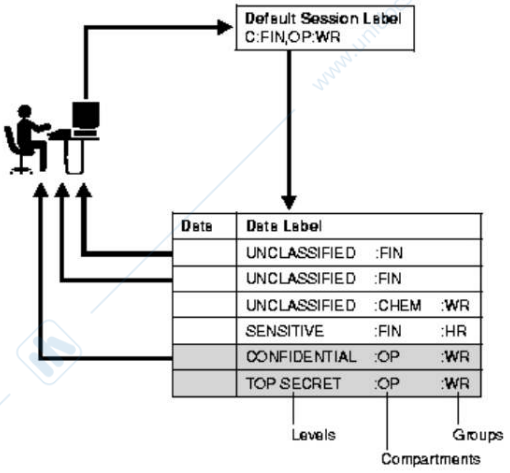


Figura 39: Esempio di etichetta di sessione

Esistono due tipi di autorizzazioni utente:

- Autorizzazioni impostate dall'amministratore (ovvero, l'amministratore imposta esplicitamente un numero di autorizzazioni utente): Livelli autorizzati, Scomparti autorizzati e Gruppi autorizzati.
- Autorizzazioni per etichette di sessione calcolate

Per ciascun utente abbiamo un insieme/range di livelli autorizzati:

- Livello massimo utente (User Max Level): La classificazione massima della sensibilità a cui un utente può accedere durante le operazioni di lettura e scrittura.
- Livello minimo utente (User Min Level): La classificazione minima della sensibilità a cui un utente può accedere durante le operazioni di scrittura. Il livello massimo dell'utente deve essere uguale o superiore al livello minimo dell'utente.
- Livello predefinito utente (User Dafault Level): Il livello assunto per impostazione predefinita durante la connessione a Oracle9i.
- Livello riga predefinito utente (User Row Default Level): Il livello utilizzato per impostazione predefinita quando si inseriscono dati in Oracle9i.

L'amministratore specifica l'elenco di scomparti che un utente può inserire nella sua etichetta di sessione. L'accesso in scrittura deve essere esplicitamente fornito per ogni compartimento. Un utente non può inserire, aggiornare o eliminare direttamente una riga che contiene un compartimento che non è autorizzato a scrivere. L'amministratore specifica l'elenco dei gruppi che un utente può inserire nella sua etichetta di sessione. L'accesso in scrittura deve essere esplicitamente fornito dall'amministratore per ciascun gruppo elencato.

Oracle Label Security calcola automaticamente un numero di etichette in base al valore dell'etichetta della sessione. Si tratta delle etichette di sessione che vengono calcolate in modo automatico dal sistema, le quali includono diverse componenti:

- Etichetta di lettura massima (Maximum Read Label): Il livello massimo dell'utente combinato con i suoi scomparti e gruppi autorizzati.
- Etichetta di scrittura massima (Maximum Write Label): Il livello massimo dell'utente combinato con i compartimenti e i gruppi per i quali all'utente è stato concesso l'accesso in scrittura.
- Etichetta di scrittura minima (Minimum Write Label): Il livello minimo dell'utente.
- Etichetta di lettura predefinita (Default Read Label): Il livello predefinito singolo combinato con scomparti e gruppi che sono stati designati come predefiniti per l'utente.
- Etichetta di scrittura predefinita (Default Write Label): Un sottoinsieme dell'etichetta di lettura predefinita, contenente gli scomparti e i gruppi a cui è stato concesso l'accesso in scrittura all'utente. Il componente di livello è uguale al livello predefinito nell'etichetta di lettura. Questa etichetta viene automaticamente derivata dall'etichetta di lettura in base alle autorizzazioni di scrittura dell'utente.
- Etichetta di riga predefinita (Default Row Label): La combinazione di componenti tra l'etichetta di scrittura minima dell'utente e l'etichetta di scrittura massima, che è stata designata come valore predefinito per l'etichetta dati per i dati inseriti.

Supponiamo che un utente richieda l'accesso in lettura su un sistema su cui vengono applicate politiche MAC. Il sistema effettua una serie di confronti:

- il livello di sessione dell'utente deve essere maggiore o uguale al livello della riga
- il gruppo di sessione dell'utente deve contenere almeno un gruppo specificato nell'etichetta dei dati della riga (o uno dei suoi antenati)
- gli scomparti della sessione dell'utente devono contenere tutti gli scomparti elencati negli scomparti dell'etichetta della riga.

La Figura 40 raffigura tutti i controlli che vengono effettuati per garantire l'accesso in lettura.

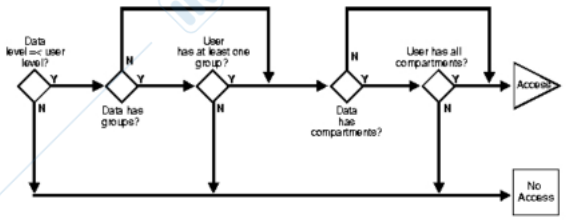


Figura 40: Accesso in lettura

I controlli che, invece, vengono effettuati per consentire il privilegio di scrittura sono:

- il livello della riga deve essere maggiore o uguale al livello minimo dell'utente
- il livello della riga deve essere inferiore o uguale al livello della sessione dell'utente
- se l'etichetta della riga non ha alcun gruppo specificato, l'utente deve avere accesso in scrittura su tutti gli scomparti dell'etichetta della riga.
- Altrimenti:
  - l'etichetta della sessione dell'utente deve contenere almeno un gruppo (o uno dei suoi antenati) con accesso in scrittura specificato nell'etichetta dei dati della riga
  - Gli scomparti specificati nelle etichette della sessione dell'utente devono contenere tutti gli scomparti nell'etichetta dei dati della riga

La Figura 41 raffigura tutti i controlli che vengono effettuati per garantire l'accesso in scrittura.

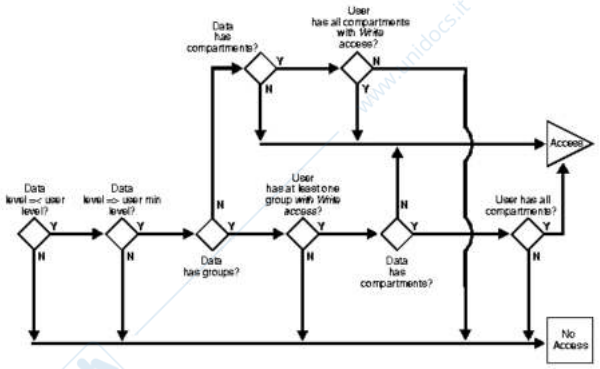


Figura 41: Accesso in scrittura

Lo scopo del riportare il seguente modello è mostrare un esempio di sistema che utilizza una politica di tipo mandatorio (MAC) e per dimostrare che, sebbene le politiche di tipo MAC non siano molto utilizzate in generale, ci possono essere dei contesti in cui è possibile utilizzarle.

1.3 Esempio di sistema che usa politica DAC: OpenStack

Vediamo ora un esempio di sistema che utilizza politiche discrezionali (DAC) e il concetto di overencryption.

Il concetto di database locale è superato, ci si sta muovendo verso un contesto aperto, dove diversi utenti spostano i dati da locale a cloud, un'infrastruttura in cui i dati non sono sotto lo stretto controllo del proprietario. Quindi, l'owner perde il controllo diretto dei file che vengono spostati sul Cloud. OpenStack (a volte indicato come O∼S) è un progetto IaaS (Infrastructure-as-a-Service) cloud computing di Rackspace Cloud e NASA. Si tratta di un software open source che offre un'infrastruttura cloud. L'architettura di Openstack è suddivisa nei seguenti moduli:

- OpenStack Compute (nome in codice Nova): gestione e fornitura di risorse virtuali (istanze VM)
- OpenStack Object Storage (nome in codice Swift): gestione della memorizzazione degli oggetti
- OpenStack Image Service (nome in codice Glance): gestione delle immagini VM
- OpenStack Identity (nome in codice Keystone): servizi di identità OpenStack (autenticazione, autorizzazione, contabilità)

- OpenStack Dashboard (nome in codice Horizon): dashboard della GUI per utenti finali
- OpenStack Networking (nome in codice Neutron, precedentemente Quantum): gestione delle risorse di rete (IP, routing, connettività)
- OpenStack Block Storage (nome in codice Cinder): gestione dei volumi di memoria a blocchi
- Heat: orchestrazione di ambienti virtualizzati (importante per fornire elasticità)

Il modulo che analizzeremo è Swift. Il progetto OpenStack Object Store, noto come Swift, offre un software di archiviazione cloud che consente di archiviare e recuperare molti dati con una semplice API. É progettato per essere scalato e ottimizzato per durabilità, disponibilità e concorrenza nell'intero set di dati. Swift è ideale per la memorizzazione di dati non strutturati che possono crescere senza limiti. Il modulo Swift è un servizio di archiviazione oggetti che consente agli utenti di archiviare e accedere ai dati sotto forma di oggetti. Gli oggetti sono organizzati secondo tre concetti:

- Tenant: client di un cloud pubblico o dominio di applicazione di un cloud privato
- Container: folder che contiene i dati veri e propri
- Object: file veri e propri

Esiste una relazione uno-a-molti tra utenti e tenant. Ogni oggetto è identificato dal percorso tenant/container/object.

Il controllo dell'accesso è basato su politiche DAC, basate su Access Control List (ACL) associate con tenant (autorizzazione di gestione sui container) e container (diverse autorizzazioni di gestione sugli oggetti). Le ACL a livello container sono composti da:

- read ACL: utenti autorizzati a leggere/scaricare oggetti dal contenitore
- write ACL: utenti autorizzati a caricare oggetti nel contenitore
- listing ACL: utenti autorizzati ad elencare il contenuto di un contenitore

In linea con l'approccio di crittografia selettiva, consideriamo le autorizzazioni di lettura. Il controllo dell'accesso si basa sull'utilizzo di diverse chiavi, che servono a proteggere il contenuto degli oggetti, tra le altre cose. Le diverse chiavi sono:

- Data Encryption Key (DEK) k_i. Ogni oggetto o_i è protetto da una crittografia simmetrica usando un k_i DEK. Si tratta di una chiave simmetrica, che critta gli oggetti.
- Master Encryption Key (MEK) m_u. Ogni utente ha una chiave di crittografia master simmetrica personale m_u.
- User encryption key pair $< p_u, s_u >$. Ogni utente u è associato a una coppia di chiavi asimmetriche $< p_u, s_u >$ per la crittografia.
- User signing key pair $< sp_u, ss_u >$. Ogni utente u è associato a una coppia di chiavi asimmetrica $< sp_u, ss_u >$ per la firma di chiavi e oggetti
- Key Encryption Key (KEK). La crittografia di un DEK che un utente può estrarre usando un segreto (chiave) che solo lui conosce

La rappresentazione delle policy è data dai seguenti passaggi:

- • Ogni utente u crea tutti i contenitori necessari, ogni contenitore è associato a un DEK univoco. Vengono raggruppate le sue risorse in gruppi, ognuno dei quali accessibile da un determinato gruppo di utenti. Si tratta di ACL con utenti autorizzati.
- Tutti gli oggetti in un contenitore sono crittografati con il DEK corrispondente.
- Ogni DEK è crittografato con la chiave master dell'utente u e il KEK risultante viene archiviato nel repository di u. $MEK(DEK) = KEK$.
- Per ciascun contenitore e per ogni utente u_j nell'ACL del contenitore, u crittografa il DEK con la chiave pubblica p_{u_j} di u_j e lo firma usando ss_u; il KEK risultante è memorizzato nel repository di u_j. Quindi, se gli oggetti di un container dovessero essere accessibili anche a utenti diversi dall'owner, l'owner critta la chiave DEK con le chiavi pubbliche degli utenti che devono avere accesso, firmandola con la sua chiave pubblica di firma. A questo punto, viene salvato nel repository dell'utente che deve avere accesso a tale risorsa.

La Figura 42 mostra un esempio di repository. Possono avere accesso al container X Alice, Beth e Carla. Alice è owner del file, perché nel repository di Beth e Carla la DEK è crittata con la chiave pubblica.

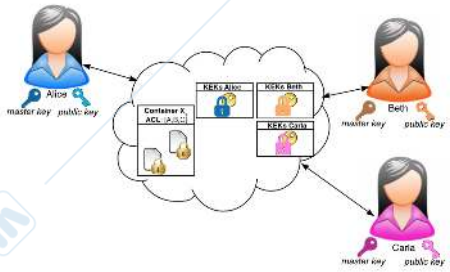


Figura 42: Crittografia basata su criteri in OpenStack Swift

Se l'ACL cambia, cioè se Alice esegue un'operazione di grant ad un nuovo utente David, il procedimento è il seguente:

- L'utente David viene aggiunto all'ACL del contenitore C
- L'utente Alice calcola una nuova chiave KEK per David, che consente di derivare il DEK del contenitore C.

La Figura 43 mostra il risultato dell'operazione di grant.

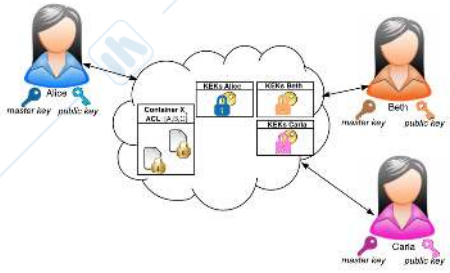


Figura 43: Cambio di policy: operazione di grant

Se, invece, Alice decide di revocare l'accesso a Carla:

- L'utente Carla viene rimosso dall'ACL del contenitore C e la sua KEK viene eliminata.
- L'utente Alice crea un nuovo DEK per il contenitore C
- L'utente Alice scarica, decodifica, crittografa con il nuovo DEK e carica nuovamente tutti gli oggetti nel contenitore C
- L'utente Alice calcola i nuovi KEK per tutti gli utenti autorizzati, che consentono loro di ricavare il nuovo DEK del contenitore C

La Figura 44 mostra il risultato dell'operazione di revoke, a partire dalla situazione iniziale proposta nella Figura 42. La Figura 44 mostra che la chiave di Alice è cambiata (è cambiato il colore del lucchetto.

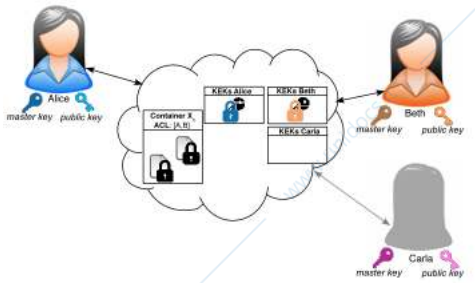


Figura 44: Cambio di policy: operazione di revoke

L'operazione di revoke, così descritta, risulta essere significativa sia dal punto di vista delle performance che dal punto di vista dell'onerosità. Per ovviare a questo processo di revoke molto oneroso possiamo utilizzare l'over-encryption. Prendiamo sempre il caso in cui Alice revoca l'accesso al container C all'utente Carla:

- L'utente Alice rimuove Carla dall'ACL del contenitore C.
- L'utente Alice chiede al server di archiviazione di crittografare doppiamente gli oggetti nel contenitore C con una chiave lato SEL che solo gli utenti non revocati possono derivare:
  - A ciascun contenitore C è associato un DEK a livello BEL che applica la politica di autorizzazione iniziale e un DEK a livello SEL che applica le revoche
  - Esiste una chiave KEK per ciascun utente inizialmente autorizzato per il contenitore C che consente di calcolare il DEK a livello BEL.
  - Esiste un KEK per ogni utente non revocato che consente di calcolare il DEK a livello SEL.

La Figura 45 presenta le tre modalità per implementare l'over-encryption:

- Immediate: risultano doppiamente crittati: lato BEL (lucchetto giallo) e lato SEL (lucchetto rosso).
- On-the-fly: La differenza è che non cambia nulla all'interno del container C, ma ogni volta che si richiede l'accesso, il sistema protegge il dato con un modulo on-the-fly di doppia cifratura. Quando il modulo fornisce la risorsa doppiamente crittata, fornisce anche una chiave KEK, se si è autorizzati, per la decrittazione lato SEL. Il modulo, quando si conclude l'azione di accesso, viene cancellato.

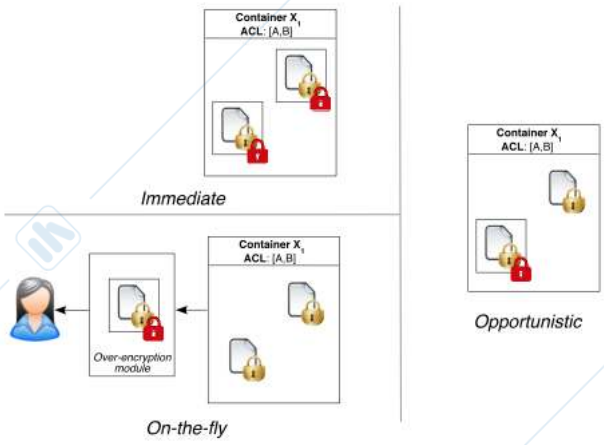


Figura 45: Implementazione dell'over-encryption

- Opportunistic: ogni qual volta che qualcuno accede ad un oggetto, viene fornito un oggetto doppiamente crittato in maniera on-the-fly. Successivamente, però, il vecchio oggetto verrà sostituito con la risorsa crittata anche lato SEL. Quindi, invece che cancellare il modulo, esso andrà a sostituire la risorsa crittata solo lato BEL.

Quando un nuovo oggetto viene inserito in un contenitore, il nuovo oggetto può essere crittografato allo stesso modo degli oggetti nel contenitore (crittografia BEL + eventualmente crittografia SEL). Se non sono state effettuate operazioni di revoca, basta unicamente la cifratura lato BEL. In alternativa, è possibile creare e utilizzare una nuova chiave BEL per crittografare tutti gli oggetti inseriti in un contenitore su cui è stata eseguita un'operazione di revoca. Nuove chiavi KEK che consentono agli utenti non revocati di generare la nuova chiave DEK vengono aggiunti al repository degli utenti. Ciò significa: proteggi la risorsa solo lato BEL, ma con una nuova chiave BEL che non sia conosciuta all'utente che ha subito la revoca. Quindi, a ciascun container vengono associate diverse chiavi BEL, a seconda del momento in cui vengono inseriti nuovi oggetti nel container.

2 Integrità delle query

Lo scenario che prendiamo in considerazione è rappresentato dalla Figura 46 ed è composto da un lato da un cloud provider/server, che ha come obiettivo quello di gestire dati che arrivano da diversi owner, e dall'altro lato diversi client che necessitano di eseguire computazioni (query) coinvolgento diverse fonti informative. Inoltre, il client necessita di controllare che la privacy e l'integrità delle query sia soddisfatta. Quello su cui andremo a soffermarci è l'integrità, quindi andremo ad analizzare tecniche e meccanismi per stabilire se il risultto di una query è integro.

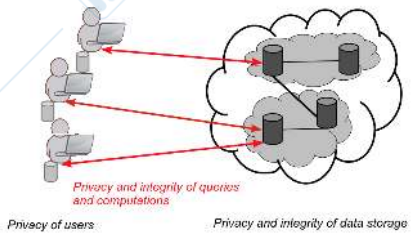


Figura 46: Privacy e integrità di query e computazioni

2.1 Integrità della computazione

La Figura 47 rappresenta la situazione in cui ci troviamo: un server che gestisce informazioni provenienti da diversi data owner. Inoltre, ci sarà qualche client che interroga il sistema, tramite query.

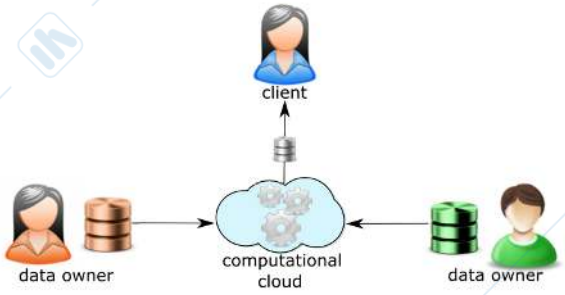


Figura 47: Collaborative queries

Sorge la necessità di elaborare dati eventualmente distribuiti tra più parti collaborative. Per consentire l'esecuzione collaborativa di query è necessario definire soluzioni che garantiscano sia la riservatezza che l'integrità dei dati.

Nella Figura 48 viene presentato un esempio di query. Il client è interessato ad eseguire la seguente query: Select * from T where $5 < A < 11$. Quindi vuole selezionare le tuple dalla tabella T con valore di A compreso tra 5 e 11.

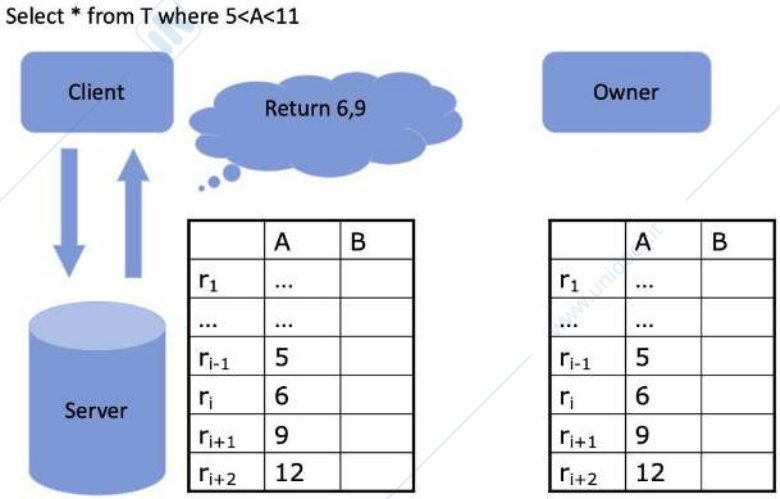


Figura 48: Esempio di query

Come fa l'utente che riceve il risultato a sapere che è integro rispetto alla tabella? Necessita di tecniche per verificare le proprietà di integrità.

Un esempio di risultato non integro è dato dalla Figura 49, in cui nel risultato vengono aggiunte tuple spurie. Il valore 7, rispetto all'istanza corrente, è un valore aggiunto dal server. Quindi, il server ha iniettato un valore spurio, cioè che non ha niente a che fare con la tabella in analisi.

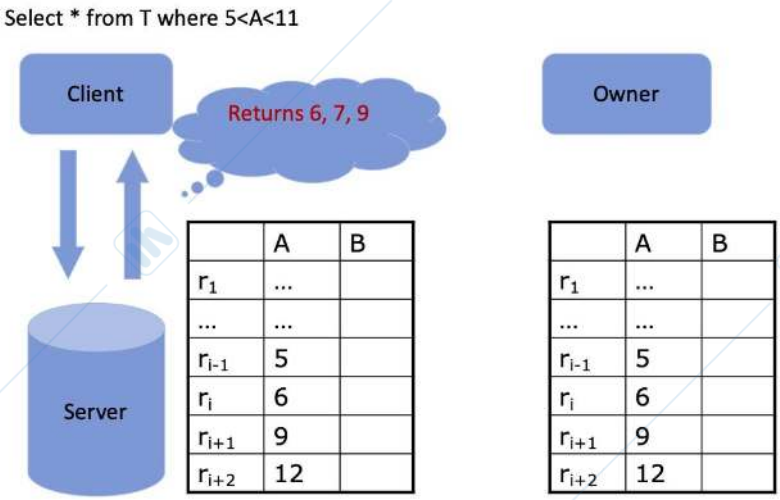


Figura 49: Esempio di injection

Un altro esempio di risultato non integro è mostrato nella Figura 50, in cui viene mostrato un risultato parziale. Il server, nel caso di drop, restituisce un risultato mancante di una o più tuple. Quindi, vengono restituite meno tuple rispetto a quelle che effettivamente soddisfano la query sottoposta.

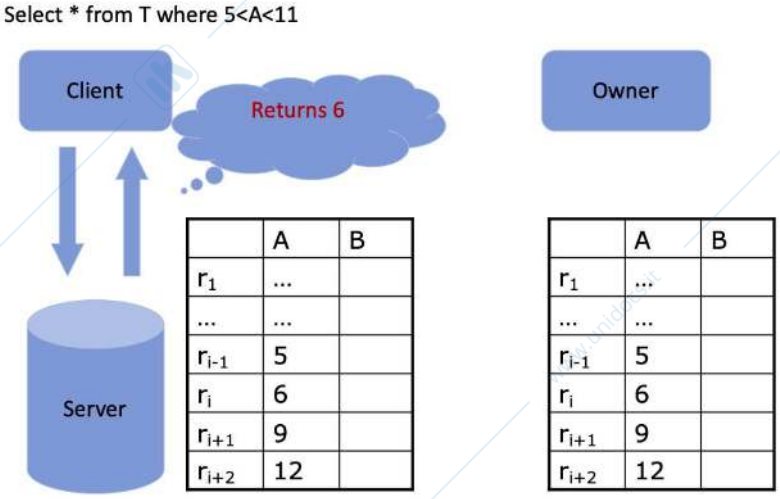


Figura 50: Esempio di drop

Infine, un ultimo esempio di risultato non integro è riportato nella Figura 51. Questo esempio riporta un'altra situazione che può avvenire, ossia quando l'owner modifica la tabella. Ciò porta ad avere una versione più recente della tabella, ma il server continua a restituire i risultati sulla base della tabella meno recente.

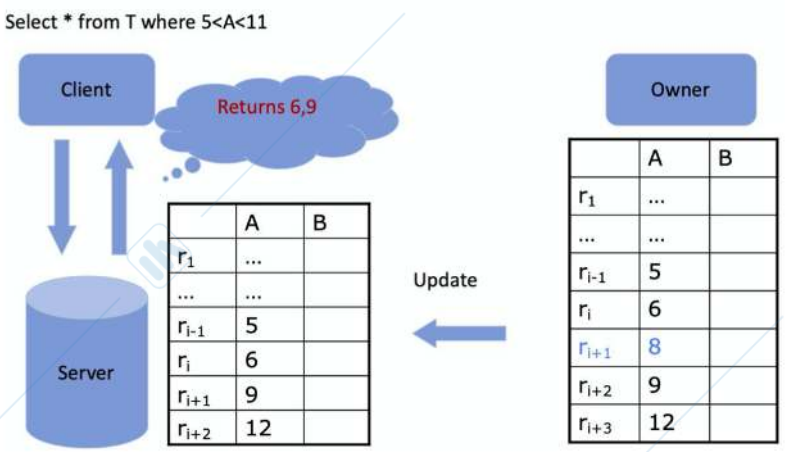


Figura 51: Esempio di omission

Quindi, verificare l'integrità del risultato significa andare a verifica il fatto che non si sia verificata nessuna delle situazioni appena presentate.

Il proprietario e gli utenti dei dati necessitano di meccanismi che forniscano integrità per i risultati delle query. Il concetto di proprietà è composto dai seguenti concetti:

- correttezza: calcolata su dati autentici (evita il caso di injection).
- completezza: calcolata sull'intera raccolta di dati (evita il caso di drop).
- freschezza: calcolata sulla versione più recente dei dati (evita il caso di omission).

Esistono due tipologie di approcci:

- deterministico: utilizza strutture di dati autenticate (ad es. Catene di firme, alberi di hash Merkle, skip list) o soluzioni basate sulla crittografia (ad es. Schema di crittografia omomorfa verificabile [LDPW-14]). Gli approcci deterministici stabiliscono con certezza assoluta l'integrità dei dati. Gli approcci deterministici consistono nell'andare ad associare ai dati che affido al server una struttura dati autentica, costruita a partire da determinati attributi presenti all'interno delle informazioni dell'owner. Le strutture dati autenticate permettono di verificare l'integrità solamente delle query calcolate sugli attributi su cui queste strutture dati sono calcolate.
- probabilistico: sfrutta l'inserimento di tuple false nei risultati di query, la replica di tuple nei risultati di query, oppure token pre-calcolati (ad es. [DFJPS-13b, DFJPS-14, DFJLPS-14b, XWYM-07]). Non stabiliscono con certezza assoluta l'integrità, ma stabiliscono se il risultato è integro con una probabilità. Gestiscono qualunque query, senza alcun tipo di limitazione. L'idea è quella di inserire tuple false all'interno del database, oppure replicare delle tuple o inserire token. Se questi artefatti inseriti non dovessero essere presenti nel risultato c'è qualche problema di integrità. Se, invece, sono presenti non si può stabilire con certezza che il risultato sia integro.

Altri approcci considerano la verifica dell'integrità dei risultati di query complesse (join):

- Tuple false [XWYM-07]:
  - tuple spurie
  - sovraccarico della rete
- Merkle hash tree o sue varianti [LHKR-06 YPPK-09]: supporta solo i join su cui è stato costruito l'albero di hash Merkle

2.1.1 Approcci deterministici

L'approccio deterministico si basa sull'introduzione di strutture autenticate per verificare l'integrità dei dati. La Figura 52 rappresenta l'idea dell'approccio deterministico. L'owner dei dati da' in gestione i dati al server. Inoltre, è presente una struttura autenticata, che viene utilizzata ogni qual volta il server risponde ad una query. Quando il server risponde ad una query, restituisce anche un verificable object (VO), per provare l'integrità.

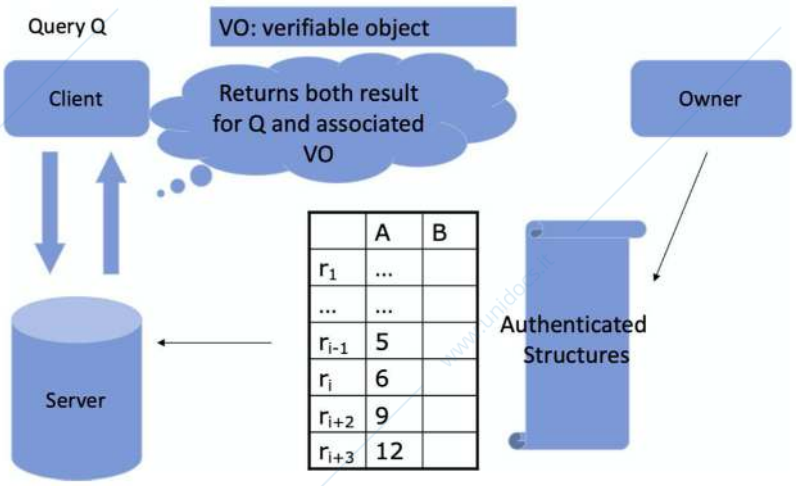


Figura 52: Approccio deterministico: Idea

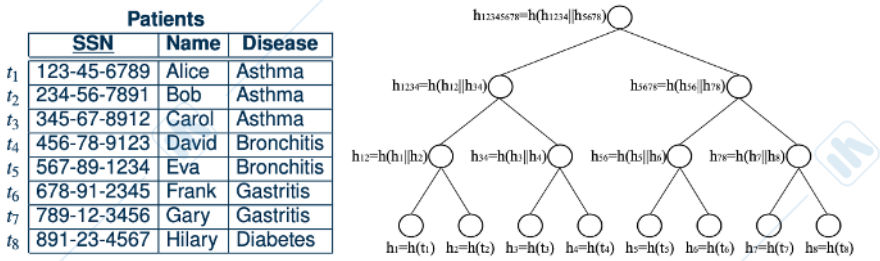
Un primo esempio di approccio deterministico è quello basato sulla firma. Le tuple nel database sono ordinate in base al valore di un attributo A. Le coppie consecutive di tuple sono firmate insieme: $(t_1, s_1); (t_2, s_2); ..., (t_m, s_m); t_x$ con $s_i = S(t_i|t_{i+1})$. $s_1, s_2, ..., s_m$ sono firme calcolate in base alla concatenzaione di t_i e t_{i+1} e così via. L'ultimo valore viene presentato senza firma. Per accelerare l'esecuzione della query sul lato server, un albero B+ viene costruito sull'attributo A. Se il server venisse interrogato e dovesser restituire le tuple comprese nel range $[a, b]$, il processo di esecuzione e verifica della query consiste in:

- Restituire le tuple nell'intervallo $[a-1, b+1]$ insieme alle firme in $[a-1, b]$ (il risultato della query include tutte le tuple in $[a, b]$)
- Verificare ogni coppia di tuple consecutive

Se il risultato è completo, significa che il controllo sulle firme è risultato corretto. Ossia, se le firma sono tutte verificate in modo corretto, significa che non si hanno buchi e che quindi il risultato è completo. Questo approccio si focalizza sul concetto di completezza.

Un secondo approccio deterministico per verificare l'integrità del risultato di una query è l'utilizzo del Merkle hash tree, il quale sfrutta la definizion dati gerarchica che viene associata ai dati veri e propri. Il Merkle hash tree è un Albero binario dove ogni foglia contiene l'hash di una tupla e ogni nodo interno contiene il risultato dell'hash della concatenazione dei suoi figli. La funzione hash utilizzata per costruire l'albero è resistente alle collisioni, ossia la probabilità di trovare due elementi diversi x e y, il cui hash sia identico sia molto bassa. La radice è firmata dal proprietario dei dati e comunicata agli utenti autorizzati. Le tuple nelle foglie sono ordinate in base al valore di attributo A su cui è definito l'albero. L'albero viene creato dal proprietario dei dati e memorizzato sul server. Il server deve gestire la tabella riportata nella Figura 53, in cui SSN rappresenta la chiave. Quindi, il Merkle hash tree è costruito in base all'attributo SSN. Di conseguenza, le tuple vengono ordinate in base all'attributo SSN. Le foglie contengono l'hash calcolato sulla singola tupla (ad esempio, $h_1 = h(t_1)$). I nodi intermedi contengono il valore di

hash dato dalla concatenazione dei due figli del nodo (ad esempio, $h_{12} = (h_1||h_2)$. Il valore di hash associato alla radice viene firmato dall'owner della tabella.

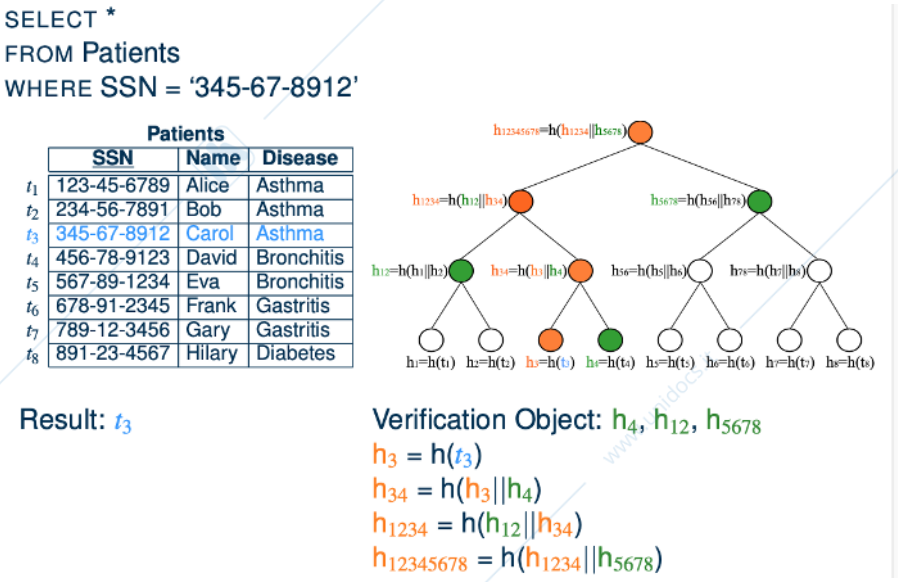


Patients

	SSN	Name	Disease
t_1	123-45-6789	Alice	Asthma
t_2	234-56-7891	Bob	Asthma
t_3	345-67-8912	Carol	Asthma
t_4	456-78-9123	David	Bronchitis
t_5	567-89-1234	Eva	Bronchitis
t_6	678-91-2345	Frank	Gastritis
t_7	789-12-3456	Gary	Gastritis
t_8	891-23-4567	Hilary	Diabetes

Figura 53: Tabella di riferimento

L'albero di hash Merkle definito su A supporta query di uguaglianza e range sull'attributo A. Il server restituisce, insieme al risultato della query, un oggetto di verifica (hash di altre tuple che consente di derivare l'hash della radice). Il client utilizza l'oggetto di verifica e il risultato della query per ricalcolare la radice dell'albero. Il risultato della query è corretto e completo se la radice calcolata è la stessa di quella che conosce; se una tupla non è corretta o manca dal risultato della query, il valore della radice ricalcolato non è lo stesso di quello noto al client.



SELECT *
FROM Patients
WHERE SSN = '345-67-8912'

Patients

	SSN	Name	Disease
t_1	123-45-6789	Alice	Asthma
t_2	234-56-7891	Bob	Asthma
t_3	345-67-8912	Carol	Asthma
t_4	456-78-9123	David	Bronchitis
t_5	567-89-1234	Eva	Bronchitis
t_6	678-91-2345	Frank	Gastritis
t_7	789-12-3456	Gary	Gastritis
t_8	891-23-4567	Hilary	Diabetes

Result: t_3

Verification Object: h_4, h_{12}, h_{5678}
$h_3 = h(t_3)$
$h_{34} = h(h_3||h_4)$
$h_{1234} = h(h_{12}||h_{34})$
$h_{12345678} = h(h_{1234}||h_{5678})$

Figura 54: Verifica mediante Merkle hash tree

Il processo di verifica per una query di uguaglianza è rappresentato nella Figura 54. Il verification object rappresenta l'insieme di valori di hash che il client deve avere a disposizione per calcolare il valore della radice. Il valore h_3 può calcolarlo a partire dal risultato della query (t_3). Combina il valore h_3 con il valore h_4, fornito nel verification object, per ottenere h_{34}. Combina il risultato con h_{12}, fornito nel verification object, per ottenere h_{1234}. Combina il risultato con h_{5678}, fornito nel verification object, per calcolare la radice dell'albero ($h_{12345678}$). I nodi arancioni sono i nodi che l'utente può derivare a partire dal risultato o dalla combinazione con i dati forniti nell'oggetto di verifica; mentre le tuple verdi sono le tuple che vengono fornite dall'oggetto di verifica.

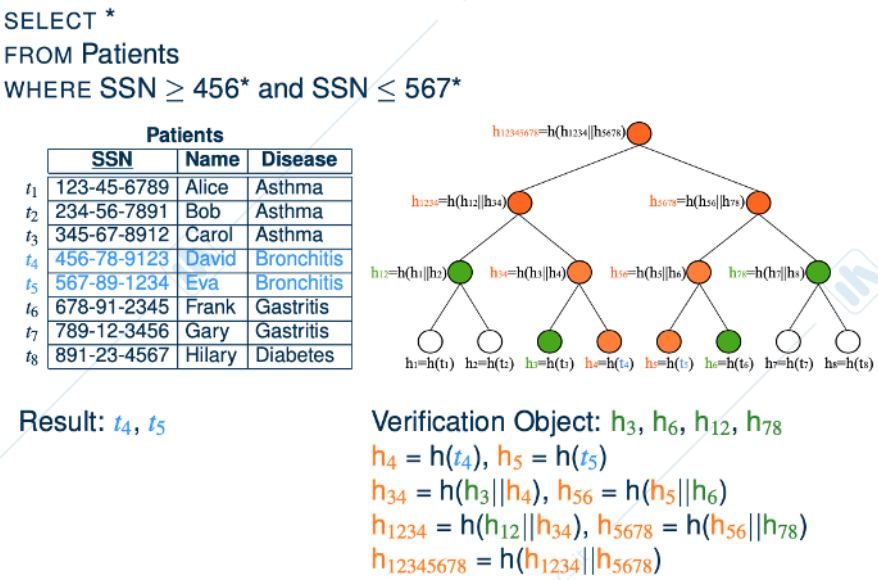


Figura 55: Verifica mediante Merkle hash tree

Il processo di verifica per una query di range è rappresentato nella Figura 55. I valore h_4 e h_5 possono essere calcolati a partire dal risultato della query (t_4, t_5). Combina il valore h_4 con il valore h_3, fornito nel verification object, per ottenere h_{34}. Combina il valore h_5 con il valore h_6, fornito nel verification object, per ottenere h_{56}. Combina h_{34} con il valore h_{12}, fornito dal verification object, per ottenere h_{1234}. Inoltre, combina h_{56} con il valore h_{78}, fornito dal verification object, per ottenere h_{5678}. Combina fra di loro i due valori appena trovati, per ottenere la radice dell'albero: $h_12345678$. Se il risultato calcolato è uguale a quello fornito dal risultato della query, il risultato è integro.

Lo schema di hashing gerarchico sotto forma di albero binario può essere adottato anche con alberi di fanout superiore e con diversi algoritmi organizzativi. L'albero MB funziona come un albero B +: i normali nodi dell'albero B+ sono estesi con un valore di hash associato ad ogni puntatore. Vengono estesi in maniera diversa, a seconda che siano nodi foglia o nodi intermedi. Un esempio è riportato nella Figura 56, in cui p_1 punta ad un nodo foglia che avrà puntatori del tipo $P_{10}, ..., p_{1f}$. Il valore di hash (h) è ottenuto applicando la funzione di hash ai valori di hash presenti nei figli. Quindi, h_{10} viene concatenato con h_{11} e così via fino a h_{1f}. Ad esempio, k_i rappresenta la chiave dell'i-esimo elemento, p_i rappresenta il puntatore e h_i rappresenta l'hash sul record che ha come valore chiave k_i.

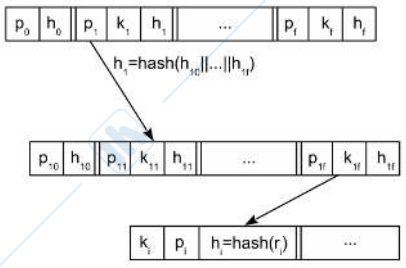


Figura 56: Merkle B-tree

La Figura 57 riporta il processo di verifica mediante l'utilizzo di alberi Merkle B. Inizialmente, stabilisco il range della query (le query di uguaglianza rappresentano un caso particolare di query di range). Stabilisco i limiti del range di interesse: left-bound (LB(q)) e right-bound (RB(q)). Estraggo un sottoalbero rappresentante il range di interesse (LCA(q)). Fino a che non

si raggiunge la radice dell'albero, ossia il valore da verificare per stabilire l'integrità del risultato.

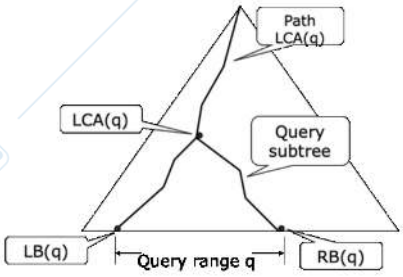


Figura 57: Merkle B-tree: processo di verifica

La Figura 58 rappresenta operativamente il processo di verifica con alberi merkle-B: non viene riportato il valore di hash per semplicità. Il range di interesse della query è dato dall'intervallo tra i valori 6 e 9. Oltre al risultato, vengono restituiti come verification object LB e RB della query in questione, e l'hash dei valori verdi (1, 3, 12, 14, 16, 20, 29, 42), altrimenti l'utente non sarebbe in grado di calcolare il valore della radice. Quindi, mediante l'hash dei valori verdi e rossi, calcolo l'hash associato al valore 10 della radice, al quale concanteno l'hash dei valori 20, 29 e 42. Lo confronto con quallo fornito dall'owner per avere un riscontro riguardo l'integrità del risultato.

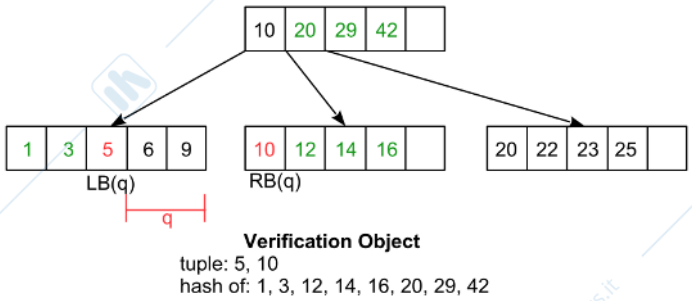


Figura 58: Merkle B-tree: processo di verifica

Un altro esempio di approccio deterministico sono le skip list. Una skip list per un insieme S di elementi distinti (chiavi) è una serie di elenchi $S_0, S_1, ..., S_k$ tali che:

- S_0 contiene le chiavi di S in ordine decrescente insieme alle sentinelle $+\infty$ e $-\infty$.
- S_i, per ogni $i = 1, ..., k$, contiene un sottoinsieme degli elementi in S_{i-1} con probabilità p (ad es. $P = \frac{1}{2}$). Le sentinelle $+\infty$ e $-\infty$ sono sempre incluse nel livello successivo.

Si utilizzano le skip list per supportare le seguenti operazioni ($O(logn)$, dove n rappresenta il numero di elementi di S):

- search (x): determina se x è in S
- insert (x): inserisci x in S
- delete (x): elimina x da S

L'operazione su cui ci concentriamo è l'operazione di ricerca. La ricerca inizia dall'elemento sentinella nella lista in alto. Si procede in orizzontale (hop forward) fino a quando l'elemento

corrente è l' elemento più grande, minore o uguale al target. A questo punto, si scorre verso il basso e si procede come prima fino a raggiungere l'elenco più basso. Quindi ci si sposta dalla lista superiore a quella inferiore fino ad arrivare alla lista S_0. L'elemento restituito nell'elenco più basso è l'elemento più grande nella skip list, ma comunque minore o uguale al target.

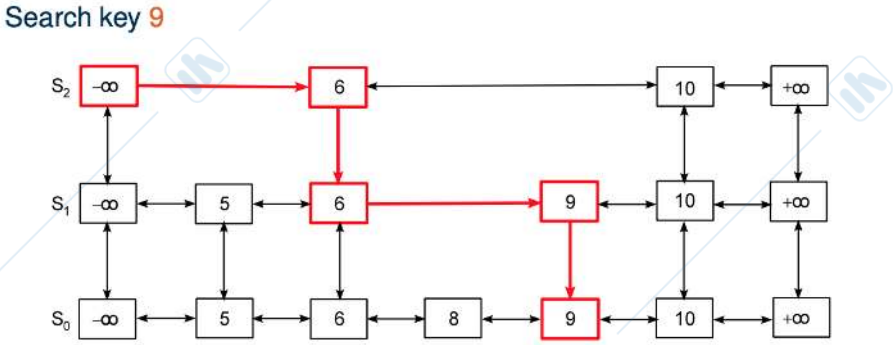


Figura 59: Skip list: operazione di ricerca

La Figura 59 riporta un esempio dell'operazione di ricerca nelle skip list. Si parte dalla lista S_2 e, in particolare, dal valore sentinella $-\infty$ e si si sposta verso destra, cercando il valore più grande, ma comunque inferiore o uguale al valore ricercato. In questo caso, questo valore è 6, dal momento che 10 è maggiore di 9. A qusto punto, ci si sposta verso la riga sottostante effettuando la stessa operazione. Si continua così fino a che non si arriva alla riga S_0.

Le skip list sono state estese per permettere la verifica del requisiti di integrità: Skip list autenticate, le quali sono basate su una funzione hash commutativa (ovvero $h(x, y) = h(y, x)$) e resistente alle collisioni. Ogni nodo n della skip list è associato a un'etichetta $f(n)$ che accumula gli elementi degli insiemi con una funzione hash commutativa h. Come viene calcolata l'etichetta? Sia n un nodo nella skip list e $w = right(n)$ e $u = down(n)$:

- Se $w = null$, allora $f(n) = 0$;
- Se il nodo $n \in S_0$: se w appartiene a S_1, allora $f(n) = h(elem(n), elem(w))$; altrimenti $f(n) = h(elem(n), f(w))$.
- Se il nodo $n \in S_i$, con $i \geq 1$: se w appartiene a S_{i+1}, allora $f(n) = f(u)$; altrimenti $f(n) = h(f(u), f(w))$.

La Figura 60 riporta un esempio di questa funzione di etichettatura: le frecce blu indicano la computazione delle etichette. Prendiamo come esempio l'elemento 5 nella riga S_0: siccome il suo elemento a destra (6) si trova anche in S_1, $f(5) = h(5, 6)$. Ora prendiamo come esempio il suo elemento a destra 6: dal momento che 8 non si trova in S_1, si ha che $f(6) = h(6, f(8))$. Infine prendiamo come esempio il valore 6 nella riga S_1: $f(6) = h(f(6), f(9))$.

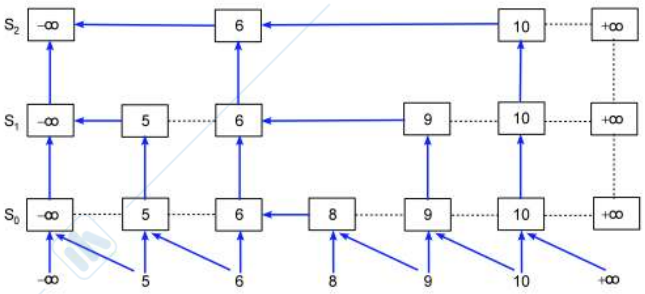


Figura 60: Skip list autenticate

L'obiettivo è produrre una verifica che un elemento x sia o non sia contenuto nell'elenco di salto:

- Se la risposta è "si", bisogna verificare la presenza dell'elemento stesso
- Se la risposta è "no" bisogna verificare la presenza di due elementi x' e x'' memorizzati in nodi consecutivi sul livello inferiore S_0 in modo tale che $x' < x < x''$. In poche parole, bisogna verificare la presenza di due elementi consecutivi che contengono all'interno del range che delimitano, il valore x. Quindi, verifico l'assenza del valore (la risposta alla query sarà).

L'idea è quella di accedere ai nodi che si attraversano in senso opposto. Bisogna utilizzare i nodi, le chiavi o le etichette che vengono restituite in base ai nodi che vengono visitati (una sorta di verification object), al fine di andare a ricostruire l'etichetta associata al nodo di partenza (la sentinella $-\infty$ nella lista di livello maggiore). Quindi, insieme ai nodi accessibili in ordine inverso $P(x) = (v_1, ..., v_m)$ viene restituita una sequenza aggiuntiva $Q(x) = (y_1, ..., y_m)$ di nodi tale che:

- $y_m = f(s)$
- $y_m = h(y_m - 1, h(y_m - 2, h(..., y_1)...)))$

La Figura 61 riporta il processo di verifica di integrità. Si verifica se l'etichetta $-\infty$ nella riga S_2 corrisponde all'etichetta che calcolo attraverso le regole precedentemente esposte. Partendo a ritroso, si calcola l'etichetta del valore 9 della riga S_0: $f(9) = h(9, 10)$. A questo punto si calcola il valore 9 della riga S_1: $f(9) = h(9, 10)$. Dopodiché si prosegue con il valore 6 della riga S_1: $f(6) = h(f(6), f(9))$. Si continua calcolando l'etichetta del valore 6 sulla riga S_2: $f(6) = h(f(6), f(10))$. Infine, si calcola l'etichetta di $-\infty$ e si controlla che corrisponda. La parte in verde della Figura 61 corrisponde al verification object.

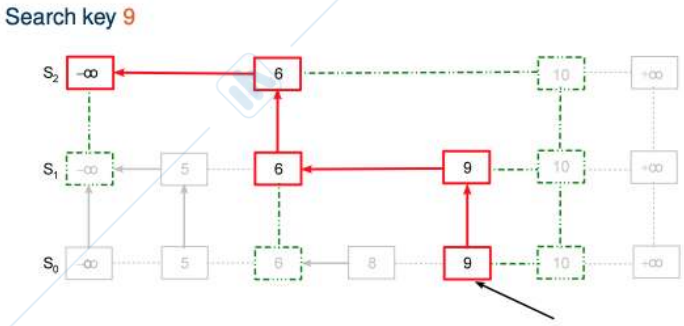


Figura 61: Verifcation of query

2.1.2 Approcci probabilistici

Gli approcci probabilistici sono delle tecniche in grado di stabilire con una certa probabilità se il risultato di una query è integro.

L'idea è quella di inserire tuple false nel database in outsourcing e quindi controllare la loro presenza nel risultato della query. I dati sono crittografati per garantire la confidenzialità dei dati. Le query possono essere eseguite direttamente su dati crittografati. Un attributo speciale viene aggiunto allo schema della relazione esternalizzata il cui contenuto viene calcolato come segue $a_h = H(tid \oplus a_1 \oplus ... \oplus a_n)$. Questo attributo viene aggiunto per garantire la correttezza. a_h rappresenta l'hash di tutti gli attributi. É una sorta di checksum che permette di garantire che la tupla sia autentica, ossia che non è stata modificata. Le tuple false vengono inserite nel database in outsourcing e il client autentica una tupla per dire se è una tupla falsa o una tupla reale:

$$a_h = \begin{cases} H(tid \oplus a_1 \oplus ... \oplus a_n) & t \text{ is real} \\ H(tid \oplus a_1 \oplus ... \oplus a_n) + 1 & t \text{ is fake} \end{cases}$$

Ciò permettere di riconoscere le tuple false da quelle autentiche. Questa tecnica garantisce la confidenzialità e la correttezza, ma non garantisce la completezza e non considera la proprietà di freschezza.

Per garantire anche la completezza viene introdotto l'approccio randomico di introduzione di fake tuples. Le tuple false vengono generate casualmente e vengono archiviate sul lato client. Quando il client ottiene il risultato R_Q per la query Q dal server, interroga Q sulla propria copia di tuple false per determinare quali sono le tuple che dovrebbero apparire in R_Q. A questo punto, si esegue uan sorta di join tra le tuple ottenute dalla query Q e le tuple false presenti in locale. Più semplicemente, il client conta le tuple false in R_q e verifica se tale numero è uguale al numero di tuple false locali corrispondenti a Q. É possibile una dimostrazione formale, che esiste un'equivalenza tra queste due operazioni. Se il numero di tuple false in R_Q è uguale al numero di tuple che ottengo, applicando la query Q sulla copia locale di tuple false, posso stabilire che il risultato sia integro con una certa probabilità. La probabilità dipende dal numero delle tuple false e dal numero di omissioni che il server fa. Più il numero di tuple false inserite aumenta, minore sarà la probabilità del server di rimanere inosservato, quando ne omette qualcuna. Allo stesso modo, più aumentano le omissioni che il server fa più diminuisce la probabilità che l'utente non si accorgerà della mancata integrità. Lo svantaggio che deriva dall'utilizzo di questa soluzione è che deve archiviare tutte le tuple false generate casualmente sul lato client, sulle quali valuta le query. Non è veramente applicabile questa tecnica.

Una variazione dell'approccio è l'approccio deterministico, che non mantiene l'intero set di tuple false, ma la funzione con cui vengono generate. Una funzione predefinita e deterministica viene utilizzata per generare le tuple false $\Im : D_1 \times ... \times D_{n-1} \rightarrow D_n$, con D_i il dominio dell'attributo i-esimo. Sono gli $n-1$ attributi presenti nella tupla, i quali generano l'n-esimo valore dell'attributo. I domini degli attributi sono divisi in griglie discretizzando ogni attributo (se l'attributo è numerico). Il risultato di una query può quindi essere visto come un insieme di griglie interamente coperte o parzialmente coperte. La completezza della query viene quindi verificata contando il numero di tuple false contenute in ciascuna griglia coperta o parzialmente coperta. Supponiamo di avere una tabella con due attributi: Price e Quantity, che hanno valori compresi tra 0 e 100. La Figura 62 riporta la suddivisione del dominio in griglie da blocchi di 25. Si creano così 9 celle. Supponendo di avere una query, stabilisco il range d'interesse della query (il bordo nero): solamente una cella è totalmente coperta. Nessuna delle altre 8 celle è completamente coperta all'interno del range di interesse.

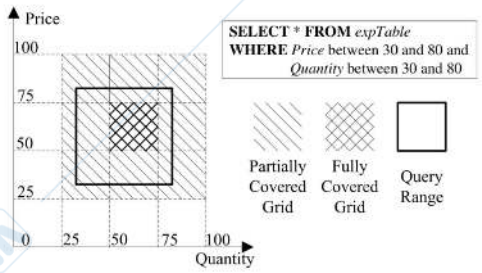


Figura 62: Approccio deterministico

Determinare il numero di tuple false in una griglia completamente coperta è facile (questo numero può essere memorizzato insieme alla griglia). Infatti, il numero delle tuple false è salvato con la cella. Per determinare il numero di tuple false in griglie parzialmente coperte, vengono imposti alcuni vincoli alla funzione $\Im$. Le tuple generate da $\Im$ in una griglia g sono continuamente coperte da una query di intervallo Q.

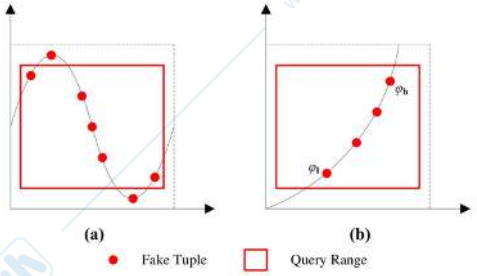


Figura 63: Approccio deterministico

Nella Figura 63 raffigura la funzione $\Im$, dove i pallini rossi rappresentano le tuple false inserite. Nella funzione a sinistra la distribuzione delle fake tuples non è continuamente coperta nel range, ossia i vicini dei punti fuori dal range ricadono entrambi all'interno del range. Nel secondo caso, quello a destra, è più semplice calcolare il numero di finte tuple, perchè le fake tuples sono contenute nel range in modo continuo. Quindi, per contare le finte tuple coperte da Q, è necessario solo trovare i due punti di intersezione tra la funzione $\Im$ e l'intervallo di Q. Una volta stabilito il numero delle tuple false bisogna controlalre che corrisponda al numero ottenuto nel risultato.

Sono disponibili sul mercato diversi CSP che offrono una varietà di servizi (ad es. Archiviazione, calcolo) a prezzi diversi. Gli utenti possono selezionare il CSP che soddisfa meglio i loro requisiti di sicurezza, economici e funzionali. CSP multipli possono aiutare a migliorare la sicurezza, ma è necessaria l'introduzione di soluzioni per verificare il comportamento corretto di questi CSP. Prendiamo come scenario quello presentato nella Figura 64, che prevede l'introduzione di un CSP che offre come servizio la computazione di query. Il problema è che questo computational CSP non è un'entità fidata, quindi bisogna verificare che ciò che fa sia corretto.

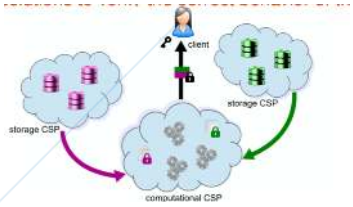


Figura 64: Computazione con provider multipli

Un client, con la collaborazione dei server di archiviazione (entità fidate), può valutare l'integrità dei join eseguiti da un cloud computazionale. Le tecniche di protezione che si introducono sono le seguenti:

- L'Encryption rende i dati incomprensibili. Serve per proteggere la confidenzialità: il cloud computazionale non deve poter leggere i dati, essendo un'entità non fidata.
- Markers, tuple false non riconoscibili come tali dal cloud computazionale (e non in collisione con tuple reali). Quindi, si introducono informazioni fasulle, che non possono essere distinte da quelle reale.
- Twins, replica di tuple esistenti.
- Salt/Bucket, repliche con salts (a lato 1) e tuple fittizie (a lato molti) per appiattire le occorrenze dei match nei join 1:n (1 a molti). Servono per eseguire join 1:n, per mascherare le varie occorrenze.

Un marker mancante o un twin che appare solo sono eventi che testimoniano la violazione dell'integrità. Quindi, posso garantire con una certa probabilità che non ci sia una violazione dell'integrità. La probabilità dipende dalla quantità di controllo (marker e twin) inserita.

La Figura 65 riporta l'approccio probabilisti per le query di tipo join. Consideriamo due server storage (entità fidate): S_l e S_r, le quali conservano rispettivamente le relazioni L e R da unire in una relazione di join. Gli storage server aggiungono twin, marker, salt e bucket alle tabelle che rappresentano le relazioni. Queste tabelle vengono cosnegnate al cloud computazione, dopo averle crittate. Il Computational cloud non è, così, in grado di leggere il contenuto delle tabelle. Il cloud computazionale esegue l'operazione di join e passa la tabella risultante al client, il quale la decritta. Il client controlla tutti i twin e tutti i marker. Se il controllo risulta corretto, mantiene le tuple.

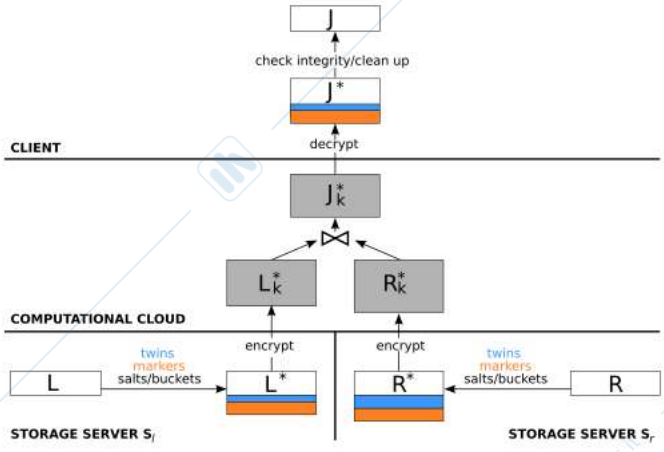


Figura 65: Approccio probabilistico per le query di join

A partire dalle relazioni R_l e R_r e dalla relazione di join J rappresentate nella Figura 66, vediamo come applicare le tecniche di protezione presentate precedentemente.

R_l

	I	Attr
l_1	a	Ann
l_2	b	Beth
l_3	c	Cloe

R_r

	I	Attr
r_1	a	flu
r_2	a	asthma
r_3	b	ulcer
r_4	e	hernia
r_5	e	flu
r_6	e	cancer

J

	L.I	L.Attr	R.I	R.Attr	
l_1	a	Ann	a	flu	r_1
l_1	a	Ann	a	asthma	r_2
l_2	b	Beth	b	ulcer	r_3

Figura 66: Relazioni e operazione join

L'encryption che si usa in questo contesto è l'on-the-fly encryption, dal momento che i server sono elementi fidati. Quindi, non è necessario che le relazioni siano crittate all'interno dei server di storage. Il server S crittografa $B(I, Att)$, ottenendo $B_k(I_k, B.Tuple_k)$. Att può rappresentare più attributi. $B.Tuple_k$ rappresenta la crittazione dell'intera tupla. Per ogni t in B, c'è τ in $B_k : \tau[I_k] = E_k(t[I])$ e $\tau[B.Tuple_k] = E_k(t)$. E è una funzione di crittografia simmetrica con chiave k. k viene definito dal client e cambia ad ogni query, per evitare attacchi di intersezione. La crittografia garantisce la riservatezza dei dati. La Figura 67 riporta le tabelle crittate.

R_{lk}

I_k	L.Tuple$_k$
α	λ_1
β	λ_2
γ	λ_3

R_{rk}

I_k	R.Tuple$_k$
α	ρ_1
α	ρ_2
β	ρ_3
ε	ρ_4
ε	ρ_5
ε	ρ_6

J_k

L.I_k	L.Attr$_k$	R.I_k	R.Attr$_k$
α	λ_1	α	ρ_1
α	λ_1	α	ρ_2
β	λ_2	β	ρ_3

Figura 67: On-the-fly encryption

I markers sono tuple artificiali iniettate in R_l da S_l e R_r da S_r, con le seguenti caratteristiche:

- non riconoscibili dal server di calcolo
- non generare tuple spurie. Nella tabella di sinistra non posso inserire un marker che non si joina con un marker nella tabella di destra.
- inseriti in modo concordato per garantire che appartengano al risultato del join

L'assenza di marker segnala l'incompletezza del risultato del join. Il join viene calcolato su dati crittati. Quindi, non mi devo inventare dati veritieri, per rendere i marker irriconoscibili. Posso anche inserire una stringa casuale, perche queste informazioni verranno crittate. Inoltre, l'attributo di join fra i marker deve essere uguale. La Figura 68 rappresenta l'introduzione dei marker.

R_l^*

	I	Attr
l_1	a	Ann
l_2	b	Beth
l_3	c	Cloe
m_1	x	$marker_1$

R_r^*

	I	Attr
r_1	a	flu
r_2	a	asthma
r_3	b	ulcer
r_4	e	hernia
r_5	e	flu
r_6	e	cancer
m_2	x	$marker_2$

J^*

	L.I	L.Attr	R.I	R.Attr	
l_1	a	Ann	a	flu	r_1
l_1	a	Ann	a	asthma	r_2
l_2	b	Beth	b	ulcer	r_3
m_1	x	$marker_1$	x	$marker_2$	m_2

Figura 68: Markers

La tecnica di protezione twin consiste nella duplicazione di tuple, che soddisfano la condizione C_{twin} che:

- è definita sull'attributo join I, perchè deve essere verificabile su entrambe le tabelle.
- regola la percentuale pt di twin. Il fattore di duplicazione viene definito da una percentuale
- è definita dal cliente e comunicata ai server di storage S_l e S_r

Le coppie gemelle non sono riconoscibili dal server computazionale. Un solo twin che appare segnala l'incompletezza del risultato del join. La Figura 69 rappresenta l'introduzione dei twin.

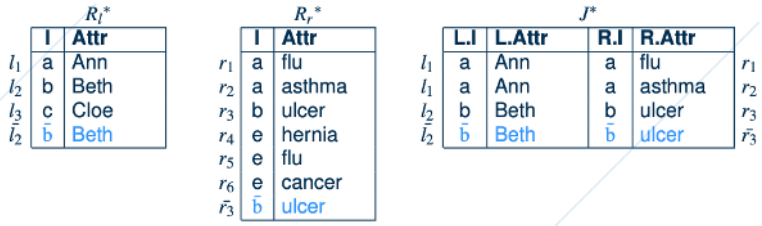


Figura 69: Twin

Se noi utilizzassimo solamente le tecniche precedentemente viste lascerebbero visibili le occorrenze al cloud computazionel. Le diverse occorrenze potrebbero essere utili per identificare twin e marker. I salts e i bucket distruggono le frequenze riconoscibili delle combinazioni in relazioni join uno-a-molti. Operano su tuple originali, marker e twin e possono essere adottati in alternativa o in combinazione. I sali mappano diverse occorrenze dello stesso valore di join sul lato "molti" del join su un diverso valore crittografato utilizzando un sale diverso. Inoltre, replicare ogni tupla sul lato "uno" del join e combinano repliche con sali diversi per garantire l'abbinamento.

I bucket inseriscono tuple fittizie (dummy) sul lato "molti" del join per garantire una distribuzione a frequenza piatta dei valori degli attributi del join. Quindi, permettono di garantire lo stesso numero di tuple per ogni occorrenza.

L'introduzione di sali e bucket non influenzano i dati, ma lo svantaggio è che aumentano la dimensione dei dati in storage. Nella Figura 70 combino l'utilizzo di sali e bucket. Il numero di sali utilizzati è 2. Il massimo numero di occorrenze è 3, quindi si introdurrano bucket da 2 tuple ciascuno. In R_l^* aggiungo i sali, quindi avrò il valore originale e un valore a cui viene applicato il sale. Nella tabella R_r^* aggiungo due elementi dummy per creare bucket da due tuple ciascuno.

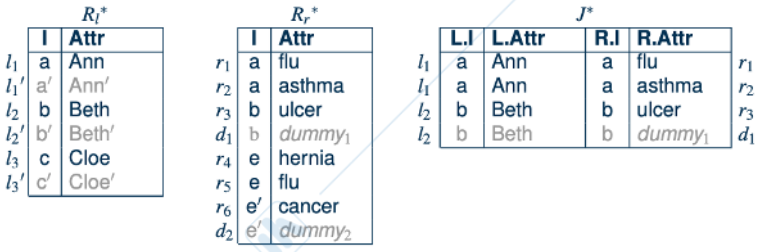


Figura 70: Salt e Bucket

Il client condivide con ogni server S_i una chiave simmetrica k_i, chiave che serve a proteggere l'interrogazione perché la richiesta passa per il computational cloud, entità non fidata. Il client invia al cloud computazionale una richiesta per eseguire un join tra le relazioni prodotte da S_l e S_r. Le relazioni che devono essere prodotte da S_l e S_r sono rappresentate come due stringhe, crittografate rispettivamente con le chiavi k_l e k_r, e devono essere inoltrate dal cloud computazionale al rispettivo server di archiviazione, contenente:

- subquery che deve essere eseguita dal server di archiviazione
- chiave di query k (crittografia immediata) che verrà utilizzata dal server di archiviazione per crittografare (on-the-fly) la relazione inviata al cloud computazionale
- numero m di markers
- percentuale p_t di gemelli
- numero di sali

La Figura 71 riporta un esempio di esecuzione dell'operazione di join. R_l e R_r sono le relazioni a cui si deve applicare l'operazione di join. Ciascun server deve effettuare il Twin di tutte le tuple, tali per cui il valore dell'attributo di join (I) non soddisfa la percentuale. Infatti, ciò che ricevono i server è la percentuale di twin che devono essere presenti. Quali sono le tuple che saranno twinate? Dipende da una condizione che viene calcolata sul valore hashato del valore di join. Nell'esempio in Figura, si suppone che il numero di marker sia 1. Viene supposto che il numero di sali è 2. Quindi, per ogni valore deve essere aggiunto un valore salato (anche per i twin). La dimensione dei bucket è data dalla seguente equazione $(numero_massimo_delle_occorrenze)/(numero_sali)$, arrotondato per eccesso, quindi in questo caso $\lceil 3/2 \rceil = 2$. Quando non riesco a riempire il bucket di tuple reali, aggiungo tuple dummy. Le relazioni, dopo le varie aggiunte e modifiche, vengono crittate e consegnate al Computational Cloud. Il Computational Cloud esegue l'operazione di join. Il Client decritta il risultato e controlla la presenza di twin, marker e sali per verificare l'integrità del risultato. Infine, ripulisce la tabella dai sali, twin e marker.

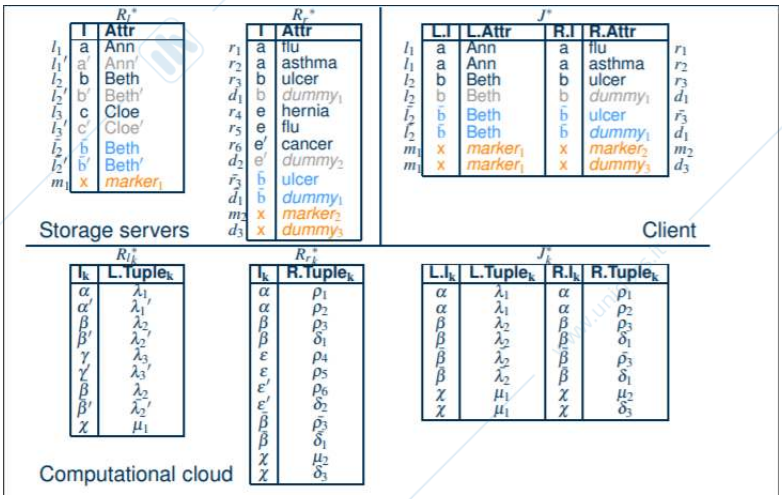


Figura 71: Esecuzione dell'operazione di Join

La garanzia offerta da markers e twin può essere misurata come la probabilità che il Computational Cloud non venga rilevato, quando omette tuple. Markers e twin offrono protezione complementare, ossia l'effetto dei twin e dei marker si bilanciano l'uno con l'altro: i twin hanno efficacia doppia rispetto ai marker, ma perdono la loro efficacia quando il Computational Cloud omette una grande frazione di tuple (caso estremo: tutte le tuple omesse), però i marker consentono di rilevare comportamenti estremi (tutte le tuple omesse) e sono efficaci quando il Computational Cloud omette una grande frazione di tuple. Nel caso di relazione 1:N la presenza di twin e marker sarebbe rilevabile. I sali e i buckets introducono overhead computazionale e di comunicazione. La strategia di esecuzione semi-joinm protegge il profilo di join senza la necessità di introdurre sali e buckets. Supporta join uno a uno, uno a molti, molti a molti e sequenze di join. La Figura 72 riporta la strategia di esecuzione del semi-join. Supponiamo di avere due

storage server S_l e S_r, ognuno contenente una relazione su cui effettuare l'operazione di join. I due storage server effettuano un'operazione di proiezione che estrae l'attributo su cui effettuare il join. Alle tabelle risultanti, vengono aggiunti twin e marker. A questo punto, gli storage server crittano i dati e li consegnano al Computational Cloud, il quale effettua l'operazione di join e passa il risultato al client. Il client decritta i dati e, successivamente, effettua un controllo dell'integrità sui dati ricevuti. Ripulisce la tabella dalla presenza di tuple spurie. Chiaramente il risultato non è completo, perché la tabella risultante è a sua volta il risultato di una proiezione eseguita dagli storage server. Si tratta di una sequenza di tuple contenenti unicamente il valore di join. Quindi, il client chiede le informazioni per completare la tupla ai server. Il server restituisce i dati e il client effettua una fusione (merge) per completare i dati.

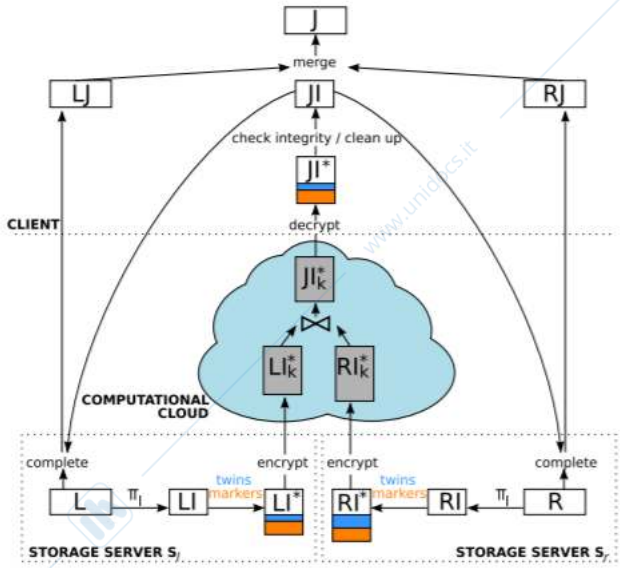


Figura 72: Esecuzione dell'operazione di semi-Join

Alcuni scenari di cloud computazionale supportano l'elaborazione di una grande quantità di dati in parallelo su un gran numero di nodi (ad es. MapReduce, un'infrastruttura distribuita su diversi nodi, chiamati worker). Sorge la necessità di ragionare sui diversi nodi coinvolti nell'applicazione dei controlli di integrità e garantire il controllo ben distribuito tra i diversi nodi e la capacità di riconoscere nodi che si comportano male (responsabilità). Un framework MapReduce supporta l'esecuzione di attività su una grande quantità di dati in parallelo da più nodi (worker), coordinato da un manager (entità fidata). Una funzione di mapping definita dall'utente traduce l'input (tuple da unire) in un set di coppie $< key, value >$. Una funzione di assegnamento f assegna coppie $< key, value >$ ai worker: tutte le coppie con la stessa chiave vanno allo stesso worker $w = f(key)$. Una funzione di riduzione (reduce) definita dall'utente (operazione di join) viene eseguita da ciascun worker e il risultato viene quindi combinato dal manager. La Figura 73 riporta un esempio di join, mediante MapReduce. All'interno del computational cloud si possono distinguere 3 workers. L'operazione di join viene elaborata mediante la strategia di semi-join. Il valore a viene mappato sul worker w_1, mentre il valore b viene mappato sul worker w_2. Il valore c e d vengono mappati sul worker w_3. Infine, il valore e viene mappato sul worker w_2. Ogni worker effettua l'operazione di join, laddove è possibile e ricombina il risultato. Il risultato viene inviato all'utente.

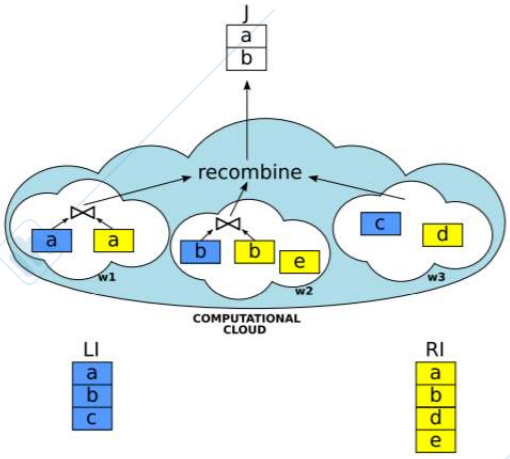


Figura 73: Funzionamento del computational cloud in MapReduce

Ora vediamo come applicare le tecniche di protezione precedentemente presentato, a questo contesto:

- La crittografia viene applicata all'attributo join delle relazioni coinvolte nel join prima che vengano passate al cloud computazionale. Ogni server di archiviazione crittografa la sua relazione B ottenendo $B^k(I_k)$, con I corrispondente all'attributo join di B:
  - per ogni distinta $t[I]$ in B, c'è τ in $B^k : \tau[I_k] = E_k(t[I])$
  - E è una funzione di crittografia simmetrica con chiave k
  - k è definito dal client e cambia ad ogni query

  I valori crittografati vengono tradotti in coppie $< key, valore >$ della forma $< \tau[I^k], B^k >$. Le tuple con gli stessi valori per l'attributo join vengono assegnate allo stesso worker. Nessuna tupla viene persa dal join a causa di un'allocazione errata.

- I marker devono essere distribuiti propriamente tra tutti i worker (per distribuire il controllo). Distribuire in modo proprio significa distribuire in modo tale da controllare tutti i worker. La strategia di distribuzione dei marker è data da $< N, N_{min}, N_{max} >$ con N che rappresenta il numero totale di marker, mentre N_{min}/N_{max} rappresenta il numero minimo/massimo di marker per worker. Esistono diverse tipologie di distribuzioni per i marker:
  - Random $< N, 0, N >$: nessuna condizione per la distribuzione dei marcatori ai worker
  - at-least-n $< N, n, n + (N - n \cdot l) >$: ogni worker deve ricevere almeno n marcatori ($n \leq \lfloor N/l \rfloor$).
  - Equilibrio perfetto $< N, \lfloor N/l \rfloor, \lceil N/l \rceil >$: i marker devono essere distribuiti uniformemente (il numero di marker in una coppia di worker può differire al massimo di uno), quindi ciascun worker deve ricevere più o meno lo stesso numero di worker.

  Tutti i server di archiviazione generano marker con la funzione μ impostata dal client. La Figura 74 rappresenta l'algoritmo di generazione dei marker. I marker spare sono marker ancora da distribuire sui vari worker.

```
Generate_Markers(N, N_min, N_max)
1: spare := N − (N_min ∗ l) /* spare markers */
2: repeat
3:    generate a new marker m via function μ
4:    let w be f(E_k(m)) and n_w be the number of markers already assigned to it
5:    if (n_w < N_min) or (n_w < N_max and spare > 0)
6:    then retain m
7:         n_w := n_w + 1
8:         if n_w > N_min then spare := spare − 1
9:    else discard m
10: until N markers have been allocated
```

Figura 74: Algoritmo per la generazione di marker

Ogni server di archiviazione genera lo stesso set di marker, ossia ogni server produce la stessa sequenza di marker. L'allocazione dei marker ai lavoratori è deterministica. I marker generati sono corretti se:

- per ogni lavoratore $w : N_{min} \leq n_w \leq N_{max}$
- il numero totale di marker assegnati è N

- Anche i twin dovrebbero essere adeguatamente distribuiti sui diversi worker. Non è possibile controllare la generazione dei twin come per i marker, perchè la generazione dei twin dipende dai valori dell'attributo join su ciascun server:
  - ogni server può gemellare tuple diverse a seconda della sua istanza
  - ogni server può osservare un diverso numero di twin per un worker
  - i server non possono coordinarsi per regolare la distribuzione dei twin

  Per l'utilizzo dei twin viene introdotta la strategia di separazione dei twin: un twin non può essere assegnato allo stesso worker della tupla originale. Tutti i server hanno la stessa visibilità su worker, quindi è possibile attuarla in modo che tutti i server la rispettino. Inoltre, la twin separation ha il vantaggio di poter applicare la two-man-rule: un worker mancante t sarebbe esposto dalla presenza di $\bar{t}$ su un lavoratore diverso (e viceversa). Anche questa regola può essere applicata da entrambi senza distinzione. Le tuple twin nei server di archiviazione sono basate sulla condizione C_{twin} e una funzione di generazione del sale σ impostata dal client. La Figura 75 riporta l'algoritmo di distribuzione dei twin all'interno dei vari worker. $\oplus S$ rappresenta l'aggiunta di sale. L'until $\bar{w} \neq w$ serve per garantire che una tupla originale e un suo twin siano sempre in due worker diversi.

```
Generate_Twins(B, C_twin)
1: for each t in B satisfying condition C_twin do
2:    let w be f(E_k(t[I]))
3:    repeat
4:       generate salt s via function σ
5:       t̄ := t
6:       let w̄ be f(E_k(t̄[I] ⊕ s))
7:    until w̄ ≠ w
```

Figura 75: Algoritmo per la distribuzione dei twin

Le coppie di twin faranno parte del risultato del join per forza, perché tutti i server generano due twin con la stessa funzione di generazione e l'allocazione dei twin agli worker è deterministica.

La Figura 76 riporta l'esecuzione di join con l'unione di tutte queste teniche. Come nell'esempio precedente, sono presenti due storage server (S_l e S_r), ognuno contenente una relazione (dove I rappresenta l'attributo di join). I due server effettuano una proiezione sull'attributo di join, sulla quale viene effettuta l'aggiunta di sali e marker da parte del server. Infine, gli storage serve crittano i dati e li distribuiscono sui vari worker, con le regole stabilite precedentemente. Il computational cloud effettua l'operazione di join (reduce), combinata dal manager. Dopodiché, il cloud fornisce al client il risultato, il quale viene decrittato. Il client controlla l'integrità e pulisce i dati dalle tuple spurie. Il client richiede il completamento delle tuple ricevute ai due server, i quali forniscono le informazioni aggiuntive ai client. Infine, il client fonde le informazioni per creare una tabella risultato.

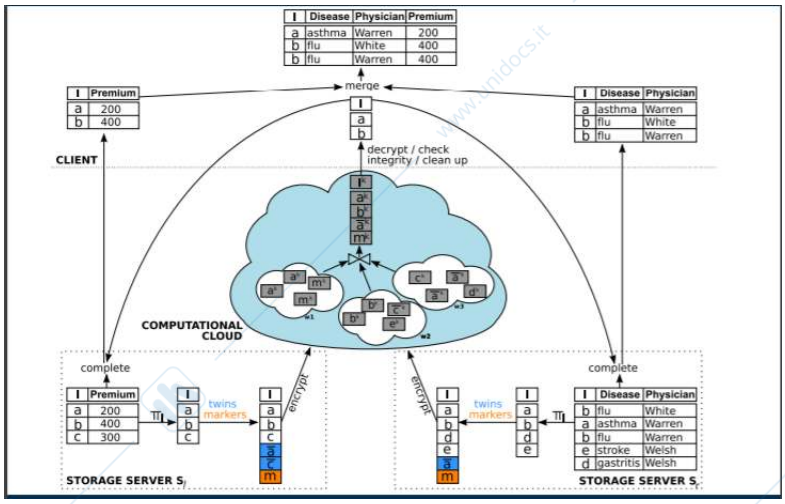


Figura 76: Esecuzione complessiva dell'operazione di Join

Una query può richiedere il calcolo di una sequenza di join su un numero arbitrario (> 2) di relazioni/server. Se l'attributo di join è lo stesso per tutte le relazioni da unire, si tratta di una semplice estensione del caso con 2 relazioni. In caso contrario, il controllo dell'integrità diventa più problematico:

- tutti i join devono rimanere uno a uno
- twin e marker devono sopravvivere alla sequenza di join

L'invio della proiezione degli attributi di join al cloud computazionale espone il profilo di join. Si utilizza il concetto di esecuzione sequenziale, per cui le informazioni inviate per l'i-esimo join devono considerare solo le tuple che sono state filtrate dai join precedenti. Ai server verrà poi richiesto di completare le informazioni. La valutazione di una sequenza di join richiede due passaggi:

- Passaggio 1: Recupera i sub-join. Si tratta di una comunicazione da parte del client e dei server, con mediazione del computational cloud. Il client deve inoltrare allo storage server i pacchetti crittati con la chiave del server, contenenti:
  - Query
  - Id Query
  - Strategia di distribuzione dei marker

  - Percentuale twin
  - Chiave usata nella query per crittare il contenuto

  Il client richeidera ai server di valutare una condizione sulla loro relazione e successivamente di effettuare una proiezione con il valore di join. Successivamente il computational cloud effettuerà l'operazione di join. Il risultato dell'i-esima tupla dei filtri di join è da valutare nell'$(i+1)$-esima.

- Passaggio 2: Completare il risultato con gli attributi rimanenti.Il processo segue un ordine inverso rispetto al Passaggio 1 per filtrare le tuple che non appartengono al risultato finale. L'ultimo join del passaggio 1 è il più restrittivo, a partire da questo risultato, andando indietro fino al primo, ai server di archiviazione viene richiesto di inviare gli attributi rimanenti per le tuple del risultato.

La Figura 69 mostra il risultato dell'operazione di join alla fine del Passaggio 1. La Figura mostra 4 relazione con diversi attributi di join:

- I_1: attributo per l'operazione di join tra la tabella 1 e la tabella 2.
- I_2: attributo per l'operazione di join tra la tabella $(1 \bowtie 2)$ e la tabella 3.
- I_3: attributo per l'operazione di join tra la tabella $((1 \bowtie 2) \bowtie 3)$ e la tabella 4.

Iniziamo con l'operazione $(1 \bowtie 2)$: i due server proiettano le due relazioni sulla base del valore I_1, comprensivo di twin e marker. Nella Figura i valori sono trattati in chiaro esclusivamente per comodità. Dopo che il computational cloud esegue il join fornisce al client il risultato. Dopodiché, seleziono nella tabella 2 le tuple che appartengono al risultato $(\sigma_{I_1} IN\{a,b\})$. A partire dalla tabella risultante dall'operazione di selezione e della tabella 3, bisogna proiettare sulla base dell'attributo I_2. Fornendo al client il risultato dell'operazione $((1 \bowtie 2) \bowtie 3)$. Allo stesso modo, nella tabella 3 selezione le tuple che appartengono al secondo risultato, quindi $\sigma_{I_2} IN\{10,20\}$. Il server, a partire dal risultato di selezione, sulla base dell'attributo I_3. Allo stesso modo, l'altro server calcola la proiezione sulla base di I_3 a partire dalla tabella 4. Ottengo così il risultato dell'operazione $(((1 \bowtie 2) \bowtie 3) \bowtie 4)$.

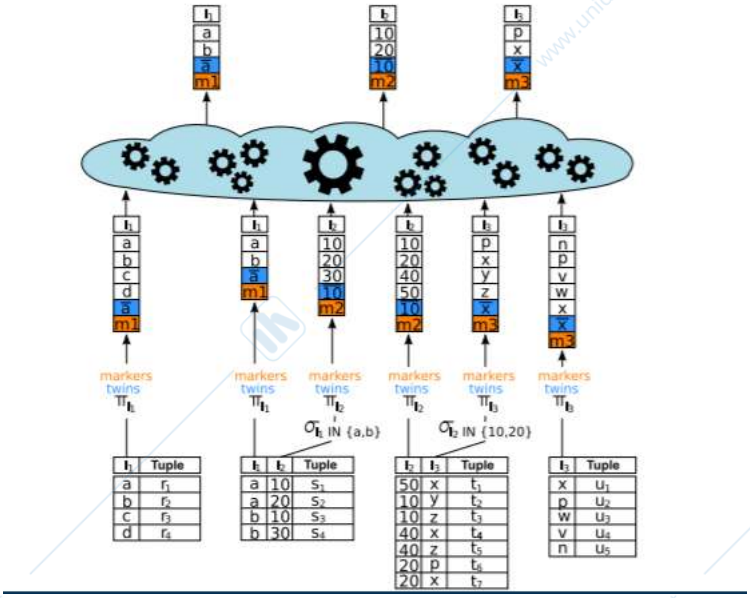


Figura 77: Esecuzione delle sequenze di Join

La Figura 78 mostra il risultato completo dell'operazione di completamento dei dati. Per prima cosa il client che riceve le sub-query controlla l'integrità, mediante la presenza di twin e marker. Se l'integrità risulta inviolata, il client ripulisce il risultato della query da twin e marker. Si parte dall'ultimo risultato, selezionando dalla tabella 4 le tuple per cui $\sigma_{I_3} IN\{p, x\}$. Successivamente, selezioniamo dalla tabella 3 le tuple che soddisfano il join sulla base di I_2, quindi si effettua un operazione di selezione sulla tabella 3 del tipo: $\sigma_{I_2} IN\{10, 20\}$. Il risultato di questa selezione deve essere sottoposto anche all'operazione di selezione su I_3: $\sigma_{I_3} IN\{p, x\}$. Di seguito, nella tabella 2 bisogna selezionare le tuple per cui vale le seguenti operazioni di selezione (collegate fra loro in AND): $\sigma_{I_2} IN\{10, 20\}$ e $\sigma_{I_1} IN\{a, b\}$. Dal momento che, in questo caso, la selezione sulla tabella 2 fornisce come risultato tuple con l'attributo I_1 uguale ad a, la selezione che si effettua sulla tabella 1 è la seguente: $\sigma_{I_1} IN\{a\}$. I risultati delle selezioni sulle varie tabelle vengono ricombinate, mediante l'operazione recombine, a formare un unico risultato.

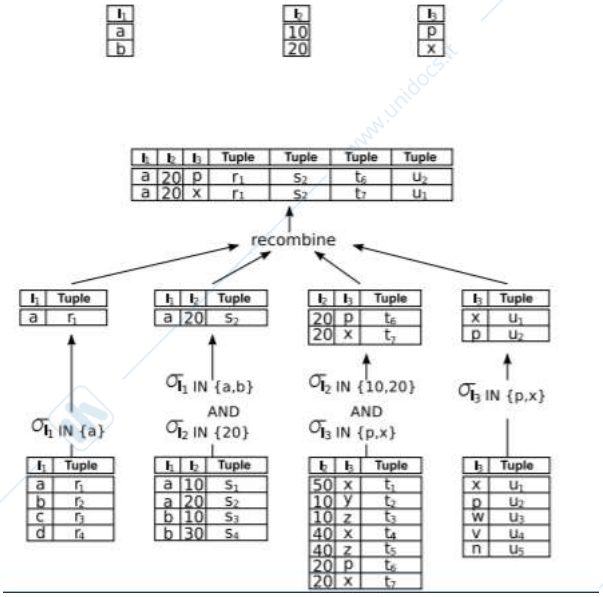


Figura 78: Esecuzione delle sequenze di Join

I problemi ancora da risolvere sono i seguenti:

- Esecuzione di un join come semi-join per supportare n:m join e proteggere il profilo join
- Applicazione delle tecniche solo a una parte dei dati (oggetto di verifica)
- Considerazione di diversi livelli di fiducia
- Rimozione dei presupposti di fiducia nei server di archiviazione

2.2 Integrità della computazione e controllo degli accessi

Fino ad ora, abbiamo presupposto che il client che richiede la computazione sia autorizzato a farlo, ma non sempre è così. Ci si sta muovendo verso una combinazione tra integrità di computazione e controllo degli accessi, al fine di rafforzare la protezione. Una direzione di ricerca promettente è rappresentata dall'integrità delle query per i database in cui viene applicato il controllo degli accessi, però, pochissime proposte affrontano questo problema e distinguono tra dati inaccessibili e dati inesistenti. Il problema è che questa distinzione comporta essa stessa la perdita di informazioni sensibili.

3 Query distribuite

Lo scenario di riferimento è rappresentato nella Figura 79. La problematica di cui ci si occuperà in questo capitolo è cercare di capire come i provider possono scambiarsi tra loro informazioni pur rispettando la privatezza e la protezione dei dati.

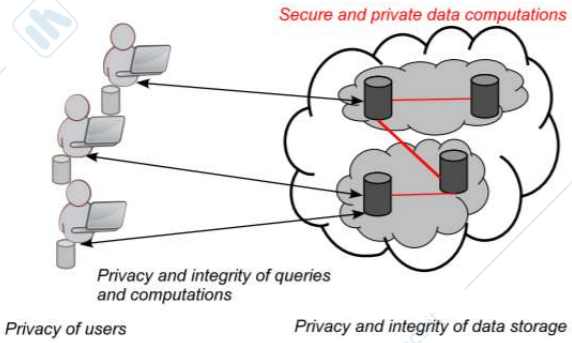


Figura 79: Calcoli privati e sicuri

L'accesso ai dati e l'esecuzione delle query sono più complessi negli scenari distribuiti, perchè possono presentarsi diverse problematiche:

- I dati possono essere archiviati al di fuori del controllo del proprietario dei dati
- Le esecuzioni di applicazioni/query possono comportare l'accesso ai dati sotto il controllo di diverse parti
- I dati possono spostarsi in posizioni diverse

Esistono diverse tecniche di specifica e applicazione dei vincoli di condivisione dei dati per la regolazione dell'esecuzione delle query in scenari multi-autorità distribuiti, al fine di garantire privatezza e protezione dei dati e della computazione. Gli approcci a supporto di queste problematiche, ossia utili per regolamentare l'accesso a diversi provider che devono collaborare tra loro sono:

- Sovereign Join: calcola un join in modo che non venga rivelato nulla oltre il risultato della query. Le tabelle e il contenuto su cui eseguo la computazione non sono visibili. Un'operazione di Sovereign Join calcola un'unione in modo che non venga rivelato nulla oltre il risultato della query. Gli scenari a cui si adatta questo approccio sono:
  - i proprietari delle due relazioni non si fidano l'uno dell'altro, nessuno di loro dovrebbe visualizzare la relazione dell'altra o il risultato della query
  - la parte che pone la domanda è diversa dai due proprietari
  - il server che esegue la query non è considerato attendibile per la riservatezza
  - il server è dotato di un coprocessore sicuro antimanomissione che è autorizzato a visualizzare il contenuto delle due relazioni. Il coprocessore è una parte fidata, protetta a livello fisico. Si tratta di un modulo hardware che ha un sistema antimanomissione. É dotato di diversi sensori che permettono di rilevare i tentativi di intrusione.

  Alcuni esempi in cui è utile utilizzare questa tecnica sono:

  - verificare che i passeggeri delle compagnie aeree non siano nella lista nera di un'agenzia federale. Non rivela chi sono i terroristi, a meno che non siano sul volo, e, contemporaneamente, non rivela chi sono i passeggeri del volo, a patto che non siano terroristi.

– controllare le sequenze di DNA rispetto ai dati dei pazienti

In entrambi i casi, si tratta di liste estremamente sensibili e devono essere rivelati solo se soddisfano determinati criteri. L'esecuzione della query consta dei diversi passaggi:

– il client invia l'operazione di join al server
– il server ha la versione crittografata delle relazioni originali, su cui deve essere effettuata l'operazione di join.
– il server invia la query e le relazioni crittografate al coprocessore.
– il coprocessore decodifica le relazioni ed esegue il join. Il coprocessore può accedere ai dati in chiaro, in quanto parte fidata.
– Il coprocessore crittografa il risultato del join con una chiave condivisa con il client e restituisce i risultati crittografati al server.
– Il server invia il risultato crittografato al client.

Non viene specificato nulla su come avviene l'operazione di join. Infatti, un'operazione di join può essere effettuata in diversi modi. Il modo più banale è: date due tabelle (A, B) e presa una tupla dalla tabella A, si cerca una corrispondenza, mediante un ciclo for, nella tabella B. Questo è necessario farlo per tutte le tuple di A. Si tratta, quindi, di due for annidati. Questo può creare problemi, perchè se il coprocessore ricevesse una tupla alla volta un osservatore potrebbere rendersi conto di quali tuple soddisfano il join e quali no, quindi si tratterebbe di un leakage di informazioni sensibili. Per questo motivo, tutte le operazioni e i dati, fatta eccezione per il risultato, devono essere manipolati all'interno del coprocessore. Questa modalità di esecuzione delle query prevede elevati costi di calcolo e necessità di un componente affidabile.

Le operazioni di join devono essere eseguite con attenzione poiché il coprocessore ha risorse e memoria limitate. Qualsiasi osservazione dell'interazione tra il coprocessore e il server non deve rivelare alcuna informazione sui risultati e sugli operandi del join. Le operazioni di join devono soddisfare due proprietà:

– fixed time: il tempo per la valutazione della condizione di join e per la composizione delle tuple è lo stesso indipendentemente dal risultato fixed size: la dimensione del risultato ottenuto confrontando le tuple è la stessa indipendentemente dal risultato

- Access Pattern: specifica le limitazioni su come accedere alle fonti di informazione, quindi stabilisce quali sono i vincoli per poter accedere. In questo approccio non è previsto nessun componente fidato. Il concetto che sta alla base dell'access pattern è che possiedo diverse relazioni su cui posso svolgere qualsiasi tipo di interrogazione e il vincolo che posseggo è che le informazioni nelle tabelle non possono essere acceduti da tutti. Quindi l'obiettivo di questo approccio è specificare le limitazioni su come accedere alle fonti di informazione. Fornisce un supporto per scenari in cui è necessario fornire valori (informazioni) per uno degli attributi di una relazione per ottenere dati. Ad esempio: in un'applicazione basata sul Web è necessario compilare alcuni campi per il recupero dei dati. Ogni relazione/vista è associata a un modello di accesso, in cui ogni attributo della relazione/vista ha un valore i (input) o o (output). Bisogna fornire un valore in corrispondenza dell'attributo definito come input, per poter accedere. É possibile accedere alle relazioni solo in base ai modelli di accesso corrispondenti. La Figura 80 riporta un esempio di richiesta di query su tre tabelle, che utilizzano l'access pattern. Le lettere i e o, che seguono il nome sono proprio l'access pattern e stabiliscono se l'attributo è di tipo input o output.

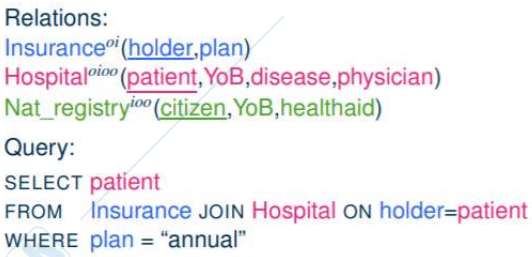


Figura 80: Esempio di access pattern

Non è possibile rispondere alla query in modo tradizionale, poiché l'attributo di input YoB della tabella Hospital non è limitato da un valore. Quindi, bisogna accedere alla tabella Insurance, per recuperare i titolari (holder). Per accedere alla tabella Insurance devo fornire un valore per l'attributo plan, che, in questo caso è specificato nella query alla voce WHERE. Per quanto riguarda la tabella Hospital devo fornire un valore per l'attributo YoB, che però non possiedo nella query. Quindi, devo accedere a Nat_registry, mediante l'attributo citizen (che coincide con holder), per recuperare l'anno di nascita (YoB). A questo punto posso fornire come input alla tabella Hospital l'anno di nascite e trovare il risultato della query. L'apprroccio Access Pattern fornisce una valutazione e ottimizzazione delle query in presenza di schemi di accesso. Le query sono in genere rappresentate in termini di registro dati. Quindi, il metodo di Access Pattern ha un doppio obiettivo:

- identificare i tipi di query supportate da determinati schemi di accesso
- determinare un piano di query che corrisponda ai modelli di accesso delle relazioni coinvolte, riducendo al minimo alcuni parametri di costo

Considera solo due attori (il proprietario dei dati e un singolo utente richiedente). Non affrontare il problema del controllo degli accessi, ma potrebbe essere sfruttato per questo. Quindi, anche in questo caso, si parte dal presupposto che chi interroga i dati è autorizzato a farlo.

- Controllo dell'accesso basato sulla viste: fornisce un controllo dell'accesso dipendente dal contenuto a grana fine nei database relazionali. Gli utenti possono interrogare le informazioni di base, senza fare esplicito riferimento alle eventuali viste definite all'interno del sistema. Questo approccio permette che le query vengano riscritte in modo trasparente sulla base della vista dell'utente. Esistono due approcci per farlo:
  - modificare delle query per rispondere alle query (modello Truman). Il nome di questo modello deriva dal titolo del film "The Truman Show". In questo modello, ogni utente possiede una vista personale e ristretta sui dati. Si possono accedere solo le informazioni per cui si è autorizzati. Un risultato di una query è corretto, limitatamente alla vista dell'utente. Gli utenti potrebbero avere informazioni esterne sul database. I risultati, che si ottengono dall'interrogazione del database, potrebbero non essere consistenti riguardo queste informazioni esterne, perché magari escluse dalla vista dell'utente. Infatti, le query vengono riscritte in maniera trasparente, quindi l'utente è ignaro della riscrittura, per soddisfare la vista dell'utente specifico. Quindi, se un utente interrogasse il database per sapere il voto medio degli esami della triennale, sulla base dei voti di tutti gli studenti del suo anno, ma avesse una vista che gli permette di accedere solo ai voti che lui stesso ha conseguito, l'utente potrebbe pensare, magari erroneamente, che la media generale è pari alla sua media personale. Per risolvere questo problema si utilizza il modello non Truman.

  - una query è valida se è possibile rispondere utilizzando le informazioni nelle viste di autorizzazione disponibili per l'utente richiedente (modello non Truman).

La Figura 81 mostra un esempio di questo approccio: si prendono in considerazione due tabelle (Treatment e Dottore) e un vincolo di integrità referenziale, per cui ogni trattamente è sempre superivisionato da un dottore. La vista dell'utente è TreatDoct. Quindi il risultato della query sarà sulla base della vista stabilita. Il risultato della query è il tipo e il costo di tutte le tuple contenute nella tabella Treatment. Dal momento che la proiezione della vista TreatDoct su T.type e T.cost è equivalente alla query. Le autorizzazioni sono

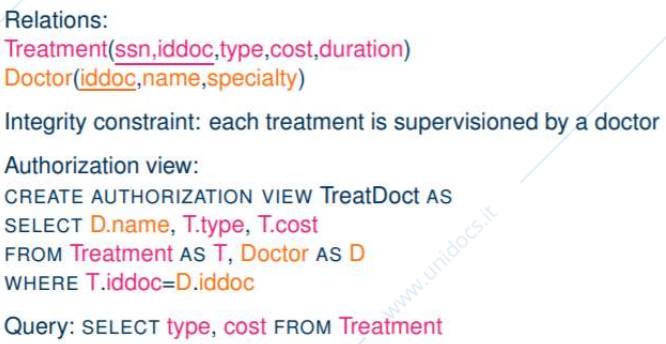


Figura 81: Esempio di autorizzazioni basate sulle viste

espresse come viste parametrizzate e viste del modello di accesso (Access Pattern View). Le viste parametrizzate sono viste relazionali, ma con parametri. Possono variare da utente a utente. Ad esempio, un paziente può vedere solo i suoi dati di ricovero (non quelli di altri pazienti). Quindi il parametro è l'ID paziente, che deve coincidere con l'ID dell'utente che richiede la query. Dato un particolare accesso, i parametri vengono fissati con i valori corrispondenti per quel determinato accesso. Anche nel caso delle Access Pattern View ci sono dei parametri, ma possono essere vincolati ad un qualsiasi valore (un parametro qualsiasi, diverso ad ogni accesso). Un esempio in questo senso è un medico che può visualizzare i dati di qualsiasi paziente se fornisce l'ID paziente (non può recuperare i dati di tutti i pazienti insieme). Se esiste una query scritta solo utilizzando le viste autorizzate istanziate dell'utente richiedente equivalente alla query originale, la query viene accettata. Se la query viola le autorizzazioni di un utente che la richiede, viene rigettata. La composizione delle viste viene utilizzato per verificare se una query è valida. I vantaggi che nascono dall'utilizzo di questa tipologia di approccio sono:

  - • Supporto delle query in modo trasparente per le autorizzazioni
  - Supporto delle autorizzazioni e dei modelli di accesso dipendenti dal contenuto
  - Analisi dell'integrazione con gli ottimizzatori DBMS
  - Si concentra su query istanziate (in cui gli attributi sono associati a valori specifici)
  - Si concentra su aspetti di applicazione di basso livello
  - Si affida all'euristica applicata da Query Optimizer e non fornisce algoritmi di composizione e garanzie di sicurezza.

- Reti di coalizione (Coalition network): diverse parti hanno lo scopo di condividere i propri dati per l'efficienza nella valutazione delle query proteggendo i dati. Le autorizzazione pairwise regolamentano il flusso di informazioni tra providere nelle coalition network. Le reti di coalizione sono sistemi distribuiti in cui diversi provider desiderano collaborare e condividere i propri dati per raggiungere un obiettivo comune. Ogni provider P ha:

  - una o più relazioni (proprietario delle relazioni)

  - uno o più server (buddies) sia per il calcolo che per l'archiviazione dei dati. I server buddies sono server che appartengono allo stesso provider.

- Il controllo dell'accesso che regola gli accessi ai dati deve consentire ai proprietari dei dati di:
  - autorizzare parti diverse per diverse porzioni di set di dati
  - mantenere il controllo completo e autonomo su chi può accedere ai set di dati
  - definire le restrizioni del controllo di accesso a livello di tupla, quindi avere un controllo dell'accesso ai dati a granularità fine.

La Figura 82 mostra un esempio di un grafico che modella una rete di coalizione. I nodi rappresentano i singoli server, mentre gli archi le connessioni tra server, a cui viene associato un costo di trasmissione da un'unità di dati tra i server.

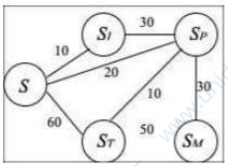


Figura 82: Esempio di un grafico che modella una rete di coalizione

Un autorizzazione a coppie (pairwise authorization), dati due provider P_i e P_j e una relazione r_i posseduta da P_i: $P_i \overset{r_x = \sigma(r_i)}{\rightarrow} P_j$ afferma che P_j (e tutti i server appartenenti a P_j) possono accedere a un sottoinsieme delle tuple in r_i secondo $\sigma(r_i)$. L'obiettivo, per eseguire una query q in una coalition network, è determinare un Safe Query Plan per una query q. Un safe query plan è un piano di query che comporta solo scambi di dati autorizzati e riduce al minimo i trasferimenti di dati in base ai costi sugli archi. Le operazioni unarie (selezione e proiezione) non richiedono alcun trasferimento di dati, quindi non richiedono alcun costo, perchè non c'è pericolo di violare le autorizzazioni. Le operazioni di join possono richiedere la cooperazione tra diversi server:

- master: server incaricato del calcolo del join
- slave: server che collabora con il master

Consideriamo la seguente operazione di join $r_x \bowtie_{ax=ay} r_y$, richiesta dal server S_Q. Esistono diverse tecniche per eseguire un join:

- Broker-join: sia S_x che S_y inviano le loro relazioni a S_Q, che calcola il risultato del join (S_x, S_y, S_Q possono essere buddies o meno).
- Peer-join: il server S_y invia la relazione r_y a S_x, che calcola la join e invia il risultato a S_Q (funziona bene quando S_x e S_Q sono buddies e S_y no)
- Semi-join: i server S_x e S_y interagiscono per calcolare il risultato del join (funziona bene quando S_x e S_y sono buddies):
  - Il master S_x invia la proiezione sull'attributo join di r_x a S_y
  - S_y calcola il join tra la relazione ricevuta da S_x e r_y e restituisce il risultato
  - S_x calcola il join tra la relazione ricevuta e r_x ottenendo il risultato del join
  - S_x invia il risultato a S_Q

– Split-join: Sia r_{x1} l'insieme di tuple in r_x a cui il server S_y può accedere, e r_{y1} sia l'insieme di tuple in r_y a cui il server S_x può accedere $(r_x \bowtie_{ax=ay} r_{y1}) \cup (r_{x1} \bowtie_{ax=ay} r_{y2}) \cup (r_{x2} \bowtie_{ax=ay} r_{y2})$ con r_{x2} l'insieme di tuple in r_x a cui S_y non può accedere, e r_{y2} l'insieme di tuple in r_y a cui S_x non può accedere. Viene eseguito nel seguente modo:
  * peer-join $(r_x \bowtie ax = ayr_{y1})$ con S_x che funge da master
  * peer-join $(r_{x1} \bowtie_{ax=ay} r_{y2})$ con S_y che funge da master
  * broker-join $(r_{x2} \bowtie_{ax=ay} r_{y2})$ con S_Q che funge da broker e calcola l'unione dei tre risultati parziali

  Questo metodo può essere applicato a prescindere che S_x, S_y e S_Q siano buddies o meno.

La Figura 83 mostra il query plan (grafico a destra) con segnato il server che è incaricato di svolgere l'operazione. La scelta della composizione del query plan dipende dai costi segnati sugli archi e sulla base delle autorizzazione, elencate sulla destra. Il query plan è strutturato per risolvere la query da eseguire, riportata in basso nella Figura.

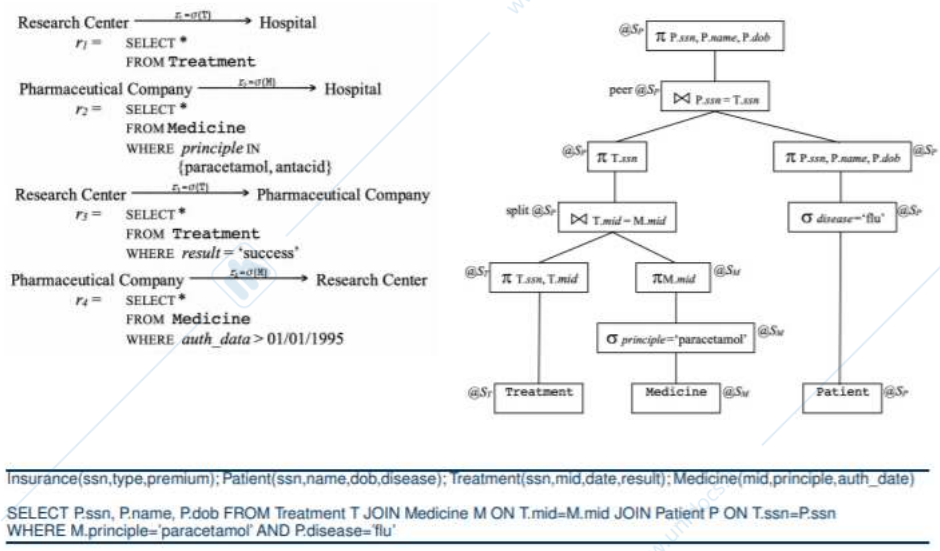


Figura 83: Esempio di Pairwise Authorization

- Restrizioni basate sull'utente: gli utenti possono definire restrizioni sulla privacy nella valutazione delle query per proteggere gli obiettivi delle loro richieste ai provider. I client che accedono ai dati potrebbero essere interessati a specificare i requisiti di riservatezza. Un richiedente potrebbe, ad esempio, voler mantenere segreto il fatto che sta unendo i dati per trovare eventualmente correlazioni nascoste (ad esempio, correlazione tra l'effetto di un nuovo farmaco e il diabete). Il richiedente può specificare le condizioni con quelle parti della query che devono essere gestite in un modo specifico. Esistono due tipi di requisiti che possono essere stabilti mediante due clausole:
  – Condizione mandatoria (REQUIRRING condition) HOLDS OVER <operation, parameters, master>
  – Condizione di preferenza (PREFERRING condition) HOLDS ON <operation, parameters, master>

  dove operation rappresenta un nodo del query plan. <operazione / parametro / master> può includere una variabile libera (@) o un carattere jolly *. La clausola REQUIRRING

può includere più condizioni connesse con AND. La clausola PREFERRING può includere più condizioni che possono essere combinate utilizzando l'operatore CASCADE, il quale definisce una sorta di precedenza tra condizioni (una relazione d'ordine parziale). Data una query q con clausole REQUIRRING, PREFERRING, il query plan deve soddisfare tutte le condizioni obbligatorie contenute nella clausola REQUIRRING e massimizzare le preferenze per le condizioni nella clausola PREFERRING.

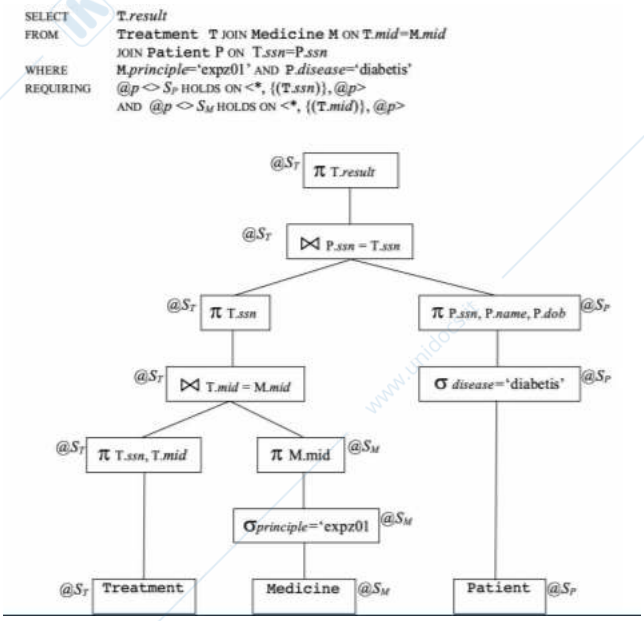


Figura 84: Esempio di preferenze nell'ottimizzazione delle query

La Figura 84 rappresenta un query plan con clausole specificate dall'utente. L'operazione di join $\bowtie_{T.mid=M.mid}$ può essere valutata unicamente da S_T, per via della clausola REQUIRRING che non permette a S_M di trattare $T.mid$. Per quanto riguarda, invece, l'operazione di join $\bowtie_{P.ssn=T.ssn}$ può essere valutata solo da S_T, per via della clausola REQUIRRING che non permette a S_P di trattare $T.ssn$.

- Valutazione distribuita delle query in base ai requisiti di protezione

3.1 Valutazione distribuita delle query in base ai requisiti di protezione

Lo scenario di riferimento è un sistema di database distribuiti. É da assumere che si tratti di un database relazionale, non si tratta di una limitazione, poiché i database relazionali sono il nucleo di qualsiasi servizio Web. Le relazioni sono distribuite su server diversi. L'esecuzione di query potrebbe richiedere la cooperazione e la condivisione delle informazioni tra server diversi. Ogni server è responsabile della definizione della politica di accesso sulle proprie risorse e della gestione dei propri dati.

Il problema consiste nel regolare le opinioni e la condivisione delle informazioni tra parti diverse, in modo che sia possibile:

- supportare la collaborazione tra le parti nell'esecuzione distribuita di query su dati soggetti a rilascio selettivo
- definire viste autorizzate basate sul contenuto informativo di una relazione
- assegnare operazioni all'interno della query a parti diverse in modo sicuro rispetto alle informazioni che possono essere visualizzate dalle parti

Supponiamo di avere a disposizione $\Re$, un insieme di relazioni $R_i(A_1, ..., A_n)$ con $K_i \subseteq \{A_{1_i}, ..., A_{n_i}\}$ chiave primaria di R_i. Esiste una dipendenza funzionale tra la chiave primaria e il resto degli attributi. Si suppone che le relazioni siano acicliche e looseless. Acicliche significa che se si esegue un'operazione di join su due relazione, si tratta di un join univoco. Con il termine looseless si intende che il join tra informazioni produce solo informazioni complete. I rappresenta l'insieme di vincoli di integrità referenziale $< F_j, K_i >$. La chiave esterna F_j può assumere solo valori in $R_i[K_i]$. Indica che l'insieme degli attributi F_j (chiave esterna) possono assumere solo i valori di K_i nella relazione R_i. J rappresenta un insieme di join $J =< A_l, A_r >$, che combinano tuple in relazioni diverse basate su condizioni di uguaglianza (si trattano unicamente i join nautrali). A_l e A_r sono attributi di relazioni diverse, ma che rappresentano la stessa entità. Il joinpath $(R_1, ..., R_n)$ rappresenta la sequenza $J_1 =< A_{l_1}, A_{r_1} >, ..., J_{n-1} =< A_{l_{n-1}}, A_{r_{n-1}} >$ dei join che collegano le relazioni. Il joinpath è definito su una sequenza di relazioni $R_1, ..., R_n$, ossia non è altro una sequenza di $(n-1)$ join. Ci si concentra su query SQL del tipo: SELECT A FROM Joined_Relations WHERE C, che rappresentano una proiezione di una selezione in base alla formula booleana C sul risultato delle operazioni di join: $\pi_A(\sigma_C(R_1 \bowtie_{J_1} ... \bowtie_{J_{n-1}} R_n))$, dove:

- A rappresenta l'insieme degli attributi
- C rappresenta la condizione di selezione
- Joined_Relations rappresenta le relazione di join tra le varie tabelle.

Il query tree plan T costituisce l'albero che rappresenta l'esecuzione della query.

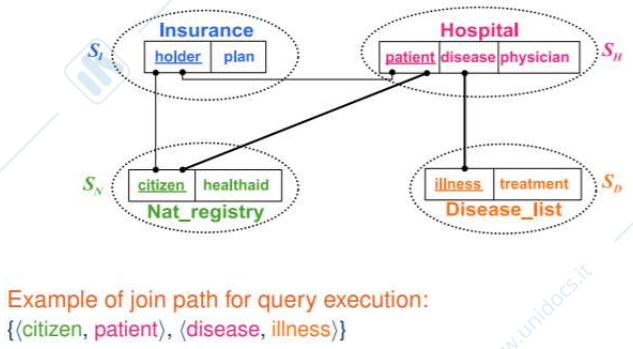


Figura 85: Esempio di joinpath e di relazioni distribuite

La Figura 85 raffigura un esempio di relazioni distribuite. Patient e Holder hanno nomi diversi, ma rappresentano la stessa entità. S_H, S_N, S_I, S_D rappresentano i server su cui si trovano le varie relazioni. I valori sottolineati rappresentano le chiavi primarie delle relazioni. Questa figura sarà il modello di riferimento per i prossimi esempi.

La Figura 86 riporta un esempi di query tree plan, che raffigura la query riportata in Figura. Si utilizza un color coding per capire a che tabella si fa riferimento.

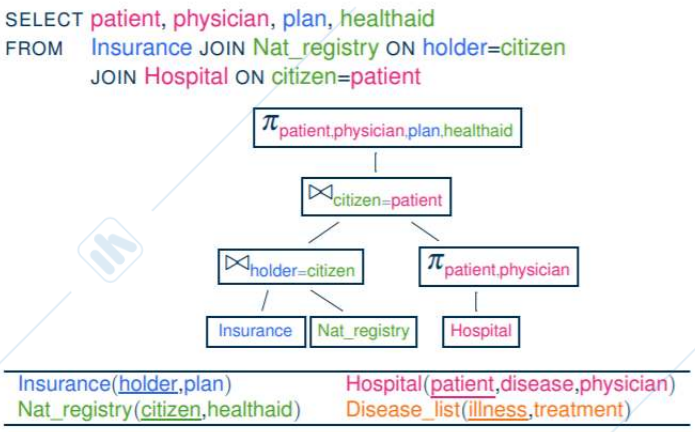


Figura 86: Esempio di joinpath e di relazioni distribuite

Questo metodo si basa su regole di autorizzazione (Permessi), che permettono di regolare l'accesso a risorse. I permessi si presentano con la seguente forma: *[Attributes, Join Path]→ Subject*, il quale autorizza il rilascio a Subject del set Attributes di attributi risultanti dal Join Path. La Figura 87 mostra due esempi di permessi: nel primo esempio il server S_N può accedere per intero alla tabella Insurance. Nel secondo esempio, S_I può vedere quali detentori di un assicurazione sono ospedalizzati e che medico li sta seguendo.

Figura 87: Esempio di permessi

Il campo Join Path può comportare relazioni che non hanno attributi in Attributes. Quindi, i Join Path permettono di modellare:

- vincoli di connettività: per costruire l'associazione tra attributi [(holder,treatment), (<I.holder,H.patient> <H.disease,D.illness>)] → S_I
- restrizioni basate sull'istanza: per limitare i valori degli attributi da rilasciare: [(holder,plan),(<I.holder,H.patient>)] → S_I

  Il server S_I, oltre ad accedere alla tabella di sua proprietà, può collegarla ad un'altra, ossia può sapere quali dei suoi clienti sono ospedalizzati.

Un Join Path in un permesso permette che il rilascio di un minor numero di tuple non implichi il rilascio di meno informazioni. Inoltre, risulta inefficace quando il join path è raggiungibile tramite vincoli di integrità referenziale.

Un profilo di relazione cattura il contenuto informativo di una relazione di base o derivata (ovvero calcolata da una query). Il profilo di relazione di R è una tripla del tipo $[R^\pi, R^\bowtie, R^\sigma]$, dove:

- R^π rappresenta lo schema della relazione R, ossia gli attributi che fanno parte della relazione
- $R^\bowtie$ rappresenta il join path usato nella definizione di R. Può essere anche vuoto.
- R^σ rappresenta l'insieme degli attributi coinvolti nelle condizioni di selezione su R.

La Figura 88 riporta un esempio di profilo di relazione, in cui vengono riportate le operazioni di proiezione, join e selezione. L'esecuzione delle operazioni ha effetto sul profilo di relazione. Il profilo di relazione dipende da come si è ottenuta la relazione R, cioè l'operazione da cui viene derivato R. L'operazione di join raccoglie i profili informativi tra le due relazioni.

	Profile		
Operation	R^{π}	$R^{\bowtie}$	R^{σ}
$R := \pi_X(R_l)$	X	$R_l^{\bowtie}$	R_l^{σ}
$R := \sigma_X(R_l)$	R_l^{π}	$R_l^{\bowtie}$	$R_l^{\sigma} \cup X$
$R := R_l \bowtie_J R_r$	$R_l^{\pi} \cup R_r^{\pi}$	$R_l^{\bowtie} \cup R_r^{\bowtie} \cup J$	$R_l^{\sigma} \cup R_r^{\sigma}$

Figura 88: Profilo di relazione

Prendiamo come esempio la seguente query:

SELECT Illness

FROM Disease_list **JOIN** Hospital **ON** Illness = Disease

WHERE Treatment = "antihistamine"

La Figura 89 riporta il profilo di relazione della query, in cui: il profilo di proiezione è sulla tabella illness; il join viene effettuato attraverso una condizione di equivalenza tra l'attributo Illness della relazione Disease e l'attributo Disease della relazione Hospital; infine, il profilo di selezione rappresenta l'attributo treatment.

Profile: $[R^{\pi}, R^{\bowtie}, R^{\sigma}]$
[(illness),(⟨D.illness,H.disease⟩),(treatment)]

Figura 89: Profilo di relazione

Il soggetto S è autorizzato a visualizzare una relazione R se e solo se $\exists$ [Attributes, Join Path] $\rightarrow S : R^{\pi} \cup R^{\sigma} \subseteq$ Attributes e $R^{\bowtie}$ = Join Path. La Figura 90 riporta un esempio di vista mediante autorizzazioni, in cui S_D richiede di accedere ad R e il profilo di relazione è quello presentato nella Figura 89. Se S_D possedesse la prima autorizzazione presentata la query sarebbe autorizzata, perchè gli permette di vedere illness e treatment ottenuti tramite il join tra Disease e Hospital. Se, invece, possedesse unicamente la seconda autorizzazione non potrebbe accedere in lettura alla query, perchè il join path dell'autorizzazione deve coincidere con il join path del profilo di relazione.

S_D requires R :
SELECT illness
FROM Disease_list JOIN Hospital ON illness=disease
WHERE treatment='antihistamine'

Relation profile: [(illness),(⟨D.illness,H.disease⟩),(treatment)]

Authorization
[(illness,treatment),(⟨D.illness,H.disease⟩)] $\rightarrow S_D$
authorizes the query

Authorization [(illness,treatment),_] $\rightarrow S_D$
does not authorize the query

Figura 90: Esempio di vista autorizzata

Ogni soggetto S_i è autorizzato a visualizzare la relazione che memorizza, quindi può visualizzare l'intera relazione se è owner di essa. Le operazioni unarie che possono essere eseguite dal soggetto S che memorizza la relazione sono:

- proiezione: $\pi_X(R)$
- selezione: $\sigma_X(R)$

Un'operazione di join può essere eseguita solo se comporta il rilascio di viste autorizzate, quindi se $R_l \bowtie_{J_{lr}} R_r$ può essere valutato come join normale o come semi-join, ossia S_l e S_r possono svolgere due ruoli: master calcola il join; lo slave collabora con il master per il calcolo. La Figura 91 raffigura un esempio di regular join. Nel caso del modello di regular join la figura dello slave non è prevista. S_H invia l'intera relazione a S_I che effettua l'operazione di join, a patto che sia autorizzato a farlo, tramite autorizzazioni esplicite.

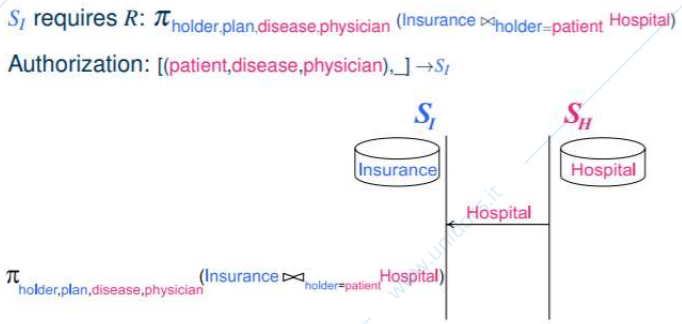


Figura 91: Esempio di Regular Join

Nella Figura 92 è possibile notare come né il server S_I né il server S_H possono eseguire un regular join, dal momento che le autorizzazione non glielo consentono. Quindi S_I esegue una proiezione dei valori della sua relazione da passare a S_H, in modo che sia autorizzato per farlo. Riducendo il numero di attributi che vengono passati, vengono soddisfatte le autorizzazioni. A questo punto S_H effettua l'operazione di join tra la sua tabella e la proiezione della tabella S_I e passa il risultato a S_I. S_I effettua una seconda operzione di join tra il risultato passato da S_H e la relazione che possiede.

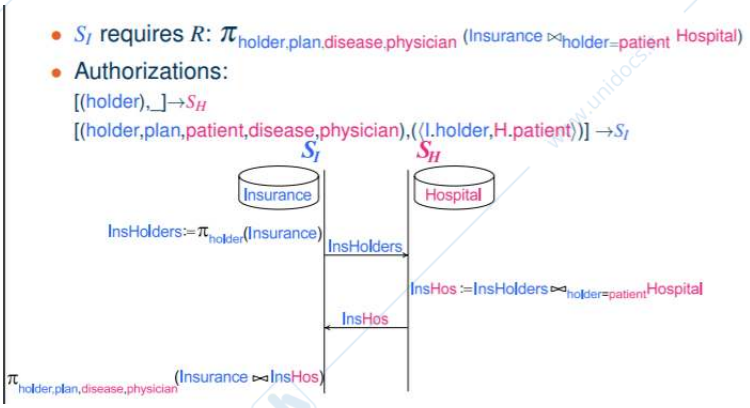


Figura 92: Esempio di Semi-Join

Un feasible query plan (piano di query fattibile) ha come obiettivo, dato un query tree plan, quello di determinare per ogni operazione un soggetto (coppia di soggetti) responsabile dell'esecuzione in modo tale che tutte le viste siano autorizzate. Per fare questo utilizziamo una funzione di assegnazione dell'esecutore λ_T, ossia una funzione che assegna ad ogni operazione il server su cui viene eseguita:

- nodo foglia n: $\lambda_T(n) = [S, NULL]$
- $n \in \{\pi, \sigma\} : \lambda_T(n) = [S, NULL]$

- $n \in \{\bowtie\} : \lambda_T(n) = [master, slave]$, dove:
  - $master \in \{S_l, S_r\}$
  - $slave \in \{S_l, S_r, NULL\}$, in caso di regular join il server slave non è previsto.
  - $master \neq slave$

λ_T è una funzione di assegnamento sicura se e solo se $\forall n \in N, \lambda_T(n)$:

- n è una foglia
- n è π o σ
- n è $\bowtie$ e tutte le viste implicite nell'incarico sono autorizzate

La Figura 93 mostra un esempio di assegnamento dell'esecutore, in cui vene mostrata la query da eseguire e avviene l'assegnamento dei server esecutori per ogni nodo. Il join tra Insurance e Nat_Registry è un regular join, mentre il $\bowtie_{citizen=patient}$ è un semi-join.

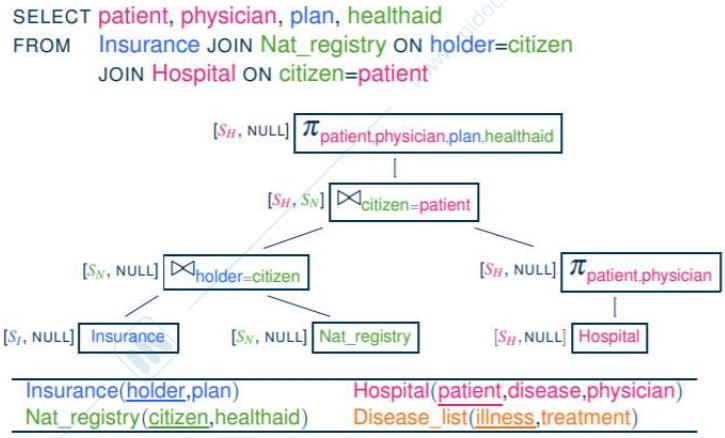


Figura 93: Esempio di assegnamento dell'esecutore

É possibile utilizzare un algoritmo per assegnare gli esecutori ai vari nodi, sviluppato in due fasi:

1. Find_candidates: visita post-ordine di T (query tree plan) per spingere verso l'alto potenziali candidati (inclusi terzi)
2. Assign_ex: visita pre-ordine di T per spingere verso il basso gli esecutori selezionati:
   - preferisce il semi-join al full-join, perchè chi agisce come slave deve solo consegnare le tuple che joinano con la proiezione e non inviare l'intera relazione
   - preferisce il soggetto coinvolto in un numero maggiore di operazioni di join

Cosa significa che un soggetto (server / utente) è autorizzato a vedere il risultato di una query? Esistono due approcci per determinare ciò:

- Sintattico. Il soggetto S è autorizzato a visualizzare una relazione R se e solo se $\exists$ $[Attributes, JoinPath] \rightarrow S : R^{\pi} \cup R^{\sigma} \subseteq Attributes$ e $R^{\bowtie} = JoinPath$. É l'approccio più semplice, ma è limitato e gravoso per quanto riguarda la specifica dell'autorizzazione, perché bisogna specificare tutte le autorizzazione per un utente.
- Semantico. Il soggetto S è autorizzato a visualizzare una relazione R se e solo se S dispone delle autorizzazioni per visualizzare il contenuto informativo portato dalla relazione. Una query dovrebbe essere autorizzata se l'insieme di autorizzazioni disponibili per l'oggetto consentirebbe al soggetto di calcolare autonomamente il risultato della query.

Prendiamo come esempio la Figura 94 e vediamo come si sviluppano i due approcci:

- Approccio sintattico: la query non è autorizzata in quanto le autorizzazioni non consentono esplicitamente l'unione tra Assicurazione e Ospedale
- Approccio semantico: la query è autorizzata poiché S_I potrebbe calcolare il risultato della query.

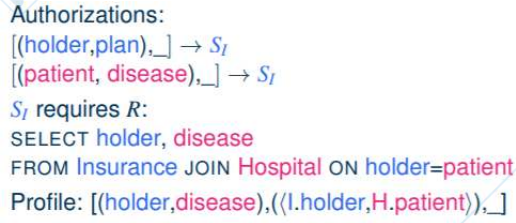


Figura 94: Confronto tra approccio semantico e sintattico

3.2 Composizione delle autorizzazioni

Vediamo ora come è possibile combinare più autorizzazioni. Una query dovrebbe essere autorizzata se l'insieme di autorizzazioni disponibili per l'oggetto consentirebbe al soggetto di calcolare in modo indipendente il risultato della query. Sorge la necessità di supportare la composizione delle autorizzazioni senza garantire il rilascio indiretto, ciò può essere causato da due fattori:

- condizioni su attributi che non vengono restituiti, catturato nella definizione del profilo. Si tratta degli attributi nella clausola di WHERE che non appaiono nella voce SELECT.
- condizioni di join che limitano le tuple restituite dalla query, acquisito in base alla colorazione dei grafici che utilizzeremo per modellare lo schema del database, le autorizzazioni e le query.

ù La tecnica di composizione delle autorizzazioni si basa su diverse assunzioni:

- Lo schema è aciclico, ossia non ci sono più modalità per unire in join due relazioni
- Gli attributi che possono essere uniti vengono visualizzati con lo stesso nome nelle diverse relazioni (join naturali)
- Limitare l'analisi a un argomento, ossia il soggetto è fissato e si va a considerare tutti i permessi per quel determinato soggetto. Un permesso $[Attr, JoinPath] \rightarrow Subject$ può essere visto come $[Attr, Rel]$, dove Rel sono le relazioni coinvolte in Join Path, che essendo univo non è necessario che venga specificato.
- Un profilo di relazione $[R^{\pi}, R^{\bowtie}, R^{\sigma}]$ può essere visualizzato come $[Attr, Rel]$, dove:
  - $Attr = R^{\pi} \cup R^{\sigma}$
  - Rel sono le relazioni coinvolte nel join path $R^{\bowtie}$

Lo schema graph è una rappresentazione grafica dei legami tra relazioni. Il diagramma dello schema (schema graph) di un insieme $\Re$ di relazioni è un grafo misto[1] $G(N, E)$, dove:

- $N = \{R_i.* | R_i \in \Re\}$ corrisponde all'insieme degli attributi
- $E = J \cup I \cup \{(R_i.K, R_i.A) | R_i \in \Re \vee A \notin K$ rappresenta gli insieme degli archi, dove:

[1] Il grafo misto è un grafo composto da archi orientati e non orientati

- join naturale (archi non orientati)
- vincoli di integrità referenziale (archi orientati)
- dipendenze funzionali (archi orientati)

La Figura 95 rappresenta un esempio di schema graph, in cui ci sono tanti nodi quanti sono gli attributi delle quattro relazioni. Gli archi orientati rappresentano le dipendenze funzionali tra la chiave di una relazione e il resto degli attributi oppure vincoli di integrità referenziale (I) tra relazioni. $T.ssn$ e $T.iddoc$ rappresentano un multinodo, ossia una chiave composta da più attributi. L'insieme J rappresenta l'insieme dei join naturali applicabili all'esempio, i quali vengono rappresentati da archi non orientati.

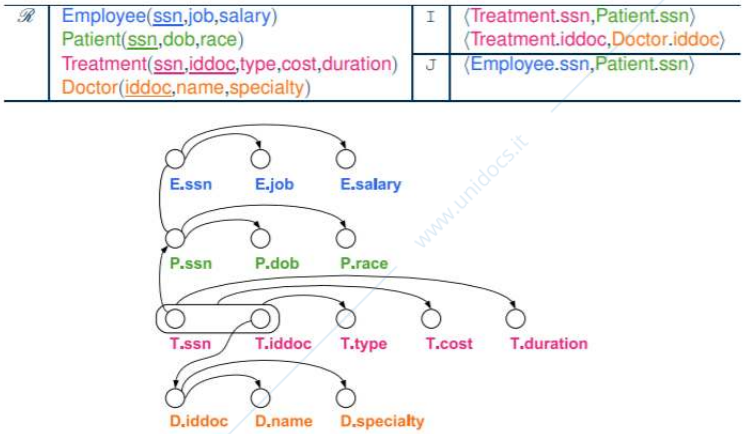


Figura 95: Esempio di schema graph

Le autorizzazioni e query $[Attr, Rel]$ sono delle viste sull'insieme di relazioni $\Re$. L'insieme Rel è esteso attraverso vincoli di integrità referenziale. La rel. Di chiusura * rappresenta le relazioni ottenute chiudendo Rel rispetto ai vincoli di integrità referenziale (join inefficaci, ossia che non aggiungono informazioni rispetto a quelle fornite dai vincoli di integrità referenziale). La Figura 96 rappresenta l'operazione di chiusura della relazione Treatment rispetto ai vincoli di integrità referenziale (Insieme I della Figura 95).

Rel = {Treatment}
Rel^* = {Treatment,Patient,Doctor}

Figura 96: Esempio di relazione di chiusura

$G_V(N, E, \lambda_V)$ rappresenta un view graph di $V = [Attr, Rel]$, ottenuto colorando (λ_V) lo schema graph. Vengono colorati di:

- nero: informazioni contenute nella vista:
  - attributi in $Attr$
  - archi in joinpath (Rel^*) o passando dalla chiave di una relazione in Rel^* agli attributi in $Attr$ o joinpath (Rel^*), quindi si tratta del jojn path definito sulla chiusura di Rel.
- bianco: attributi non neri delle relazioni in Rel^* e archi che li collegano alla chiave primaria
- colorati di chiaro: qualsiasi altro attributo o arco, ossia attributi e archi che non hanno impatto sul contenuto informativo della vista.

Consideriamo la vista riportata in Figura 97. Nell'esempio di sinistra, viene effettuata la chiusura rispetto ai vincoli di integrità referenziale della relazione Patient. La chiusura rappresenta la tabella Patient, perché non ci sono vincoli di integrità referenziali dichiarati sulla tabella Patient. Nell'esempio di destra, invece, viene effettuata la chiusura in base ai vincoli di integrità referenziale della tabella Treatment, quindi vengono colorati di nero gli attributi della tabella Treatment che sono parte della vista e Patient.ssn che rappresenta il vincolo di integrità referenziale. Treatment.iddoc viene colorato di bianco perché non fa parte della vista. Viene colorato di nero l'arco orientato rappresentante il vincolo di integrità referenziale tra Treatment e Doctor, ma i nodi uniti rimangono bianchi, perché non fanno parte della vista.

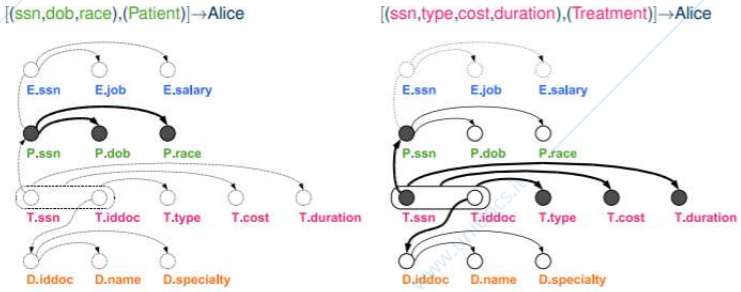


Figura 97: Grafico della vista

La Figura 98 mostra un altro esempio di view graph, che è la rappresentazione della query riportata. Gli attributi colorati di nero sono gli attributi restituiti dalla query e l'attributo su cui si va a verificare la condizione WHERE. Inoltre, coloro di nero i nodi degli attributi da selezionare e gli archi di dipendenza funzionale e i vincoli di integrità referenziale.

```
SELECT E.ssn,salary
FROM   Employee AS E JOIN Patient AS P ON E.ssn=P.ssn
       JOIN Treatment AS T ON T.ssn=P.ssn
WHERE  cost> 250
```

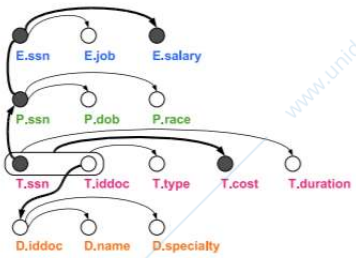


Figura 98: Esempio di view graph

L'autorizzazione p autorizza la query q se le informazioni dirette o indirette rilasciate da q sono un sottoinsieme delle informazioni autorizzate da p. $p = [Attr_p, Rel_p]$ è applicabile a $q = [Attr_q, Rel_q]$ se e solo se $Rel_p^* \subseteq Rel_q^*$. Vale a dire che l'autorizzazione si riferisce all'insieme completo di tuple richiesto dalla query (non contiene join aggiuntivi oltre a quelli corrispondenti ai vincoli di integrità referenziale). $p = [Attr_p, Rel_p]$ autorizza $q = [Attr_q, Rel_q]$ se e solo se:

- p è applicabile a q
- G_q e G_p hanno gli stessi archi di integrità referenziale/join neri, dove G_p e G_q sono rispettivamente il grafo relativo al permesso e quello relativo all query. Quindi, il permesso deve avere almeno gli stessi archi/nodi neri della query.

- tutti i nodi neri in G_q sono neri anche in G_p

La Figura 99 riporta tre esempi di permessi applicabili e autorizzati. Il primo esempio rappresenta una query autorizzata sulla base del permesso p, il quale è applicabile a q. Nel secondo esempio il permesso non si applica a q, perché il permesso possiede un arco e un nodo nero che non sono presenti nella query. Nel terzo esempio la query non è autorizzata perché richiede un join che il permesso non possiede.

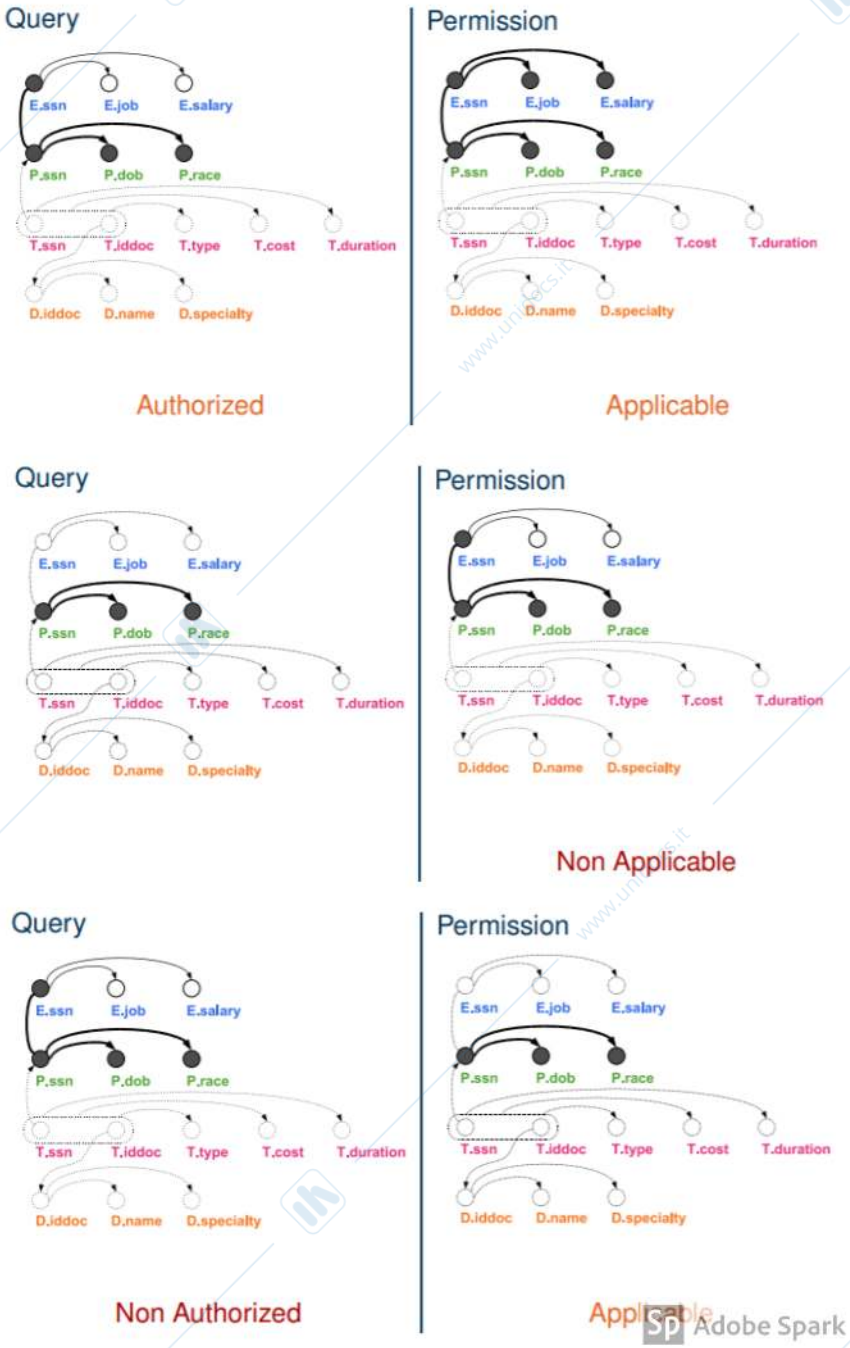


Figura 99: Autorizzazione applicabile, non autorizzante - Esempio

Una query può essere eseguita se l'oggetto dispone delle autorizzazioni per visualizzare il contenuto delle informazioni trasportato dalla query. Una query deve essere autorizzata se l'insieme di autorizzazioni disponibili per l'oggetto consentirebbe al soggetto di calcolare in modo indipendente il risultato della query. Data una query potrebbe non esserci un singolo permesso

che permette l'esecuzione, ma l'informazione che viene rilasciata (direttamente o indirettamente) può essere autorizzata da una composizione di permessi, però è necessario prestare attenzione a non attuare la divulgazione indiretta. La Figura 100 mostra due esempi di composizione di permessi. Si potrebbe pensare che effettuare una composizione sia una sorta di fusione, ma non si tratta di questo, perché, effettuando una fusione, si fornirebbero più informazioni di quelle autorizzate.

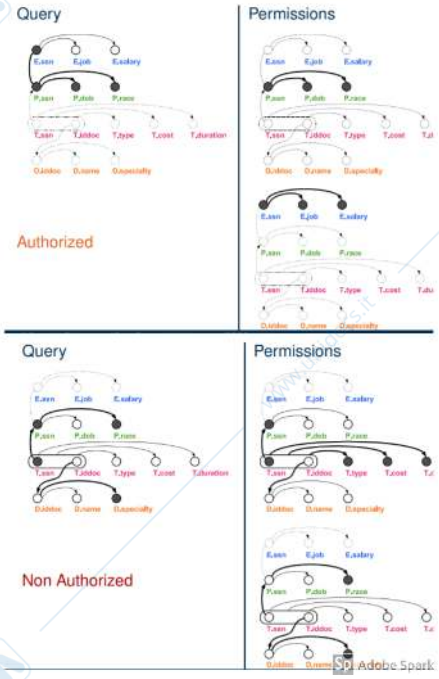


Figura 100: Esempi di composizione dei permessi

Due autorizzazioni possono essere composte in modo sicuro se la loro composizione non aggiunge informazioni: $p_i \rightarrow p_j : p_j = [Attr_j, Rel_j]$ dipende da $p_i = [Attr_i, Rel_i]$ se e solo se esiste un percorso dai nodi corrispondenti agli attributi in $Attr_i \cap Attr_j$ a tutti i nodi neri in G_{p_j}. Due permessi p_i e p_j possono essere composti in sicurezza se almeno uno dei $p_i \rightarrow p_j$ o $p_i \rightarrow p_j$ è valido. Il permesso che ottengo è un autorizzazione composta $p_i \otimes p_j = [Attr_i \cup Attr_j, Rel_i \cup Rel_j]$, in cui la colorazione grafica dell'autorizzazione composta è derivata dai colori degli operandi.

La Figura 101 mostra un esempio di safe composition, in cui l'attributo che hanno in comune i due permessi è ssn, il quale è la chiave sia di Patient che di Employee. Ciò significa che dall'attributo ssn dipendono funzionalmente tutti gli attributi di almeno una relazione.

Invece, nella Figura 102 l'attributo che hanno in comune è race, ma nessuno degli attributi rimanenti nelle due relazioni dipende funzionalmente da race, quindi la composizione non è possibile.

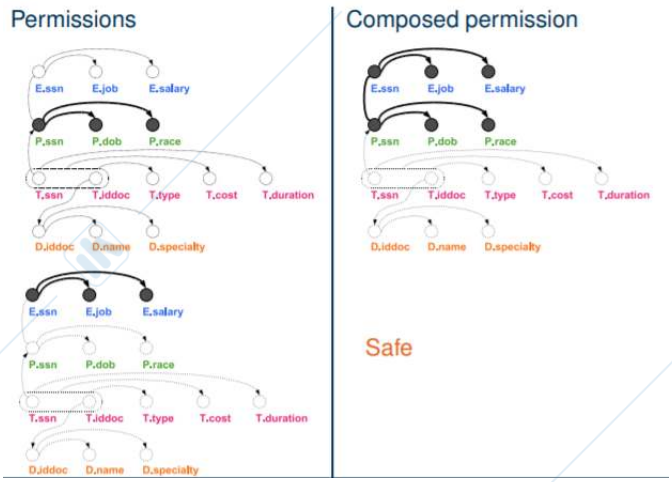


Figura 101: Esempi di composizione dei permessi sicura

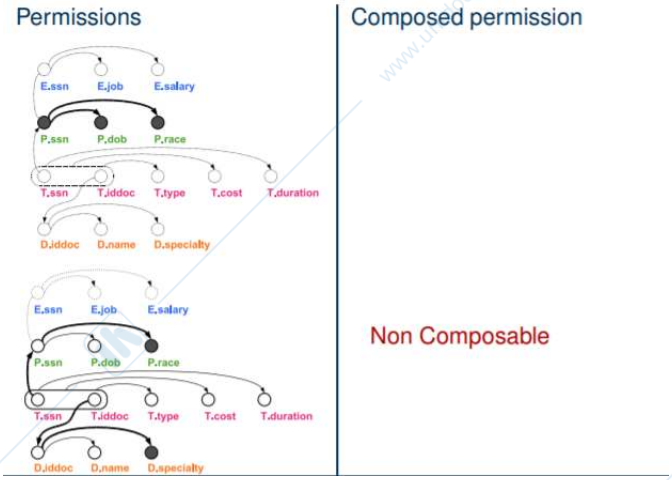


Figura 102: Esempi di composizione dei permessi sicura

L'applicazione del controllo di accesso può essere effettuata, mediante:

1. Determinare il set di autorizzazioni P applicabile alla query q
2. Calcolare la chiusura sulla composizione di P (calcolo del punto fisso)
3. q è autorizzato se $\exists p \in P^{\otimes}$: tale che p autorizza q.

L'efficienza è garantita sfruttando le implicazioni dell'autorizzazione: se $p_j \to p_i$, $\forall p_k \in P$, tale che:

- $p_j \to p_k \implies (p_i \otimes p_j) \to p_k$
- $p_k \to p_j \implies p_k \to (p_i \otimes p_j)$

Quando si aggiunge $p_i \otimes p_j$, p_j può essere rimosso da P. Non è necessario calcolare tutte le 2^{n-1} composizioni di n autorizzazioni. La complessità computazionale rimane polinomiale: $O(n^3)$.

3.3 Un modello di autorizzazione per query multi-provider

Questo modello differisce perché è possibile coinvolgere parti esterne nell'esecuzione della query, se vantaggioso dal punto di vista economico.

Il problema su cui ci focalizziamo è regolamentare l'accesso ai dati forniti da un cloud provider. L'input del problema è un piano di query (query plan), un insieme di provider di servizi cloud e le autorizzazioni definite dalle autorità competenti. Il risultato del modello è un'assegnazione di operazioni a soggetti (autorità, provider, utenti) che soddisfa le autorizzazioni e riduce i costi. Tratteremo anche query che possono essere raggruppate con un clausola HAVING. La Figura 103 rappresenta il query tree plan della query riportata. La proiezione viene inglobata nel nodo foglia, dal momento che non ha costo e viene eseguita direttamente dall'owner della relazione.

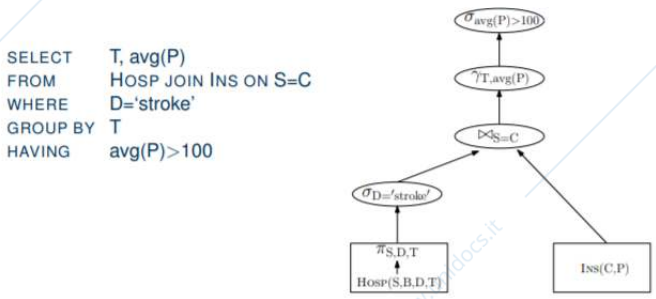


Figura 103: Esempi di query tree plan

Le relazione, nei modelli precedentemente presentati, specificano anche i join path (ossia i join che potevano essere eseguiti dal soggetto per fondere informazioni). La specifica di queste operazioni prevede che tutti gli owner coinvolti nel join devono essere d'accordo per dichiarare un join path, ma in questo modello pretendiamo che ogni owner di dati sia indipendente, quindi il modello base di questo approccio prevede che ogni autorità di dati specifica in modo indipendente le autorizzazioni che regolano l'accesso alla sua relazione. Per la relazione R, l'autorizzazione $[P, E] \rightarrow S$, con $P \subseteq R$ ed $E \subseteq R$, concede a S:

- visibilità in chiaro sugli attributi in P
- visibilità crittografata sugli attributi in E
- nessuna visibilità su tutti gli altri attributi in R

Tutti i modelli precedentemente visti o permettono l'accesso o non lo permettono, mentre in questo modello viene introdotto un livello intermedio, per cui è possibile interrogare i dati crittati, quindi il server dovrà gestire dati crittati. La Figura 104 mostra un modello di autorizzazione, in cui si prendono come esempio le relazioni Hospital (sul server H) e Insurance (sul server I). Per ogni soggetto vengono specificate le autorizzazioni per ogni attributo delle due relazioni. Da notare che il server H ha visibilità plain su tutti gli attributi della sua relazione (Hospital); la stessa cosa vale per le autorizzazioni del server I sulla relazione Insurance.

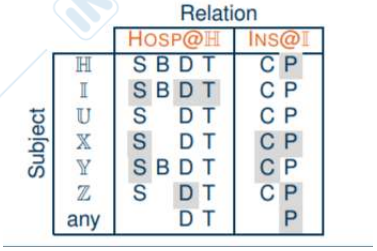


Figura 104: Esempi di modello di autorizzazioni

Siccome si può avere accesso a valori crittati o plain, cambia il profilo di una relazione, rispetto ai modelli precedenti. Il profilo di una relazione cattura il contenuto informativo di una generica relazione R e include:

- v: attributi visibili: testo in chiaro o crittografato nello schema di R
- i: attributi impliciti: trasmessi, in chiaro o criptati, da R. Quindi, quell'insieme di attributi che anche se non appaiono in maniera esplicita, fanno parte del calcolo per ricavare R
  - selezione: SELEZIONA S DALL'HOSP DOVE D = "stroke" perde il valore di D, anche se D non appartiene allo schema
  - raggruppamento: SELEZIONA COUNT (*) DA HOSP JOIN INS SU S = C GROUP BY T perde informazioni su tuple con lo stesso valore per T, anche se T non appartiene allo schema
- $\simeq$: relazione di equivalenza tra gli attributi collegati nel calcolo di R. Confronto degli attributi: SELEZIONA S DA HOSP JOIN INS SU S = C perde i valori di C, anche se C non appartiene allo schema

La Figura 105 riporta la rappresentazione grafica del profilo, in cui i valori colorati di grigio rappresentano i valori crittati.

Figura 105: Rappresentazione grafica del profilo di una relazione

La Figura 106 raffigura le variazioni del profilo informativo sulla base dell'operazione:

- Proiezione: restituisce un sottoinsieme di attributi. Il profilo di relazione contiene nell'insieme degli attributi visibili solo gli attributi proiettati. Gli attributi impliciti e le equivalenze rimangono invariati.
- Selezione: restituisce il sottoinsieme delle tuple sulla base dell'operando. Non ha alcun effetto sullo schema della relazione operando. L'insieme degli attributi impliciti rimane ivnariato. Invece, per quanto riguarda gli altri componenti del profilo, dipendono dalla condizione che viene valutata. Nel primo caso, a viene aggiunto ai valori impliciti; mentre nel secondo caso, a_i e a_j vengono aggiunti nelle equivalenze. a_i e a_j devono presentarsi nella stessa forma (encrypted o plain) e devono essere entrambi visibili.
- Prodotto Cartesiano: restituisce il prodotto cartesiano delle relazioni R_l e R_r. Tutte le possibili combinazioni delle loro tuple. Gli attributi visibili o impliciti o nella relazione risultato non sono altro che l'unione degli attributi visibili, impliciti e equivalenze dei due operandi.
- Join: restituisce come risultato una relazione che contiene la concatenazione delle tuple degli operandi R_l e R_r che soddisfa la condizione di join del tipo $\bowtie_{a_i op a_j}$. É equivalente all'operazione di selezione, rispetto alla condizione eseguita sul prodotto cartesiano dei due operandi. a_i e a_j devono essere entrambi nella stessa forma.
- Group By(A): raggruppamento sulla base di un attributo della relazione operando. Viene eseguita una valutazione di una funzione aggregata a. Il profilo di relazione risultante contiene nell'insieme degli attributi visibili solamente gli attributi che appaiono nel raggruppamento e la funzione aggregata, che abbiamo supposto chiamarsi a. Gli attributi che appaiono in A devono essere aggiunti negli attributi impliciti, perchè fare un raggruppamento su questi attributi, significa applicare una sorta di condizione su di essi.

- User Defined Function (ν): sono delle funzioni che possono apparire nella query (costosi sia per computazione che per dimensione). Vengono considerati gli attributi A, sui quali viene valutata la funzione, mentre a è il risultato dell'applicazione. Gli attributi visibili contengono il risultato dell'applicazione (a) e anche gli attributi su cui non ho operato. Gli attributi impliciti coincidono con l'operando. Nell'insieme delle equivalenze viene aggiunto anche A, su cui ha operato la funzione.
- Encryption: il profilo risultato coincide con quello dell'operando, ad eccezione degli attributi su cui encryption si applica, che passano da essere plain ad essere encryption. Il profilo per decryption funziona allo stesso modo, ma inversamente.

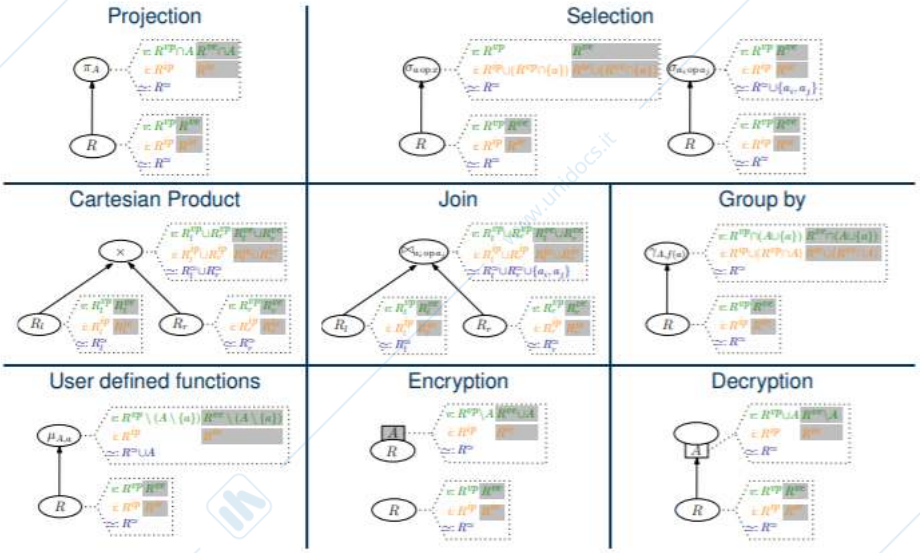


Figura 106: Profili risultanti da operazioni

La Figura 107 mostra un esempio di query tree plan con l'aggiunta dei profili di relazione. Quindi, viene aggiunto un profilo per ogni nodo dell'albero. Nel caso della selezione dell'attributo "Stroke", l'attributo D deve essere aggiunto all'insieme degli attributi impliciti. Risalendo l'albero, dopo l'esecuzione del primo join vengono aggiunti nel campo equivalenze gli attributi su cui calcolo la codnizione di equivalenza del join. Dopo l'operazione di group by, rimangono solo gli attributi direttamente interessati tra gli attributi visibili; mentre T deve essere aggiunti agli attributi impliciti. Dopo l'ultima operazione di selezione viene aggiunto P ai valori impliciti.

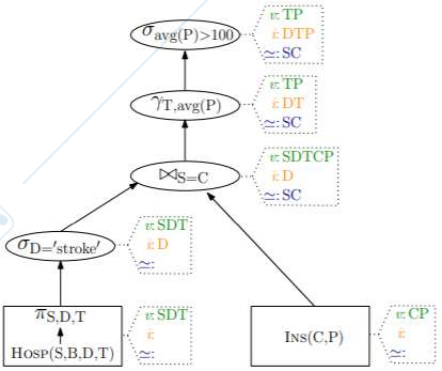


Figura 107: Query tree plan con l'aggiunta dei profili

Il soggetto S è autorizzato per R se e solo se:

- se possiede l'autorizzazione per la visibilità in chiaro su attributi in chiaro (visibili o impliciti)
- se possiede l'autorizzazione per la visibilità in chiaro o crittografata su attributi crittografati (visibili o impliciti). L'autorizzazione di accesso ai dati plaintext è maggiore di encrypted, quindi se ho visibilità plain posso accedere a valori crittati.
- se possiede l'autorizzazione per la visibilità uniforme (in chiaro o crittografata) su attributi equivalenti. Ciò deriva dal fatto che per poter valutare un'equivalenza i due attributi devono essere nella stessa forma.

La Figura 108 riporta un esempio di profilo di relazione e la relativa tabella con le autorizzazioni di accesso per i vari soggetti. Il profilo della relazione prevede l'accesso plain all'attributo P; mentre l'accesso encrypted agli attributi BSC. Gli attributi S e C devono essere nella stessa forma, perchè possano essere confrontati. Vediamo chi ha accesso alla relazione:

- H non ha accesso, perché l'attributo P deve essere accessibile in modalità plain.
- I non ha accesso, perché gli attributi S e C nelle sue autorizzazioni non sono uniformi (S è crittato, mentre C è plain).
- U non ha accesso perché non ha alcun tipo di visibilità sull'attributo B.
- X non ha accesso perché non ha alcun tipo di visibilità sull'attributo B e non ha accesso plain all'attributo P.
- Y è autorizzato ad accedere alla relazione
- Z non ha accesso perché non ha alcun tipo di visibilità sull'attributo B e non ha accesso plain all'attributo P.

I candidati vengono assegnati ai nodi dell'albero sulla base delle autorizzazioni in Figura 108. La crittografia degli attributi non necessari in testo normale per la valutazione degli operandi può aumentare i candidati per un nodo. La Figura 109 riporta due esempi di metodi di assegnazione. Nel primo caso, per trovare i candidati, voglio massimizzare la visibilità, quindi riduco l'encryption, a meno che non sia strettamente necessaria. Ciò può evitare operazioni di encryption e decryption, ma riduco il numero possibile di candidati per nodo. Il secondo esempio

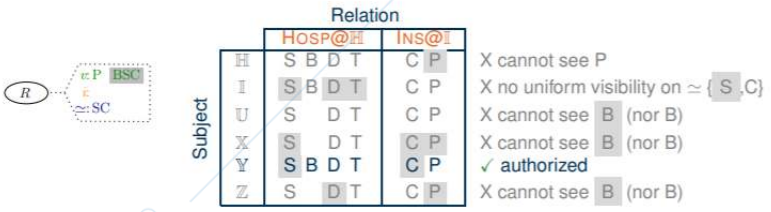


Figura 108: Visibilità autorizzate

riporta un altro tipo di scelta, ossia minimizzare il visibile (need-to-know), per fare questo critto tutto e lascio decrittato solo se strettamente necessario. Questo porta ad un eccessivo uso di encryption e decryption Ad esempio, si suppone che non sia conveniente lasciare P crittato.

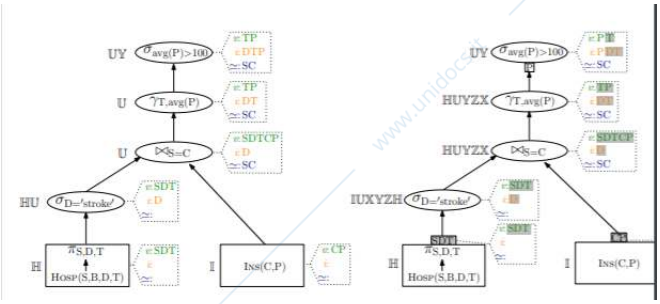


Figura 109: Assegnamento computazione

Quindi, si applica una via di mezzo tra le due strategie, chiamata Minimally extended query plan. Scelto un candidato per ciascun nodo:

- crittografa gli attributi quando necessario per obbedire alle autorizzazioni
- decifra gli attributi quando necessario per l'esecuzione di un'operazione

La Figura 110 mostra un esempio di questa strategia. I candidati vengono scelti per motivi economici. Tra i candidati scelgo chi deve svolgere l'operazione e modello l'encryption o la decryption sulla base delle autorizzazioni del candidato. Il server H, essendo owner della tabella Hospital può eseguire sia la proiezione che la selezione su di essa. Allo stesso modo, il server I può eseguire il nodo foglia relativo alla relazione Insurance, perché owner della tabella. X, scelto come esecutore del $\bowtie_{S=C}$, può accedere in modalità encrypted su S, C e P, quindi devo crittare questi tre attributi. L'attributo P rimane crittata, così che X possa eseguire l'operazione di GROUP BY. Per l'operazione di selezione finale P non può essere crittata, quindi viene decrittato l'attributo P e Y può eseguire l'operazione.

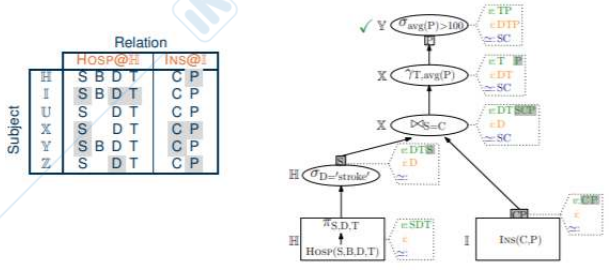


Figura 110: Query Plan minimally extended

Sarà necessario crittare gli attributi con un algoritmo di cifratura che permetta di svolgere l'operazione che deve essere applicata su quei dati. Gli attributi, in condizioni che li confrontano, devono usare la stessa chiave, ossia gli attributi nella stessa equivalenza impostata nella radice usano la stessa chiave. Le chiavi vengono distribuite ai soggetti responsabili della crittografia/decrittografia. Per quanto riguarda la Figura 110, k_{SC} è la stessa chiave per S e C fornita a H per la crittografia S e ad I per la crittografia C. k_P è la chiave per P data a I per la crittografia e a Y per la decrittografia.

Viene valutato sperimentalmente il costo economico della valutazione delle query presupponendo che, oltre alle autorità:

- UA: solo l'utente può accedere alle relazioni di base
- UAP_{enc}: i provider cloud possono accedere a tutti gli attributi crittografati
- UAP_{mix}: i provider cloud possono accedere ad alcuni degli attributi in chiaro (gli altri crittografati)

Il coinvolgimento dei cloud provider consente risparmi economici significativi.

Alcuni problemi ancora da risolvere sono:

- Amministrazione delle autorizzazioni
- Soluzioni architettoniche
- Coinvolgimento di provider esterni per l'archiviazione delle relazioni di base
- Archiviazione delle relazioni di base in forma crittografata
- Supporto per garanzie di correttezza sui risultati della query
- Propagazione di restrizioni di accesso e metadati insieme ai dati (politiche adesive, provenienza)
- Supporto per politiche multi-proprietà

4 Requisiti e preferenze degli utenti per la selezione del piano cloud

Il mercato cloud si sta muovendo verso uno scenario ricco e diversificato, con maggiori possibilità per i clienti in prospettiva. Il processo di selezione, però, può essere difficile. Subentrano, infatti, diversi problemi:

- identificazione degli attributi di qualità del servizio
- definizione di metriche per determinare i piani preferiti
- considerazione delle proprietà di sicurezza negli SLA
- necessario supporto utente per
  - la definizione di requisiti (hard) di sicurezza / privacy e preferenze (soft)
  - la definizione di specifiche di alto livello a supporto del ragionamento sui requisiti di sicurezza / privacy

4.1 Supporto dei requisiti e delle preferenze degli utenti

L'obiettivo è fornire un linguaggio semplice e ad alto livello e una tecnica che consenta di creare una sorta di ranking tra i cloud plan.

Lo scenario di riferimento (Figura 111) prevede la presenza di:

- un utente, che deve scegliere un cloud plan sulla base della sue necessità
- diversi cloud provider che forniscono diversi servizi
- un broker, che funge da intermediario tra il client e i cloud provider. Esso riceve in input una collezione di tutti i servizi offerti dai vari cloud provider e tutti i requisiti da parte dell'utente. Fornisce un ranking dei cloud plan sulla base dei requisiti dell'utente.

Quindi, lo scenario si basa su:

- l'identificazione di possibili requisiti e preferenze.
- il linguaggio espressivo e intuitivo per esigenze e preferenze. Si utilizza un linguaggio ad alto livello.
- classifica dei piani accettabili in base alle preferenze.

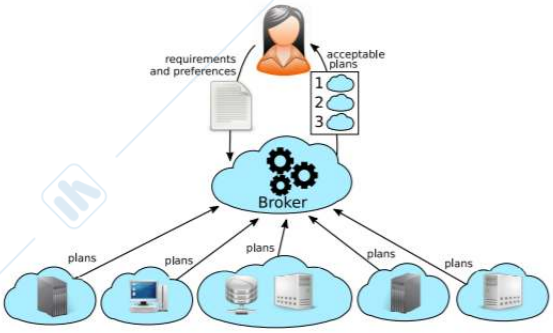


Figura 111: Un approccio basato sulla mediazione

Un piano P è modellato come un insieme $[a_1, ..., a_n]$ di attributi di interesse. La Figura 112 mostra un piano riassuntivo per i vari cloud plan. In particola, nella Figura si hanno 5 cloud plan: P_1, P_2, P_3, P_4, P_5. Gli attributi di interesse presi in analisi sono:

- *prov*: Qual è il provider
- *loc*: Dove sono i server che offrono il servizio
- *encr*: L'algoritmo di encryption usato
- *avail*: Garanzia di disponibilità (M = media, H = alta, VH = molto alta)
- *test*: Qual è l'autorità che si occupa di eventuali test, ed esempio pentration test
- *cert*:Qual è la certificazione di sicurezza utilizzata nel cloud plan
- *aud*: Frequenza di audit

I valori potrebbe non essere dichiarati, perché non noti o perché non hanno significato nel contesto.

I requisiti limitano i valori che possono essere assunti da un piano. I requisiti devono essere espressi attraverso un linguaggio intuitivo e espressivo, che offre costrutti semplici. La Figura 113 mostra i principali costrutti del linguaggio per la specifica di requisiti:

	P_1	P_2	P_3	P_4	P_5	
prov	Ghost	Ghost	GoGo	GoGo	GoGo	(provider)
loc	US	US	EU	US	EU	(server location)
encr	3DES	AES	3DES	AES	AES	(encryption algo)
avail	M	H	VH	H	VH	(availability)
test	authC	authB	authB	authA	authA	(pen test authority)
cert	certB	certC	certB	certC	certA	(security certification)
aud	1Y	_	_	_	_	(audit frequency)

Figura 112: Modello astratto di piani cloud

- IN e $NOT\ IN$ rappresentano gli attribute term, ossia i blocchi base per definire i requisiti. $IN/NOT\ IN$ rappresenta se il provider deve essere o non deve essere uguale all'insieme specificato.
- ANY significa che almeno uno degli attribute term all'interno di ANY deve essere soddisfatto.
- ALL: Tutti gli attribute term dichiarati devono essere soddisfatti.
- $IF\ THEN$: Se la clausola dichiarata dopo l'IF significa che la clausola dichiarata dopo il $THEN$ è vera, ossia richiede il soddisfacimento della formula che segue il $THEN$ se la formula che segue l'IF è vera.
- $FORBIDDEN$: Non tutti gli attribute term dichiarati possono essere soddisfatti: almeno uno non deve esserlo.
- $AT\_LEAST(N, \{AT_1, AT_2, ..., AT_N\})$: nell'insieme di attribute term almeno 2 devono essere soddisfatti.
- $AT\_MOST(N, \{AT_1, AT_2, ..., AT_N\})$: nell'insieme di attribute term al massimo 2 devono essere soddisfatti.

simple	prov IN {Ghost, GoGo, MHard} avail NOT IN {VL, L}
alternatives	ANY({test IN {authA, authB}, cert IN {certA, certB}})
conjunctions	ALL({loc IN {EU, US}, encr NOT IN {DES}})
implications	IF ALL({loc IN {US}, encr IN {3DES}}) THEN ANY({audit IN {3M, 6M}, cert IN {certA}})
exclusions	FORBIDDEN({loc NOT IN {EU}, test IN {authC}})
at least n	AT_LEAST(2, {loc IN {EU}, encr IN {AES}, prov IN {GoGo, Ghost}})
at most n	AT_MOST(2, {prov IN {Ghost}, avail IN {M, MH}, encr IN {3DES}})

Figura 113: Linguaggio per la specifica dei requisiti

Un piano è accettabile se e solo se soddisfa tutti i requisiti utente. La Figura 114 mostra un esempio in questo senso. c_1, c_2, c_3, c_4, c_5 sono dei requisiti stabiliti dall'utente. I cloud plan devono essere valutati sulla base di questi requisiti. Tutti i piani soddisfano i primi 3 requisiti. Il quarto requisito non è soddisfatto da P_1, perché un piano non può non essere in Europa e allo stesso tempo essere testa dall'AuthC. Il quinto requisito è soddisfatto da tutti i piani, fatta eccezione per il primo, per cui solo un attribute term di quelli espressi è soddisfatto.

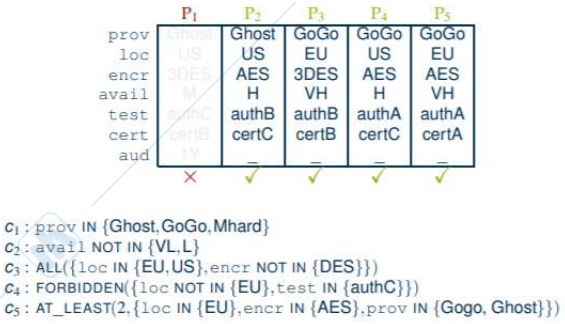


Figura 114: Piano accettabile

L'utente, oltre ai requisiti, può specificare anche delle preferenze, ossia requisiti soft, che rappresentano il fatto che alcune caratteristiche sono preferite ad altre. Vengono applicati sui vari piani accettabili per classificarli. Esistono due tipi di preferenze:

- sui valori degli attributi
- sugli attributi stessi

I vincoli sono requisiti mandatori, mentre le preferenze subentrano nel caso in cui ci siano più cloud plan che soddisfano i requisiti. Fissato un attributo, ci sono alcuni valori che sono preferiti ad altri valori per un attributo. Ciò viene modellato attraverso una relazione di preferenza, andando a definire una relazione di ordinamento totale su set di valori accettabili. Significa mettere in ordine di preferenza le varie occorrenze di un attributo. Ad esempio, per quanto riguarda l'attributo *prov* si ha la seguente relazione di preferenza: $\{MHARD\} \succ \{GoGo\} \succ \{Ghost\}$. Inoltre, associa una funzione di punteggio (scoring) che riflette la posizione relativa dei valori. Quindi, associa un valore numerico ai valori, perché possono facilitare il controllo delle preferenze. La Figura 115 mostra l'associazione della funzione di punteggio ai vari componenti dell'insieme dell'attributo.

Figura 115: Preferenze sui valori degli attributi

Il vettore di punteggio Π_i include i punteggi dei valori di P_i. Quindi il vettore di score, rappresenta i valori di score dei vari attributi per un cloud plan. La Figura 116 mostra un esempio di vettore di score per il piano P_2: i quadrati gialli sono quelli relativi al vettore. I quadrati rappresentano uno o una classe di valori.

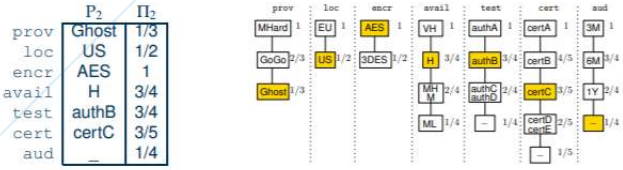


Figura 116: Preferenze sui valori degli attributi

Alcuni attributi sono più importanti di altri. Questo viene modellato tramite una funzione di peso: pesi più elevati implicano un'importanza maggiore. Ad esempio, $w(prov) = 1, w(disp) = 10$ significa che l'attributo *avail* è più importante rispetto all'attributo *disp*.

Esistono tre possibili approcci per creare un ranking tra i cloud plan:

- Dominio di Pareto
- D-dominance (basata sulla distanza)
- WD-dominance (basata sulla distanza ponderata)

Ogni strategia definisce il dominio tra coppie di piani: P_i domina $P_j \implies P_i$ è preferito a P_j. Per quanto riguarda la dominanza di Pareto P_i domina P_j se e solo se per tutti gli attributi, valori uguali o più preferiti di quelli in P_j e per almeno un attributo, un valore più preferito di quello in P_j. La Figura 117 mostra un esempio in cui P_5 domina P_4, infatti i valori degli attributi di P_5 sono maggiori o uguali rispetto ai valori degli attributi di P_4, secondo la gerarchia di ordine parziale dichiarata precedentemente.

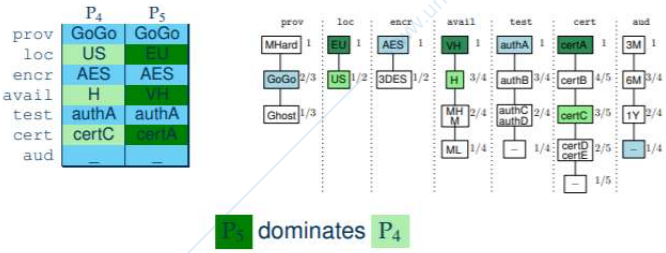


Figura 117: Pareto

Si tratta della definizione classica di dominanza, con cui, però, non riesco sempre a creare un ranking. Infatti, nella Figura 118, P_2 e P_3 non sono comparabili, perché hanno lo stesso valore di equivalenza per gli attributi $prov, encr, test$ e $audit$, ma bisognerebbe preferire P_2 per l'attributo *encr* e P_3 per i rimanenti. La relazione si invertirebbe, quindi non è più vero che P_3 domina P_2, ma allo stesso tempo P_2 non domina P_3. Quindi quando due piani non sono comparabili non si è in grado di fornire un ranking.

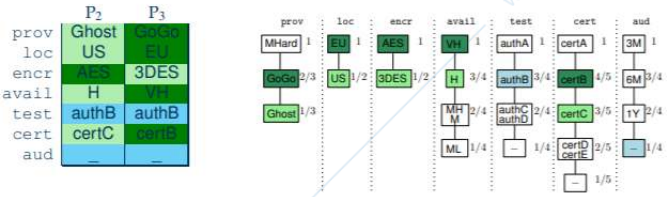


Figura 118: Pareto

Per ovviare a questa problematica, si utilizza un'altra tecnica di ranking. La seconda tecnica per stabilire un ranking tra cloud plan è basata sulla distanza. L'idea che sta alla base è quella di stabilire un ranking tra i plan sulla base della distanza tra questi plan e un piano ideale, in cui gli attributi assumono tutti il valore più preferito. Ci possono essere più cloud plan ideali, perché il valore preferito può essere rappresentato da una classe di valori e non da un singolo attributo. I piani sono rappresentati come punti in uno spazio n-dimensionale, dove n è il numero di attributi. Le coordinate del piano P sono i valori nel vettore di punteggio Π. Il dominio è dato dalla distanza da un piano ideale $P_\top$, nel quale per tutti gli attributi, $P_\top$ ha i valori più alti ($\Pi_\top$ è rappresentato da tutti 1). La Figura 119 mostra un esempio semplificato per avere una

visualizzazione grafica del procedimento. I punti vengono riportati sul piano cartesiano e viene calcolata la distanza euclidea con la formula riportata in basso. In questo caso, viene calcolata la distanza sulla base degli attributi *aud* e *encr*. Il calcolo della distanza calcola la distanza dal piano ideale $P_\top$ da P_2 e P_3. Si scegli il cloud plan più vicino, quindi quello con il valore della distanza più basso. In questo caso si preferisce P_2 a P_3.

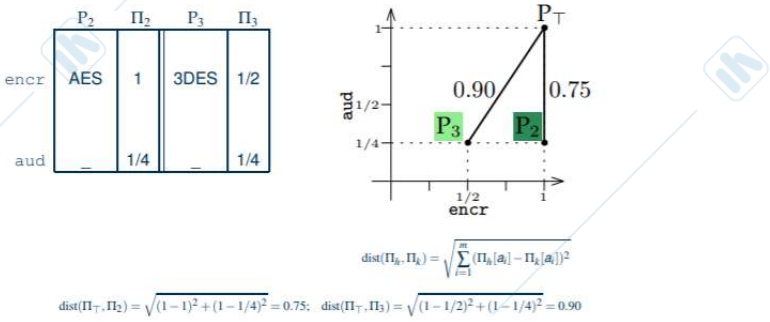


Figura 119: Dominanza basata sulla distanza

La Figura 120 mostra il calcolo della distanza per i piani P_2 e P_3 sulla base di tutti gli attributi (quindi viene considerato un piano a 7 dimensioni). La Figura riporta le distanza rispetto al piano $P_\top$. Secondo queste distanza, il piano P_3 è da preferire.

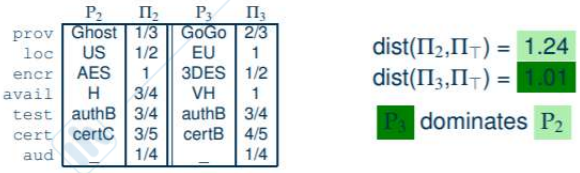


Figura 120: Dominanza basata sulla distanza

Non potrà mai accadere che due piani non siano comparabili. Può accadere che due piani siano equidistanti dal piano ideale, quindi si tratterebbe di due piani totalmente equivalenti lato utente.

Una variazione rispetto alla tecnica precedente consiste nell'aggiungere un peso agli attributi, quindi stabilendo una priorità tra attributi. La priorità tra gli attributi avviene ridimensionando lo spazio n-dimensionale. Il fattore di ridimensionamento lungo la dimensione per l'attributo a è $w(a)$. Nella Figura 121 viene riportato un esempio di dominanza basata sulla distanza pesata, in cui viene ridimensionata l'ascissa. Infatti, il valore di *encr* ha un peso pari a 2, mentre il valore di *aud* uguale a 1. Non si fa altro che allungare l'ascissa di un fattore pari a 2, ossia i valori di *encr* vengono moltiplicati per 2.

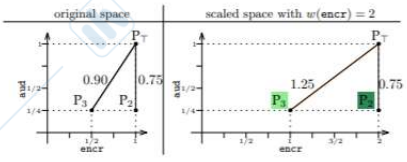


Figura 121: Dominanza basata sulla distanza pesata

In entrambi i casi, viene rispettata la dominanza di Pareto, ossia se P_i domina P_j secondo Pareto, allora anche tramite queste tecniche P_i domina P_j. Non è vero il contrario, o meglio non è possibile stabilire nulla. La complessità del processo di ranking dipende dalla relazione di dominazione scelta:

- Il caso peggiore per la dominanza di pareto è $O(n^2)$
- Il caso peggiore per la dominanza basata sulla distanza è $O(n(log(n)))$

4.2 Logica Fuzzy

L'appartenenza di un elemento in un insieme, può essere rappresentata da una funzione di membership, con una funzione a scalino. Ad esempio, vogliamo stabilire l'insieme delle persone alte, considerando come alte le persone con un'altezza maggiore o uguale a 180 cm. Il valore crisp può essere 0 o 1, a seconda che appartenga o meno all'insieme "Persone alte". Se, invece, andiamo a considerare l'insieme fuzzy, gli elementi non sono collegati a una funziona a scalino, ma una funzione che risulta più graduale. Quindi, esiste un grado di appartenenza all'insieme che stabilisce quanto un utente appartiene all'insieme. Ad esempio, una persona alta 205 cm appartiene di certo all'insieme delle persone alte; così come una persona alta 152 cm non può appartenere all'insieme delle persone alte. Quindi:

- Crisp Set Representation (funzione caratteristica) - visione bianca o nera: restituisce 1 se appartiene ad A, 0 altrimenti.

$$f_A(x): X \to 0,1$$
$$f_A(x) = \begin{cases} 1, \text{if } x \in A \\ 0, \text{if } x \notin A \end{cases}$$

Figura 122: Crisp Set Representation

- Fuzzy set representation (Funzione di appartenenza): $\nu_a(x)$ rappresenta la funzione di appartenenza. Restituisce un valore appartenente al dominio compreso tra 0 e 1, e viene stabilito sulla base di quanto un elemento appartena ad un insieme. Le tipiche funzioni di

$$\mu_A(x): X \to [0,1]$$
$$\mu_A(x) = 1 \text{ if } x \text{ is totally in } A;$$
$$\mu_A(x) = 0 \text{ if } x \text{ is not in } A;$$
$$0 < \mu_A(x) < 1 \text{ if } x \text{ is partly in } A.$$

Figura 123: Fuzzy Set Representation

appartenenza sono:

  - Funzione triangolare
  - Funzione trapezoidale
  - Funzione Gaussiana
  - Funzione di Bell

Bisogna distinguere la funzione di appartenenza dalla probabilità che si verifichi un evento. Sia la funzione di appartenenza che la probabilità mappano l'elemento ad un dominio compreso tra 0 e 1, ma corrispondono a concetti completamente diversi tra loro. La probabilità misura la conoscenza di verità dell'evento per cui possa appartenere ad un determinato dominio (si occupa di verosimiglianza e incertezza). Mentre, la funzione di appartenenza misura il grado di appartenenza di x all'insieme A, quindi l'evento x si è già verificato e si è interessati a vedere quanto questo evento appartenga all'insieme. Fa riferimento ad ambiguità e vaghezza.

4.2.1 Operatori Fuzzy

Vengono composte diverse funzioni di membership, attraverso gli operatori booleani: AND, OR e NOT. L'operatore OR corrisponde a calcolare il valore massimo tra i valori di membership. L'operatore AND significa calcolare il valore minimo tra le due funzioni di appartenenza. Il complemento rappresenta la funzione inversa rispetto all'asse x della funzione di membership.

4.2.2 Variabili Fuzzy

La logica fuzzy serve per rappresentare e trattare concetti o informazioni espressi in forma qualitativa. Quindi, vengono utilizzati variabili linguistiche e modificatori (hedges). La proposizione fuzzy utilizza variabili linguistiche. Si è portati ad utilizzare valori linguistici per due ragioni principali: l'uomo elabora meglio concetti espressi con valori linguistici. Inoltre, i valori linguistici condensano in una sola parola più informazioni, fra loro equivalenti per quanto concerne la reazione umana. Perciò è di importanza fondamentale definire una metodologia per strutturare informali leggi linguistiche. Una variabile linguistica è una variabile che ha come valori attributi linguistici, cioè parole utilizzate nel linguaggio comune. Il concetto, invece, è caratterizzato attraverso insiemi fuzzy definiti nell'universo all'interno del quale la variabile assume significato: un insieme fuzzy per ogni valore (attributo linguistico). La variabile viene fuzzyficata indicando il grado di soddisfacimento (o DOF,degree of fulfillment) di ogni attributo. Una variabile linguistica è definita in termini di una variabile base (variabile in senso classico), i cui valori sono numeri reali all'interno di un dato intervallo. Una variabile linguistica è definita attraverso i suoi termini primari. I termini primari sono etichette di insiemi fuzzy: "bassissima", "bassa", "media", "alta" e "altissima" (in caso di rappresentazione della temperatura, per esempio). I modificatori (hedges) sono avverbi del linguaggio naturale, come "molto", "abbastanza", "leggermente", ecc. e modificano i termini in $T(x)$ e i relativi insiemi fuzzy che ne determinano il significato. Le modifiche avvengono applicando dei modelli matematici al grado di appartenenza A assunto dal termine in assenza di modificatori. Nel caso di più modificatori applicati ad uno stesso termine, vale la seguente gerarchia per le precedenze:

- NOT e modificatore fuzzy
- AND
- OR

4.2.3 Sistema di inferenza fuzzy

Un sistema di inferenza fuzzy realizza un mapping di I/O mediante l'inferenza di regole linguistiche (fuzzy) del tipo IF-THEN: Dato un input (colore di un frutto), il modello definito dalle

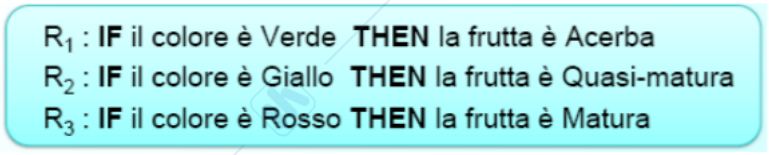


Figura 124: Regole linguistiche

tre regole fuzzy può essere usato per derivare l'output corrispondente, ovvero il grado di maturazione, mediante un processo di inferenza (implicazione).Il meccanismo di inferenza si suddivide in diverse fasi:

- Fuzzificazione: i valori in input saranno tipicamente 2. La Figura 125 rappresenta le funzioni di membership. I valori di input devono essere trasformati sulla base delle funzioni di membership.

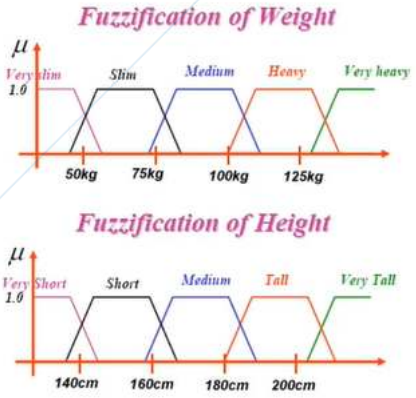


Figura 125: Fuzzificazione

- Applicazione delle regole. Le regole vengono tabulate, usando i valori linguistici per determinare il grado di salute, in questo caso. Dati i miei valori di input, devo calcolare il mio grado di salute. I valori crisp vengono fuzzificati, ossia vengono trasformati in valori

Height \ Weight	Very Slim	Slim	Medium	Heavy	Very Heavy
Very Short	H	SH	LH	U	U
Short	SH	H	SH	LH	U
Medium	LH	H	H	LH	U
Tall	U	SH	H	SH	U
Very Tall	U	LH	H	SH	LH

Figura 126: Forma tabellare delle regole

linguistici.

- Calcolare la mia funzione di membership sulla base dei dati di input. La Figura 127 mostra l'altezza pari a 185 cm e il peso 49 kg, mappando i valori di input sui valori fuzzy.

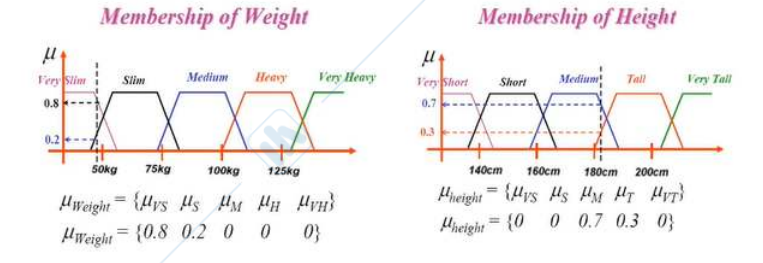


Figura 127: Calcolo della funzione di membership

Prendendo questi valori, li utilizzo nella tabella per calcolare il grado di salute. Il valore della variabile Fuzzy è dato dall'intersezione tra altezza e peso.

Height \ Weight	0.8	0.2	Medium	Heavy	Very Heavy
Very Short	H	SH	LH	U	U
Short	SH	H	SH	LH	U
0.7	LH	H	H	LH	U
0.3	U	SH	H	SH	U
Very Tall	U	LH	H	SH	LH

Height \ Weight	0.8	0.2	Medium (0)	Heavy (0)	V.Heavy (0)
V. Short (0)	0	0	0	0	0
Short (0)	0	0	0	0	0
0.7	0.7	0.2	0	0	0
0.3	0.3	0.2	0	0	0
V. Tall (0)	0	0	0	0	0

Figura 128: Valore

Ho i seguenti gradi di appartenenza: 0,7 per LH; 0.2 per H e SH; e 0.3 per U. Devo riportare questi valori sulle funzioni di membership della mia variabile di utput, posso riportarli:

- tagliando le funzioni di membership sulla base dei valori calcolati.
- scalando le funzioni di membership sulla base dei valori calcolati.

Infine, avviene il processo di defuzzificazione, quindi le funzioni devono essere tradotte in un numero, attraverso il calcolo del centroide. Riporto il centroide sul grafo che contiene le funzioni di membership non scalate e determino a che insieme appartengo.

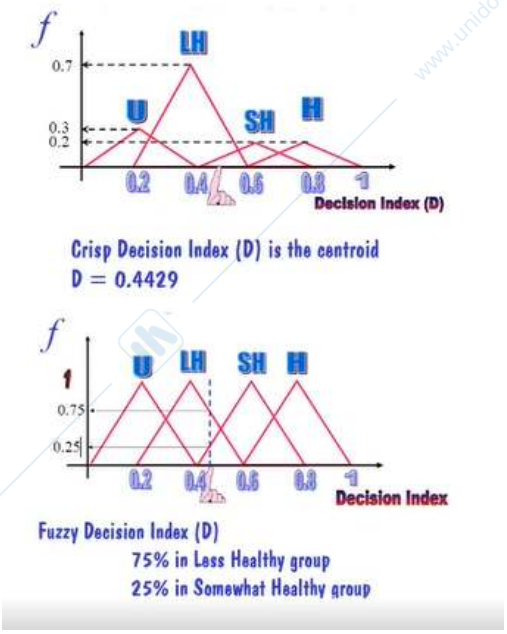
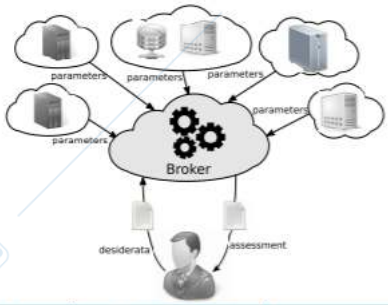


Figura 129: Calcolo del centroide

4.3 Approccio basato su logica fuzzy per la specifica di requisiti utente

L'approccio basato su logica fuzzy rappresenta un altro approccio che permette agli di utenti di esprimere requisiti. Lo scenario di riferimento è simile a quello dell'approccio precedente, ossia diversi cloud provider con diversi cloud plan, rappresentanti diversi parametri di configurazione. Permane la presenza di una figura di intermediazione tra cloud provider ed utente (Broker) e di un utente che fornisce un desiderata (ossia dei requisiti di scelta). La Figura 130 fornisce uno schema di questo scenario con i parametri dei cloud provider:

- *uptime*: percentuale di tempo media in un mese durante il quale il cloud provider è disponibile
- *replicas*: numero di repliche garantite per i dati in outsourcing
- *throughput*: capacità di trasmissione
- *bandwidth*: banda di trasmissione
- *reputation*: reputazione



Parameter p	Domain $D(p)$	Lingustic values $\mathcal{L}(p)$
uptime	[96.00, 99.90]	{low,med,high}
replicas	[1, 10]	{scarce,many}
throughput	[1, 16]	{low,med,high}
bandwidth	[0.2, 25]	{small,large}
reputation	[0, 1]	{bad,avg,good}

Figura 130: Scenario di riferimento e esempio di parametri

In questa tecnica vengono introdotti due nozioni:

- I parametri astratti consentono agli utenti di esprimere i propri desideri senza fare riferimento a specifici valori nitidi dei parametri di configurazione (valori linguistici, di più semplice comprensione).
- Il concetto astratto è una proprietà di alto livello in grado di caratterizzare i piani di servizio cloud (ad es. Prestazioni o affidabilità). Le regole di implicazione collegano concetti e parametri e indicano quali valori per parametri implicano quali valori di concetti di alto livello.

La Figura 130 presenta i parametri astratti, ossia i parametri rappresentati medianti valori linguistici, anziché attraverso valori numerici. L'utente può, quindi, esprimere i propri desiderata senza utilizzare valori a basso livello.

Parameter p	Domain $D(p)$	Lingustic values $\mathscr{L}(p)$
uptime	[96.00, 99.90]	{low,med,high}
replicas	[1, 10]	{scarce,many}
throughput	[1, 16]	{low,med,high}
bandwidth	[0.2, 25]	{small,large}
reputation	[0, 1]	{bad,avg,good}

Figura 131: Parametri astratti

I concetti astratti possono essere espressi attraverso i valori linguistici associati ad essi. I concetti vengono dichiarati, come in Figura 132, attravero i valori linguistici e i parametri coinvolti nella definizione. Il fatto che un concetto dipenda da parametri è invisibile all'utente, cioè l'utente non è a conoscenza del fatto che il concetto dipende dai parametri coinvolti. Inoltre, i concetti vengono dichiarati mediante le regole di implicazione, che permettono di specificare in modo preciso come sono concatenati i parametri per formare i concetti. Le regole di implicazione sono nel formato IF...THEN.

Concept c	Linguistic values $\mathscr{L}(c)$	Involved parameters $\phi(c)$
reliability	{low,med,high}	{uptime, replicas}
performance	{low,med,high}	{throughput, bandwidth}

Implication rules for concept reliability

$\langle \texttt{uptime} = \text{high} \rangle \vee \langle \texttt{replicas} = \text{many} \rangle \Longrightarrow \langle \texttt{reliability} = \text{high} \rangle$
$\langle \texttt{uptime} = \text{med} \rangle \Longrightarrow \langle \texttt{reliability} = \text{med} \rangle$
$\langle \texttt{uptime} = \text{low} \rangle \vee \langle \texttt{replicas} = \text{scarce} \rangle \Longrightarrow \langle \texttt{reliability} = \text{low} \rangle$

Implication rules for concept performance

$\langle \texttt{throughput} = \text{high} \rangle \vee \langle \texttt{bandwidth} = \text{large} \rangle \Longrightarrow \langle \texttt{performance} = \text{high} \rangle$
$\langle \texttt{throughput} = \text{med} \rangle \Longrightarrow \langle \texttt{performance} = \text{med} \rangle$
$\langle \texttt{throughput} = \text{low} \rangle \vee \langle \texttt{bandwidth} = \text{small} \rangle \Longrightarrow \langle \texttt{performance} = \text{low} \rangle$

Figura 132: Concetti astratti

I desiderata degli utenti esprimono le sue preferenze rispetto alle configurazioni dei piani di servizio cloud che soddisfano le sue esigenze. Riflettono i diversi livelli di soddisfazione (ad esempio, Low, Medium, High) dell'utente rispetto alle diverse configurazioni del piano di servizio cloud. Il broker consente agli utenti di utilizzare i valori linguistici per specificare i livelli di soddisfazione. Ovviamente, l'utente può specificare i valori dei parametri anche in modalità crisp. I desiderata sono espressi dagli utenti attraverso regole che dichiarano il livello di soddisfazione implicito da una combinazione di parametri e concetti. La Figura 133 mostra un esempio di regole di satisfaction dell'utente.

Desiderata of user u_1

$\langle \texttt{reliability} = \text{high} \rangle \vee \langle \texttt{reputation} = \text{good} \rangle \Longrightarrow \langle \texttt{sat.} = \text{high} \rangle$
$\langle \texttt{reliability} = \text{med} \rangle \vee \langle \texttt{reputation} = \text{avg} \rangle \Longrightarrow \langle \texttt{sat.} = \text{med} \rangle$
$\langle \texttt{reliability} = \text{low} \rangle \vee \langle \texttt{reputation} = \text{bad} \rangle \Longrightarrow \langle \texttt{sat.} = \text{low} \rangle$

Figura 133: Desiderata dell'utente

Il modello Fuzzy è una delle tecniche adottabili per stabilire un grado di ranking tra i cloud plan selezionabili. É basata sulla logica fuzzy. • Il parametro Fuzzy modella un parametro in un fuzzy set. Ogni valore linguistico l del parametro p è associato a una funzione di appartenenza

$\nu_{p,l} : D(p) \rightarrow [0,1]$. L'input della funzione corrisponde al dominio del parametro p e come risultato fornisce una valutazione del grado di appartenenza del valore crisp a cui viene applicata la funzione. Quindi, dato un determinato crisp del parametro, esso può avere un grado di membership maggiore di 0 per più di un valore linguisto e questo significa che appartiene a più fuzzy set con diversi gradi di appartenenza. Il concetto fuzzy estende un concetto c con un dominio $D(c)$ di valori crisp (che non è naturalmente associato a un concetto astratto) e un insieme di funzioni di appartenenza. Il dominio dei valori crisp è $[0,1]$. Ogni valore linguistico l (rappresentanti diversi valori di compliance) del concetto c è associato a una funzione di appartenenza $\nu_{c,l} : D(c) \rightarrow [0,1]$. Per valutare i desiderata con le caratteristiche del cloud plan, il broker deve poter stabilire una corrispondeza tra valori linguistici e valori crisp. Un metodo intuitivo potrebbe essere quello di partizionare il dominio sulla base del numero di valori possibili, ma le differenze tra un valore e l'altro non sono evidenti. Si utilizza, quindi, il fuzzy modeling, per cui il grado di soddisfazione dei desideri dell'utente è modellato come una variabile fuzzy definita in modo simile ai concetti fuzzy. Il dominio dei valori crisp è $[0,1]$: esso rappresenta la varibile necessaria per quantificare il grado di soddisfabilità. Ogni valore linguistico l di *sat*. è associato a una funzione di appartenenza $\nu_{sat,l} : D(sat) \rightarrow [0,1]$. La Figura 134 propone un esempio di fuzzy modeling. Per ogni valore linguistico si ha una funzione di membership. Tracciando righe verticali a partire dai valori crisp ottengo dei valori di appartenenza in prossimità con il punto di intersezione con le varie funzioni. Quindi, per un valore di *uptime* pari a 97.5 si hanno valori di appartenenza:

- Leggermente maggiore a 0.2 per il valore Low
- Tra 0.6 e 0.8 per il valore Med

Le ascisse, nella Figura, rappresentano il dominio del parametro, mentre le ordinate il grado di appartenenza. L'insieme dei concetti è un superinsieme rispetto a $[0,1]$. Si utilizza un superinsieme per garantire che il centroide di una qualsiasi area definita dalle combinazioni di diversi funzioni di membership copra sempre l'intervallo $[0,1]$. Il broker deve valutare questi valori rispetto ai desiderata e produrre un livello di soddsifacibilità.

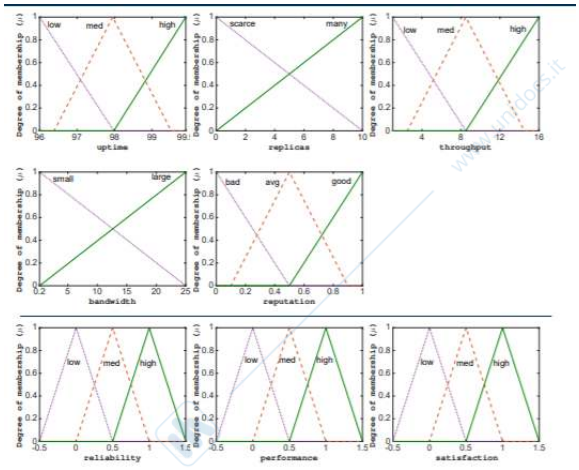


Figura 134: Esempi di fuzzy modeling

La valutazione del piano cloud è rappresentata nella Figura 135 ed avviene in due fasi. Entrambe le fasi utilizzano un Fuzzy Inference System (FIS), che utilizza il metodo Mandani:

- determinare un insieme di regole fuzzy
- fuzzificazione degli input utilizzando le funzioni di appartenenza degli input. Consiste nel mappare i valori di input crisp in gradi di appartenenza per ogni fuzzy set dei parametri e concetti fuzzy, a cui i valori di input fanno riferimento.

- Inference reasoning, che consiste nel valutare le diverse regole rispetto ai valori fuzzy. La base di regole è ottenuta effettuando un mappaggio delle regole in input (concetti e desiderata) nella forma IF...THEN che il FIS è in grado di elaborare.
- Aggregation e Defuzzification: combina i risultati per determinare un singolo valore crisp, se è necessario un output crisp, altrimenti viene mantenuto il valore fuzzy.

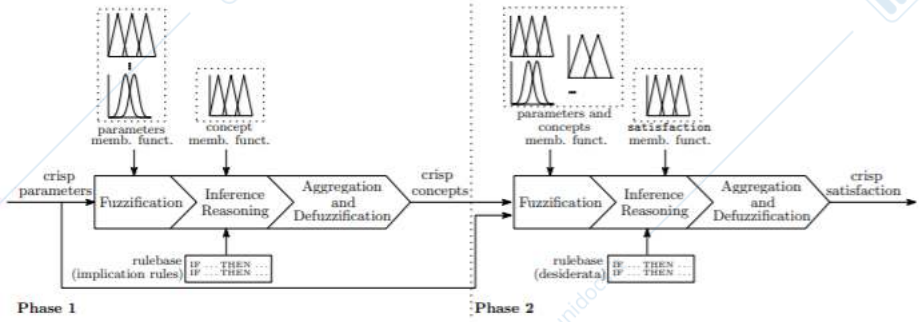


Figura 135: Valutazione dei cloud plan

La Figura 136 mostra un esempio di valutazione per i vari cloud plan. Prendiamo come esempio i valori di *uptime* del server s_1, rappresentano il fatto che 99 abbia un grado appartenenza al valore high per 0.531 e al valore med per 0.326.

			Cloud service plans				
			s_1	s_2	s_3	s_4	s_5
Parameters	uptime	crisp	99	99.9	99	97	96
		high	0.531	1	0.531	0	0
		med	0.326	0	0.326	0.391	0
		low	0	0	0	0.494	1
	replicas	crisp	5	10	0	2	1
		many	0.500	0	1	0.800	0.900
		scarce	0.500	1	0	0.200	0.100
	throughput	crisp	14	8.5	16	8.5	8.5
		high	0.733	0	1	0	0
		med	0.083	1	0	1	1
		low	0	0	0	0	0
	bandwidth	crisp	18.5	12.5	25	12.5	12.5
		large	0.738	0.496	1	0.496	0.496
		small	0.262	0.504	0	0.504	0.504
	reputation	crisp	1	0.5	1	0.5	0.25
		good	1	0	1	0	0
		avg	0	1	0	1	0.375
		bad	0	0	0	0	0.500

Figura 136: Valutazione dei cloud plan

La Figura 137 il grado di appartenenza dei vari provider ai concetti. I valori crisp che vengono forniti sono l'output del primo passo del processo di inferenza. I concetti e i parametri vengono combinati secondo le desiderata proposte dell'utente a formare la variabile di soddisfacibilità.

			Cloud service plans				
			s_1	s_2	s_3	s_4	s_5
Concepts	reliability	crisp	0.509	1	0.435	0.319	0.159
		high	0.018	1	0	0	0
		med	0.982	1	0.870	0.638	0.318
		low	0	0	0.130	0.362	0.682
	performance	crisp	0.671	0.498	1	0.498	0.498
		high	0.342	0	1	0	0
		med	0.658	0.996	0	0.996	0.996
		low	0	0.004	0	0.004	0.004
Sat.	sat.	u_1	0.669	0.750	0.685	0.325	0.265

Figura 137: Valutazione dei cloud plan

La Figure 138 mostra come vengono valutati i valori composti, sulla base delle implication rule elencate nella Figura 132. La prima regola di implicazione è un OR, quindi si prende il valore massimo tra i gradi di appartenenza delle due sottoregole. Quindi prendendo come esempio il server S_1 traccio una riga verticale per il valoure $uptime = 99$ e per $replicas = 5$. Viene preso il punto di intersezione tra questa riga verticale e la curva che rappresenta la funzione le occorrenze dei vari attributi. Quindi, nel caso del server S_1, si prende il grado di appartenenza 0.531, che mi permette di individuare la parte blu. Combino le aree blu e calcolo il centroide: in questo caso è pari a 0.509. Quindi il concetto di reliability ha un valore di 0.509. Questo procedimento è da applicare ad ogni concetto per avere un valore crisp per ogni concetto.

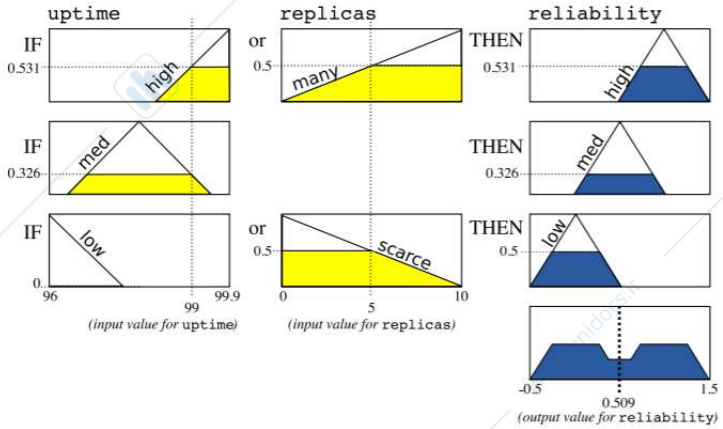


Figura 138: Valutazione dei cloud plan

La Figur 139 mostra la fase 2, in cui prendo tutti i valori dei concetti e procedendo in modo analogo a quanto fatto prima, calcolo il valore di satisfatction. Viene utilizzato il valore di reliability e, a partire da quel valore, traccio la riga verticale per determinare il grado di appartenenza alle categorie di reliability. La stessa cosa viene applicata al concetto di reputation. Combino le aree blu e calcolo il centroide, che rappresenta il grado di soddisfazione per il cloud S_1. Calcolo questo valore per ogni cloud plan.

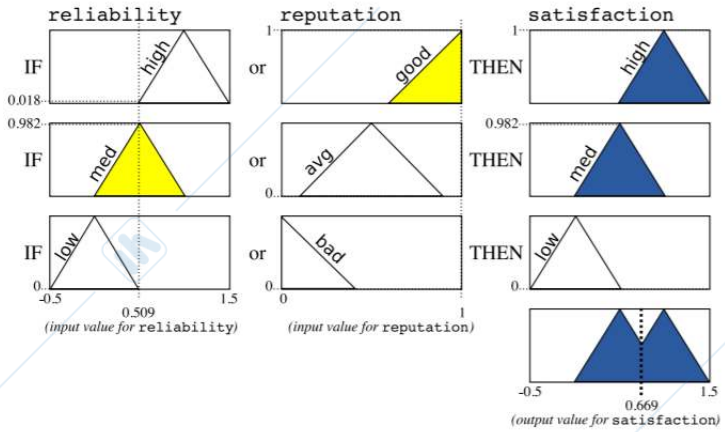


Figura 139: Valutazione dei cloud plan

4.4 Allocazione dei dati consapevole della sicurezza in ambienti multi-cloud

L'idea è quella di selezionare i servizi in modo che venga assicurato il soddisfacimento di tutti i requisiti di protezione, tenendo anche presente il costo economico. Bisogna andare a considerare più cloud provider, i quali offrono diversi piani di servizi cloud. I proprietari dei dati esternalizzano la gestione delle loro risorse a fornitori di cloud esterni. L'obiettivo è andare a selezionare i cloud service, eventualmente più di uno, in modo da soddisfare al meglio i requisiti imposti dall'utente. I servizi cloud sono caratterizzati da un set di attributi A. Le risorse possono essere esternalizzate in testo semplice o in forma crittografata. La forma crittografata può essere letta anche da server meno fidati, ma con un molto più conveniente. La Figura 140 mostra i set di attributi relativi ai 7 cloud service. I valori vuoti rappresentano valori irrilevanti o non disponibili per il servizio in esame.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
provider	Ghost	Cloudy	Amaron	Mist	Mhard	GoGo	NewCloud
loc	EU	US	EU	US	EU	US	EU
cert	certA	certA		certC	certA		certB
audit			auditA				auditB
uptime	99.999	99.998	99.970	99.980		99.997	99.960

Figura 140: Scenario di riferimento

L'assegnazione di una risorsa a un servizio cloud deve anche specificare se la risorsa è esternalizzata in testo semplice o crittografata: $\alpha : \mathcal{R} \rightarrow \mathcal{S} \times \{\circ, \bullet\}$, dove $\alpha(r) =< s, m >$ afferma che la risorsa r è allocata a s se quella r è visibile ($m = \circ$) o crittografata ($m = \bullet$). L'obiettivo è supportare i proprietari dei dati nell'allocazione delle risorse ai servizi cloud che soddisfano meglio le loro esigenze.

Le proprietà di sicurezza astratte sono concetti di alto livello associati a un dominio di etichette. Ogni etichetta corrisponde alla soddisfazione di determinate formule sugli attributi (le formule sono indipendenti). Il fatto che le formule siano indipendenti significa che non c'è nessuna implicazione tra le formule. Per un cloud service che soddisfa L_i non è necessariamente vero che soddisfi L_j, tale per cui L_j domina L_i. I valori associati alle proprietà sono etichette di alto livello che caratterizzano un livello di soddisfazione rispetto alla proprietà. I vantaggi che sorgono dalla definizione di queste proprietà sono:

- Il data owner può ragionare sulle sue necessità senza dover far riferimento agli attributi del cloud service.
- Il mapping tra requisiti e attributi viene fatto una volta sola.

La Figura 141 mostra un esempio di proprietà di confidenzialità C e di disponibilità A. Il simbolo _ nella colonna delle etichette rappresenta il valore di default, che sta ad indicare che l'aspetto non di interesse per l'utente. La colonna Expression $e^p(l)$ riporta le condizioni che devono esser soddisfatte, perché possa essere assegnate quella determinata etichetta. Quindi, ad esempio, per far sì che la confidenzialità sia HC, il cloud service deve essere localizzato in Europa e deve avere come ente di certificazione CertA.

Property p	Label l	Expression $e^p(l)$
C	HC	loc=EU ∧ cert=certA
	MC	cert=certA ∨ audit=auditA
	LC	cert=certB ∨ audit=auditB
	_	true
A	HA	uptime>99.99%
	MA	uptime>99.95%
	LA	uptime>99.90%
	_	true

Figura 141: Proprietà di sicurezza astratte

L'utilizzo di queste proprietà astratte permette al data owner di non preoccuparsi degli attributi che caratterizzano i cloud service. Una classe di sicurezza è una tupla di etichette, con un'etichetta per ogni proprietà. Posso andare a definire un reticolo (Lattice), in cui troviamo le classi di sicurezza ($\mathcal{C}$) con una relazione di ordine parziale indotta dalla relazione di ordine totale delle etichette delle diverse proprietà. La classe di sicurezza $\bot : \bot$ è l'elemento bottom del reticolo. La Figura 142 mostra un esempio di reticolo delle classi di proprietà, dove il primo elemento di una classe di sicurezza rappresenta l'etichetta di confidenzialità, mentre il secondo l'etichetta di disponibilità.

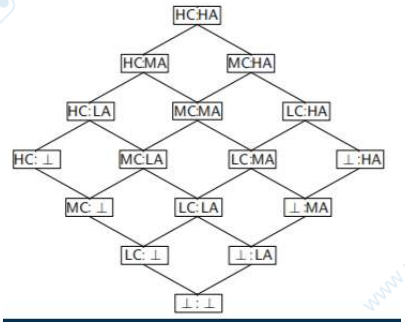


Figura 142: Reticolo di proprietà di sicurezza astratte

Le condizioni associate a un'etichetta devono essere soddisfatte da un servizio cloud per essere considerate per le risorse che hanno quell'etichetta, ossia deve sussistere una mappatura tra servizi ed etichette nelle classi di sicurezza. La classe di sicurezza di un servizio cloud rappresenta la massima garanzia di sicurezza che offre. Data una classe di sicurezza, l'espressione associata è l'AND delle espressioni corrispondenti alle etichette che la compongono. Anche le espressiono sono indipendenti. Se la classe associata ad un cloud service domina un'altra classe, esso soddisfa anche quella, anche se non soddisfa l'espressione associata ad essa. La Figura 143 mostra un esempio di assegnazione delle classi di sicurezza relative ai diversi cloud server. Ad esempio, la classe di sicurezza di s_1 è $[HC, HA]$.

I requisiti di protezione delle risorse sono specificati associando a ciascuna risorsa due classi di sicurezza:

- requisiti per la risorsa in chiaro, $\lambda^\circ : R \to \mathcal{C} \cup \{-\}$
- requisiti per la risorsa crittografata, $\lambda^\bullet : R \to \mathcal{C} \cup \{-\}$

Property p	Label l	Expression $e^p(l)$
C	HC	loc=EU ∧ cert=certA
	MC	cert=certA ∨ audit=auditA
	LC	cert=certB ∨ audit=auditB
	_	true
A	HA	uptime>99.99%
	MA	uptime>99.95%
	LA	uptime>90.90%
	_	true

s_1=[Ghost,EU,certA,_,99.999]
$\lambda(s_1) = [HC, HA]$
s_2=[Cloud,US,certA,_,99.998]
$\lambda(s_2) = [HC, HA]$
s_3=[Amaron,EU,_,auditA,99.970]
$\lambda(s_3) = [MC, MA]$
s_4=[Mist,US,certC,_,99.980]
$\lambda(s_4) = [\_, MA]$
s_5=[Mhard,EU,certA,_,_]
$\lambda(s_5) = [HC, \_]$
s_6=[GoGo,US,_,_,99.997]
$\lambda(s_6) = [\_, HA]$
s_7=[NewCloud,EU,certB,auditB,99.960]
$\lambda(s_7) = [LC, MA]$

Figura 143: Classificazione dei cloud service

La funzione λ associa ad una risorsa la classe di sicurezza associata ad essa. La Figura 144 riporta i requisiti di protezioni minimi, ossia gli attributi devono avere almeno una classi di sicurezza pari a quella nella tabella. Per quanto riguarda l'attributo crittato, la confidenzialità si abbasa perché la risorsa crittata è intrinsecamente protetta, quindi in caso di encryption la confidenzialità ha una etichetta minore.

	λ°	$\lambda^\bullet$
admin	[MC, HA]	[LC, HA]
agreements	[LC, MA]	[_, MA]
suppliers	[MC, MA]	–
projects	[HC, HA]	–
archive	–	[_, LA]

Figura 144: Requisiti di protezione delle risorse

Con il termine allocazione sicura si intende un'allocazione tale che una risorsa sia allocata solo a un servizio la cui classe di sicurezza domina la classe della risorsa. La Figura 145 mostra un esempio di safe allocation. Tramite il reticolo, si ha una visione immediata di qual è la classe di sicurezza dei cloud service e qual è il livello di protezione minimo richiesto. I valori su sfondo grigio rappresentano i valori crittati. L'attributo *projects* non potrà esser assegnato a nessun altro server diverso da s_1. admin potrebbe essere gestito anche da s_1, perché la classe di sicurezza di s_1 domina il livello di protezione richiesto per l'attributo *admin*.

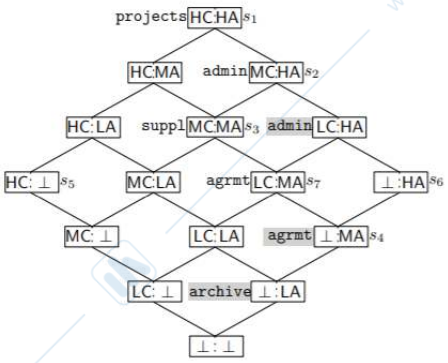


Figura 145: Allocazione sicura

I requisiti di allocazione globale sono requisiti che impongono condizioni sull'allocazione delle risorse che non si riferiscono alle singole risorse ma a gruppi di esse o all'assegnazione complessiva. Esistono tre tipi di requisiti globali:

- Co-location: le risorse in un insieme di *loc* devono essere gestite dallo stesso servizio. Non impone alcuna restrizione sul formato delle risorse salvate.

- Separazione: le risorse in un set specifico non dovrebbero essere tutte visibili dallo stesso servizio, per evitare conflitti. Quindi, almeno una delle risorse deve essere encrypted o allocata su un altro servizio.

- Service usage: coppia $< max\_services, min\_storage >$ impone a un'allocazione da usare al massimo $max\_services$ servizi, usando ciascuno di essi per l'occupazione almeno $min\_storage$ di archiviazione. Il fatto di utilizzare più cloud service può generare overhead, quindi si impone un numero massimo di servizi e la quantità minima che deve essere data in gestione ad un singolo cloud service.

I primi due tipi impongono una condizione sull'allocazione delle risorse, mentre l'ultimo impone condizioni sul numero di servizi che si possono coinvolgere per la gestione delle risorse.

Un'allocazione è compliant quando soddisfa i requisiti di allocazione globale per tutte le risorse. La Figura 146 mostra un esempio di allocazioni safe e compliant. Nella Figura si specifica mediante i requisiti globali che:

- *admin* e *projects* devono essere allocati sullo stesso cloud

- *agreements* e *suppliers* devono essere allocati su cloud diversi oppure devono avere forme differenti (uno crittato e l'altro plain)

- Le risorse devono essere affidatae ad un massimo di tre server e su ognuno di essi devono esserci almeno 30 GB di risorse.

La tabella in Figura mostra alcuni esempi di allocazioni, sulla base dei requisiti di protezione in Figura 145 e sui requisiti globali in Figura 146:

- α_1 e α_2 sono sicuri e compliant

- α_3 è compliant, ma non è sicuro, ossia non soddisfa i requisiti di protezione

- α_4 è sicuro, ma non compliant.

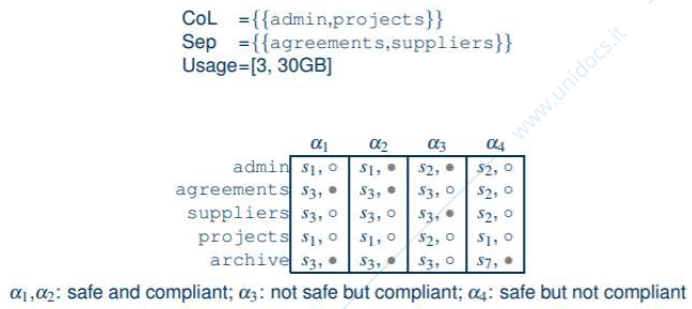


Figura 146: Allocazione sicura e compliant

L'allocazione, inoltre, deve essere minima e corretta. Un'allocazione corretta è un'allocazione che che soddisfa la protezione delle risorse e i requisiti di allocazione globale, quindi che sia safe e compliant. Un'allocazione minima è un'allocazione corretta con un costo economico minimo. Il costo economico preso in considerazione può essere di due tipi:

- Costo di archiviazione: calcolato moltiplicando la dimensione delle risorse (in GB) per il prezzo di archiviazione (in GB) applicato dal servizio cloud

- Costo di trasferimento dati: calcolato moltiplicando la quantità di traffico in uscita (in GB) da un servizio cloud (dato dalla dimensione delle risorse), per il costo di trasferimento dati corrispondente, ponderato per la frequenza con cui si accede alle risorse

Il problema di calcolo di un'allocazione minima e corretta può essere tradotta in un problema di programmazione binario (Figura 147), in cui minimizzare il costo è la funzione obiettivo, con i vincoli che rappresentano i requisiti imposti dall'utente.

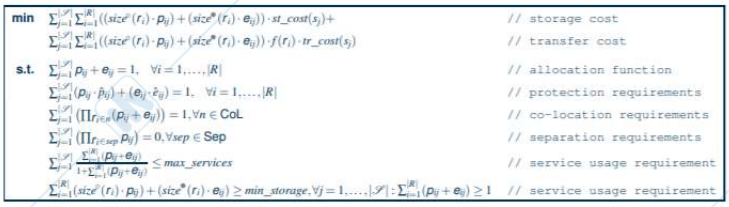


Figura 147: Minimizzare il costo di allocazione

5 Differential privacy

5.1 Introduzione

La differential privacy è una tecnica semantica per la protezione dei dati. La privacy differenziale è un sistema per la condivisione pubblica di informazioni su un set di dati descrivendo i modelli di gruppi all'interno del set di dati mentre si trattengono informazioni sugli individui nel set di dati. Un altro modo per descrivere la privacy differenziale è come un vincolo agli algoritmi utilizzati per pubblicare informazioni aggregate su un database statistico che limita la divulgazione di informazioni private di record le cui informazioni sono nel database. Il mondo in cui viviamo è guidato dai dati. Si tratta di grandi quantità di dati salvati e analizzati per diversi scopi (malevoli e benevoli). I dati sono preziosi. Il concetto di big data è stato adottato da molte aziende ed è entrato nel vocabolario pubblico. I dati riguardano principalmente persone la cui privacy deve essere garantita. Le violazioni di dati possono avere un impatto sulla reputazione, sul lato economico dell'utente, etc. Come possiamo lavorare su dati privati? Bisogna trovare un compromesso tra condivisione e privatizzazione dei dati. Esistono diverse leggi per proteggere i dati più sensibili (PII). I dati personali, noti anche come informazioni personali o informazioni di identificazione personale (PII) sono qualsiasi informazione relativa a una persona identificabile. Il concetto di PII è diventato prevalente in quanto la tecnologia dell'informazione e Internet hanno reso più semplice la raccolta di PII che porta a un mercato redditizio nella raccolta e nella rivendita di PII. Le informazioni personali possono anche essere sfruttate dai criminali per inseguire o rubare l'identità di una persona o per aiutare nella pianificazione di atti criminali. In risposta a queste minacce, molte politiche sulla privacy dei siti Web si rivolgono in modo specifico alla raccolta di informazioni personali e legislatori come il Parlamento europeo hanno adottato una serie di leggi come il regolamento generale sulla protezione dei dati (GDPR) per limitare la distribuzione e l'accessibilità di PII. Il processo che possiamo applicare per trattare dati sensibili è quello di anonimizzarli e condividerli. Il fatto è che, spesso, l'anonomizzazione non è sufficiente. L'autorità che pubblica i dati non può garantire al 100% la privacy degli utenti, dal momento che possono essere sfruttate conoscenze o collegamenti che l'autorità non considera come sensibili, ma da cui è possibile risalire all'identità, o comunque fare inferenze, sugli utenti.

Per rispondere a queste problematiche, vengono sviluppate sempre nuove tecniche per proteggere al meglio la privacy. L'obiettivo che si pongono coloro che le sviluppano è capire come dati sensibili di utenti possono essere condivisi senza violare la privacy di essi. Bisogna sempre trovare un compromesso tra privacy e utilità dei dati pubblicati. La Figura 148 mostra lo scenario di base a cui stiamo facendo riferimento. Da un lato abbiamo un database, formato da un insieme di record. Ogni record rappresenta un utente. I dati presenti dal database vengono analizzati e sanitizzati da un'entità fidata che si interpone tra i dati rilasciati e il database stesso.

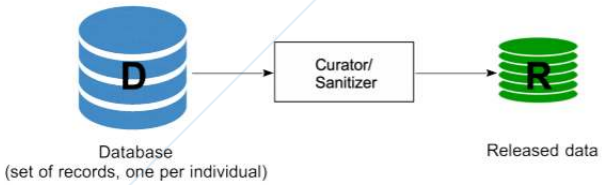


Figura 148: Scenario base

La privacy è una proprietà dell'analisi stessa e non solo dell'output. La privacy è legata alla correlazione tra l'input e l'output. Supponiamo che Alice, insegnante, voglia rilasciare statistiche sui suoi studenti ed, in particolare, voglia rilasciare il voto medio degli studenti del secondo anno. Formalmente, questi dati non hanno alcun impatto sulla privacy degli utenti, ma supponiamo che un utente sia interessato al metodo con cui viene calcolata questa media: se conoscessi il metodo di computazione potrei avere degli indizi sui singoli record. Se il voto medio è associato ai dati del primo studente in ordine alfabetico, potrei inferire qualcosa su di lui.

Una classica intuizione sul concetto di privacy è che un utente si sente sicuro in database D se:

- Sa che i suoi dati non hanno alcun impatto sui risultati rilasciati, ossia se il calcolo su "D senza di me" è uguale al calcolo su "D".
- Sa che le informazioni apprese su un individuo dai risultati pubblicati R non sono altro che le informazioni che possiamo imparare su quell'individuo senza accesso a R.

Questo ragionamento, però, è da applicare a tutti gli utenti presenti nel database, e non al singolo utente, ma se nessuno degli utenti in R computa il risultato, allora i dati che vengono pubblicati non hanno alcuna utilità, ma si tratta di valori generati casualmente. Per quanto riguarda il secondo punto, proviamo a fare un esempio. Viene pubblicata una ricerca che sostiene che fumare e il fumo passivo causano il cancro. Questa ricerca non contiene nessuna informazione sensibile, ma immaginiamo di conoscere Alice, una fumatrice incallita. Sappiamo, inoltre, che Alice è ricoverata in ospedale per una qualche malattia. Queste conoscenze mi possono far inferire che Alice abbia il cancro. Quindi, seppur il risultato sia di per sé innocuo, può permettermi di fare inferenza sugli utenti presenti in esso.

5.2 Definizione

Il concetto alla base della differential privacy è che con o senza l'inclusione di Alice nel database, il suo rischio di privacy non dovrebbe cambiare molto. Quindi, la privacy di una persona è protetta ogni volta che il risultato R non dipende dalle sue informazioni specifiche, o per lo meno non in modo forte. Le interferenze su un individuo da un calcolo differenzialmente privato sono (essenzialmente) limitate a ciò che potrebbe essere dedotto dai dati di tutti gli altri senza che i suoi dati vengano inclusi nel calcolo. Graficamente, consolido due scenari (Figura 149):

- Scenario reale, contenente i dati di tutti gli utenti
- Scenario ideale per l'utente X, ossia il database originale in cui è stato rimosso il record relativo ad X. Fornisce un output ideale, perché non dipende dai valori di X.

Si effettua la stessa analisi su entrambi gli scenari, calcolando due output. La differenza tra i due output deve essere al massimo ε (privacy budget), che rappresenta il livello di privacy desiderato. Questo livello di privacy ha impatto anche sull'accuratezza dei dati rilasciati. ε è un valore piccolo, solitamente compreso tra $[0, 1]$. Più si discosta da 0, più si pttiene un risultato accurato, ma meno protetto.

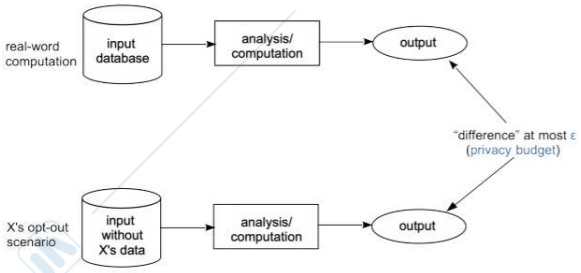


Figura 149: Scenario differential privacy

Come è possibile che i due output si discostino al massimo di un valore ε? Bisogna introdurre del rumore, che maschera le differenze tra il mondo reale e il mondo ideale di un individuo, tenendo presente che ci sono molti mondi ideali, uno per ogni utente rappresentato nel database. Quindi, l'output è un valore approssimato. Se ripeto una stessa analisi più volte, ottengo risultati diversi perché il rumore cambia ad ogni analisi. Se ripeto la stessa analisi più volte, si presenta un'altra problematica, perché è possibile calcolare una media dei risultati ottenuti, che si avvicina sempre di più al valore reale del risultato all'aumentare del numero di ripetizioni della stessa analisi.

La Figura 150 presenta un esempio di applicazione di differential privacy. Le due tabelle rappresentano la stessa analisi in due momenti diversi: al momento x (tabella sopra) abbiamo quattro record, mentre al momento y si è aggiunto il record relativo ad Alice. Gli istogrammi al centro sono istogrammi creati sui valori originali, senza l'introduzione di rumore. Il cambiamento del grafico è subito visibile, quindi possiamo inferire che è stata aggiunta una tupla di un utente con età compresa tra 40 e 45 anni con l'HIV. Quindi, relativamente all'aggiunta di Alice, possiamo notare un cambiamento sul grafico. Questo significa che l'introduzione di Alice ha un impatto sull'output. Se, invece, guardiamo i grafici a destra possiamo vedere che il fatto che Alice sia presente o meno non ha alcun impatto sugli istogrammi, dal momento che già di per sé mascherano i dati con rumore. Le righe nere sono i valori dei vari range a cui è stato aggiunto un fattore rumore.

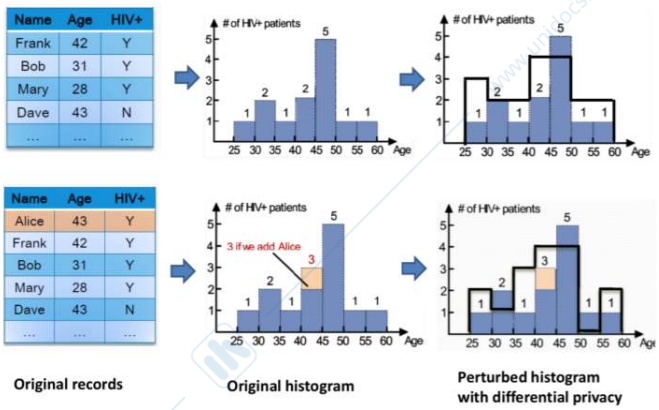


Figura 150: Esempio di differential privacy

Le analisi, effettuate mediante l'utilizzo della differential privacy, aggiungono rumore casuale al risultato. Il rumore maschera le differenze tra il calcolo del mondo reale e lo scenario ideale di ciascun individuo nel database. Il risultato di un'analisi differenzialmente privata non è esatto ma è un'approssimazione. Un'analisi differenzialmente privata può, se eseguita due volte sullo stesso set di dati, restituire risultati diversi e spesso è possibile calcolare i limiti di precisione per l'analisi. Quindi, il risultato che ottengo dalla differential privacy è un risultato offuscato da

rumore casuale, il cui obiettivo è andare a mascherare la differenza di risultato tra il mondo reale e il mondo ideale di un utente. Questo deve essere vero per ogni utente. Se il rumore aggiunto è troppo, riduco l'utilità dei dati; mentre, se il rumore è poco, non è garantita la privacy.

La definizione formale di differential privacy prevede che ci siano due database vicini D e D'. Due database vicini sono due database uguali, ma che differiscono fra loro per al massimo un utente. Un algoritmo A soddisfa la privacy ε-differenziale se per tutte le coppie di database vicini D, D' e per tutti gli output o: $P[A(D) = o] \leq e^{\varepsilon} P[A(D') = o]$. Prevee che un avversario non dovrebbe essere in grado di usare o per distinguere tra D e D'. La formula presentata significa che la probabilità che o sia il risultato dell'algoritmo A applicato a D sia minore o uguale alla probabilità, moltiplicata per e^{ε}, che l'output o sia anche uguale al risultato dell'algoritmo A su D'. Quindi, la differenza tra i due risultati è compresa tra $1 - \varepsilon$ e $1 + \varepsilon$. Riassumendo, significa che il fatto di avere o non avere un utente all'interno del database ha un impatto minimo sul risultato.

Il privacy budget ε determina la quantità di rumore aggiunta al calcolo. La scelta di questo ε è un compromesso tra privacy e accuratezza: più piccolo (più grande) è il ε, maggiore (minore) è il rumore:

- per un ε piccolo si ha più privacy, ma meno utilità
- per un ε grande si ha meno privacy, ma più utilità

Ad esempio:

- Un $\varepsilon = 0$ comporta un'analisi che non può fornire alcun risultato significativo, dal momento che fornisce solo rumore. Si tratta della situazione con privacy perfetta, perché i risultati non rappresentano neanche minimamente gli utenti.
- Un $\varepsilon = 0.1$ fornisce forti garanzie sulla privacy e statistiche utili.
- Un $\varepsilon = 1$ fornisce alta precisione, ma bassa privacy.

La Figura 151 mostra come cambia un grafico rappresentante la distribuzione cumulativa di probabilità dei guadagni delle famiglie nel distretto q. Il grafico in alto a sinistra rappresenta il grafico originale, in cui non è stato aggiunto rumore. Il grafico alla sua sinistra prevede un $\varepsilon = 0.05$, il quale ha un'accuratezza minore rispetto agli altri, ma una privacy maggiore degli utenti rappresentati. Il grafico in basso a sinistra ha un ε pari a 0.01, mentre quello in basso a destra ha $\varepsilon = 0.1$. Si può vedere come all'aumentare di ε, le curve si avvicinino sempre di più alla curva reale, perdendo così di privatezza dei dati, ma andando ad aumentare l'accuratezza di essi.

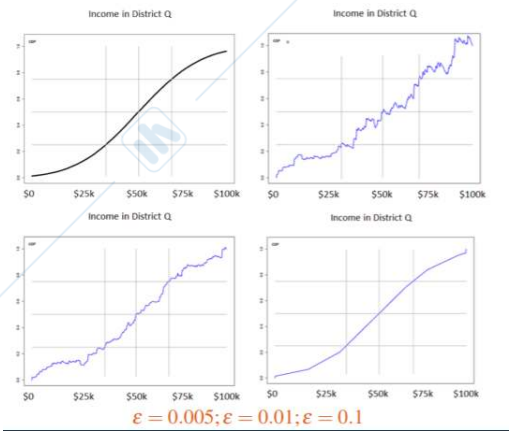


Figura 151: Differential privacy e accuratezza

Per ottenere una privacy differenziale, è necessario calibrare il rumore sull'influenza che un individuo può avere sul risultato. Si introduce quindi il concetto di sensibilità globale (GS), che ha come obiettivo quello di andare a valutare l'influenza massima che un utente può avere sul risultato. La sensibilità globale caratterizza la scala dell'influenza di un individuo (caso peggiore), e stabilisce, quindi, quanto rumore dobbiamo aggiungere. La Figura 152 mostra un esempio di calcolo di sensibilità globale. Nell'output D' manca la tupla dell'individuo di cui sono interessato a mantenere la privacy. L'impatto massimo che questa tupla può avere è 1. Qualunque sia l'operazione di conteggio, l'impatto peggiore è sempre pari ad 1. Il caso migliore è che non rientri nella query di conteggio sottoposta (non ha il diabete) e che, quindi, non abbia impatto sul risultato della query.

Figura 152: Esempio di Global Sensitivity

La Figura 153 mostra due query da vedere come un tutt'uno, quindi una sequenza di richieste. Per entrambi il numero di modifiche nei due conteggi presi singolarmente, nel caso peggiore è 1. Però, è possibile inferire più dati, quindi il GS aumenta. Più è grande il GS maggiore è l'impatto che l'utente può avere sul risultato e quindi maggiore sarà il rumore che sarà necessario inserire.

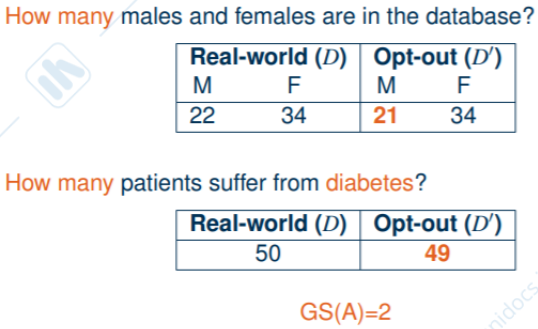


Figura 153: Esempio di Global Sensitivity

Il rumore viene aggiunto usando diverse distribuzioni: la più utilizzata è la distribuzione di Laplace. Il risultato R viene campionato da una distribuzione di Laplace con il risultato reale medio e una scala λ (determinata da ε e la sensibilità globale del calcolo): $R = A(D)+Z$, dove Z rappresenta il fattore di scala (rumore). Z è una variabile casuale estratta dalla distribuzione di Laplace. La Figura 154 mostra un esempio di distribuzione di Laplace, al variare di λ. Vengono quindi utilizzati diversi valori di scala: maggiore è il valore di scala, più la funzione si appiattisce. Come possiamo vedere dalla formula riportata in Figura, il fattore di scala è proporzionale al grado di sensibilità globale: $> GS \implies > \lambda$. Inoltre, *lambda* è inversamente proporzionale a ε: $< \varepsilon \implies > \lambda$, ciò significa che più è grande ε, meno rumore devo aggiungere. Da queste due relazioni deriva che: $\lambda = \frac{GS(A)}{\varepsilon}$.

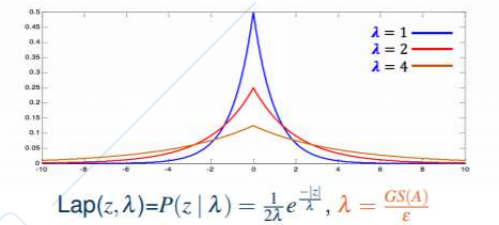


Figura 154: Meccanismo di Laplace con sensibilità

5.3 Proprietà

La privacy differenziale è resiliente alla post-elaborazione, utile quando il risultato è palesemente sbagliato rispetto al dominio di definizione. Quindi, il calcolo di una funzione sul risultato di un calcolo differenziato privato non può renderlo meno differenzialmente privato. Nella Figura 155 viene mostrato un esempio, nel quale il grafico a sinistra mostra l'istogramma reale, relativo al numero di utenti sulla base di range di età. L'istogramma a destra mostra il risultato dell'applicazione della differential privacy. Se venissero rilasciati questi dati, con un valore negativo di utenti per il range di età che va da 80 a 89, chi li consulta capirebbe subito che i dati non sono reali, ma frutto di un offuscamento. Il grafico in basso è frutto di un arrotondamento (post-processing) dei dati risultanti dal processo di differential privacy. Le operazioni post-processing non cambiano il grado di privacy fornito.

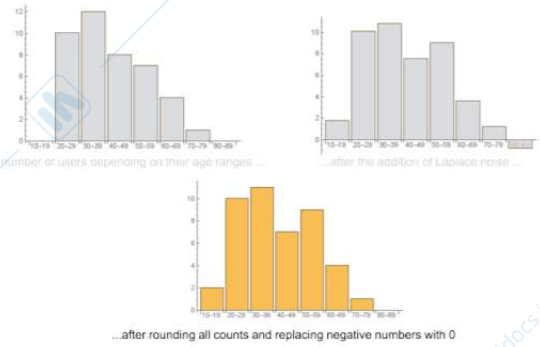


Figura 155: Proprietà di chiusura rispetto alle operazioni di post-processing

La privacy differenziale si combina bene con se stessa. Se parliamo di composizione in differential privacy, dobbiamo specificare fra due tipi di composizioni:

- Composizione sequenziale: sequenza di m calcoli sul database D con risultati sovrapposti. Più analisi sequenziali vengono fatte, più aumentano i rischi di privacy, per via di inferenze. Il caso peggiore, ossia il grado di privacy peggiore, è rappresentato dalla somma dei gradi di privacy delle singole m composizioni: $\varepsilon_1 + \varepsilon_2 + ... + \varepsilon_m$.
- Composizione parallela: sequenza di m calcoli su sottoinsiemi disgiunti di un database D. Il grado di privacy è rappresentato dal valore massimo calcolato sulle singole computazioni: $max(\varepsilon_1, \varepsilon_2, ..., \varepsilon_m$.

La Figura 156 mostra un esempio di composizione sequenziale, in cui viene fissato un privacy budget pari a ε. La query richiede il numero (query di conteggio) di pazienti donne e il numero di pazienti che soffrono di diabete. Le celle possono sovrapporsi (ad es. una femmina che soffre di diabete). Ogni conteggio deve essere rilasciato in modo tale che ε_1 (primo conteggio) + ε_2 (secondo conteggio) sia uguale a ε. Le analisi soddisfano il grado di privacy se il totali dei gradi delle singole computazioni è uguale a ε.

# Females	# Diabetes
34	23

Figura 156: Composizione sequenziale

Invece, la Figura 157 mostra un esempio di composizione parallela, in cui viene sempre fissato un privacy budget pari a ε. La query richiede il conteggio delle persone suddivise per mano con cui scrivono e colore dei capelli. Ogni cella è un insieme disgiunto di individui. Ogni cella può essere rilasciata con la privacy ε-differenziale, ossia ogni composizione può avere un grado di privacy massimo pari a ε, solo così è possibile stabilire se le proprie computazioni non hanno violato il grado di privacy prestabilito. Infatti, dal momento che un utente può rientra solo in un conteggio, si prende il massimo valore delle singole computazioni.

	Redhead	Blond	Brunette
Left-handed	23	35	56
Right-handed	215	360	493

Figura 157: Composizione parallela

La privacy differenziale è stata introdotta per ragionare sulla privacy di un singolo individuo, ma consente anche ragionamenti sulla privacy dei gruppi. Le garanzie sulla privacy che si applicano a un individuo con ε si applicano a un gruppo di dimensioni n con il parametro privacy che diventa $n \cdot \varepsilon$.

5.4 Modelli

Esistono diversi modelli per l'applicazione della differential privacy. Il primo modello distingue due tipologie di modelli:

- Modello non interattivo, il quale calcola le statistiche, che vengono rilasciate e possono essere consultate da tutti gli utenti interessate. Questo modello, però, presuppone la presenza di una parte fidata che computa il risultato a partire dai valori in chiaro presenti nel database. Si tratta di un intermediario tra i valori rilasciati e il database.
- Modello interattivo, in cui le statistiche non sono computate a priori, ma l'utente interagisce direttamente interrogando il database. Prevede, anch'esso, la presenza di una parte fidata, che funga da intermediario. Questa parte fidata, oltre ad interrogare e prelevare i dati dal database, aggiunge del rumore per mascherare i dati degli utenti presenti in esso.

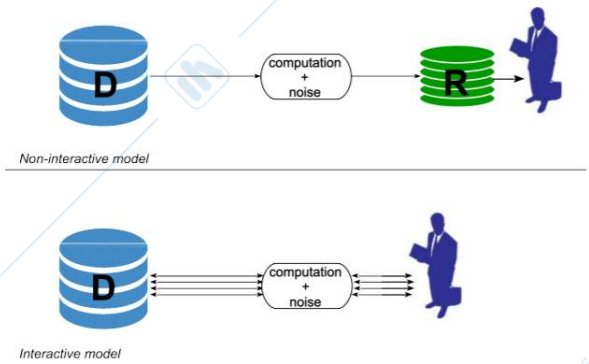


Figura 158: Modello interattivo vs Modello non-interattivo

Un secondo modello consiste nell'applicazione della differential privacy in due modalità differenti:

- Global differential privacy, la quale prevede una parte fidata che ha accesso ai dati in chiaro di diversi individui e che calcola computazione, aggiungendo rumore.
- Local differential privacy, nella quale non esiste una parte fidata, quindi ciascun individuo è responsabile dell'offuscamento dei propri dati. L'intermediario tra gli utenti e l'utente finale che interroga i dati colleziona i dati (con rumore) ed esegue la computazione, non aggiungendo rumore. Quindi, nella local differential privacy, ogni utente esegue un algoritmo privato differenziale sui propri dati. Una parte esterna (non necessariamente trusted) combina tutti i dati (disturbati) ricevuti dagli utenti per ottenere un risultato finale. Il rumore può essere cancellato o sottratto

La global differential privacy è più accurata, perché le analisi vengono effettuate sui dati reali senza rumore. Allo stesso tempo, però, la local differential privacy presenta il vantaggio di non dover possedere una parte fidata. Nella local differential privacy il rumore aggiunto è maggiore (anche se non in maniera così marcata) rispetto alla global differential privacy.

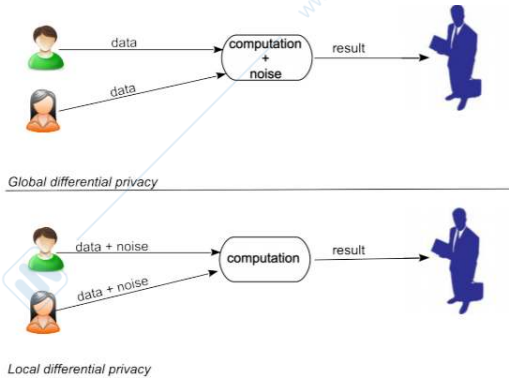


Figura 159: Global differential privacy vs Local differential privacy

Un algoritmo randomizzato A soddisfa la privacy differenziale ε-locale se e solo se per tutti gli input x, x' e output o di A: $P[A(x) = o] \leq e^{\varepsilon} P[A(x') = o]$. Ciò signica che qualsiasi output non dovrebbe dipendere dal segreto dell'utente. Nel caso di global differential privacy, ossia quella di cui abbiamo parlato fino ad ora, non si parla mai di singolo utente, ma si parla a livello di database D. Quindi, la differenza principale tra questa definizione e la definizione standard di privacy differenziale è che nella privacy differenziale le probabilità sono degli output di un algoritmo che prende i dati di tutti gli utenti e qui è su un algoritmo che accetta i dati di un singolo utente. La local differential privacy è sempre più utilizzata (Google, Apple, etc.). La local differential privacy è una definizione introdotta nel 2011, più recente rispetto alla global differential privacy sviluppata nel 2006.

5.5 Applicazioni nel mondo reale

Nel 2008 il censimento degli Stati Uniti ha distribuito OnTheMap, un'applicazione basata sul Web che mostra dove sono impiegati i lavoratori e dove vivono. Quindi, dato un blocco, mostrare i blocchi di residenza in cui vivono i lavoratori del blocco in analisi. Questa piattaforma è basata su una variante di privacy ε-differenziale, chiamata privacy differenziale approssimativa ((ε, δ)-privacy differenziale), dove:

- ε è il budget per la privacy.
- δ è correlato alla confidenza $(1 - \delta)$, per cui il risultato soddisfa la privacy ε-differenziale.

La Figura 160 mostra le differenze tra la privacy differenziale ε e (ε, δ). Il grafico a sinistra riporta tre linee: la linea tratteggiata rossa (B) è il risultato offuscato mediante ε-differential privacy. e^{ε} è il parametro di ε-differential privacy. Il grafico di destra riporta anch'esso tre linee: la linea tratteggiata rossa (B) è il risultato offuscato di a mediante la (ε, δ)-differential privacy. Non soddisfa la ε-differential privacy, perché B si trova sopra $e^{\varepsilon} \cdot A$. Più δ è grande, maggiore sarà il discostamento.

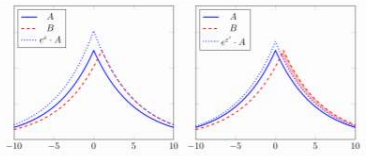


Figura 160: La privacy in pratica - OnTheMap

OnTheMap utilizza come valori:

- $\varepsilon = 8.99$
- $\delta = 0.000001$

Esperimenti interni hanno confermato che i microdati riservati del censimento del 2010 possono essere ricostruiti in modo abbastanza accurato. L'Ufficio censimento degli Stati Uniti ha adottato un nuovo meccanismo differenzialmente privato per il controllo della divulgazione statistica nel censimento del 2020. Non è chiaro esattamente come imposteranno ε, un comitato politico (il comitato politico esecutivo per la gestione dei dati, non il personale tecnico) deciderà sul valore di ε.

La privacy differenziale basata sul lancio di monete è ampiamente diffusa:

- RAPPOR, in cui Google ha utilizzato la privacy differenziale locale per raccogliere dati dagli utenti, come altri processi in esecuzione e le home page di Chrome
- Private Count Mean Sketch (e varianze) del conteggio privato in cui Apple ha utilizzato la privacy differenziale locale per raccogliere dati sull'utilizzo delle emoji, sull'utilizzo delle parole e altre informazioni dagli utenti iPhone.

Viene applicata la location differential privacy, perché vengono raccolti dati dagli individui. Tutte le distribuzioni si basano su una risposta randomizzata, che permette di raccogliere informazioni statistiche su informazioni sensibili (ad esempio, mediante sondaggi). La risposta randomizzata è un metodo di raccolta dati che garantisce che venga raccolta solo una versione rumorosa del set di dati vero. La Figura 161 mostra come viene calcolato il dato quando devo rispondere a una domanda sensibile. Si suppone di avere una moneta onesta. Quando si deve registrare una risposta, si simula il lancio di una moneta: se esce testa viene registrata la risposta veritiera, altrimenti si rilancia la moneta. Se dal secondo lancio esce testa si registra la risposta veritiera, altrimenti una bugia. Le probabilità con cui viene registrata una risposta vera o falsa sono le seguenti:

- $P(\text{true answer}) = 0.75 = 0.5 + (0.5 \times 0.5)$
- $P(\text{lie}) = 0.25 = 0.5 \times 0.5$

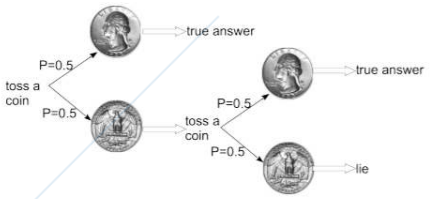


Figura 161: Randomized response

I ricercatori di Google hanno creato una tecnologia chiamata RAPPOR (Randomized Aggregatable Privacy-Preserving Response Ordinal Response) che utilizza una risposta randomizzata per studiare i dati degli utenti senza compromettere la privacy di ogni singolo utente. Supponiamo di voler raccogliere un determinato valore da tutti gli utenti di un software (ad esempio l'età dell'utente). Quindi, ogni utente ha un valore v su una serie molto ampia di possibilità. La soluzione Rappor si basa su:

- Un bloom filter
- Due livelli di randomized response: permanente e istantaneo. Sono necessari due livelli perché Rappor deve essere robusto sia rispetto agli attacchi one-time e agli attacchi longitudinali.

La Figura 162 mostra il funzionamento della randomized response per Rappor. Le risposte degli utenti possono essere riassunti in una stringa di k bit. Ciascun bit è una risposta randomizzata riguardante una proprietà di un utente. Quello che bisogna proteggere è la stringa di bit. Si sviluppa in tre passaggi:

1. Compressione: utilizzare le funzioni hash h per eseguire l'hash della stringa di input nel vettore k-bit (filtro Bloom). Un filtro Bloom è una struttura dati probabilistica per verificare se un elemento appartiene a un insieme. É possibile che si generino falsi positivi, ma non falsi negativi. Nell'esempio in Figura, vengono applicate 2 funzioni di hash su una stringa da 10 bit.

2. Risposta randomizzata permanente: da B (bloom filter) a B' (fake bloom filter) la risposta randomizzata permanente viene creata con il parametro di probabilità (sintonizzabile dall'utente) f. B' viene memorizzato e verrà utilizzato per tutti i rapporti futuri. Quindi se l'utente sceglie lo stesso v non è necessario ricalcolare B'. Il fake bloom filter viene creato sulla base della seguente funzione:

$$B'_i = \begin{cases} 1 & \text{with probability } \frac{1}{2}f \\ 0 & \text{with probability } \frac{1}{2}f \\ B_i & \text{with probability } 1-f \end{cases}$$

3. Risposta randomizzata istantanea: invia un rapporto al server di dimensione k bit generato da B', calcolato al punto precedente, in cui
   - Viene capovolto il valore 1 con probabilità $1-q$
   - Viene capovolto il valore 0 con probabilità p

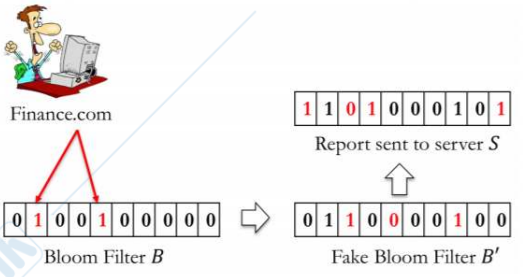


Figura 162: Funzionamento di Rappor

Apple raccoglie dati dagli utenti iOS e OS X:

- Emoji popolari: (cuore) (risata) (sorriso) (pianto) (faccia triste)
- Parole "nuove": bruh, hun, bae, tryna, despacito, mayweather
- Su quali siti Web attivare il muto e su quali riprodurre automaticamente l'audio

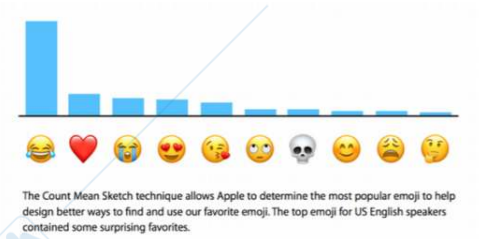


Figura 163: Funzionamento di local differential privacy su Apple

É importante capire quale sia il privacy budget ε, perché ad ε è correlato il grado di privacy relativo agli utenti. I valori di ε per Google e Apple sono:

- Google
  - $\varepsilon = 2$ per dati particolari caricati
  - $\varepsilon = 8 - 9$ è un limite superiore per la durata dell'utente
- Apple
  - $\varepsilon = 6$ per macOS
  - $\varepsilon = 14$ per iOS 10
  - $\varepsilon = 43$ per la versione beta iOS 11 (versione sconosciuta)

Si dice che la privacy sia garantita da differential privacy, ma con valori di ε così granfi non è supportata. Di seguito un commento di uno degli inventori della differential privacy: "Supponiamo che qualcuno abbia detto all'app di salute del proprio telefono di avere una condizione medica e che il loro telefono carichi i dati su base giornaliera, usando la privacy differenziale con un epsilon di 14. Dopo un caricamento offuscato con un'iniezione di dati casuali, gli analisti di dati dell'azienda sarebbero in grado di capire con certezza del 50 percento se la persona aveva la condizione. Dopo due giorni di caricamenti, gli analisti avrebbero saputo di quella condizione medica con certezza praticamente del 100 percento."

5.6 Problemi nell'applicazione della differential privacy

La privacy differenziale funziona bene (la presenza/assenza di un singolo record può modificare leggermente il risultato) nei casi di conteggio e calcoli dell'istogramma. Quindi, fino a che si tratta di computazioni semplici funziona bene, ma già con computazioni di somma sorgono i primi problemi. L'applicazione della privacy differenziale può essere un problema in casi di somme, ad esempio: qual è il reddito totale guadagnato da uomini contro donne? Se ci fosse un singolo reddito molto elevato, bisognerebbe introdurre molto rumore per questo individuo nel caso peggiore. La global sensitivity è alta nel caso di utenti molto significativi rispetto al risultato. L'introduzione di molto rumore comporta una perdita in termini di utilità. Un altro problema si pone sulla scelta di ε, che dovrebbe essere un parametro piccolo tipicamente compreso tra $[0, 1]$. Ma, cosa succede quando il budget per la privacy è esaurito? Infatti, se continuo a rispondere a query (composizione sequenziale) il grado di privacy potrebbe saturarsi. Se si continua a rispondere ad ulteriori richieste, il grado di privacy è violato, ma, se non continuo, l'utilità dei dati è nulla.

6 Blockchain

6.1 Introduzione

La blockchain (letteralmente "catena di blocchi") è una struttura dati condivisa e immutabile. È definita come un registro digitale le cui voci sono raggruppate in blocchi, concatenati in ordine cronologico, e la cui integrità è garantita dall'uso della crittografia. Sebbene la sua dimensione sia destinata a crescere nel tempo, è immutabile in quanto, di norma, il suo contenuto una volta scritto non è più né modificabile né eliminabile, a meno di non invalidare l'intera struttura. La prima blockchain fu introdotta, nel 2008, ad opera di Satoshi Nakamoto (pseudonimo di un autore la cui identità è tuttora sconosciuta), e implementata l'anno seguente, con l'obiettivo di fungere da libro mastro (registro di tutte le transazioni) della nascente valuta digitale Bitcoin. La tecnologia blockchain ha un grande potenziale per trasformare i modelli operativi aziendali nel lungo periodo. L'uso della blockchain promette di portare significativi miglioramenti alle catene di fornitura globali, alle transazioni finanziarie, ai beni contabili e ai social network distribuiti. Questa nuova tecnologia può essere integrata nelle aree più disparate e i suoi protocolli facilitano alle aziende l'uso di nuovi metodi per processare e gestire le transazioni digitali.

Come dice già il nome, una blockchain è una catena di blocchi che contiene dati. Lo scopo è quello di avere un timestamp dei documenti digitali in modo che non sia possibile retrodatarli o manometterli. La blockchain viene utilizzata per il trasferimento sicuro di articoli (ad es. Denaro, proprietà, contratti, ma in realtà può essere qualunque cosa) senza richiedere un intermediario di terze parti (ad es. Banca o governo). Una volta che un dato viene registrato all'interno di una blockchain, è molto difficile e oneroso modificarlo. Un errore che spesso si fa è associare il termine blockchain a bitcoin, ma in realtà le blockchain sono un concetto molto più generale.

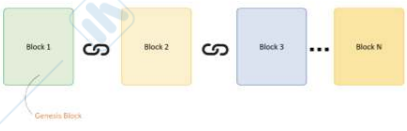


Figura 164: Architettura blockchain

Una blockchain è un registro digitale aperto e distribuito, in grado di memorizzare record di dati (solitamente, denominati "transazioni") in modo sicuro, verificabile e permanente. Una volta scritti, i dati in un blocco non possono essere retroattivamente alterati senza che vengano modificati tutti i blocchi successivi ad esso e ciò, per la natura del protocollo e dello schema di validazione, necessiterebbe del consenso della maggioranza della rete. La blockchain è quindi

rappresentabile come una lista, in continua crescita, di blocchi collegati tra loro e resi sicuri mediante l'uso della crittografia. Il primo blocco della catena viene chiamato genesis block. I dati memorizzati in un blocco dipendono dal tipo di blockchain. Ad esempio, la Figura 165 mostra un esempio di un blocco di una blockchain per gestire i bitcoin. Viene indicato il mittente, il destinatario e la somma da trasferire.

Figura 165: Architettura blockchain

A livello astratto, il blocco è composto da due parti:

- Header: contiene i dati sul blocco:
  - Versione (4 Byte): numero di versione del software utilizzato. Indica implicitamente anche le regole di validazione che il blocco segue.
  - hash del blocco precedente calcolato sull'header tramite l'algoritmo SHA256. Campo a 32 byte che contiene una sorta di puntatore al blocco precedente.
  - Timestamp (4 byte): indica un valore di tempo calcolato secondo le epoche di Unix. Nei sistemi operativi Unix e Unix-like il tempo viene rappresentato come offset in secondi rispetto alla mezzanotte (UTC) del 1◦ gennaio 1970 (detta epoca).
  - Merkle root: hash della radice dell'albero merkle del blocco. Il merkle tree viene costruito sulla base delle transazioni del blocco.
  - Difficult target (4 byte): soglia per cui l'hash dell'header del blocco deve essere minore.
  - Nounce (8 byte): contatore utilizzato per l'algoritmo proof-of-work. numero arbitrario che i miners possono andare a modificare.
- Elenco delle transazioni

Il nounce e il difficult target sono le informazioni necessarie per il funzionamento dell'algoritmo proof-of-work.

La Figura 166 mostra il funzionamento ad alto livello di una blockchain. Supponiamo che l'utente A voglia inviare denaro all'utente B. La transazione è rappresentata come un blocco. Il blocco viene trasmesso a tutti i nodi della rete, all'interno della quale sufficienti miners devono approvare la transazione. A quel punto, la transazione è aggiunta alla blockchain e diventa non modificabile. Infine B riceve i soldi da A.

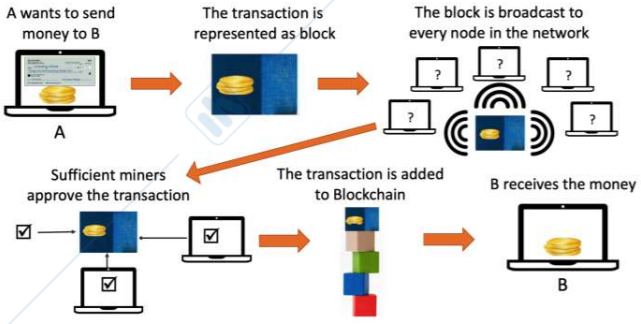


Figura 166: Funzionamento di una blockchain

Una tecnologie alla base di una blockchain è la funzione di hash, che viene usata per calcolare l'hash dell'header o per la costruzione del merkle tree. Si tratta di un algoritmo matematico che

mappa dei dati di lunghezza arbitraria (messaggio) in una stringa binaria di dimensione fissa chiamata valore di hash, ma spesso viene indicata anche con il termine inglese message digest (o semplicemente digest). Tale funzione di hash è progettata per essere unidirezionale (one-way), ovvero una funzione difficile da invertire: l'unico modo per ricreare i dati di input dall'output di una funzione di hash ideale è quello di tentare una ricerca di forza-bruta di possibili input per vedere se vi è corrispondenza (match). La funzione crittografica di hash ideale deve avere alcune proprietà fondamentali:

- deve identificare univocamente il messaggio, non è possibile che due messaggi differenti, pur essendo simili, abbiano lo stesso valore di hash;
- deve essere deterministico, in modo che lo stesso messaggio si traduca sempre nello stesso hash;
- deve essere semplice e veloce calcolare un valore hash da un qualunque tipo di dato;
- deve essere molto difficile o quasi impossibile generare un messaggio dal suo valore hash se non provando tutti i messaggi possibili.

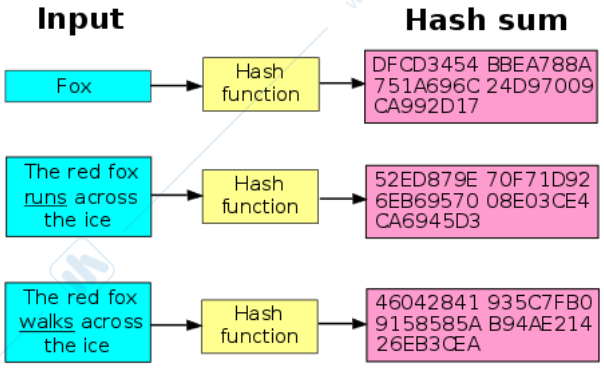


Figura 167: Funzionamento di una funzione di hash

Un altro aspetto fondamentale è che le blockchain sono immutabili. Infatti, ogni blocco della catena contiene l'hash di quello precedente. Non è possibile modificare alcun blocco senza cambiare l'intera catena. La catena funziona come una struttura dati immutabile. Supponiamo di avere la blockchain in Figura 168, un utente malintenzionato anonimo rimuove, aggiunge o modifica qualsiasi transazione nel primo blocco. La modifica modifica l'header del primo blocco, però, non può avvenire in modo trasparente senza la modifica anche del contenuto dell'hash del blocco precedente nel secondo blocco. Di conseguenza modificando il secondo blocco, dovremmo modificare l'hash dell'header del blocco precedente anche nel terzo blocco, e così via. Questo attacco che prevede la modifica di una transazione è veramente costoso, perché significherebbe cambiare tutti i blocchi della catena. Infatti, l'errore si propagherà su ogni blocco della catena dopo il blocco sotto attacco.

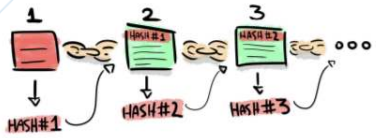


Figura 168: Struttura dati immutabile

Una blockchain è un registro digitale aperto e distribuito, in grado di memorizzare record di dati (solitamente, denominati "transazioni") in modo sicuro, verificabile e permanente. I

dati Blockchain sono distribuiti tra tutti gli utenti (rete peer-peer distribuita).La blockchain decentrata sfrutta il passaggio di messaggi ad-hoc e un networking distribuito per fare in modo di memorizzare i dati su tutta la sua rete ed evitare di avere un single point of failure in modo che non esista una centralizzazione che i cracker potrebbero sfruttare per abbattere l'intero sistema. Ogni nodo o miner nel sistema decentrato ha una copia della blockchain: difatti la qualità dei dati è mantenuta grazie a una massiva replicazione del database. Non esiste nessuna copia ufficiale centralizzata e nessun utente è più credibile di altri, tutti sono allo stesso livello di credenziali. I nodi miner, ovvero gli utenti, validano le nuove transazioni e le aggiungono al blocco che stanno costruendo dopo aver verificato l'intera blockchain. Una volta completato il blocco, lo trasmettono agli altri nodi della rete. La Figura 169 mostra un esempio, nel quale ognuno degli utenti detiene la stessa copia completa della blockchain.

Figura 169: Rete P2P distribuita

Le transazioni vengono firmate da chi le produce per garantire l'integrità dei dati. Si utilizza una crittografia asimettrica. Il mittente utilizza la sua chiave privata per firmare i dati, i quali vengono decrittografati mediante la chiave pubblica del mittente.

Un albero di hash o Merkle è un albero in cui ogni nodo foglia è etichettato con l'hash crittografico di un blocco di dati, e ogni nodo non foglia è etichettato con l'hash crittografico delle etichette dei suoi nodi figlio . Gli hash tree consentono una verifica efficiente e sicura dei contenuti di grandi strutture di dati. La costruzione del merkle tree avviene usando una funzione di hash per autenticare un insieme di messaggi mediante un numero logaritmico di valori. Da notare che se cambia una transazione nel blocco, si potrebbe non essere più in grado di ricostruire la radice dell'albero. La Figura 170 mostra un esempio di Merkle Tree.

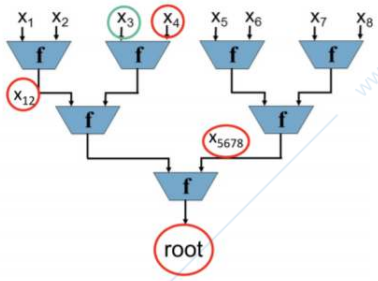


Figura 170: Merkle Tree

Esistono diversi tipi di blockchain:

- Pubblica: chiunque può partecipare, senza autorizzazione e chiunque può verificare e aggiungere un blocco di transazioni nella blockchain.
- Privata: la partecipazione è regolata da regole di governance e solo persone specifiche di un'organizzazione possono verificare e aggiungere blocchi di transazioni; i blocchi di transazione sono generalmente disponibili a chiunque.

- Consorzio: solo un gruppo di organizzazioni (ad es. Banche) può partecipare e può verificare/aggiungere un blocco di transazioni e la struttura dati può essere aperta o limitata a gruppi selezionati.

6.2 Blockchain pubbliche

Il grande vantaggio della blockchain aperta, permissionless o pubblica, è che non è necessaria nessuna protezione verso utenti malintenzionati e non si necessita di nessun controllo degli accessi. Questo significa che è possibile aggiungere le applicazioni alla rete senza l'approvazione di nessuno, usando la blockchain come un livello di trasporto. Bitcoin e le altre criptomonete rendono sicure le loro blockchain richiedendo una prova di lavoro (proof-of-work) per poter scrivere su di esse. Le blockchain pubbliche si basano su due concetti estremamente diversi tra loro:

- Convalida: consiste nel verificare che le transazioni siano legali (ad esempio, non dannose, doppie spese). Con doppie spese si intende che un mittente utilizza il suo bitcoin, o altra valuta, per due transazioni differenti.
- Consenso: tutti i partecipanti alla rete raggiungono un accordo sullo stato della struttura dati e comporta la determinazione dell'ordine degli eventi nella blockchain.

I miners sono coloro che si occupano di queste operazioni. I miners convalidano ogni transazione. Costruire e archiviano tutti i blocchi, raggiungendo un consenso su quali blocchi includere nella blockchain. l'obiettivo dei miner è quello di estendere la catena in lunghezza e guadagnano ricompense per questo lavoro. In particolare, i compiti di cui si occupa un miner sono:

- Ascoltare le transazioni che vengono inoltrate sulla rete di cui fa parte e convalidarle
- Mantenere la catena di blocchi e rimanere in attesa di nuovi blocchi:
  - chiedere agli altri nodi tutti i blocchi storici che fanno già parte della blockchain prima del join
  - ascoltare i nuovi blocchi che vengono trasmessi alla rete
  - convalidare ciascun blocco ricevuto, convalidando ogni transazione nel blocco e verificando che il blocco contenga un nonce valido
- Assemblare un blocco candidato che contiene un gruppo di transazioni di cui il minatore sente parlare e che estende l'ultimo blocco di cui il minatore è a conoscenza. Il miner deve assicurarsi che ogni transazione inclusa nel blocco sia valida
- Trova un nonce che renda valido il blocco candidato (questo passaggio richiede più lavoro). Il nonce è una sorta di numero magico che sblocca il blocco.
- Deve sperare che il blocco candidato venga accettato, ovvero che altri miners accettino il blocco e inizino a scavare sopra di esso, anziché sui blocchi di alcuni concorrenti
- Ottenere profitto (se tutti gli altri minatori accettano il blocco)

6.2.1 Come vengono validate le transazioni?

Viene creata una nuova transazione e quindi trasmessa a tutta la rete. I miners prendono una serie di transazioni, confermano che sono "legittime" e le inseriscono in un blocco. L'intero processo di validazione dei blocchi è chiamato mining. Sussistono dei problemi da queste operazioni:

- In che modo i minatori verificano una transazione?

- Ogni minatore sta costruendo il proprio blocco: come possiamo concordare un unico registro comune?

Per verificare una transazione utilizzo la crittografia asimmetrica, in particolare le firme digitali. Ogni transazione viene inviata insieme alla firma del mittente. Ogni nodo nella rete verifica il mittente di una transazione tramite la sua firma digitale. La Figura 171 mostra questo procedimento: Alice crea una transazione (Paga 100$), dopo di che firma questa transazione con la sua chiave privata. La firma viene allegata alla transazione: la presenza della firma di Alice permette la non ripudiazione da parte di Alice. Andare a verificare una transazione significa utilizzare la chiave pubblica del mittente. Quindi, l'utente Bob deve utilizzare la chiave pubblica di Alice per verificare la corrispondenza.

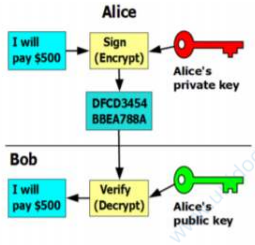


Figura 171: Verifica di una transazione

Il miner compila un insieme di transazioni valide da un insieme di transazioni in sospeso in un albero Merkle. Quindi, crea un blocco unendo diverse transazioni pendenti. In poche parole, un nodo, dopo aver verificato l'intera blockchain, raccoglie e colleziona le nuove transazioni generate ancora non validate e suggerisce alla rete quale dovrebbe essere il nuovo blocco. Il numero di transazioni dipende dalla dimensione totale del blocco. Il blocco creato ha un'intestazione che punta al blocco precedente. Questa operazione di raccolta delle transazioni pendenti viene fatta da più miner, dal momento che è proprio la creazione di nuovi blocchi che permette di produrre profitto. Cosa succede se più blocchi fanno contemporaneamente questa operazione? Supponiamo di concentrarci su due miner: Alice e Bob. Il miner Alice crea un blocco con transazioni valide formato da T_A, T_B, T_C, mentre il miner Bob crea un blocco con transazioni valide T_Z, T_Y, T_A. Le opzioni possibili di aggiunta del blocco sono:

- Viene aggiunto il blocco di Alice
- Viene aggiunto il blocco di Bob
- Vengono aggiunti entrmabi

Ogni qual volta un blocco viene inserito nella blockchain il miner che ha creato il blocco viene ricompensato per il lavoro fatto. Ovviamente, per i miner sarebbe più conveniente creare due blocchi, dal momento che verrebbero ricompensati entrambi, ma ciò non è possibile. Infatti, i due blocchi condividono la transazione T_A, ma se questa fosse un pagamento e venisse aggiunta due volte, l'utente dovrebbe pagare due volte il destinatario della transizione. Quindi, come facciamo a stabilire quale dei due blocchi viene aggiunto alla catena? Si applica l'algoritmo di consenso. La figura 172 mostra il funzionamento generale del protocollo di consenso. Deve essere aggiunto un protocollo alla blockchain contenente i tre blocchi a sinistra. Ogni utente genera un blocco e viene instaurato un protocollo di consenso tra i nodi della rete. Verrà, quindi, scelto uno dei blocchi proposti e verrà aggiunto alla blockchain. É possibile selezionare qualsiasi blocco valido, anche se proposto da un singolo nodo.

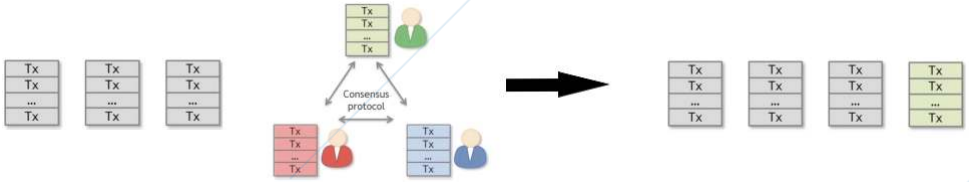


Figura 172: Protocollo di consenso

Il protocollo di consenso è un problema complesso per i seguetni motivi:

- I nodi potrebbero bloccarsi/crashare
- I nodi potrebbero essere dannosi, cioè non operare per fini benevoli
- La rete è imperfetta:
  - non tutte le coppie di nodi sono collegati fra loro
  - guasti nella rete
  - latenza: nessuna nozione di tempo globale. Il protocollo di consenso, quindi, non può lavorare sul primo messaggio arrivato in termini di tempo, perché non esiste una nozione di tempo globale. Diversi protocolli di consenso non possono essere utilizzati proprio per questo motivo.

Come dicevamo alcuni protocolli di consenso non possono essere applicati, tra cui:

- Il problema dei generali bizantini è un problema informatico su come raggiungere consenso in situazioni in cui è possibile la presenza di errori. Il problema consiste nel trovare un accordo, comunicando solo tramite messaggi, tra componenti diversi nel caso in cui siano presenti informazioni discordanti. Informalmente il problema è esemplificato dalla situazione in cui tre o più generali bizantini debbano decidere se attaccare o ritirarsi dato un ordine da un comandante superiore. Uno o più dei generali potrebbero essere dei traditori con l'intenzione di confondere gli altri, quindi potrebbe verificarsi il caso in cui il comandante dia ordini discordanti ai generali oppure il caso in cui uno dei generali comunichi ai propri colleghi un ordine differente da quello impartito dal comandante. La soluzione al problema permette ai generali leali di evitare queste trappole. É dimostrato che non esiste soluzione al problema se il numero di processi non corretti è maggiore o uguale a un terzo del numero totale di processi.
- Fischer-Lynch-Peterson è un problema che prende il nome dai tre autori che lo hanno sviluppato. Se si considera un contesto in cui i nodi agiscono in modo deterministico è impossibile trovare un consenso anche solo con un nodo malizioso/dannoso.

Le blockchain però introducono dei concetti differenti che permettono di scegliere determinati protocolli di consenso:

- Introduce incentivi
- Abbraccia la casualità, ossia:
  - elimina la nozione di un punto finale specifico
  - il consenso si verifica su scale temporali lunghe (circa 1 ora)

  Nonostante la scala temporale sia lunga, non c'è alcuna garanzia deterministica, ma solo una garanzia probabilistica che il blocco interessato sia nella catena.

6.3 Protocollo di consenso nell'ambito delle blockchain

Prima di entrare nel dettaglio del funzionamento, vediamo quali sono le caratteristiche del sistema in analisi. Il primo aspetto legato al funzionamento è che i nodi non hanno identità. Esistono 2 motivi per cui i nodi non hanno identità:

- L'identità è difficile in un sistema P2P, perché non esiste un identità centralizzata in grado di assegnare un'identità ai componenti di una rete. L'attacco di Sybil è un attacco informatico dove i sistemi di reputazione sono sovvertiti falsificando le identità di una persona in una rete p2p.
- Inoltre, la pseudonimia è l'obiettivo tipico delle blockchain.

Per ovviare al problema della mancanza di identità, si fanno delle assunzioni. Un'assunzione più debole prevede che esista la possibilità di selezionare nodo casuale per l'assegnazione dei "meriti". Quando il monitoraggio e la verifica delle identità sono difficili, vengono forniti token, ticket e così via. Si tratta di un algoritmo smart, in cui a tanti nodi sybil viene assegnato un token. Dopo di che viene selezionato un ID casuale e scelto il nodo corrispondente, sulla base di una condizione random. La possibilità di selezionare casualmente un nodo rende possibile il concetto di consenso implicito. In ogni round (blocco aggiunto alla blockchain), viene scelto un nodo casuale. Il nodo selezionato propone il blocco successivo nella catena, non c'è alcun consenso, né meccanismo di voting. Gli altri nodi accettano/rifiutano implicitamente questo blocco: estendendolo (accetta) o ignorandolo ed estendendo la catena dal blocco precedente (rifiuto). Quindi, gli altri nodi decidono implicitamente se accettarlo, andando ad estendere la blockchain, includendo il blocco, oppure ignorandolo. L'algoritmo di consenso può essere semplificato nei seguenti passaggi:

1. Le nuove transazioni vengono trasmesse a tutti i nodi
2. Ogni nodo raccoglie nuove transazioni in un blocco
3. In ogni round, un nodo casuale arriva a trasmettere il suo blocco
4. Altri nodi accettano il blocco solo se tutte le transazioni in esso contenute sono valide
5. I nodi esprimono la loro accettazione del blocco includendone l'hash nel blocco successivo che creano. L'hash dell'header del blocco precedente è sempre incluso nel blocco successivo, ed è questo che permette di capire se un blocco è stato incluso o escluso dalla catena.

Supponiamo che Alice sia maliziosa e voglia sovvertire questo algoritmo. Esistono diversi attacchi che può tentare di attuare. Alice può rubare dei soldi e utilizzarli per una transazione, però, per fare questo, dovrebbe essere in grado di generare una transazione in cui spende quel determinato denaro (bitcoin), ma questo è impossibile, a meno che Alice non sia in grado di replicare la firma del proprietario originale del bitcoin. Quindi, Alice dovrebbe riuscire a rubare l'identità del destinatario per rubare i bitcoin di sua proprietà. Un altro attacco che Alice potrebbe voler attuare è un denial of service: Alice vuole mettere in crisi Bob, allora decide di non includere mai le transazioni di Bob nei suoi blocchi (negare il servizio a Bob). Anche se le transazioni di Bob non rientrano mai nei blocchi di Alice, Bob aspetta che un altro miner, onesto, le includa nei suoi blocchi. Il terzo attacco prevede un'assunzione: Alice paga Bob, il quale fornisce servizi sotto pagamento di bitcoin; a seguito del pagamento, si può effettuare il download del prodotto. La Figura 173 mostra il funzionamento di questo attacco. Alice crea una transazione finanziaria e fa il broadcast in rete di essa. Supponiamo che un nodo onesto includa la transazione nel suo blocco. La transazione, firmata da Alice, prevede un pagamento alla chiave pubblica di Bob. La transazione contiene un hash, che rappresenta un puntatore all'output di una transazione ricevuta da Alice, il cui output viene speso in questa specifica transazione. Alice

(utente malizioso) crea un blocco contenente una transazione, firmata da Alice, e direzionata all'entità A', ossia un'entità sempre controllata da Alice, il quale punta sempre allo stesso output della transazione precedente. Al round successivo, il nodo che viene selezionato è proprio quello di Alice, la quale propone un blocco che ignori il blocco in cui avviene il pagamento a Bob, ossia ignora il blocco contenente la transazione direzionata a Bob. Dal punto di vista tecnologico, i blocchi sono identici, ossia sono entrambi validi, mentre sono diversi dal punto di vista morale. Come facciamo a sapere concretamente se l'attacco della doppia spesa è andato a buon fine? L'attacco va a buon fine, a seconda del blocco che viene accettato e, quindi, andrà a far parte effettivamente della blockcahin:

- Se i blocchi successivi vengono aggiunti a seguito del nodo di Alice, l'attacco è andato a buon fine
- Altrimenti, l'attacco fallisce

I nodi onesti applicano come politica l'estensione del ramo valido più lungo, ma se hanno entrambi la stessa lunghezza scelgono chi includere e chi no.

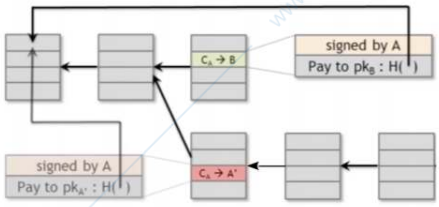


Figura 173: Cosa può fare un nodo dannoso?

La Figura 174 mostra l'attacco dal punto di vista di Bob. Bob è in ascolto sulla rete, appena vede la transazione potrebbe completare il processo di vendita e consentire immediatamente ad Alice di eseguire il download del prodotto, senza aspettare che il blocco faccia effettivamente parte della chain. Questo comportamento si chiama 0 confermations. Questo comportamento aumenta la probabilità di attuare un attacco di double spending con successo. Alice potrebbe riuscire a portare a termine l'attacco anche senza possedere un nodo malizioso. Infatti, a Bob basta vedere la transazione sulla rete, per permettere il download. Per ovviare, bob dovrebbe aspettare che ci sia almeno 1 conferma, ossia che ci sia un nodo contenente la transazione. Ma è sufficiente? I venditori cauti aspetterebbero di avere altre conferme. Se Bob vede che Alice lancia un attacco di double spending, Bob potrebbe realizzare il fatto che il blocco contenente la sua transazione potrebbe rimanere orfano (senza successori) e decidere di bloccare la possibilità di download. Se, invece, Bob vede che il blocco benevolo ottiene dei consensi, ci sarà sempre una maggiore confidenza che l'attacco non sia andato a buon fine. Questo perché i nodi onesti estendono sempre la catena più lunga. La probabilità di successo di un attacco decresce esponenzialmente all'aumento della confidenza al nodo benevolo. Se la transazione benevola ha ricevuto k cofnerme, la possibilità di un attacco double spending decresce di una funzione k. Qual è un buon trade-off tra il tempo attesso e la probabilità che la transazione non finisca nella blockchain? L'assunzione di basse è che almeno la metà dei nodi siano onesti. L'euristica comune prevede che si aspettino 6 conferme.

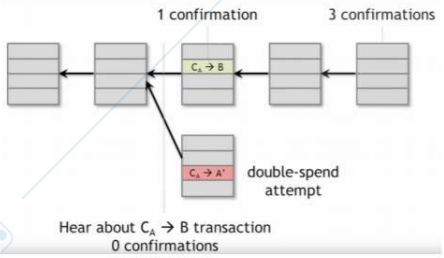


Figura 174: Dal punto di vista di Bob

Come possiamo incentivare i nodi a comportarsi onestamente? Possiamo penalizzare il nodo che ha creato il blocco malevolo? No, perché è difficile conoscere dal punto di vista morale quale sia legittima, dal momento che dal punto di vista tecnologico sono entrambe valide. Inoltre, è difficile punire i nodi, perché non hanno identità. Rigiriamo la questione, possiamo premiare il creatore del nodo benevolo? L'identità non la conosco, quindi è possibile utilizzare unicamente degli incentivi. Esistono due diversi tipi di incentivi:

- Ricompensa blocco: il creatore di blocco ottiene l'inclusione in una speciale transazione per la creazione di monete nel blocco. Sceglie un indirizzo di destinazione di questa transazione, che gli appartenga. La fornitura di bitcoin deve essere fissa. Nuovi bitcoin vengono creati ad ogni nuovo blocco. I blocchi vengono creati ogni 10 minuti (in media). Il bitcoin deve essere un bene scarso. Per rendere la scarsità il creatore della moneta decise che ogni 210mila blocchi la fornitura di bitcoin emessi dovesse dimezzarsi, tramite un evento chiamato halving. Il primo halving ebbe luogo nel 2012 quando fu dimezzata la quantità di bitcoin emessi per blocco, passando da 50 a 25 bitcoin. Un secondo halving ebbe luogo nel 2016, dimezzando il numero da 25 a 12,5. Alle 19.23 Utc (le 21.23 in Italia) di lunedì 11 maggio 2020 è avvenuto il terzo halving. Il terzo halving ha portato a dimezzare la quantità di bitcoin coniati ogni blocco da 12,5 a 6,25. Tramite il meccanismo dell'halving, Satoshi fissa anche un tetto massimo di 21 milioni di bitcoin coniati (tetto che raggiungeremo circa nel 2140). Chi trova il nuovo blocco della blockchain ottiene il "reward del blocco" ovvero l'insieme dei nuovi bitcoin coniati (più le fee di ogni transazione validata all'interno del blocco) che vengono accreditati come ricompensa ai minatori che per primi trovano i nuovi blocchi. Potrebbe sembrare che chiunque crei un blocco riceva una ricompensa, ma la transazione di pagamento del miner viene avviata solo se il blocco viene effettivamente inserito nella catena.

- Commissione di transazione (Transaction Fee): il creatore di una transazione può scegliere di rendere il valore di output inferiore al valore di input. É una commissione di transazione che va al creatore del blocco (una sorta di mancia). La fee di transazione non è un valore prestabilito ma viene deciso dal mittente della transazione a seconda del traffico di rete di quel momento. Anche se dal punto di vista del codice di Bitcoin fornire commissioni di transazione non è richiesto per il completamento di una transazione, queste incentivano i miners a prendere in considerazione una transazione piuttosto che un altra. Quando ci si avvicinerà al 2140 potrebbe diventare obbligatoria, come unica ricompensa dei miner.

I protocolli di consenso sono un aspetto chiave che permette ad una blockchain di funzionare e di esistere. Possiamo considerare la blockchain una sorta di libro mastro contenente informazioni, è quindi fondamentale che venga data l'assoluta garanzia circa la veridicità ed accuratezza di queste informazioni. L'algoritmo seleziona i nodi in proporzione a una risorsa che nessuno può monopolizzare. A seconda della risorsa si può parlare di:

- In proporzione al calcolo: proof-of-work. La selezione viene fatta sulla base del lavoro svolto, quindi sulla base del loro potere computazionale. Quindi, i nodi vengono scelti sulla base delle loro capacità.
- In proporzione alla proprietà: proof-of-stake. In consenso Proof of Stake offre una sorta di garanzia su chi sarà colui che verrà scelto quale validatore del blocco successivo. Tale scelta avviene sulla base dell'ammontare dei token detenuti da ogni individuo e determinata da un sistema random.
- In proporzione a autorità: proof-of-authority. La Proof of Authority è un meccanismo di consenso autorizzato in cui il consenso è riferito ad un gruppo di validatori noti e rispettabili. I validatori sono nodi membri autorizzati a partecipare alla convalida delle transazioni e all'aggiunta di blocchi alla blockchain.

6.4 Proof of work

Proof-of-work è il protocollo con cui funziona il protocollo di consensi, ma rimangono aperti diversi problemi:

- Prendere per buono il fatto che si possa selezionare un nodo
- Legato al fatto che si possiede questo meccanismo di incentivi, potrebbe portare il sistema ad essere instabile, perché tutti vogliono far parte della rete ed avere remunerazioni
- Un attaccante potrebbe creare nodi sybil per sovvertire il funzionamento della rete

Esistono tecniche opportune per ovviare a questi problemi.

Proof-of-work è un protocollo di consenso introdotto da Bitcoin che attualmente è molto utilizzato da parecchie criptovalute. Questo processo è conosciuto come "mining" e di conseguenza i nodi della rete sono noti come "miners". La Proof of Work, che è un tipo di consenso trustless e distribuito, si presenta sotto forma di una risposta ad un problema matematico che richiede un notevole lavoro da parte dei computer per giungere ad una soluzione, Hash puzzles. Il procedimento è rappresentato nella Figura 175. Un nodo, dopo aver verificato l'intera blockchain, raccoglie e colleziona le nuove transazioni generate ancora non validate e suggerisce alla rete quale dovrebbe essere il nuovo blocco. I computer usano la funzione crittografica di hash per stimare l'output fino a che non risulta inferiore al valore di target (valore dato dal campo 'bits' nell'header del blocco). Il primo nodo che risolve il blocco lo trasmette nella rete dove viene accettato come blocco successivo nella catena. Sia X la funzione di hash fissata della rete e si consideri x come le transazioni pendenti e n rappresenti il valore di nonce. L'hash di output deve iniziare con degli zero e deve essere inferiore al valore di target, il problema è determinare il valore del nonce per far si che questo avvenga. Il numero degli zeri all'inizio dell'output costituisce la difficoltà per risolvere il blocco. La difficoltà è dovuta al fatto che la funzione crittografica di hash genera numeri casuali, difatti la variazione anche di un solo bit nell'input della funzione porta ad un output completamente differente rispetto a quello calcolato senza la variazione; in questo modo diventa praticamente impossibile predire il nonce. Una volta che il blocco è stato risolto, il suo hash, come un'impronta digitale, lo rappresenta univocamente ed è usato anche come riferimento al blocco precedente. Successivamente alla risoluzione del blocco, la rete automaticamente aggiusta il valore di target. L'intero processo di validazione dei blocchi è chiamato mining. É possibile, per diversi nodi, validare più blocchi contemporaneamente, portando così ad una biforcazione della catena. In questo caso, i miner (coloro che fanno mining) lavorano per la validazione dei blocchi su entrambe le biforcazioni della catena, ma appena in una delle due viene validato ed aggiunto un nuovo blocco, tutti i miner che lavoravano sull'altra si spostano su quella a cui è stato aggiunto un blocco nuovo, trasformando così il blocco abbandonato in un blocco orfano. Questo accade perché l'obiettivo dei miner è quello di estendere la catena in

lunghezza. Quindi, il miner deve trovare il nonce di un blocco tale per cui cada nel target space. Lo spazio target è molto piccolo, quindi il nodo deve fare molti tentativi. L'unico modo per risolvere questi enigmi matematici è attraverso i nodi della rete che eseguono un lungo e casuale processo di output delle risposte basate su tentativi ed errori. Tecnicamente, questo significa che il problema potrebbe essere risolto al primo tentativo, anche se ciò è estremamente improbabile, per non dire impossibile.

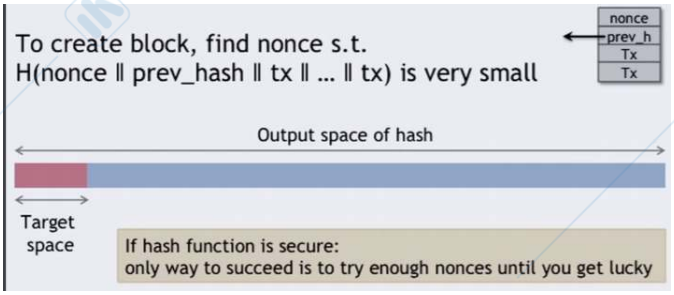


Figura 175: Hash puzzles

Le ricompense sono particolarmente importanti per via della complessità degli enigmi da risolvere, dato che, il processo è estremamente costoso, sia in termini di tempo che di potenza di calcolo. Mantenere i miners motivati è una funzione chiave di un protocollo in quanto sono, in un certo senso, le fondamenta che mantengono in funzione il sistema. La Proof of Work è garante del fatto che le transazioni non possono essere modificate, poiché i dati necessari per farlo sono estremamente difficili da produrre.

Le proprietà del protocollo di consenso del proof of work sono:

- Difficile da calcolare: circa 10^{20} hash / blocco. La dimensione dello spazio target è $\frac{1}{10^{20}}$. La difficile risoluzione comporta un grosso potere computazionale, al di fuori della capacità di un laptop. La ricerca di nounce è il processo di mining. Chi effettua mining si chiama miners. Potenzialmente, tutti possono essere miners. In realtà, non tutti possono esserlo, per via della potenza computazionale richiesta.
- Costo parametrizzabile: i nodi ricalcolano automaticamente l'obiettivo (lo spazio target) ogni due settimane in modo tale che il tempo medio tra i blocchi sia di 10 minuti. Perché proprio 10 minuti? Se si potessero creare blocchi più frequentemente, perderei in efficienza.
- Banale da verificare: il nonce deve essere pubblicato come parte del blocco; altri minatori verificano che $H(nonce||prev\_hash||tx||...||tx) < target$.

La difficoltà di mining cambia ogni 2016 blocchi, che si verifica circa una volta ogni 2 settimane. La nuova difficoltà è data dalla seguente equazione: $NF = (PD * 2016 * 10)/T)$, dove:

- NF: nuova difficoltà
- PD: difficoltà precedente
- 10: 10 minuti
- T: tempo di mining degli ultimi 2016 blocchi

La Figura 176 mostra come cambia la difficoltà del target space (riga rossa) in un periodo di 2 mesi. Mano a mano che aumenta la difficoltà bisogna avere una potenza computazionale (hash rate, ossia le operazioni di hash al secondo) sempre maggiore.

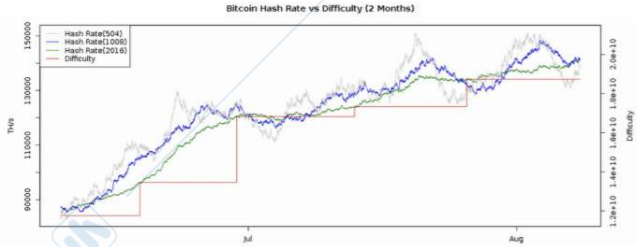


Figura 176: Difficoltà di mining nel tempo

La Figura 177 mette in relazione la difficoltà del target e il tempo necessario a risolvere il problema. Come si può vedere, il tempo necessario si aggiusta ogni volta che la difficoltà viene aumentata. Ogni due settimane il tempo si resetta. Con il passare del tempo, i miner aggiungono sempre più potenza per risolvere il problema.

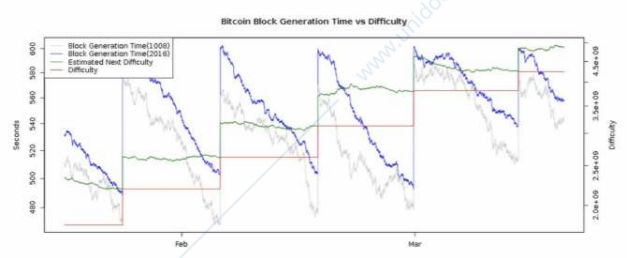


Figura 177: Tempo necessario per trovare il blocco