Protezione e Integrità dei Dati nel Cloud

Parte V

Indice

Capitolo 1

Encryption

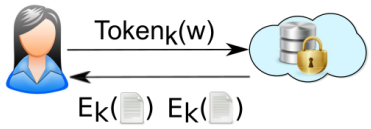
Il *server* potrebbe essere ***honest-but-curious***, non dovrebbe avere accesso alle risorse; voglio garantire confidenzialità anche rispetto a lui.

Un modo per ottenerla è utilizzare l'*encyption*: si aggiunge un livello di protezione attorno ai dati sensibili che li rende non leggibili a chi non è autorizzato.

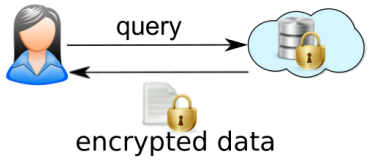
Di base voglio avere una criptazione dei dati; il problema è il **bilanciamento tra protezione e funzionalità**, ovvero sulle *query* che è possibile fare sui dati.

Approcci per accesso a diversi livelli di granularità

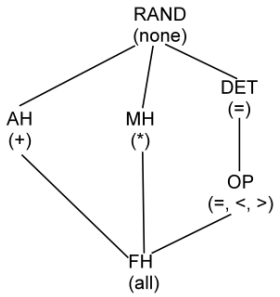
- ***Keyword-based searching:*** passo un *token* già criptato che viene usato per fare ricerca sui dati criptati (voglio trovare dove c'è una certa parola/espressione booleana)



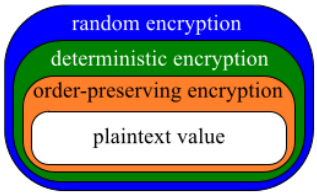
- **Crittografia omomorfica:** crittografia che supporta le operazioni direttamente sul cifrato

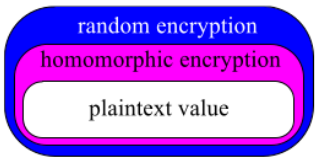


- ***Encryption Schemas:*** ogni colonna può essere cifrata con un diverso schema crittografico (*random, add homomorphic, deterministic, order preserving, ...*)



- ***Onion Encryption:*** cifro i dati con diversi livelli *a cipolla*, ognuno dei quali supporta l'esecuzione di una specifica *query SQL*; l'idea è che *scopro il dato solo quando mi serve*





- **Indicizzazione:** associo degli indici ai metadati Nella seconda tabella:

Accounts

Account	Customer	Balance
Acc1	Alice	100
Acc2	Alice	200
Acc3	Bob	300
Acc4	Chris	200
Acc5	Donna	400
Acc6	Elvis	200

Accounts$_1^k$

Counter	Etuple	I_A	I_C	I_B
1	x4Z3tfX2ShOSM	π	α	μ
2	mNHg1oC010p8w	ϖ	α	κ
3	WsIaCvfyF1Dxw	ξ	β	η
4	JpO8eLTVgwV1E	ρ	γ	κ
5	qctG6XnFNDTQc	ς	δ	θ
6	4QbqCeq3hxZHklU	ι	ε	κ

nella seconda colonna c'è la tupla criptata; nelle ultime tre ci sono gli attributi; si possono avere diversi tipi di indicizzazione:

  - ***Direct*** $(1:1)$

+ riesco a fare query precise

- soggetto ad attacchi di frequenza

Patients

SSN	Name	Illness	Doctor
123…89	Alice	Asthma	Angel
234…91	Bob	Asthma	Angel
345…12	Carol	Asthma	Bell
456…23	David	Bronchitis	Clark
567…34	Eva	Gastritis	Dan
232…11	Eva	Stroke	Ellis

Patients^k

Tid	Etuple	I_S	I_N	I_I	I_D
1	x4Z3tfX2ShOSM	π	κ	α	δ
2	mNHg1oC010p8w	ϖ	ω	α	δ
3	WsIaCvfyF1Dxw	ξ	λ	α	ν
4	JpO8eLTVgwV1E	ρ	υ	β	γ
5	qctG6XnFNDTQc	ι	μ	α	σ
6	kotG8XnFNDTaW	χ	o	β	ψ

– ***Bucket*** $(n:1) \rightarrow$ indicizzazione con collisione; ho diversi valori che sono **mappati allo stesso indice**

+ non ho più attacchi di frequenze

+ supporta query di uguaglianza (*se un valore è uguale ad un altro*)

- i risultati avranno delle tuple spurie

- è ancora possibile fare qualche leakage *In questo caso sono comunque*

Patients

SSN	Name	Illness	Doctor
123…89	Alice	Asthma	Angel
234…91	Bob	Asthma	Angel
345…12	Carol	Asthma	Bell
456…23	David	Bronchitis	Clark
567…34	Eva	Gastritis	Dan
232…11	Eva	Stroke	Ellis

Patients^k

Tid	Etuple	I_S	I_N	I_I	I_D
1	x4Z3tfX2ShOSM	π	κ	α	δ
2	mNHg1oC010p8w	ϖ	ω	α	δ
3	WsIaCvfyF1Dxw	ξ	λ	α	ν
4	JpO8eLTVgwV1E	ρ	υ	β	γ
5	qctG6XnFNDTQc	ι	μ	α	σ
6	kotG8XnFNDTaW	χ	o	β	ψ

esposto perché asma ha 3 occorrenze, dunque sarà per forza associata ad α

– ***Flattened*** $(1:n) \rightarrow$ ciascun indice deve avere lo stesso numero di occorrenze; significa che i valori che hanno più occorrenze sono associati ad indici diversi

+ rimuovo la possibilità di fare attacchi di inferenze

- sono esposto ad osservazioni dinamiche (magari certi dati sono sempre cercati assieme)

Patients

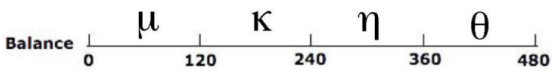
SSN	Name	Illness	Doctor
123…89	Alice	Asthma	Angel
234…91	Bob	Asthma	Angel
345…12	Carol	Asthma	Bell
456…23	David	Bronchitis	Clark
567…34	Eva	Gastritis	Dan
232…11	Eva	Stroke	Ellis

Patients^k

Tid	Etuple	I_S	I_N	I_I	I_D
1	x4Z3tfX2ShOSM	π	κ	α	δ
2	mNHg1oC010p8w	ϖ	ω	α	δ
3	WsIaCvfyF1Dxw	ξ	λ	α	ν
4	JpO8eLTVgwV1E	ρ	υ	β	γ
5	qctG6XnFNDTQc	ι	μ	α	σ
6	kotG8XnFNDTaW	χ	o	β	ψ

– ***Partition-based:***

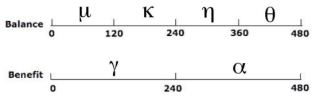
1. si partiziona il dominio di un attributo
2. a ciascuna partizione si assegna un'etichetta
3. il valore in chiaro viene sostituito dall'etichetta



Supporta *query* dove le condizioni sono espressioni booleane del tipo:
- *Attribute* `op` *Value*
- *Attribute* `op` *Attribute*

dove `op`$= \{=, <, >, \leq, \geq\}$

Example



$Map_{cond}(\text{Balance=Benefit}) \Longrightarrow (I_{Balance}=\mu \wedge I_{Benefit}=\gamma) \vee (I_{Balance}=\kappa \wedge I_{Benefit}=\gamma) \vee (I_{Balance}=\eta \wedge I_{Benefit}=\alpha) \vee (I_{Balance}=\theta \wedge I_{Benefit}=\alpha)$

Esecuzione delle query:

Ogni query Q sul DB in chiaro viene tradotta in:

1. una query Q_s da eseguire sul server $\rightarrow$ query sull'indice per ottenere le tuple criptate
2. una query Q_c da eseguire sul client $\rightarrow$ decriptare il risultato della query precedente e filtrare le tuple spurie

La traduzione dovrebbe essere fatta in modo tale che il server sia responsabile della maggior parte del lavoro.

Accounts

Account	Customer	Balance
Acc1	Alice	100
Acc2	Alice	200
Acc3	Bob	300
Acc4	Chris	200
Acc5	Donna	400
Acc6	Elvis	200

Accounts$^{k}_{2}$

Counter	Etuple	I_A	I_C	I_B
1	x4Z3tfX2ShOSM	π	α	μ
2	mNHg1oC010p8w	ϖ	α	κ
3	WslaCvfyF1Dxw	ξ	δ	θ
4	JpO8eLTVgwV1E	ρ	α	κ
5	qctG6XnFNDTQc	ς	β	κ
6	4QbqC3hxZHklU	ι	β	κ

Original query on Accounts	Translation over Accounts_2^k
Q := SELECT * FROM Accounts WHERE Balance=200	Q_s := SELECT Etuple FROM Accounts_2^k WHERE I_B=κ Q_c := SELECT * FROM *Decrypt*(Q_s, *Key*) WHERE Balance=200

– ***Hash-based:*** basate sul concetto di *one-way hash function*; ogni attributo viene mappato ad un indice utilizzando una funzione di hash sicura.

Dat una funzione h e il dominio degli attributi D_i, diciamo che h è **sicura** se:

1. $\forall x, y \in D_i \implies h(x) = h(y)$ (**determinismo**)
2. dati due valori $x, y \in D_i$ tali che $x \neq y$, potremmo avere che $h(x) = h(y)$ (**collisione**, per proteggermi da attacchi di frequenza)
3. la distanza dei valori in chiaro deve essere **indipendente** dalla distanza dei valori di hash (*strong mixing*)

Questo metodo supporta *query* dove le condizioni sono espressioni booleane del tipo:

* $Attribute = Value$
* $Attribute_1 = Attribute_2$, se sono indicizzati con la stessa funzione di hash

La traduzione funziona come nel metodo *partion-based*; non sono supportate *query di range*.

Interval-based queries

- Le tecniche di indicizzazione che preservano l'ordine supportano query di range, ma sono esposte ad inferenza
- Le tecniche di incizzazione che *non* preservano l'ordine non sono esposte ad inferenza, ma non supportano query di range

$\rightarrow$ viene calcolato un $B_+ - tree$ dal client, ed ogni nodo viene criptato come un tutt'uno; successivamente per rispondere alle query l'albero viene visitato (in ambiente trusted).

1.1 *Searchable Encryption*

1.1.1 *Order preserving encryption*

- ***Order Preserving Encryption Schema (OPES):*** prende in input una distribuzione target di valori per gli indici ed applica una trasformazione che preserva l'ordine e rispecchia la distribuzione di input.

  \+ la comparazione può essere fatta direttamente sui dati criptati

  \+ le query non producono tuple spurie

  \- vulnerabile ad attacchi di inferenza

- ***Order Preserving Encryption with Splitting and Scaling (OPESS):***

  Questo schema crea degli indici in modo tale che la loro distribuzione delle frequenze sia piatta.

1.1.2 *Fully homomorphic encryption*

- Permette una performante computazione specifica sui dati criptati
- Decriptando il risultato, si ottiene lo stesso risultato delle stesse operazioni sui dati in chiaro

1.2 Esposizione all'inferenza

Ci sono due requisiti conflittuali quando si parla di *indicizzare* dati:

- gli indici dovrebbero fornire una **esecuzione delle query efficiente**
- gli indici non dovrebbero aprire porte ad attacchi di **inferenza** e *linking*

→ diventa importante misurare quantitativamente il livello di esposizione dovuto alla pubblicazione degli indici:

$$\epsilon = \textit{Coefficiente di Esposizione}$$

La computazione del *Coefficiente di Esposizione* dipende da diversi fattori:

- **Metodo di incizzazione utilizzato**
  - *direct encryption*
  - *hashing*
- **Conoscenza pregressa dell'attaccante**
  - $Freq + DB^k$
  - $DB + DB^k$

  In entrambi i casi l'attaccante può risalire alla funzione di incizzazione.

1.2.1 Direct Encryption

Freq + DB^k

- La corrispondenza tra indice e valore in chiaro può essere determinata sulla base del numero di occorenze di indice/valore

  → **Protezione base:** i valori con lo stesso numero di occorenze sono indistinguibili per l'attaccante
- Valutazione dell'esposizione dell'indice basata sulla relazione di equivalenza in cui i valori di indice/valore con lo stesso numero di occorrenze appartengono alla stessa classe

  → L'esposizione di un indice nella classe di equivalenza C è $1/|C|$

A.1 = $\{\pi, \varpi, \xi, \rho, \varsigma, \iota\}$ = {Acc1,...,Acc6}

C.1 = $\{\beta, \gamma, \delta, \varepsilon\}$ = {Bob,Chris,Donna,Elvis}

C.2 = $\{\alpha\}$ = {Alice}

B.1 = $\{\mu, \eta, \theta\}$ = {100,300,400}

B.3 = $\{\kappa\}$ = {200}

INDEX_VALUES

I_A	I_C	I_B
π	α	μ
ϖ	α	κ
ξ	β	η
ρ	γ	κ
ς	δ	θ
ι	ε	κ

QUOTIENT

qt_A	qt_C	qt_B
A.1	C.2	B.1
A.1	C.2	B.3
A.1	C.1	B.1
A.1	C.1	B.3
A.1	C.1	B.1
A.1	C.1	B.3

INVERSE CARDINALITY

ic_A	ic_C	ic_B
1/6	1	1/3
1/6	1	1
1/6	1/4	1/3
1/6	1/4	1
1/6	1/4	1/3
1/6	1/4	1

$$\mathscr{E} = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k} \mathrm{IC}_{i,j} = 1/18$$

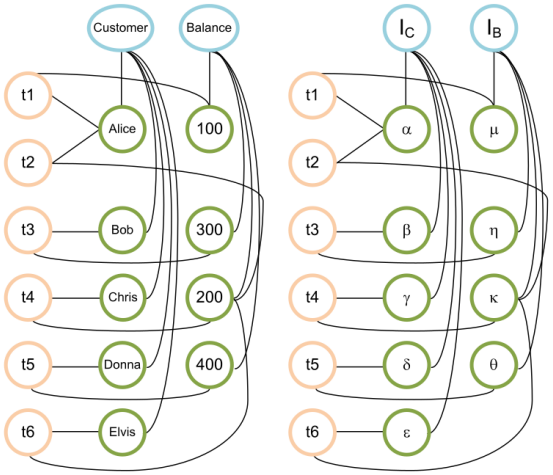
- nella tabella *Quotient* ci sono le classi di equivalenza a cui appartengono gli indici
- nella tabella *Inverse Cardinality* c'è $1/|C|$, si interpreta come:
  - *c'è 1 di 6 valori che non so distinguere*
  - *c'è 1 di 4 valori che non so distinguere*
  - Sta esprimendo l'incertezza; più sarà grande $|C|$, più avrò incertezza → quelli con 1/1 rappresentano un problema dato che non c'è incertezza
- A **livello di tupla** l'incertezza è il **prodotto** delle incertezze
- A **livello di tabella** faccio la **media** dell'esposizione delle tuple (ϵ)

DB + DB^k

- Grafo ***Row-Column-Value*** non-direzionato a 3 colori
  - un vertice di colore *column* per ogni attributo
  - un vertice di colore *row* per ogni tupla
  - un vertice di colore *value* per ogni valore distinto in una colonna
  - un arco connette ogni valore alla riga e colonna in cui compare
- RCV sui valori in chiaro è uguale a quello sugli indici
- posso avere una misura del grado di esposizione guardando quanto *un nodo si confonde* (automorfismo)

Customer	Balance
Alice	100
Alice	200
Bob	300
Chris	200
Donna	400
Elvis	200

I_C	I_B
α	μ
α	κ
β	η
γ	κ
δ	θ
ε	κ



Inference

- I_C = Customer
- I_B = Balance
- α = Alice
- μ = 100
- κ = 200
- $\{\gamma, \varepsilon\}$ = {Chris,Elvis}
- $\{\langle\beta,\eta\rangle,\langle\delta,\theta\rangle\}$= {⟨Bob,300⟩,⟨Donna,400⟩}

Equitable partition: $\{(\alpha),(\beta,\delta),(\gamma,\varepsilon),(\mu),(\eta,\theta),(\kappa)\}$
$\mathscr{E} = 6/9 = 2/3$

Per *Equitable Partion* si intende un insieme di vertici che costituiscono un automorfismo.
L'esposizione si calcola come il rapporto tra il numero di *equitable partition* e il numero totale degli elementi.

1.2.2 Hashing

Freq + DB^k

- La funzione di hash è caratterizzata da un *fattore di collisione*, ovvero il numero di valori che in media collidono sullo stesso indice
- Sono possibili diversi mapping dei valori negli indici, in relazione ai vincoli imposti dalle frequenze
- Per ogni mapping si calcola il coefficiente di esposizione

DB + DB^k

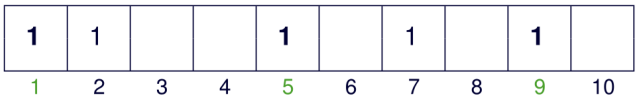
- i grafi RCV tra dati in chiaro e criptati non sono uguali, dato che *vertici diversi* nel grafo in chiaro potrebbro collassare nello *stesso vertice* nel grafo criptato
- il numero di archi che collega i vertici *row* ai vertici *value* è lo stesso
- il problema diventa trovare un *matching corretto* tra gli archi del grafo in chiaro e quello criptato

1.3 Bloom Filter

Il *Bloom Filter* sta alla base della costruzione di alcune tecniche di indicizzazione; è un metodo efficiente per codificare l'appartenenza a un insieme.

- set di n elementi (n è grande)
- vettore di l bit (l è piccolo)
- h funzioni di hash indipendenti $H_i : \{0,1\}^* \to [1, l]$
- **Insert x:** set a 1 i bit corrispondenti a $H_1(x), H_2(x), \ldots, H_h(x)$
- **Search x:** Computare $H_1(x), H_2(x), \ldots, H_h(x)$ e verificare se quei valori sono settati a 1 nel vettore

Let $l = 10$ and $h = 3$



- Insert sun: H_1(sun)=2; H_2(sun)=5; H_3(sun)=9
- Insert frog: H_1(frog)=1; H_2(frog)=5; H_3(frog)=7
- Search dog: H_1(dog)=2; H_2(dog)=5; H_3(dog)=10
  $\Longrightarrow$ No
- Search car: H_1(car)=1; H_2(car)=5; H_3(car)=9
  $\Longrightarrow$ Maybe Yes; false positive!

- è una generalizzazione dell'hashing (*bloom filter* con 1 funzione di hash equivale all'hash ordinario)

  \+ efficiente nello spazio

  \- gli elementi non possono essere rimossi

- ha una costante di probabilità di ottenere un falso positivo

  \- teoricamente non accettabile

  \+ nella pratica è accettabile perché il costo viene messo in relazione ai guadagni in termini di spazio

1.4 Integrità dei Dati

Due aspetti:

- **Integrità in Storage:** i dati devono essere protetti da modifiche non autorizzate

  → update non autorizzate devono essere rilevati

  - si ottiene utilizzando la **firma digitale** a livello di tupla (a livello di cella sarebbe troppo costoso)

- **Integrità nelle query:** i risultati delle query devono essere corretti e completi

  → un comportamento non corretto del server deve essere rilevato

1.5 *Selective-Encryption* e *Over-Encryption*

1.5.1 Selective Encryption

Utenti diversi potrebbero necessitare di viste diverse dei dati nel cloud
→ **Selective Encryption:** la politica di autorizzazione definita dal proprietario dei dati viene tradotta in una politica di encryption equivalente

Desiderata:

- i dati stessi dovrebbero regolare i controlli di accesso
- dovrebbero essere usate chiavi differenti per criptare i dati
- l'autorizzazione di accesso a una risorsa viene tradotta nella **conoscenza della chiave** con cui la risorsa è criptata
- ad ogni utente vengono comunicate le chiavi per decriptare i dati a cui ha diritto di accesso

Politiche di Autorizzazione

Il *data owner* definisce delle politiche di autorizzazione per regolare l'accesso ai dati.

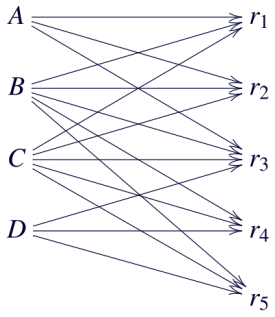
- Una politica di autorizzazione $\mathcal{A}$ è un set di permessi della forma $\langle user, resource \rangle$

  Può essere rappresentata sotto forma di:

  - matrice
  - grafo diretto bipartito

- L'idea è che diverse autorizzazioni di accesso ai dati implicano diverse chiave per criptare

	r_1	r_2	r_3	r_4	r_5
A	1	1	1	0	0
B	1	1	1	1	1
C	1	1	1	1	1
D	0	0	1	1	1



Politica di Encryption

La *politica di autorizzazione* definita dal data owner viene tradotta in una *politica di encyption* equivalente.
Due possibili soluzioni:

- criptare ogni risorsa con una chiave diversa e dare all'utente le chiavi che decriptano le risorse a cui ha accesso

  - l'utente deve gestire tante chiavi quante sono le risorse a cui ha accesso

- usare un **metodo di derivazione delle chiavi** per permettere di derivare dalla propria chiave utente tutte le chiavi a cui hanno accesso

  + ad ogni utente viene rilasciata una sola chiave

Metodi di Derivazione delle Chiavi

- Basata sulla definizione di una **gerarchia di derivazione delle chiavi** $(\mathcal{K}, \leq)$

  - $\mathcal{K}$ è il set di chiavi

  - $\leq$ è la relazione d'ordine parziale definita su $\mathcal{K}$

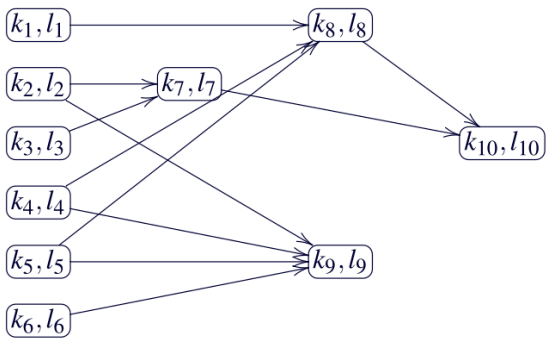
- $(\mathcal{K}, \leq)$ può essere rappresentata come un grafo con un vertice per ogni $x \in \mathcal{K}$ e un percorso da x a y sse $y \leq x$

Metodi di Derivazione delle Chiavi basati su Token

- Le chiavi sono assegnate arbitrariamente ai vertici
- Una label l_i (pubblica) viene assegnata a ciascuna chiave k_i
- Un token $t_{i,j}$ (pubblico) viene associato ad ogni arco nella gerarchia
- Dato un arco (k_i, k_j), il token $t_{i,j}$ viene calcolato come $k_j \oplus h(k_i, l_j)$, dove:
  - $\oplus$ è l'operatore `xor`
  - h è una funzione di hash sicura
- $+$ i token sono pubblici e permettono agli utenti di derivare più chiavi, ma dovendosi preoccupare solo di una
- $+$ possono essere storati su un server così che ogni utente vi può accedere

Le relazioni delle chiavi tramite token possono essere rappresentate con un grafo:

- un vertice per ogni coppia $\langle k, l \rangle$, dove $k \in \mathcal{K}$ è una chiave e $l \in \mathcal{L}$ è l'etichetta associata
- un arco dal vertice $\langle k_i, l_i \rangle$ a $\langle k_j, l_j \rangle$ se esiste un token $t_{i,j} \in \mathcal{T}$ che permette la derivazione di k_j a partire da k_i



Traduzione della politica di autorizzazione in una di encryption:

- *Desiderata:*
  - ad ogni utente viene rilasciata una sola chiave

  - le risorse vengono criptate una sola volta con una sola chiave

- Una funzione $\phi : \mathcal{U} \cup \mathcal{R} \rightarrow \mathcal{L}$ che descrive:
  - l'associazione tra un utente la (etichetta della) sua chiave
  - l'associazione tra una risorsa e la (etichetta della) chiave usata per criptarla

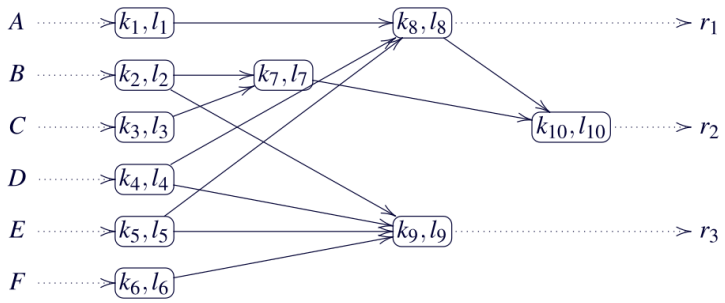
Definzione Formale della Politica Crittografica

Una **politica di encryption** su utenti $\mathcal{U}$ e risorse $\mathcal{R}$, denotata come $\mathcal{E}$, è una 6-tupla $\langle \mathcal{U}, \mathcal{R}, \mathcal{K}, \mathcal{L}, \phi, \mathcal{T} \rangle$, dove:

- $\mathcal{K}$ è il set di chiavi del sistema e $\mathcal{L}$ l'insieme delle chiavi corrispondenti
- ϕ è la funzione di assegnamento delle chiavi e schema crittografico
- $\mathcal{T}$ è il set di token definiti su $\mathcal{K}$ e $\mathcal{L}$

La politica di encryption può essere rappresentata come un grafo estendo quello di chiavi e token per includere:

- un vertice per ogni utente e ogni risorsa
- un arco da ogni vertice utente u a $\langle k, l \rangle$ tale che $\phi(u) = l$
- un arco da ogni vertice $\langle k, l \rangle$ a ogni vertice risorsa r tale che $\phi(r) = l$



- user A can access $\{r_1, r_2\}$
- user B can access $\{r_2, r_3\}$
- user C can access $\{r_2\}$
- user D can access $\{r_1, r_2, r_3\}$
- user E can access $\{r_1, r_2, r_3\}$
- user F can access $\{r_3\}$

Politica di Trasformazione

Obiettivo: trasformare una politica di autorizzazione $\mathcal{A}$ in una politica di encryption $\mathcal{E}$ **equivalente**.
$\mathcal{A}$ e $\mathcal{E}$ si dicono equivalenti se garantiscono gli stessi accessi.

- **Soluzione nativa**
  - ad ogni utente viene associata una chiave
  - ogni risorsa viene criptata con una chiave
  - per ogni permesso $\langle u, r \rangle$ viene generato un token $t_{u,r}$

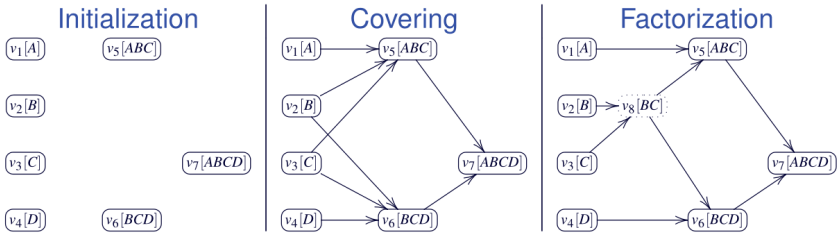
  $\rightarrow$ produrre e gestire un token per ogni singolo permesso non è realizzabile
- $\rightarrow$ **Si sfruttano i gruppi di utente**
  - si raggruppano gli utenti con gli stessi privilegi
  - si cripta ogni risorsa con la chiave associata al set di utenti che può accedervi
- È possibile creare un grafo sfruttando la gerarchia tra insiemi di utenti, indotta dalla relazione d'ordine parziale di inclusione di insieme ($\subseteq$)
- **Osservazione:** i gruppi che non corrispondono a nessun accesso non hanno bisogno di una chiave
- $\rightarrow$ **Obiettivo:** computare una politica di encryption minima, equivalente a una politica di autorizzazione data, che minimizza il numero di token gestiti dal server

Costruzione di un grafo per chiavi e token

Partendo da un politica di autorizzazione $\mathcal{A}$:

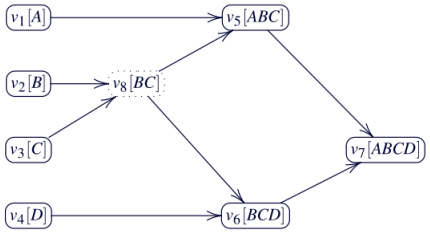
1. **Inizializzazione:** si crea un vertice (chiave) per ogni utente e gruppi di utenti (acl)
2. **Covering** minimo; mi fa in modo che ciascun utente possa raggiungere le sue chiavi
3. **Fattorizzazione** di antenati comuni (se ho n nodi da una parte e m dall'altra, mettendo un *hub* in mezzo passo da $n * m$ a $n + m$)

	r_1	r_2	r_3	r_4	r_5
A	0	1	0	1	1
B	1	1	1	1	1
C	0	1	1	1	1
D	0	0	1	1	1



- gli utenti riceveranno:
  - $A = \langle k_1, l_1 \rangle$
  - $B = \langle k_2, l_2 \rangle$
  - $C = \langle k_3, l_3 \rangle$
  - $D = \langle k_4, l_4 \rangle$

  tutto il resto è sul server
- la funzione ϕ mi dice rispettivamente quali chiavi hanno gli utenti e quali chiavi sono associate alle risorse, facendo riferimento alle *label*



u	$\phi(u)$
A	$v_1.l$
B	$v_2.l$
C	$v_3.l$
D	$v_4.l$

r	$\phi(r)$
r_1	$v_2.l$
r_2	$v_5.l$
r_3	$v_6.l$
r_4,r_5	$v_7.l$

source	destination	token_value
$v_1.l$	$v_5.l$	$t_{1,5}$
$v_2.l$	$v_8.l$	$t_{2,8}$
$v_3.l$	$v_8.l$	$t_{3,8}$
$v_4.l$	$v_6.l$	$t_{4,6}$
$v_5.l$	$v_7.l$	$t_{5,7}$
$v_6.l$	$v_7.l$	$t_{6,7}$
$v_8.l$	$v_5.l$	$t_{8,5}$
$v_8.l$	$v_6.l$	$t_{8,6}$

Quando le autorizzazioni cambiano dinamicamente, il data owner deve:

- scaricare la risorsa dal server
- creare una nuova chiave
- decriptare la risorsa con la vecchia chiave
- criptare la risorsa con la nuova chiave
- upload della risorsa e comunicare l'update
  → Non efficiente;
- Possibile soluzione **over-encryption**

1.5.2 Over-Encryption

Le risorse vengono criptate due volte:

- dall'*owner*, con una chiave condivisa a tutti gli utenti e sconosciuta dal server (**Base Encryption Layer - BEL**)
- dal *server*, con una chiave condivisa agli utenti autorizzati (**Surface Encryption Layer - SEL**)
  → per accedere a una risorsa un utente deve conoscere sia la chiave BEL che SEL

BEL

A livello BEL distinguiamo due tipi di chiavi: chiavi di **accesso** k_a e di derivazione k

- ogni nodo viene associato ad una coppia di chiavi (k, k_a) dove $k_a = h(k)$ (h funzione hash sicura *one-way*) e ad una coppia di labels (l, l_a)
- la chiave k e la label l sono usate per la derivazione
- la chiave k_a e la label l_a sono usate per criptare le risorse associate al nodo
- la distinzione delle chiavi separa i due ruoli: derivazione delle chiavi e accesso alle risorse

SEL

A livello SEL viene fatta una politica di encryption come mostrato precedentemente; ci si può dividere in due scenari:

- **FullSEL:** inizia da un SEL identico al BEL e tiene il SEL sempre aggiornato per rispecchiare la politica corrente
- **DeltaSEL:** inizia da un SEL vuoto e aggiunge elementi man mano che la politica evolve, in modo tale che la coppia BEL SEL rispecchi la politica

L'evoluzione di BEL e SEL è gestita da:

- procedura **over-encrypt** che regola il processo di update facendo over-encryption delle risorse a livello SEL

  chiedo aiuto al server per andare a coprire qualcosa

- procedure **grant** e **revoke** per gestire i privilegi

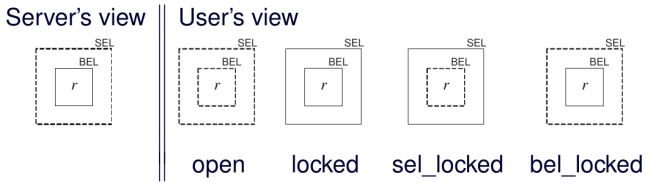
  l"andare a coprire qualcosa" mi serve sia in operazioni di grant che di revoke

  grant $\rightarrow$ *coprire delle risorse che altrimenti resterebbero scoperte*

  revoke $\rightarrow$ *coprire per non rendere più accessibile*

1.5.3 Collusione

La collusione si verifica quando due entità, unendo le loro conoscenze, acquisicono conoscenza a cui prima nessuna delle due aveva accesso.
Ci può essere collusione tra utenti o con il server; dipende dalla visione che gli utenti hanno delle risorse.



Capitolo 2

Fragmentation e Encryption

L'encryption rende la valutazione delle query e l'esecuzione delle applicazioni più costosa e non sempre possibile.
Spesso ciò che è sensibile è l'**associazione** tra valori di attributi diversi, piuttosto che i valori stessi.
→ si proteggono le associazioni **spezzandole**, piuttosto che criptando

Constraint di confidenzialità

- **Attributi sensibili:** il **valore** di alcuni attributi potrebbe essere considerato sensibile e non dovrebbe essere visibile

  → *singleton constraint*

- **Associazioni sensibili:** l'**associazione** dei valori di attributi dati è sensibile e non dovrebbe essere sensibile

  → *non-singleton constraint*

2.1 Coppia di server non-comunicanti

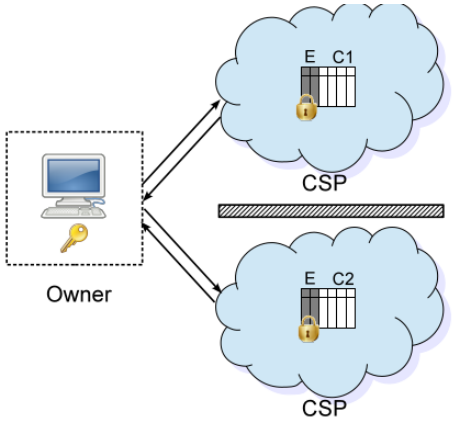
I vincoli di riservatezza sono rispettati dividendo le informazioni su **due server indipendenti che non comunicano**.

- le associazioni sensibili sono protette distribuendo i valori su i due server
- l'encryption viene usata per coprire i valori che non possono stare su nessuno dei due server (constraint oppure esporrebbe almeno una associazione sensibile)

I constraint di confidenzialità $\mathcal{C}$ definiti su una relazione $\mathcal{R}$ sono applicati scomponendo $\mathcal{R}$ come $\langle R_1, R_2, E\rangle$, dove:

- R_1 e R_2 hanno un *tuple ID* per garantire un join senza perdita di informazioni

- $R_1 \cup R_2 = R$
- E è il set di attributi criptati, con $E \subset R_1, E \subset R_2$
- Nessun costraint di confidenzialità può essere contenuto insieme in chiaro



2.1.1 Esecuzione delle query

Per rispondere alle query sarà necessario interrogare entrambi i server; è possibile farlo in due modi:

- si mandano delle *sub-queries* a S_1 e S_2 in parallelo, per poi joinare i risultati sul client
- si manda solo una delle *sub-queries*, ad esempio a S_1; il *tuple ID* del risultato di S_1 è usato poi per fare un semi-join con il risultato della query di S_2

2.1.2 Identificare la scomposizione ottimale

Si utilizza una **matrice di affinità:** indica quanto spesso due attributi sono acceduti insieme e la frequenza delle query.
Matrice di affinità M:

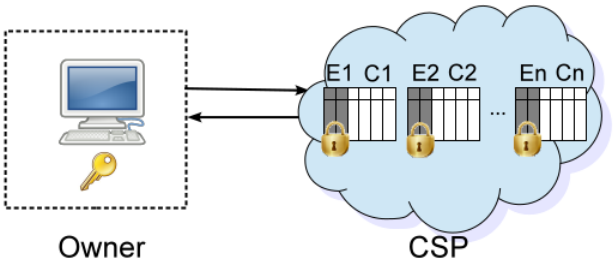
- $M_{i,j}$: costo di mettere attributi i e j in chiaro in frammenti diversi
- $M_{i,i}$: costo di criptare attributo i (e metterlo in entrambi i frammenti)
- **Obiettivo:** Minimizzare $\sum M_{i,j} + \sum M_{i,i}$ (la somma delle affinità e criptazione)

2.2 Frammenti non comunicanti multipli

L'assunzione di avere (solo) due server non comunicanti:

- - difficile da realizzare
- - limita il numero di associazioni che si possono risolvere solo con la frammentazione

→ si utilizzano frammenti non comunicanti multipli



- Una frammentazione di $\mathcal{R}$ è un set di frammenti $\mathcal{F} = \{F_1, \ldots, F_m\}$ dove $F_i \subset R$
- Una frammentazione $\mathcal{F}$ soddisfa correttamente un set constraint di confidenzialità $\mathcal{C}$ se:
  - ogni frammento soddisfa i constraint
  - i frammenti non hanno attributi in comune
- Ogni frammento viene mappato in un *frammento fisico* contenente:
  - i suoi attributi in chiaro
  - tutti gli altri attributi criptati (per ogni criptazione viene applicato un *salt*)

PATIENTS

	SSN	Name	YoB	Job	Disease
t_1	123456789	Alice	1980	Clerk	Asthma
t_2	234567891	Bob	1980	Doctor	Asthma
t_3	345678912	Carol	1970	Nurse	Asthma
t_4	456789123	David	1970	Lawyer	Bronchitis
t_5	567891234	Eva	1970	Doctor	Bronchitis
t_6	678912345	Frank	1960	Doctor	Gastritis
t_7	789123456	Gary	1960	Teacher	Gastritis
t_8	891234567	Hilary	1960	Nurse	Diabetes

c_0 = {SSN}
c_1 = {Name, Disease}
c_2 = {Name, Job}
c_3 = {Job, Disease}

F_1

salt	enc	Name	YoB
s_{11}	Bd6!l3	Alice	1980
s_{12}	Oij3X.	Bob	1980
s_{13}	9kEf6?	Carol	1970
s_{14}	ker5/2	David	1970
s_{15}	C:mE91	Eva	1970
s_{16}	4lDwqz	Frank	1960
s_{17}	me3,op	Gary	1960
s_{18}	zWf4g>	Hilary	1960

F_2

salt	enc	Job
s_{21}	8de6TO	Clerk
s_{22}	X'mlE3	Doctor
s_{23}	wq.vy0	Nurse
s_{24}	nh=l3a	Lawyer
s_{25}	hh%kj)	Doctor
s_{26}	;vf5eS	Doctor
s_{27}	e4+YUp	Teacher
s_{28}	pgt6eC	Nurse

F_3

salt	enc	Disease
s_{31}	ew3)V!	Asthma
s_{32}	LkEd69	Asthma
s_{33}	w8vd66	Asthma
s_{34}	1"qPdd	Bronchitis
s_{35}	(mn2eW	Bronchitis
s_{36}	wD}x1X	Gastritis
s_{37}	0opAuEl	Gastritis
s_{38}	Sw@Fez	Diabetes

2.2.1 Esecuzione delle query

Dato che ogni frammento contiene tutti gli attributi, per rispondere a una query basta accedere soltanto a uno; se la query coinvolge un attributo criptato, potrebbe essere necessario fare delle query aggiuntive sul client.

Original query on R	**Translation over fragment F_3**
Q := SELECT SSN, Name FROM PATIENTS WHERE (Disease='Gastritis' OR Disease='Asthma') AND Job='Doctor'	Q^3 := SELECT salt, enc FROM F_3 WHERE (Disease='Gastritis' OR Disease='Asthma') Q' := SELECT SSN, Name FROM *Decrypt*(Q^3, *Key*) WHERE Job='Doctor'

2.2.2 Criteri di ottimizzazione

L'obiettivo è trovare una frammentazione che renda l'esecuzione delle query efficiente.
Ci sono diversi criteri di ottimizzazione:

- **Numero di frammenti minimo**
- **Affinità tra attributi**
- **Query workload** (minimizzare il costo dell'esecuzione delle query)

Ciascuno di questi criteri obbedisce alla **visibilità massima:**

- solo gli attributi che appaiono in *singleton constraint* vengono criptati
- tutti gli attributi che non sono sensibili appaiono in chiaro in un frammento

Minimo numero di frammenti

- Si definisce una nozione di minimalità per computare in modo efficiente la soluzione
  - $\mathcal{F}$ è minimale se prendendo due frammenti qualsiasi da $\mathcal{F}$ e unendoli si viola almeno un constraint di confidenzialità
- Iterativamente si seleziona un attributo con il numero massimo di constraint non risolti e lo si inserisce in un frammento esistentente se non viene violato alcun constraint; se ne crea uno nuovo altrimenti

Massima affinità tra attributi

Idea:

- preservare le associazioni tra alcuni attributi
- **matrice di affinità** per rappresentare il vantaggio di avere una coppia di attributi nello stesso frammento

Obiettivo:

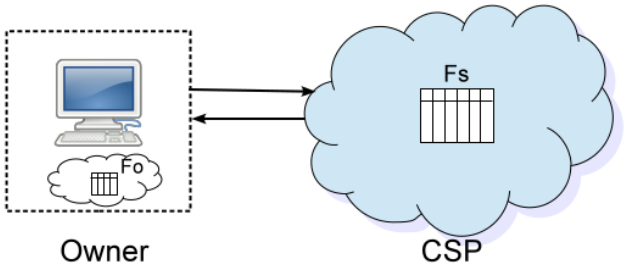
- Computare una frammentazione corretta con affinità massima
- Iterativamente si combinano i frammenti che hanno l'affinità massima e che non violano alcun constraint

Capitolo 3

Frammentazione

L'idea di base è:

- - la cifratura rende l'esecuzione delle query più costosa e non sempre possibile
- - la cifratura comporta un sovraccarico per la gestione delle chiavi
- → si abbandona la cifratura, per coinvolgere l'**owner come una parte fidata che mantiene una quantità limitata di dati**



Dati uno schema relazionale R e dei constraint $\mathcal{C}$ su di esso, si determina una frammentazione $\mathcal{F} = \langle F_O, F_S \rangle$, dove lo storage di F_O è affidato all'owner e quello di F_S al server, e:

- $F_O \cup F_S = R$ (**completezza**)
- $\forall c \in \mathcal{C}, c \notin F_S$ (**riservatezza**)
- $F_O \cap F_S = \emptyset$ (**non ridondanza**, non è essenziale a fini di sicurezza)

A livello fisico i frammenti F_O e F_S hanno un attributo in comune (*tuple ID* oppure attributo-chiave) per garantire un join senza perdita di informazioni.

3.1 Valutazione delle query

Le query formulate su R devono essere tradotte in **query equivalenti** su F_O e/o F_S
$\rightarrow$ `SELECT` A `FROM` R `WHERE` C, dove C è una congiunzione di condizioni, che possono essere del tipo:

- C_O: congiunzioni che involvono solo attributi storati nel client
- C_S congiunzioni che involvono solo attributi storati nel server
- C_{SO} congiunzioni che involvono attributi storati sia nel client che nel server

$$C_o = \{\text{Disease} = \text{“Bronchitis”}\}$$

$$C_s = \{\text{YoB} = \text{“1970”}\}$$

$$C_{so} = \{\text{Name} = \text{Job}\}$$

3.1.1 Strategia Server-Client

1. **Server:** valuta C_S e ritorna il risultato al client
2. **Client:** riceve il risultato dal server con F_O (frammento dell'owner)
3. **Client:** valuta C_O e C_SO dopo aver fatto il join

```
q = SELECT SSN, YoB
    FROM Patients
    WHERE (Disease = “Bronchitis”)
          AND (YoB = “1970”)
          AND (Name = Job)
```

C_o={Disease = “Bronchitis”}
C_s={YoB = “1970”}
C_{so}={Name = Job}

```
q_s = SELECT tid,Name,YoB
      FROM F_s
      WHERE YoB = “1970”
```

```
q_so = SELECT SSN, YoB
       FROM F_o JOIN r_s
            ON F_o.tid=r_s.tid
       WHERE (Disease = “Bronchitis”) AND (Name = Job)
```

3.1.2 Strategia Client-Server

1. **Client:** valuta C_O e manda i *tuple ID* al server
2. **Server:** fa il join dell'input con F_S, valuta C_S, e ritorna il risultato al client
3. **Client:** fa il join del risultato con F_O e valuta $C_S O$

q_o = SELECT tid
FROM F_o
WHERE Disease = "Bronchitis"

q_s = SELECT tid,Name,YoB
FROM F_s JOIN r_o ON F_s.tid=r_o.tid
WHERE YoB = "1970"

q_{so} = SELECT SSN, YoB
FROM F_o JOIN r_s ON F_o.tid=r_s.tid
WHERE Name = Job

3.1.3 Strategie a confronto

- se il server **conosce o può inferire la query**, allora la strategia Client-Server rilascia informazioni

  $\rightarrow$ il server può inferire che alcune tuple sono associate a valori che soddisfano C_O

- se il server **non conosce e non può inferire la query**, allora entrambe le strategie sono sicure

  $\rightarrow$ si sceglie quella più performante, valutando per prime le condizioni più selettive

3.2 Frammentazione minima

- L'obiettivo è **minimizzare il carico di lavoro dell'owner**, gestendo F_O
- Si stabilisce una funzione peso w che prende una coppia $\langle F_O, F_S \rangle$ come input e ritorna il carico di lavoro dell'owner
- Una frammentazione $\mathcal{F} = \langle F_O, F_S \rangle$ si dice minimale sse:
  - $\mathcal{F}$ è corretta (soddisfa le proprietà di correttezza, confidenzialità e non-ridondanza)
  - $\nexists \mathcal{F}'$ tale che $w(\mathcal{F}') < w(\mathcal{F})$ e $\mathcal{F}'$ è corretto

3.2.1 Metriche per la frammentazione

Possono essere usate diverse metriche per dividere gli attributi tra F_O e F_S, per minimizzare:

- **Storage**
  - Numero di attributi in F_O (***Min-Attr***):
    $\rightarrow w_a(\mathcal{F}) = |F_O|$
  - Dimensione degli attributi in F_O (***Min-Size***):
    $\rightarrow w_s(\mathcal{F}) = \sum_{A \in F_O} size(A)$
- **Computazione/Traffico**
  - Numero di query in cui l'owner viene coinvolto (***Min-Query***):
    Si definisce un *query workload profile*:
    $\mathcal{Q} = \{(q_1, freq(q_1), Attr(q_1), \ldots, q_n, freq(q_n), Attr(q_n))\}$, con:
    - $q_1, \ldots, q_n$ query da eseguire
    - $freq(q_i)$ frequenza attesa di q_i
    - $Attr(q_i)$ attributi che compaiono nella clausola `WHERE` di q_i

    $\rightarrow w_q(\mathcal{F}) = \sum_{q \in \mathcal{Q}} freq(q)$ t.c. $Attr(q) \cap F_O \neq \emptyset$
  - Numero di condizioni nelle query in cui è necessario coinvolgere l'owner (***Min-Cond***):
    Si definisce un *query workload profile*:
    $\mathcal{Q} = \{(q_1, freq(q_1), Cond(q_1), \ldots, q_n, freq(q_n), Cond(q_n))\}$, con:
    - $q_1, \ldots, q_n$ query da eseguire
    - $freq(q_i)$ frequenza attesa di q_i
    - $Cond(q_i)$ condizioni che compaiono nella clausola `WHERE` di q_i

    $\rightarrow w_c(\mathcal{F}) = \sum_{c \in Cond(\mathcal{Q})} freq(c)$ t.c. $c \cap F_O \neq \emptyset$

3.2.2 Modellizzazione del problema di mininimzzazione

Tutti i problemi di minimizzazione mirano ad identificare un ***hitting set***; metriche differenti corrispondo a criteri differenti secondo cui l'*hitting set* deve essere minimizzato.
Tutti i criteri sono rappresentati in modo uniforme con il seguente modello:

- **target set:** elementi (attributi, query, condizioni) su cui è definito il problema di minimizzazione
- **funzione peso:** funzione che associa un peso a ciascun elemento
- **peso di un set di attributi:** somma dei pesi dei target che intersecano con il set

→ si vuole computare un hitting set con peso minimo

3.2.3 Algoritmo Euristico

- **Input:**
  - $\mathcal{A}$: set di attributi che non appaiono in *singleton constraint*
  - $\mathcal{C}$: set di constraint ben definiti
  - $\mathcal{T}$: set di target
  - w: funzione peso definita su $\mathcal{T}$
- **Output:**
  - $\mathcal{H}$: set di attributi che, uniti a quelli che appaiono in *singleton constraint*, formano F_O
  - F_S computato come R/F_S
- **Struttura Dati:** *Priority-queue* PQ con un elemento E per ogni attributo:
  - $E.A$: attributo
  - $E.C$: puntatore a constraint non soddisfatti che contengono $E.A$
  - $E.T$: puntatore ai target che non intersecano $\mathcal{H}$ che contengono $E.A$
  - $E.n_c$: numero di constraint puntati da $E : C$
  - $E.w.$ peso totale dei target puntati da $E.T$

  → la priorità è dettata da $E.w/E.n_c$, ovvero il miglior rapporto *costo / numero di vincoli risolti*

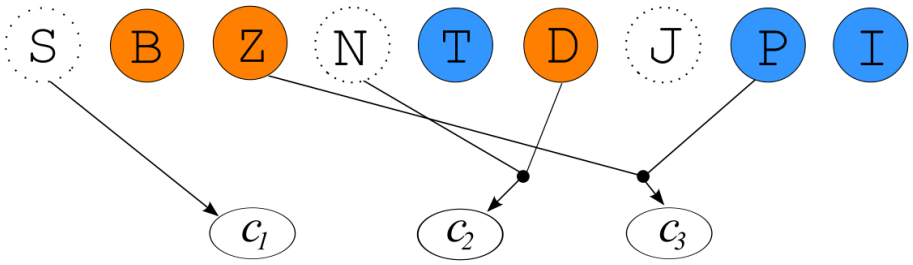
Procedura

- `while` $PQ \neq \emptyset$ e $\exists E \in PQ$ t.c. $E.n_c \neq 0$
  - estrai da PQ elemento E con $E.w/E.n_c$ minimo
  - inserisci $E.A$ in $\mathcal{H}$
  - $\forall c$ puntato da $E.C$, rimuovi i puntatori a c da ogni elemento E', ed aggiorna $E'.n_c$
  - $\forall t$ puntato da $E.T$, rimuovi i puntatori a t da ogni elemento E', ed aggiorna $E'.w$
  - aggiusta PQ secondo i nuovi $E.w/E.n_c$
- `for` $A \in \mathcal{H}$
  - se $\mathcal{H}/A$ è un *hitting set* per $\mathcal{C}$, allora rimuovi A da $\mathcal{H}$

3.3 Frammentazione e inferenza

La frammentazione assume che gli attributi siano tra loro indipendenti; in caso di dipendenza tra attributi, potrebbe verificarsi l'esposizione indiretta di attributi/associazioni sensibili.

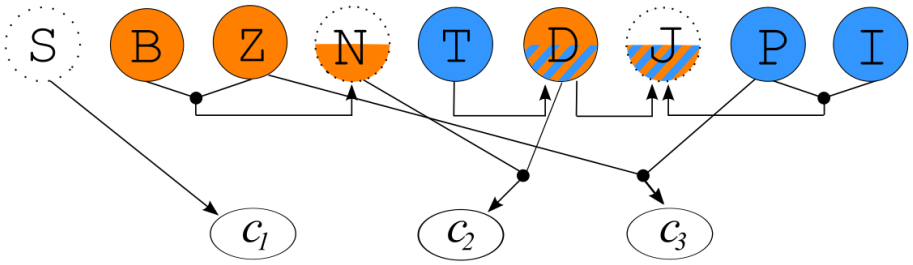
R(SSN, Birth, ZIP, Name, Treatment, Disease, Job, Premium, Insurance)



Constraints

c_1 = {SSN}
c_2 = {Name, Disease}
c_3 = {ZIP, Premium}

R(SSN, Birth, ZIP, Name, Treatment, Disease, Job, Premium, Insurance)



Constraints

c_1 = {SSN}
c_2 = {Name, Disease}
c_3 = {ZIP, Premium}

Dependencies

d_1 = {Birth, ZIP}$\rightsquigarrow$ Name
d_2 = {Treatment}$\rightsquigarrow$ Disease
d_3 = {Disease}$\rightsquigarrow$ Job
d_4 = {Insurance, Premium}$\rightsquigarrow$ Job

Avere attributi in comune significa essere linkabili; ma non è vero che essere linkabili significhi per forza dire avere attributi in comune.
I frammenti non dovrebbe contenere attributi/associazioni sensibili **né direttamente che indirettamente.**

Capitolo 4

Pubblicazione di Associazioni Offuscate

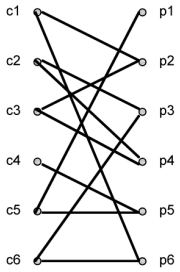
4.1 Anonimizzazione di Grafo Bipartito

Si *maschera* il mapping tra entità e nodi del grafo, conservando la struttura del grafo (e quindi delle relazioni).

Customer	State
c1	NJ
c2	NC
c3	CA
c4	NJ
c5	NC
c6	CA

Product	Avail
p1	Rx
p2	OTC
p3	OTC
p4	OTC
p5	Rx
p6	OTC

Customer	Product
c1	p2
c1	p6
c2	p3
c2	p4
c3	p2
c3	p4
c4	p5
c5	p1
c5	p5
c6	p3
c6	p6



Pubblicando una versione anonimizzata del grafo bipartito si può rispondere a diverse query:

- **Tipo 0 - Struttura del grafo**
  Qual è il numero medio di prodotti comprati?
- **Tipo 1 - Predicati su attributi di un solo lato**
  Qual è il numero medio di prodotti comprati da clienti del NJ?

- **Tipo 2 - Predicati su attributi di entrambi i lati**

  Qual è il numero medio di prodotti OTC acquistati dai clienti del NJ?

La privacy delle associazioni viene rispettata.

4.1.1 *(k, l)* grouping

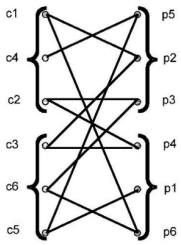
L'idea è quella di preservare la struttura del grafo mappando le singole entità a nodi:
→ **(k, l) grouping di un grafo bipartito G=(V,W,E)**

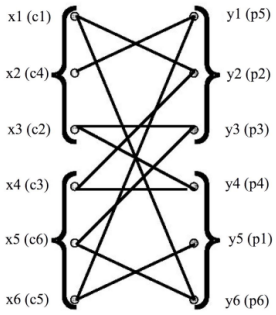
- Si partiziona V (W, rispettivamente) in sottoinsiemi che non intersecano tra loro di dimensione $\leq k$ (l, rispettivamente)
- Si pubblicano gli archi E', isomorfi ad E, dove la mappatura da E a E' è anonimizzata grazie alle partizioni di V e W

Customer	State
c1	NJ
c2	NC
c3	CA
c4	NJ
c5	NC
c6	CA

Product	Avail
p1	Rx
p2	OTC
p3	OTC
p4	OTC
p5	Rx
p6	OTC

Customer	Product
c1	p2
c1	p6
c2	p3
c2	p4
c3	p2
c3	p4
c4	p5
c5	p1
c5	p5
c6	p3
c6	p6





x1	y2
x1	y6
x2	y1
x3	y3
x3	y4
x4	y2
x4	y4
x5	y3
x5	y6
x6	y1
x6	y5

E'

Customer	Group
c1	CG1
c2	CG1
c3	CG2
c4	CG1
c5	CG2
c6	CG2

H_V

Product	Group
p1	PG2
p2	PG1
p3	PG1
p4	PG2
p5	PG1
p6	PG2

H_W

X-node	Group
x1	CG1
x2	CG1
x3	CG1
x4	CG2
x5	CG2
x6	CG2

R_V

Y-node	Group
y1	PG1
y2	PG1
y3	PG1
y4	PG2
y5	PG2
y6	PG2

R_W

Per fare dei **raggruppamenti sicuri**, i nodi nello stesso gruppo di V non dovrebbero essere connessi a uno stesso nodo di W.

4.2 Frammenti e *Loose Associations*

Per incrementare l'utilità delle informazioni pubblicate, si possono fare delle associazioni tra gruppi di valori.

4.2.1 Frammentazione Corretta e Minima

- Una frammentazione è **corretta** se:
  - ogni constraint di confidenzialità è soddisfatto da **tutti** i frammenti
  - ogni requisito di visibilità è soddisfatto da **almeno** un frammento
  - i frammenti non hanno attributi in comune
- Una frammentazione è **minimale** se:
  - il numero di frammenti è **minimo**

4.2.2 *Loose Associations*

Dati due frammenti F_l e F_r, una *loose association* tra F_l e F_r:

- partiziona le tuple dei frammenti in **gruppi**
- dà informazioni sulle associazioni a **livello di gruppo**
- **non permette di ricostruire** esattamente le associazioni originali tra tuple
- aumenta l'utilità dei dati pubblicati

4.2.3 Grouping

- Data l'istanza f_i di un frammento F_i, un k-grouping partiziona le tuple in gruppi di dimensione $\geq k$
- Un $k-grouping$ è minimale se massimizza il numero di gruppi
- La notazione (k_l, k_r)-grouping denota i gruppi su f_l e f_r
- (k_l, k_r)-grouping è minimale se sia k_l-grouping che k_r-grouping sono minimali

Esempio - minimal (2,2)-grouping

Birth	City	Illness	Doctor
56/12/9	Rome	diabetes	David
53/3/19	Paris	gastritis	Daisy
58/5/18	Oslo	flu	Damian
53/12/9	Oslo	asthma	Daniel
56/12/9	Rome	gastritis	Dorothy
57/6/25	Paris	obesity	Drew
53/12/1	NY	measles	Dennis
60/7/25	Rome	diabetes	Daisy

c_0={SSN}
c_1={Patient,Illness}
c_2={Patient,Doctor}
c_3={Birth,City,Illness}
c_4={Birth,City,Doctor}

F_l

Birth	City
53/3/19	Paris
53/12/9	Oslo
56/12/9	Rome
57/6/25	Paris
58/5/18	Oslo
56/12/9	Rome
53/12/1	NY
60/7/25	Rome

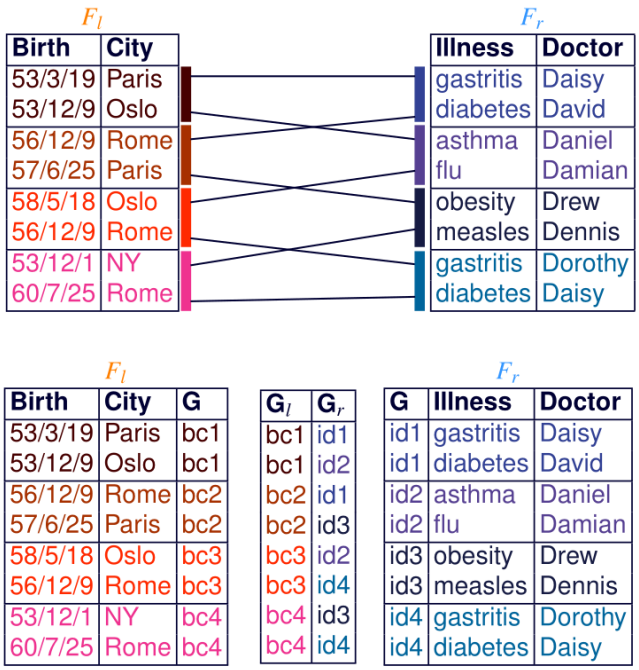
F_r

Illness	Doctor
gastritis	Daisy
diabetes	David
asthma	Daniel
flu	Damian
obesity	Drew
measles	Dennis
gastritis	Dorothy
diabetes	Daisy

Associazione tra Gruppi

(k_l, k_r)-grouping induce una associazione A tra gruppi di f_l e f_r;
definiamo una associazione A come un set di coppie di *identificatori di gruppo* tale che:

- A ha la stessa cardinalità della relazione originale
- ogni tupla nella relazione originale è mappata in modo biietivo ad una coppia (l, r) con $l \in f_l$ e $r \in f_r$



4.2.4 *k*-loose association

Una associazione tra gruppi si dice k**-loose** se ogni tupla corrisponde indistintamente ad almeno k associazioni tra tuple nei frammenti.

4.2.5 *Alikeness*

Due tuple sono ***alike*** rispetto a un constraint di confidenzialità c, denotata come $l_i \simeq_c l_j$ se:

- c è coperto da $F_l \cup F_r$
- l'intersezione di c sulle due tuple ha uguale valore

Birth	City	Illness	Doctor
56/12/9	Rome	diabetes	David
53/3/19	Paris	gastritis	Daisy
58/5/18	Oslo	flu	Damian
53/12/9	Oslo	asthma	Daniel
56/12/9	Rome	gastritis	Dorothy
57/6/25	Paris	obesity	Drew
53/12/1	NY	measles	Dennis
60/7/25	Rome	diabetes	Daisy

c_0={SSN}
c_1={Patient,Illness}
c_2={Patient,Doctor}
c_3={Birth,City,Illness}
c_4={Birth,City,Doctor}

F_l

Birth	City
56/12/9	Rome
53/3/19	Paris
58/5/18	Oslo
53/12/9	Oslo
56/12/9	Rome
57/6/25	Paris
53/12/1	NY
60/7/25	Rome

F_r

Illness	Doctor
diabetes	David
gastritis	Daisy
flu	Damian
asthma	Daniel
gastritis	Dorothy
obesity	Drew
measles	Dennis
diabetes	Daisy

$\simeq_{c_4}$ $\simeq_{c_3}$

4.2.6 Proprietà di Eterogeneità

C'è una relazione tra k_l, k_r e il grado di k-looseness:

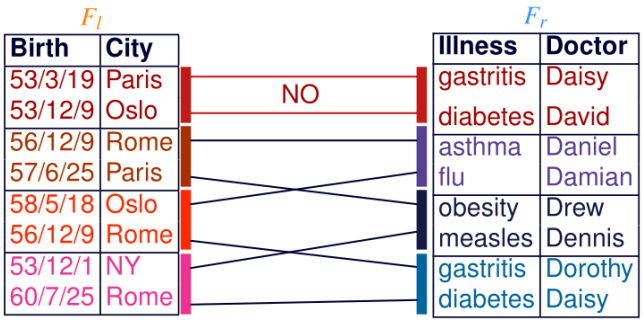
- un (k_l, k_r)-grouping può indurre una k-loose association con al massimo $k = k_l * k_r$
- il valore $k \leq k_l * k_r$ dipende da come sono definiti i gruppi
- Se un (k_l, k_r)-grouping rispetta le proprietà di eterogeneità, la relazione tra gruppi sarà k-loose con $k = k_l * k_r$

Eterogeneità dei gruppi

Nessun gruppo può contenere tuple che sono *alike*
→ assicura diversità tra le tuple all'interno dei gruppi

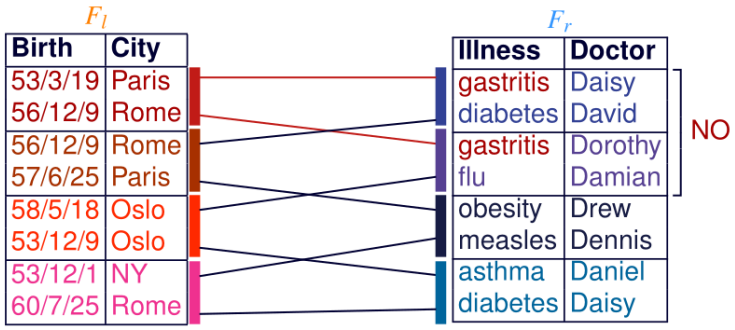
Eterogeneità delle associazioni

Nessun gruppo può essere associato due volte allo stesso gruppo.
→ assicura che per ogni tupla della relazione originale, ci siano almeno $k_l * k_r$ coppie a cui potrebbe corrispondere



Eterogeneità profonda

Nessun gruppo può essere associato a due gruppi che contengono tuple *alike*.
$\rightarrow$ assicura che tutte le $k_l * k_r$ a cui ogni tupla potrebbe corrispondere abbiano valori diversi (per quelli coinvolti nei constraint)



4.2.7 *Flat grouping* vs *Sparse grouping*

Un (k_l, k_r)-grouping si dice:

- *piatto* se uno tra k_l o k_r è uguale a 1
- *sparso* se sia k_l che k_r sono diveri da 1

4.2.8 Privacy vs Utilità

- La pubblicazione delle *loose associations* incrementa l'**utilità** dei dati; permette di valutare le query in modo più preciso rispetto a se solamente i frammenti fossero pubblicati
- Una maggiore utilità corrisponde anche ad una maggiore **esposizione** (meno sicurezza)