Protezione e Integrità dei Dati nel Cloud

Parte V

Indice

Capitolo 1

Encryption

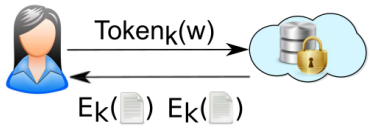
Il *server* potrebbe essere ***honest-but-curious***, non dovrebbe avere accesso alle risorse; voglio garantire confidenzialità anche rispetto a lui.

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.

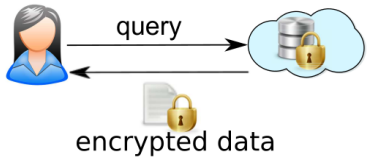
Di base voglio avere una criptazione dei dati; il problema è il **bilanciamento tra protezione e funzionalità**, ovvero sulle *query* che è possibile fare sui dati.

Approcci per accesso a diversi livelli di granularità

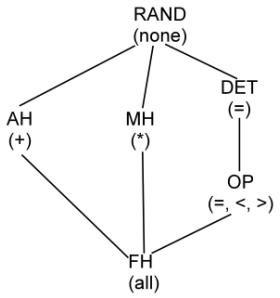
- ***Keyword-based searching:*** passo un *token* già criptato che viene usato per fare ricerca sui dati criptati (voglio trovare dove c'è una certa parola/espressione booleana)



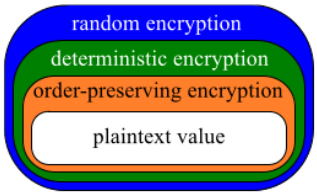
- **Crittografia omomorfica:** crittografia che supporta le operazioni direttamente sul cifrato

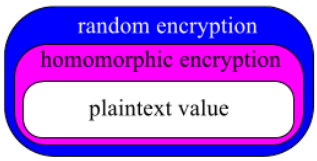


- ***Encryption Schemas:*** ogni colonna può essere cifrata con un diverso schema crittografico (*random, add homomorphic, deterministic, order preserving, . . .*)



- ***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$^{k}_{I}$

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

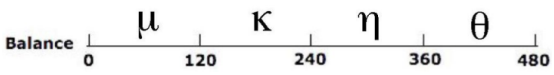
SSN	Name	Illness	Doctor
123…89	Alice	Asthma	Angel
234…91	Bob	Asthma	Angel
345…12	Carol	Asthma	Bell
456…23	David	Bronchitis	Clark
567…34	Eva	Gastritis	Dan
232…11	Eva	Stroke	Ellis

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:***

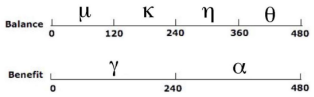
1. si partiziona il dominio di un attributo
2. a ciascuna partizione si assegna un'etichetta
3. il valore in chiaro viene sostituito dall'etichetta



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$^{\mathbf{k}}_{\mathbf{2}}$

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

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

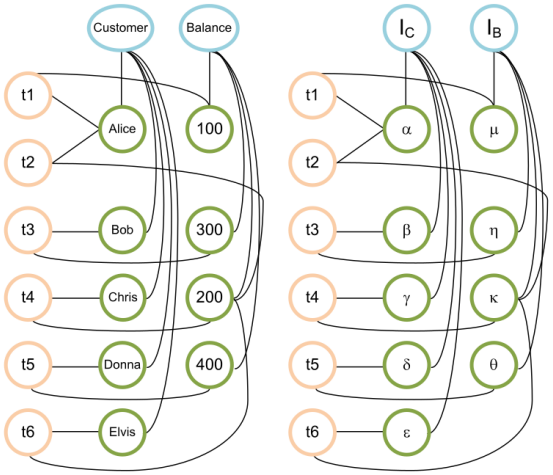
- nella tabella *Quotient* ci sono le classi di equivalenza a cui appartengono gli indici
- nella tabella *Inverse Cardinality* c'è $1/|C|$, si interpreta come:
  - *c'è 1 di 6 valori che non so distinguere*
  - *c'è 1 di 4 valori che non so distinguere*
  - Sta esprimendo l'incertezza; più sarà grande $|C|$, 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

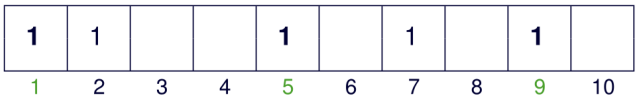
- i grafi RCV tra dati in chiaro e criptati non sono uguali, dato che *vertici diversi* nel grafo in chiaro potrebbro collassare nello *stesso vertice* nel grafo criptato
- il numero di archi che collega i vertici *row* ai vertici *value* è lo stesso
- il problema diventa trovare un *matching corretto*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

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.

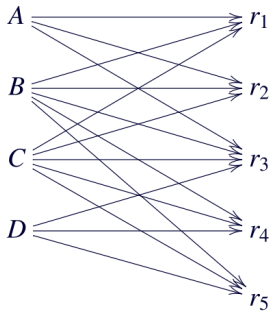
- Una politica di autorizzazione $\mathcal{A}$ è un set di permessi della forma $\langle user, resource \rangle$

  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 $\mathcal{K}$

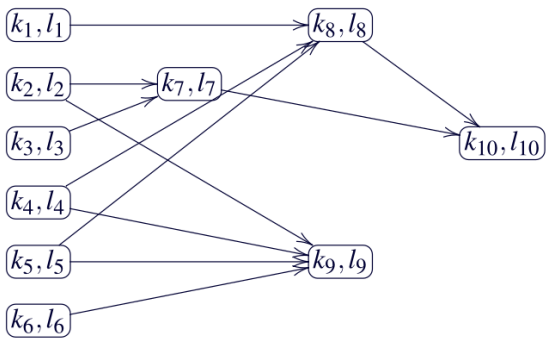
- $(\mathcal{K}, \leq)$ può essere rappresentata come un grafo con un vertice per ogni $x \in \mathcal{K}$ e un percorso da x a y sse $y \leq x$

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

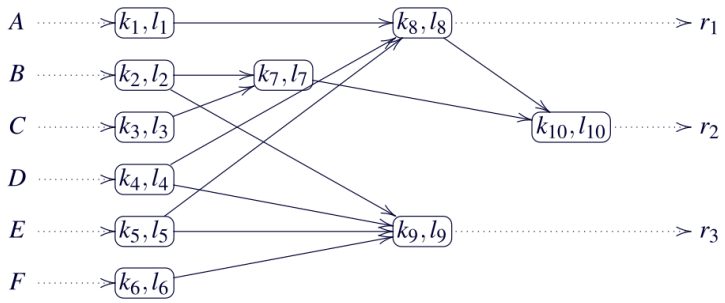
Definzione Formale della Politica Crittografica

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 $\{r_3\}$