**Privacy and Data Protection in Emerging Scenarios**

Per proteggere la confidentiality dei dati si ricorre a:

1. Encryption
2. Frammentazione
3. Encryption e frammentazione

**Encryption**

L'encryption ci protegge da CSP honest but curious, cioè cerco di proteggere la confidenzialità dei dati anche se il CSP svolge il proprio dovere. Un livello di encryption viene applicato ai dati sensibili per poterli proteggere.

Ovviamente il CSP non può conoscere la chiave per la decryption, mi servono metodi efficaci per proteggere i dati.

1. Keyword-based searches directly on the encrypted data
2. Homomorphic encryption: supports the execution of operations directly on the encrypted data
3. Encryption schemas (su ogni colonna)/onion encryption: a diversi livelli di encryption corrispondono diverse operazioni, problema: una volta tolta l’encryption non si può più riapplicare.

Sono possibili varie operazioni a seconda del layer es: deterministic (x = y -> e(x) = e(y)) order preserving (x > y -> e(x) > e(y)). Random = nessuna funzione supportata

(det = equality queries, OP = query di range)

1. Indici: metadati allegati alle tuple criptate che servono per interrogare il CSP mantenendo i dati criptati.

Query evaluation process: un client (trusted) crea una query che viene tradotta tramite translator e passata tramite query engine al CSP, questa query prendera i dati criptati dal CSP e li riporterà al client dove verrano decriptati e sulla quale verrà ri-eseguita la query originale per eliminare eventuali tuple spurie.

Possibili valori degli indici:

1:1 perfetto per equality queries ma permette inferenza (Direct)

N:1 supporta le equality queries e presenta collisione per rimuovere la distinguibilità, comunque soggetto a inferenza (Bucket)

1:N: (Flattened) diminuisce inferenza ma è comunque possibile fare attacchi di osservazione dinamica

Partition-based index: il dominio dell’attributo viene diviso in partizioni e ad ogni partizione viene dato un id, i valori del dominio dell’attributo verranno mappati all’interno di questi intervalli, la mappatura può essere order preserving oppure random.

Il partition bases index permette query di confronto booleane del tipo: attributo op. Valore oppure attributo op. Attributo, dove op. È un’operazione booleana < > = >= <=.

• Ai = v. The mapping is defined as: Mapcond(Ai = v) =⇒ Ii =MapAi (v)

Mapcond(Balance< 200) =⇒ IBalance ∈ {µ,κ} (mappa balance ad avere un certo valore)

Ai = Aj: vengono considerate tutte le combinazioni che si sovrappongono negli intervalli: (IBalance=µ ∧ IBenefit=γ) ∨ (IBalance=κ ∧ IBenefit=γ) ∨ (IBalance=η ∧ IBenefit=α) ∨ (IBalance=θ ∧ IBenefit=α)

Ai < Aj dipende se le mapping sono order preseverving oppure no.

Procedimento:

• Each query Q on the plaintext DB is translated into: ◦ a query Qs to be executed at the server ◦ a query Qc to be executed at client on the result • Query Qs is defined by exploiting the definition of Mapcond(C) • Query Qc is executed on the decrypted result of Qs to filter out spurious tuples

Hash-based index: Si basano sul concetto di hash: one way function.

Important properties of any secure hash function h are:

◦ ∀x,y ∈ Di : x = y =⇒ h(x) = h(y) (**determinism**)

◦ given two values x,y ∈ Di with x != y, we may have that h(x) = h(y) (**collision**)

◦ given two distinct but near values x,y (| x−y |< ε) chosen randomly in Di , the discrete probability distribution of the difference h(x)−h(y) is uniform (**strong mixing**) (distribuzione di probabilità sulla differenza dei valori di hash è uniforme)

Le query supportato sono di tipo: attributo = valore, attributo = attributo (solo se la hash function è la stessa) (questo perché l’hashing non è order preserving).

La query translation funziona allo stesso modo del partion based index.

Interval-based queries: gli indici che permettono query di range hanno maggiore esposizione ad attacchi di inferenza.

Le interval base queries sono supportato grazie all’utilizzo di B-tree (quelli con valori solo nelle foglie) però quelli del server non possono essere utilizzati perciò:

O encripto ogni nodo dell’albero e lo metto sul server o eseguo la navigazione dell’albero quando sono al client.

Se devo mettere l’albero sul server, encripto il contenuto e tengo libero l’id dei nodi

Se devo portarmi l’albero a casa per fare la navigazione su un trusted client allora devo ricostruirmi l’albero, se mi serve Bob che è una foglia, devo prima prendermi la root, poi il nodo intermedio e poi la foglia dove c’è Bob, così avverrà la query su bob. (range query).

**Searchable encryption:**

Order Preserving Encryption Schema (OPES): prende in input una distribuzione e applica una trasformazione order preserving agli indici forniti in modo che seguano la distribuzione target

Order Preserving Encryption with Splitting and Scaling (OPESS): i valori degli indici seguono una distribuzione flat (piatta)

Fully homomorphic encryption schema: le operazioni vengono svolte sui dati criptati e portano agli stessi risultati che fare tali operazioni sui dati plaintext. (troppo computaion intensive)

**Inference exposure**

Ci sono due richieste constrastanti nell’utilizzo di index:

1. Si vuole che le query forniscano più informazione possibile
2. Non si vuole che i dati siano esposti a linking ed inference attacks

ε = Exposure Coefficient.

The computation of the exposure coefficient ε depends on two factors:

• the indexing method adopted, e.g., ◦ direct encryption ◦ hashing

• the a-priori knowledge of the intruder, e.g.,

◦ Freq+DBk : − the frequency distribution of plaintext values in the original database (Freq) − the encrypted database (DBk )

◦ DB+DBk : − the plaintext database (DB) − the encrypted database (DBk )

Le inferenze che possono essere fatte riguardano associazioni di tuple da enc a plain o associazioni di indici a valori plaintext.

Direct encryption – freq+DBk

La correlazione tra indice e valore viene determinata dal conteggio delle occorrenze, una semplice protezione è: i valori con le stesso numero di occorrenze sono indistinguibili dall’attaccante, es: tutti gli id hanno occorrenza 1, se ho 6 id ogni id avrà prob di correlazione pari a 1/6, hanno tutti la stessa occorrenza, se invece avessi un id ripetuto due volte, esso verrebbe determinato subito.

Exposure of values in equivalence class C is 1/ | C |

Una classe C è un gruppo di elementi in cui tutti hanno la stessa occorrenza, se gli elementi hanno occorrenza 1 sono nella classe occorrenza 1 se hanno occorenza 2 nella classe occorrenza 2 ecc..

|C| = numero di classi con la stessa occorrenza.

Direct encryption – DB+DBk

Autotomorfismo =?

Equitable partition = nodi che possono essere scambiati con altri, sono cioè equivalenti

ε = 6/9 , 6 partizioni su 9 nodi, 6 gruppi all’interno dei quali i nodi possono essere scambiati.

(rivedere veloce videolez)

**Bloom Filter**

Caratteristiche:

• Set of n elements (n is large)

• Vector of l bits (l is small)

• h independent hash functions Hi : {0,1} ∗ → [1,l]

Le hash functions sono h con valori che vanno da 1 a l, il vettore nella posizione in cui viene computato il valore viene messo 1 altrimenti 0

Insert element x: • Sets to 1 the bit values at index positions H1(x),H2(x),...,Hh(x)

Search element x: • Compute H1(x),H2(x),... ,Hh(x) and check whether those values are set in the bit vector

Richiede poco spazio ma gli svantaggi sono che possono verificarsi falsi positivi(elementi che mi dice che sono presenti anche se mai inseriti) e non ha la possibilità di rimuovere elementi.

**Data Integrity**

Due aspetti fondamentali:

1. Integrity in storage (unathourized update)
2. Integrity nella computazione di queries (server misbehaviour)

Integrity in storage: si affida alle firme digitali tipicamente computate a livello di tupla, utilizzate anche per le query

**Selective Encryption and Over-Encryption**

Problema: voglio avere utenti differenti che accedono a viste differenti, ma voglio che il data owner rimanga sempre in controllo di tutto.

Selective information sharing: Approaches:

ABE: Attribute based encryption: La derivazione della key dipende da certe caratteristiche dei soggetti, è basata sul paradigma di asymmetric cryptography.

Selective encryption: la politica di autorizzazione viene trasformata in una politica di encryption di pari livello (stessi risultati).

Ad ogni chiave corrisponde una certa view sui dati, sono presenti chiavi differenti per ruoli e views differenti.

Il data owner crea una authorization policy del tipo <user, risorsa> può essere rappresentata come: access matrix oppure come grafo bipartito. (user da una parte e risorse dall’altra e arco per indicare permesso)Quindi si avrebbe una key different per ogni ACL; ACL = RIGA ACCESS MATRIX

Acces matrix = righe users, colonne risorse.

Per passare da una authorization policy ad una encryption policy si utilizza un metodo di derivazione di chiave, quindi tramite la user key in base ai permessi che si hanno si possono derivare altre chiavi per accedere ad altre risorse.

Nel caso volessi criptare ogni risorsa con chiavi diverse dovrei dare ad ogni utente così tante keys come le risorse a cui può accedere.

• Based on a key derivation hierarchy (K ,) ◦ K is the set of keys in the system ◦ partial order relation defined on K • The knowledge of the key of vertex v1 and of a piece of information publicly available allows the computation of the key of a lower level vertex v2 such that v2 v1 • (K ,) can be graphically represented as a graph with a vertex for each x ∈ K and a path from x to y iff y x

• Keys are arbitrarily assigned to vertices • A public label li is associated with each key ki • A piece of public information ti,j , called token, is associated with each edge in the hierarchy.

Nel grafo un arco è presente solo se è presente un token tra le due chiavi per permetterne la derivazione.

Ad ogni user è data solo una key ed ogni risorsa è criptata solamente con una key.

• An encryption policy over users U and resources R, denoted E , is a 6-tuple hU ,R,K ,L ,φ,T i, where: ◦ K is the set of keys defined in the system and L is the set of corresponding labels ◦ φ is a key assignment and encryption schema ◦ T is a set of tokens defined on K and L

• The encryption policy can be represented via a graph by extending the key and token graph to include: ◦ a vertex for each user and each resource ◦ an edge from each user vertex u to the vertex hk,li such that φ(u)=l ◦ an edge from each vertex hk,li to each resource vertex r such that φ(r) = l

Policy transformation: trasformare una authorization policy in una encryption policy:

Le due polici sono equivalenti se permettono gli stessi accessi.

Per trasformare le due policy posso applicare un’approccio naive in cui genero un token differente per ogni permesso possibile, però posso sfruttare un’approccio migliore in cui al posto di generare un token per utente e risorsa genero un token per gruppo di utenti che hanno gli stessi permessi (molto più menageble).

Se però faccio una key per ogni gruppo creo keys inutili, mi basta creare una key per ogni gruppo che è associato con una ACL. -> minimum encryption policy.

Viene computata tramite degli algoritmi euristici, possono essere aggiunti anche nodi intermedi che mi permettono di ottimizzare il numero di keys che posso creare per la derivazione di keys.

Steps per la creazione della minimum enc policy:

Start from an authorization policy A

1. Create a vertex/key for each user and for each non-singleton acl (initialization)

2. For each vertex v corresponding to a non-singleton acl, find a cover without redundancies (covering) - for each user u in v.acl, find an ancestor v ′ of v with u ∈ v ′ .acl (dal fondo al sopra)

3. Factorize common ancestors (factorization)

Il problema principale della selective encryption è il seguente: se le autorizzazioni cambiano si ha la necessità di scaricare i dati, decriptarli, creare una nuova key , recriptarli e dare la nuova key agli utenti e fare l’upload dei dati, MOLTO inefficiente.

Perciò si utilizza la over-encryption per permettere changes dinamiche

**over-encryption**

• Resources are encrypted twice ◦ by the owner, with a key shared with the users and unknown to the server (Base Encryption Layer - BEL level) ◦ by the server, with a key shared with authorized users (Surface Encryption Layer - SEL level)

• To access a resource a user must know both the corresponding BEL and SEL keys

• Grant and revoke operations may require ◦ the addition of new tokens at the BEL level ◦ the update of the SEL level according to the operations performed

A livello BEL ogni nodo possiede due chiavi: una per la derivazione di altre keys e una per l’accesso alle risorse. Oltre ad avere ovviamente un paio di labels.

SEL è caratterizzato da una encryption policy.

Full\_SEL: starts from a SEL identical to the BEL and keeps the SEL always updated to represent the current policy

Delta\_SEL: starts from an empty SEL and adds elements to it as the policy evolves, such that the pair BEL-SEL represents the policy.

L’evoluzione di BEL e SEL dipendono da operazioni di grant e di revoke. (algoritmi di over-ecnryption)

Grant = over encrypto risorse alle quali non voglio che il nuovo utente acceda e poi over encrypto tutti quelli che possono accedere alla risorsa

Revoke = over encripto solo chi ci può accedere.

Protection evaluation: vulnerable to collusion attack : server and users, only users is not a problem.

La collusione avviene quando due individui mettono insieme le loro conoscenze per ottenere dei risultati ai quali singolarmente non potevano arrivare, mentre insieme riescono ad arrivarci.

• Four views: ◦ open: the user knows the key at the BEL level as well as the key at the SEL level ◦ locked: the user knows neither the key at the BEL level nor the key at the SEL level ◦ sel\_locked: the user knows only the key at the BEL level but does not know the key at the SEL level ◦ bel\_locked: the user knows only the key at the SEL level but does not know the one at the BEL level

Posso avere collusione sul SEL-locked perché io utente ho accesso e quindi ho lo strato BEL, mi metto insieme col server che mi fornisce lo strato SEL e insieme riusciamo a fare collusione e a reperire l’informazione. Se riesce ad accedere a qualcosa a cui aveva accesso prima (colludendo) non è un problema.

Con full sell si ha la sincronizzazione server e client quindi o parto con entrambi open o con entrambi locked. Quello che può succedere è con grant e revoke arrivo in una situazione sel locked.

Questo succede quando ho una policy split. Prima erano criptati con la stessa chiave, ora invece devo separare. Grant mi espone.

Nel delta\_sel non parto sincronizzato: open oppure bel\_locked(gli chiedo protezione) il problema è sempre il policy split (grant), se ho già la mia protezione, il server non aggiunge la propria, se ho bel\_locked il server non aggiunge il sel.

Quando faccio la grant chiedo al server di farmi over encryption su r4,r5 quindi di proteggere quelle risorse in modo che D non possa accedere e possa accedere solo a r3, ma su r3 ho già la protezione BEL quindi il server non aggiunge la propria, la aggiunge solo a quelle sulla quale non posso accedere. Problema è solo quando la collusione mi da accesso a qualcosa che non potevo accedere prima, non qualcosa di past.

Con Delta\_sel può essere collusione un utente con sé stesso (caso in cui l’utente si tiene il db) questo perché in delta sel posso partire con solo BEL locked invece in full sel parte o completely locked oppure open (quindi se parto bel\_locked posso io utente scaricarmi tt il db e successivamente fare collusione con me stesso). Quindi non posso avere questa situazione con Full\_sel (però ho comunque policy split). Mentre in delta ho entrambi i casi.(utente con se stesso e policy split)

**Fragmentation and Encryption**

L’encryption a volte non permette di fare queries o comunque di spendere più risorse per la computazione del risultato, è però importante notare che a volte non sono i valori ad essere sensibili ma le associazioni tra i valori. Perciò si punta a frammetare e quindi a rompere queste associazioni per proteggere i dati.(l’encryption è ancora necessaria ma meno utilizzata).

Attributi sensibili: sensibili singolarmente (singleton constraint)

Associazioni sensibili: ciò che è sensibile non è il dato in sé ma l’associazione con altri dati.

Non-communicating pair of servers:

L’informazione viene divisa su due server che non comunicano, l’uno non deve essere a conoscenza dell’altro. L’encryption viene tenuta per gli attributi sensibili o se non si può fare altro perché verrebbero a crearsi comunque associazioni da proteggere.

Le associazioni sono sensibili se hanno elementi in comune (esempio id in comune sul quale fare join). Questo id in comune è presente sui due server indipendenti in modo da porter ricostruire l’associazione quando si va a fare una query e non escano tuple spurie.

Tutto il confidentiality contraint in chiaro su un solo server non può esserci (definisce protezione).

<R1, R2, E> <- conf constraint.

Query execution: Le selezioni e le proiezioni le spingo in basso il più possibile, il lavoro viene fatto ai nodi e quindi diminuisce il risultato che porta a casa.

Posso fare due subqueries in parallelo sui due server e unire i risultati. Oppure valuto una condizione su uno dei due servers, mi porto a casa i risultati e poi valuto la condizione dall’altra parte. (utilizzando i risultati già ottenuti) (semijoin).

La soluzione non parallela è peggio dal punto di vista della privacy perché posso fare leakage di informazioni (do il tuple id al server quindi può inferire che ad esempio quelle tuple sono dottori ecc... Non si parlano però è comunque un leakage possibile perché si ha assuzione di osservabilità)

Se la query è nota l’altro server sa a cosa corrispondono le tuple (così sono fritto ). Se non si sa la query sai che quelle tuple soddisfano tutte una certa condizione che però non sai qual è.

Identifying the optimal decomposition:

Devo utilizzare le affinity matrix: misura quanto un attributo è affine con un altro, se li separo pago il prezzo della affinity matrix (quando li separo e li metto in 2 server diversi). <-- caso un attributo con un altro.

Caso attributo con sé stesso mi dice quanto costa piazzare quell’attributo perché devo criptarlo, non posso scegliere altrimenti, violerei le constraints su tutti e due i server altrimenti. (sarebbe quello sul quale faccio meno selezioni che preferisco criptare).

Ciò che devo minimizzare è la somma degli Mi,i e per gli Mi,j dove mi,i sono gli attributi che sono stato costretto a criptare e mi,j è per tutti quegli i e quei j che sono stato costretto a dividere sui server e quindi non ho potuto tenere insieme.

Può essere visto come un hypergraph coloring problem: • attributes are vertexes • affinity value Mi,j =⇒ weight of arc (i,j) • affinity value Mi,i =⇒ weight of vertex i • confidentiality constraints C represent a hypergraph H(R,C ) on the same vertexes

Colori diversi significano server diversi. In un hypergraph un arco può collegare più nodi al contrario del grafo normale in cui un arco collega solamente 2 nodi. Il problema consiste nel colorare i vertici in modo che stiano o su un server o sull’altro (NP Hard).

Multiple non-linkable fragments :

Al posto che avere due server che non comunicano, l’idea è creare più frammenti sullo stesso server che non siano linkabili. (tuple id)

Ogni frammento deve soddisfare la constraint e i frammenti non devono essere linkabili (non devono avere attributi in comune).

Ogni frammento contiene gli attributi del frammento in chiaro e tutti gli altri attributi in modo criptato. Se il frammento è composto da due attributi, quei due attributi saranno in chairo e poi avrò tutto il resto dei dati salvati in modo criptato uniti insieme a quei due attributi in chiaro.

Quindi mentre nei due server indipendenti ho i singoli attributi criptati, qua ho singoli attributi in chiaro e poi tutto il resto criptato. (con un salto per rendere le cose più difficili da osservare)

Optimization criteria: riuscire a computare query nel modo più efficiente possibile, si hanno vari approcci ma tutti puntano alla maximum visibility: se un attributo è sensibile(singleton constraints) preso singolarmente allora lo cripto (attributo sensibile ex: SSN) altrimenti divido l’associazione che lo rende sensibile e lo metto in chiaro.

Primo criterio di ottimizzazione: Minimal number of fragments:

Utilizzo algoritmi euristici altrimenti il problema è NP-Hard

◦ F is minimal if all the fragmentations that can be obtained from F by merging any two fragments in F violate at least one constraint

• iteratively select an attribute with the highest number of non-solved constraints and insert it in an existing fragment if no constraint is violated; create a new fragment otherwise

Secondo criterio : Maximum affinity:

Cerco di ottenere la divisione che mi permette di avere la affinità più alta possibile sommando tutti i frammenti, cerco di risolvere il problema con algoritmi euristici.

Iteratively combine fragments that have the highest affinity and do not violate any confidentiality constraint

Terzo criterio: Query workload:

Cerco di computare le query in modo più efficiente possibile.

query cost model: based on the selectivity of the conditions in queries and queries’ frequencies

**Fragmentation**

Keep a few: L’encryption porta overhead perciò mi sbarazzo dell’encryption tenendo una parte dei dati sul client dell’owner evitando di appunto utilizzare encryption.

Formalità su slides. 151.

Fo = frammento client, Fs = frammento server.

Chiaramente le partizioni devono avere una tid in comune per far sì che si possa appliccare un loseless join, altrimenti non riesco a collegare le informazioni.

Queries of the form: SELECT A FROM R WHERE C where C is a conjunction of basic conditions

◦ Co: conditions that involve only attributes stored at the client

◦ Cs : conditions that involve only attributes stored at the sever

◦ Cso: conditions that involve attributes stored at the client and attributes stored at the server.

Server-Client strategy

• server: evaluate Cs and return result to client

• client: receive result from server and join it with Fo

• client: evaluate Co and Cso on the joined relation

Client-Server strategy

• client: evaluate Co and send tid of tuples in result to server

• server: join input with Fs , evaluate Cs , and return result to client

• client: join result from server with Fo and evaluate Cso

Server-client = valuto cs, join al client e valuto co e cso.

Client-server = valuto co, joino al server e valuto cs, ritorno al client joino e valuto cso.

Se le quaery è conosciuta la strategia client-server può creare lekeage di informazione, se non è conosciuta allora le strategie sono uguali a livello di privacy.

Minimal fragmentation

Il goal è minimizzare il workload dell’owner. (menagment di Fo).

I principi su cui basarsi sono: il traffico oppure lo storage.

Traffico: faccio in modo di minimizzare le query in cui l’owner gestisce Fo (numero di query o numero di condizioni), storage: faccio sì che l’owner abbiamo il minor numero di attributi oppure la minor size di attributi.

Weight function: w takes a pair <Fo,Fs> as input and returns the owner’s workload.

Min-Attr: minimize the number of attributes in Fo.

Min-Size: minimize the physical size in Fo.

Min-query: minimize the number of query executions that require processing at the owner

Min-Cond: minimize the number of conditions that require processing at the owner

All the problems of minimizing storage or computation/traffic aim at identifying a hitting set ◦ Fo must contain at least an attribute for each constraint.

In sostanza tutti e 4 i casi possono essere affrontati con questa soluzione dell’hitting set.

Deve colpire tutti i target e non esistere un altro hitting set che abbia un peso più piccolo.

Min 49 lez 8/11 spiega hitting set esempio pratico.

T = target, cambia a seconda dell’obiettivo (min query, min cond , min size, min attr).

I Constraints C sono uguali per tutti. Nel mezzo ho la coda di priorità che cambia per ciascuno, varia in base agli attributi. Li tengo tutti ma in un certo ordine.

Numero in alto nella PQ = numero constraints. Numero in basso = targets. (somma dei target puntati).

Voglio tenere a casa quello che mi costa di meno, la migliore è che mi risolve di più e che costa di meno, sto utilizzando un’euristica, questa è l’euristica.

La priorità è dettata da: Quanto paghi / quello che risolvi = numerino sotto / numerino sopra. (peso rispetto ai constraints)

La priorità all’interno della coda dipende da questa euristica = quanto paghi rispetto a quanto risolvi. sotto/ Sopra.

Sono in ordine dal + basso al + alto : quelli che risolvi rispetto a quello che paghi.

Algoritmo: prendo il valore + basso e me lo porto a casa, i vincoli collegati ad esso li considero risolti; a questo punto cambio il numero relativo ai constraints e ai target puntati (è come se fossero stati consumati da ciò che ho portato a casa) degli altri elementi e aggiorno la queue;

Se ho un il primo in queue legato a 2 target e 2 constraints, me lo prendo e al giro dopo quei due target e quei due constraints non devo + considerarli quindi devo aggiornare gli altri elementi togliendo il puntatore a quei target e constraints che sono stati consumati.(diminuendo i relativi numeri) e aggiorno la queue in base a quanto pago rispetto a quanto risolvo.

Nel unlikable fragmentations sullo stesso server abbiamo detto che gli attributi non sono linkabili se non hanno attributi in comune, è davvero così? Il problema rimane la dipendenza tra i dati, quindi non è per forza detto che siano unlinkable se non hanno attr. In comune.

Es: medicina dice qualcosa su malattia, malattia dice qualcosa sul lavoro e così via, i dati hanno delle dipendenze tra loro. I frammenti non sono così unlinkable come pensavi. (attenzione a inferenze che potrebbero esserci).

Non ho mai privacy, ho il soddisfacimento delle proprietà che tu mi hai indicato.

**Publishing Obfuscated Associations (quindi riguarda frammentazione in publishing)**

Voglio poter avere una protezione relativa alle associazioni sensibili, riuscendo però a permettere certi tipi di queries.

Anonymizing Bipartite Graph:

Tipi di queries possibili:

1. Tipo 0: struttura del grafo soltanto: es numero medio di prodotti per cliente (guardo solo la topologia del grafo a prescindere dal cliente, guardo solo gli archi incidenti.)
2. Tipo 1: voglio poter fare una query che rispecchia delle condizioni da una parte del grafo (indipendente quale)
3. Tipo 2: voglio porter fare delle queries che rispecchiano delle condizioni da entrambe le parti.

**(k,l) grouping:**

Preservo la struttura del grafo (associazioni) però offuscare le informazioni a chi si riferiscono.

Raggruppo i nodi da una parte del grafo bipartito e dall’altra parte del grafo almeno k da una parte e almeno l dell’altra. Archi isomorfi = stessa topologia. Preservo gli archi

Ai clienti ed ai prodotti do degli alias le l’associazione la pubblico a livello di Alias.

L'unica cosa che non pubblico è l’associazione all’alias, però comunque pubblico il fatto che un certo utente appartiene ad un certo gruppo.

(rilascio lista clienti e in che gruppi sono, lista prodotti e in che gruppo sono, associazioni, le alias dei prodotti in che gruppo sono, le alias dei clienti in che gruppo sono).

Se faccio aliasing perché devo fare anche grouping? Per mantenere utilità se l’aliasing lo facessi a livello di tabella non potrei rispondere a query del tipo 1 e 2, mentre grazie al grouping riesco a rispondere alle query di quel tipo. (impongo condizioni). Un certo errore ce l’avrò ma dipende da come faccio grouping.

Raggruppare = Dare utilità

Safe grouping: dei nodi nello stesso gruppo V non sono collegati allo stesso nodo nel gruppo W.

Se il grouping è safe ho incertezza di W\*V

Più il gruppo è piccolo, più utilità do, però più espongo l’associazione.

L’agoritmo per risolvere questo safe grouping NP-hard è un algoritmo greedy, aggiungi un nodo ad un gruppo se è safe altrimenti produci un altro gruppo.

**Fragments and Loose Associations:**

Confidentiality constraint stessa cosa già vista o attributo sensibile (singelton) non posso pubblicarlo oppure non posso pubblicare l’associazione tra attributi (vincolo non singleton).

Visibility requirements: dipendono un po’ da chi ti sta chiedendo i dati che cosa vuole, sono formule booleano che mi indicano cosa posso o non posso pubblicare in base a ciò che ha bisogno il cliente.

Le formule booleano sono monotone, o and o or, la negazione la decide il data owner, sono i confidentiality constraints. Permettono di specificare : attributi, associazioni(più attributi (and)) o attributi in alternativa ad una certa richiesta se non è possibile darla (con OR ad esempio).

La frammentazione può essere utilizzata per i confidentiality constraints e i visibility requirements,

Devo stare attento a non dare al cliente ciò che sarebbe un attributo sensibile oppure un’associazione sensibile.

Volere le cose in vincoli di visibilità separate significa, io quelle cose le voglie ma non necessariamente in associazione, quindi facendo due fragments e mettendo dentro gli attributi richiesti proteggo l’associazione e riesco a soddisfare il mio cliente.

Per questo dare un rilascio con due fragments è diverso da fare un rilascio con l’associazione (AND) di quei fragments perché nel primo caso proteggo l’associazione nel secondo caso no. Quindi se ho due vincoli di visibilità separati posso soddisfarli facendo + fragments differenti.

• A fragmentation is correct if

◦ each confidentiality constraint is satisfied by all fragments

◦ each visibility requirement is satisfied by at least a fragment

◦ fragments do not have attributes in common (to prevent joins on fragments to retrieve associations)

• A correct fragmentation is minimal if

◦ the number of fragments is minimum (i.e., any other correct fragmentation has an equal or greater number of fragments)

• The Min-CF problem of computing a correct and minimal fragmentation is NP-hard.

Il min-CF problem viene risolto tramite SAT SOLVER. (riformulo i constraints come formule booleane, per i conf constraints basta mettere degli and tra gli attributi.).

Se itero sulle valutazioni del sat solver aggiungendo ad ogni iterazione una frammentazione, allora so che la valutazione è minimale appena ho soluzione.

Loose associations: al posto di specificare le associazioni a livello di singoli specifico l’associazione a livello di gruppo.

Proteggo in questo modo le associazioni specifiche. Pubblico a livello di gruppo rispetto che di aliasing.

Per il grouping devo cercare di rimanere ad almeno quanto specificato da k e da l, nel caso andassi sotto allora avrei un problema nel rispettare il livello di privacy che dovevo garantire, nel caso invece facessi un grouping + alto allora perderei solo utilità.

Il grouping è minimale se la quantità di gruppi che viene fatta è massimizzata (massima utilità), comunque se non riesco, cerco di stare il più vicino possibile al k indicato.

Per quando creo il grouping , l’associazione di gruppo è indotta, non la calcolo, è dipendenza di che tuple sono finite in che gruppo. (rispetto alle associazioni singole in sostanza).

Invece che pubblicare a livello di alias, pubblico a livello di grupp8o, come se ci fosse un muretto e mi aggancio al muretto.

Due tuple sono **alike** rispetto a un constraint se:

L’unione dei frammenti copre completamente il constraint (il constraint è presente nell’unione dei frammenti, quindi sarebbe un’associazione sensibile) e il frammento preso singolarmente è una intersezione con la constraint, cioè non la rappresenta del tutto.

Due tuple sono alike sullo stesso frammento se rispetto all’intersezione con il constraint sono uguali.

Quindi ad esempio F1 =Name, DoB F2 = Illness, Doctor con constraint = N,D,I

F1 U F2 copre completamente la constraint, e i frammenti sono una intersezione della contraint, due tuple sono alike se nel frammento F2 (ma può essere anche in F1) hanno lo stesso valore per illness , quindi se ho due tuple con malattia gastrite, allora quelle tuple sono alike, cioè hanno lo stesso valore di attributo per illness e quindi non rappresentano una vera incertezza a livello di grouping, in quanto è vero che sono due tuple differenti ma l’informazione che rappresentano è la stessa quindi non ho una vera e propria incertezza, Stesso concetto di l-diversity, se l’informazione che rappresento è la stessa non ho una vera incertezza tra quelle due tuple.

In sostanza mi dice che l’incertezza non deve riguardare le tuple ma i valori che ci sono dentro, allora è una vera incertezza.

Nell’esempio fatto se ho un doctor e una illness diversa allora le tuple non sono alike né rispetto l’associazione con illness ne rispetto a quella con doctor.

È importante notare che l’alikeness si basi sull’intersezione con il constraint presa in considerazione.

Cioè se prendo in considerazione solo il constraint con illness le tuple con doctort uguale non sono alike, perché doctor per quella alikeness non mi interessa. Se invece guardo l’alikeness sul doctor allora sono alike.

k-loose association: un’associazione a livello di gruppo è k-loose e ad essa possono corrispondere k associazioni. Quindi 5 tuple di qua, 10 tuple di la, tutte distinte allora ho k = 5\*10 = 50, cioè all’associazione di quel gruppo corrispondono 50 associazioni.

Se ho 4 e 10 mi aspetto una k-loose di 40 però per rispettarla devo utilizzare i 3 livelli di heterogenity per avere una k-loose association.

È una proprietà indotta, devo far in modo che raggruppando a sinistra e a destra la associazione che ottengo è k-loose.

Per ottenere una certa safety a livello di gruppo (ovvero non scendo al di sotto di k dove k è quella definita dalla k-loose) utilizzo un’eterogeneità che può essere di 3 livelli:

1. group heterogeneity
2. association heterogeneity
3. deep heterogeneity

group heterogeneity: non puoi mettere nello stesso gruppo tuple che sono ALIKE.

association heterogeneity: nessuno gruppo può essere associato due volte con lo stesso gruppo.

Deep heterogeneity: nessun gruppo può essere associato con due gruppi che hanno tuple alike.

Se ho la stessa illness su due gruppi in cui sono collegato allora ho k looseness a livello di tupla ma non di illness quindi non va bene, non rispetta la deep heterogeneity.

Tutte queste heterogeneity per avere la massima k-looseness cioè garantito essere il prodotto del gruppo 1 per il gruppo 2.

Se mi chiedono k-looseness di 12 è inutile fare gruppo di 20 da una parte e di 50 dall’altra, potrei fare meglio es: potrei fare 3,4 e rispetterei k-looseness di 12 se rispetto i principi di eterogeneità,

Se faccio gruppi di 1,12 o 12,1 ho sempre incertezza, si chiama gruppo flat, ma ce l’incertezza ce l’ho tutta da una parte invece che sui due frammenti.

Privacy vs utiliy: perché stiamo parlando di tutte queste minimalità ecc.. ? Perché c’è sempre trade-off tra privacy e utility.

Associazione a livello loose = incremento utilità, diminuisco incertezza rispetto a pubblicare gli alias.

Le tabelle delle slides dicono che ho incertezza di 1/8 se pubbliche le associazioni di tuple, se pubblico quelle di gruppo ho ¼ perché se il gruppo di sx è associato con due gruppi a quello di dx ho incertezza 2x2 = 4 e gli altri 2 gruppi che rimangono ho incertezza 0, perché so che con quei due gruppi non ho alcun tipo di associazione possibile. Nel caso dei gruppi ho loose association.

La prima tabella è 1/8 su tutto, mi dice associazione tupla con tupla. Se considero l’alikeness, allora cambia perciò diminuisce, esempio gastrite compare 2 volte e anche diabete perché considero le semituple perché considero i confidentiality constraints quindi le intersezioni e quindi le tuple alike rispetto alle confidentiality constraints.

La probabilità quindi diventa una probabilità composta nella tabella, e si riferisce ai confidentiality constraints che considero. (le tuple uguali le devo combinare, cioè probabilità di uno + probabilità dell’altro - il prodotto delle due, se ho 1/8 e 1/8 diventa 1/8 +1/8 - (1/8 \*1/8)); collasso insieme colonne e righe.

In sostanza 1/8 a livello di tupla dopo aver pubblicato le associazioni, e l’altra considerando le associazioni sensibili (conf. constraints).

Lo stesso ragionamento lo faccio per la looseness association, quindi a livello di gruppo. (lo chiama a livello di associazione)