ALGORITHMS FOR MASSIVE DATA, CLOUD AND DISTRIBUTED COMPURING

MACRODATA & MICRODATA PROTECTION

- *Respondent* = person whose data (refer) have been released. [patient]
- *Recipient* = subject receiving and looking at the data which have been published. [pharmaceutical company]

Often statistical data are released → these data can be used to infer information that was not intended for disclosure.

Disclosure can:

- Occur based on the released data alone
- Result from combination of the released data with publicly available
- Different data sources combined: combination of the released data with detailed external (public) data source

With all these cases we have a disclosure risk, that we need to counteract.

Statistical data can be released in 2 different forms:

- **Statistical DBMS**: the DBMS responds only to statistical queries → need run time checking to control information (indirectly) released. Data stored in local DBMS. Anyone outside can connect to my DBMS (result of aggregate, perform queries). More dynamic interaction.
    - Permit the final recipient to perform queries on statistics that combine together groups of individuals, but no query should reveal information about any particular individual. Anyway can refer to a group, but not single individual. Confidential information can be deduced from single or combination of multiple queries or external knowledge.
- **Statistical data**: computing some statistics that they think are interesting/relevant and publish them [INSTAT] → before releasing, make sure that the analysis does not contain sensitive information. One shot of release of the data from institution to the recipient.

When we release macrodata: releasing result of aggregation of statistical computation perform over the data.

Microdata are more subject to risk of privacy breaches (linking attacks).

Macrodata

Can leak pieces of sensitive information.

Can be classified into two groups (types of tables) depending on the content:

A. **Count/Frequency**. Each cell contains the number (result of count query) or the percentage (frequency) of respondents that have the same value over all attributes in the table.
B. **Magnitude data**. Each cell contains an aggregate (sum, mean…) value of a quantity of interest over all attributes in the table.

Tables of counts or frequencies

Sampling: conduct and publish a sample survey rather than census. → the fact of knowing a person doesn't mean that that person is present in the sample (no privacy issues). Estimates are made by multiplying individual responses by a sampling weight before aggregating them; if weights are not published, weighting helps to make an individual respondent's data less identifiable from published totals [secret key]. Estimates must achieve a specified accuracy (unless not published).

Special rules: more often applied when you are publishing data on the whole population (less in case of only a sample of the population). Special rules define restriction on the level of details available in a table [geographical details]. The rules are defined by company or institution, sometimes depend on Laws. Example: SSA prohibit publishing if in the table we have a value of a cell that is equal to the marginal total [or some types of small levels/threshold].

- Solutions: the table can be restructured and rows/columns combined (rolling-up categories).

Threshold rules: fix a threshold to the number of respondents in a cell, to be less than some specific number. → table restructuring and category combination, cell suppression, random rounding, controlled rounding, confidentiality editing.

> **Cell suppression**: Suppressing only sensitive cells (primary suppression) is not sufficient, at least, because we could retrieve/infer the real value/number that we suppressed → we need to suppress multiple cells for each row/column
>
> **Rounding**: reduce data loss due to suppression, using rounding of values. Linear programming methods are used to identify a controlled rounding for a table; it requires to use specialized computer programs and controlled type may not always exist. Can be:
>
> - Random: random decision whether the value has to be rounded up or down, the sum of values in row/column may be different from published marginal totals
> - Controlled: ensure that sum of cells of rows/columns respect published marginal total.
>
> **Confidentiality editing**: protect macrodata table working directly on collected data, not changing results on counts, but operating on data before the count. Switch some microdata table: takes sample of records from microdata file, finds a match in geographic region/attributes, swaps all attributes on the matched records.

Magnitude data

Magnitude data are the result of an aggregate computed over a numerical value/attribute.

Magnitude data are typically nonnegative quantities and the distribution of these values is not flat distribution, but skewed.

Disclosure limitation techniques focus on preventing precise estimation of the values of outliers [small village in Tuscany – Bocelli]. Sampling is less likely to provide protection. The units that are most visible because of their size do not receive any protection from sampling.

If frequencies are peculiar/uncommon, different from the others, are called outliers and are more likely to be identified. Sampling is less likely to provide protection in this case.

Primary suppression rules: determine whether a cell could reveal individual respondent information; such cells are considered sensitive and cannot be released. The most common suppression rules are: p-percent rule; pq rule; (n,k) rule.

P-percent rule: disclosure of magnitude data occurs if the user can estimate the contribution of a respondent too accurately, if upper and lower estimates for the respondent's value are closer to the reported value than a pre-specified percentage p. A cell is protected if:

$$\sum_{i=c+2}^{N} x_i \geq \frac{p}{100} x_1$$

The largest value x_l is the most exposed.

P is a parameter of protection. Higher p: more protected. Lower p: less protected.

We assume that there was no prior knowledge about respondent's values.

P-percent rule starts from the assumption that people looking at the table does not have any knowledge about respondent's values.

Pq rule: agencies can specify how much prior external additional knowledge there is by assigning a value **q** which represents how accurately respondents can estimate another respondent's value before any data are published ($p < q^1 < 100$).

$$\frac{q}{100} \sum_{i=c+2}^{N} x_i \geq \frac{p}{100} x_1$$

(n,k) rule: I cannot release a cell if less then n respondents/people contribute more than k percent of the total cell value, regardless of the number of respondents in a cell.

If cell is dominated by one respondent, the published total is an upper estimate for her value. n is selected to be larger than the number of any suspected coalitions.

Count/Frequency table= typical protection technique is sampling, special rules, threshold rules

Magnitude data= aggregate of data

[1] Ability in estimation people have before the cell is published.

Secondary suppression → how to protect macrodata table

Once sensitive cells have been identified, there are two options:

- **Table restructuring and cells collapsing**: recombining cells over different rows/ columns, until no sensitive cells remain
- **Cell suppression**: decide not to publish sensitive cells (primary suppression), to blank them out (complementary suppression). BUT we need complementary/ secondary suppression because the marginal total risk to show them anyway → guarantee that sensitive cells cannot be derived nor estimated

Another way to avoid suppression (administrative way) is to ask respondents' permission.

We identify additional cells that need to be removed before the release of the table to guarantee that the original rules of protection are still satisfied after we applied our protection techniques.

For small tables the selection of complementary cells can be done manually.

Audit

If totals are published, the sum of the primary or secondary suppressed cells can be derived.

We apply audit techniques, apply the sensitivity rule to these sums, to ensure that they are not sensitive. → to see if the solution computed automatically or manually for secondary suppression satisfies our privacy requirements.

Information loss and information in parameter values

The selection of the complementary cells should result in minimum information loss.

We can try to minimize the sum of suppressed cells or the total number of suppressed cells.

While the suppression rules can be published, parameter values should be kept confidential. For example, in p-percent, we should not reveal the p value. Once the value for one suppressed cell has been uniquely determined, other cell values can be easily derived.

Microdata

Many situations require today that the specific stored data themselves (microdata) be released.

Precise table with row for each of the respondent [patient, citizen].

Advantage:

- Flexibility and availability of information for the recipients.
- No precomputed type of statistics, but can apply every type of computation
- Utility higher than data released in macrodata tables.

To protect the anonymity of the respondents, data holders often remove or encrypt explicit identifiers such as names, addresses, phone numbers.

BUT de-identifying data is not sufficient, does not guarantee anonymity. Anonymize = not able anymore to use any kind of information to reconstruct the identity of the specific subject, even if it is not explicitly written. De-identify = remove explicit identifiers → more dangerous than macro

In data collection, released information often contains other **quasi-identifying** data [race, birth date, ZIP code – anagraphical information]: combined together (or using other external knowledge) permit to uniquely re-identify a subject.

We need to reduce the information content or change the data in such way that the information content is maintained as much as possible.

To limit disclosure risk, the following procedures should be applied:

- Include data from a sample of the whole population only
- Remove identifiers
- Limit geographic details
- Limit the number of variables

Geographic details always appear in micro-data, because they are anagraphycal details, and can be used for re-identifying respondents. → To limit these details, we can define that cannot be shown areas in which the number of people living there is smaller than a certain number.

Let's classify the protection techniques we can use in case of microdata. Techniques for microdata protection, to limit the ability to identify people:

- **Masking technique**: releasing data that I have collected, possibly modified/generalized/with reduced details/with some noise injected, BUT starting from data collected from population.
- **Synthetic data generation technique**: Not releasing any of data collected, BUT building a model over data and producing a sample of made-up data, extracting information from the model build on collected data. Not corresponding to identity of real people, but with the same statistical characteristics.

Another differentiation is done with respect to the kind of domain of attribute on which we operate:

- **Continuous attribute**: domain is numerical, so make sense to perform arithmetic operations [date of birth, temperature]
- **Categorical attribute**: values for which does not make sense to operate arithmetical operations [phone number, gender]

Masking techniques

= based on idea of releasing data collected, possibly preserving confidentiality of respondents.

The original data are transformed to produce new data that are valid for statistical analysis and such that they preserve the confidentiality of respondents.

We distinguish between **non perturbative** (not lying, but omitting) or **perturbative** techniques (original data are modified).

Non-perturbative		
Technique	**Continuous**	**Categorical**
Sampling	yes	yes
Local suppression	yes	yes
Global recoding	yes	yes
Top-coding	yes	yes
Bottom-coding	yes	yes
Generalization	yes	yes

Perturbative		
Technique	**Continuous**	**Categorical**
Resampling	yes	no
Lossy compression	yes	no
Rounding	yes	no
PRAM	no	yes
MASSC	no	yes
Random noise	yes	yes
Swapping	yes	yes
Rank swapping	yes	yes
Micro-aggregation	yes	yes

Sampling

= protected microdata is obtained as a sample of the original data [survey] table/ whole population.

Since there is uncertainty about whether or not a specific respondent is in the sample, re-identification risk decreases.

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Asian	64/03/09	M	94138	Married	10	190
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/03/18	M	94141	Married	60	290
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/02	M	94138	Single	22	140
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	12	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

Local suppression

= if there is an attribute value that is particularly sensitive and I don't want to release it, I simply remove it and replace with blank/missing value.

Usually we remove information which contribute more than the others in re-identification of a person, that makes a person stand out with respect with the others.

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F			15	200

Global recoding

= organize the domain of an attribute in disjoined intervals, associating a label with each interval, substituting the label instead of the precise value.

The protection is obtained by removing/replacing the values of an attribute with the label associated with the corresponding interval, that has a higher level of abstraction.

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	med
		Asian	64/09/30	F	94139	Divorced	1	low
		Asian	64/04/18	M	94139	Married	40	med
		Asian	64/04/15	M	94139	Married	17	med
		Black	63/03/13	M	94138	Married	2	low
		Black	63/03/18	M	94138	Married	13	low
		Black	64/09/13	F	94141	Married	15	med
		Black	64/09/07	F	94141	Married	60	high
		White	61/05/14	M	94138	Single	17	low
		White	61/05/08	M	94138	Single	10	high
		White	61/09/15	F	94142	Widow	15	med

Top-coding Bottom-coding

= works on the upper and lower limit, with respect to a certain threshold. Above or below a certain threshold we do not publish the exact value.

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

Generalization

= representing the values of a given attribute by using more general values with respect to the ones collected.

We need to build a hierarchy of generalization and for each value we know exactly how can be generalized.

[remove day of the date of birth, or have only the year, or only the range of years →depends on the level of generalization we want]

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09	F	94139	Divorced	13	260
		Asian	64/09	F	94139	Divorced	1	170
		Asian	64/04	M	94139	Married	40	200
		Asian	64/04	M	94139	Married	17	280
		Black	63/03	M	94138	Married	2	190
		Black	63/03	M	94138	Married	13	185
		Black	64/09	F	94141	Married	15	200
		Black	64/09	F	94141	Married	60	290
		White	61/05	M	94138	Single	17	170
		White	61/05	M	94138	Single	10	300
		White	61/09	F	94142	Widow	15	200

Random noise

= it perturbs a sensitive attribute by adding or multiplying it with a random variable with a given distribution (the distribution selected shouldn't be published).

The chosen value shouldn't affect data [average]. For example, if we want to have additive noise over the attribute *Holidays*, to preserve average:

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Noise	Income
		Asian	64/09/27	F	94139	Divorced	13	+2	260
		Asian	64/09/30	F	94139	Divorced	1	+1	170
		Asian	64/04/18	M	94139	Married	40	-10	200
		Asian	64/04/15	M	94139	Married	17	+3	280
		Black	63/03/13	M	94138	Married	2	+5	190
		Black	63/03/18	M	94138	Married	13	+8	185
		Black	64/09/13	F	94141	Married	15	+4	200
		Black	64/09/07	F	94141	Married	60	-11	290
		White	61/05/14	M	94138	Single	17	-2	170
		White	61/05/08	M	94138	Single	10	-3	300
		White	61/09/15	F	94142	Widow	15	+3	200

Swapping

= a small percent of records (small sample of tuples) are matched with other records in the same file. The values of all other variables on the file are swapped between the two records. This reduces risk, because we introduce uncertainty about the true value of the respondent's data.

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	2	190
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	40	200
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	60	290
		Black	64/09/07	F	94141	Married	15	200
		White	61/05/14	M	94138	Single	10	300
		White	61/05/08	M	94138	Single	17	170
		White	61/09/15	F	94142	Widow	15	200

Micro-aggregation (blurring)

= creating group of tuples characterized by similar or same values for some of the quasi-identifying attributes, grouping into small aggregates of a fixed dimension k. The average over each aggregate is published instead of individual values.

The groups are formed by using maximal similarity criteria.

There are different variations of micro-aggregation:

- The average can substitute the original value only for a tuple in the group or for all of them
- Different attributes can be protected through micro-aggregation using the same or different grouping

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	260
		Asian	64/09/30	F	94139	Divorced	1	170
		Asian	64/04/18	M	94139	Married	40	200
		Asian	64/04/15	M	94139	Married	17	280
		Black	63/03/13	M	94138	Married	2	190
		Black	63/03/18	M	94138	Married	13	185
		Black	64/09/13	F	94141	Married	15	200
		Black	64/09/07	F	94141	Married	60	290
		White	61/05/14	M	94138	Single	17	170
		White	61/05/08	M	94138	Single	10	300
		White	61/09/15	F	94142	Widow	15	200

SSN	Name	Race	DoB	Sex	ZIP	MarStat	Holidays	Income
		Asian	64/09/27	F	94139	Divorced	13	215
		Asian	64/09/30	F	94139	Divorced	1	215
		Asian	64/04/18	M	94139	Married	40	213
		Asian	64/04/15	M	94139	Married	17	213
		Black	63/03/13	M	94138	Married	2	213
		Black	63/03/18	M	94138	Married	13	213
		Black	64/09/13	F	94141	Married	15	245
		Black	64/09/07	F	94141	Married	60	245
		White	61/05/14	M	94138	Single	17	235
		White	61/05/08	M	94138	Single	10	235
		White	61/09/15	F	94142	Widow	15	200

Synthetic techniques

Release the statistical model instead of data. So is safer, but the model should be built in a proper manner.

Since the statistical content of the data is not related to the information provided by each respondent, a model well representing the data could in principle replace the data themselves.

In some situations, we may want to release only synthetic data, or mix them.

Fully Synthetic

Technique	Continuous	Categorical
Bootstrap	yes	no
Cholesky decomposition	yes	no
Multiple imputation	yes	yes
Maximum entropy	yes	yes
Latin Hypercube Sampling	yes	yes

Partially Synthetic

Technique	Continuous	Categorical
IPSO	yes	no
Hybrid masking	yes	no
Random response	no	yes
Blank and impute	yes	yes
SMIKe	yes	yes
Multiply imputed partially synthetic dataset	yes	yes

PRIVACY IN DATA PUBLICATION

Information disclosure = process through which someone is able to re-identify someone or extract information.

We can have different levels of disclosure:

1. Simple identity disclosure: someone looking at the data can reconstruct back the name of the subject.
2. Attribute disclosure: sensitive information about the respondent can be retrieved through the released data, not only identity.
3. Inferential disclosure: the ability of malicious recipient of data to determine the characteristic of respondent, because there is correlation between what is released and a specific characteristic (even if not in the attributes released). Infer data staring from correlations based on common knowledge.

Identity disclosure

It occurs if a third party can re-identify at least one of the respondents from the released data.

Revealing that an individual is respondent in a data collection may or may not violate confidentiality requirements.

- Macrodata: quite not a problem, unless identification leads to divulging confidential information (attribute disclosure).
- Microdata: more dangerous, because re-identifying in microdata table, where records are detailed, usually implies also attribute disclosure.

More common to release certain type of attributes in microdata than macrodata.

Attribute disclosure

It occurs when it is possible to retrieve the association between the identity and one piece of **confidential information**/sensitive attribute about respondents.

Confidential information may be revealed exactly or closely estimated.

Inferential disclosure

It occurs when information can be inferred with high confidence from statistical properties of the released data.

For example, data may show that there is a correlation between income and purchase price of house, which is a public information that a third party might use to infer income of the respondent.

Restricted data and restricted access

How to protect microdata in a public or semi-public manner? We use the protection techniques we mentioned:

- Remove explicit identifiers, represents the first step we need to do [name, address, SSN]

→ We can restrict data (reduce the amount of info released) or restrict access (limit the number of people that can have access, can see the data or which part of it).

Anonymity problem = information about finances, interests, demographics is increasing every day somewhere on the web.

De-identified table ≠ Anonymous table → If it is possible to reconstruct the identity of a subject, the table is not anonymous.

Classification of attributes in a microdata table:

- **Identifiers**: single attribute that uniquely identify a microdata respondent [SSN, university ID number].
- **Quasi-identifiers**: set of attributes that in combination can be connected to external information to identify all or some of the respondents to whom information refers, or reduce the uncertainty over their identities [DoB, ZIP and Gender, preferences]. Not really a key, in general any kind of information which can be used to reduce uncertainty about identity of subjects, even if they are not unique.
- **Confidential**: attributes of the microdata table that contain sensitive information [disease].
- **Non-confidential**: attributes that the respondents do not consider sensitive and whose release does not cause disclosure.

Factors contributing to increase disclosure risk of microdata:

- **Existence of high visibility records**. Outliers or records that stand out with respect to the others, of people having characteristics very different from others, like US President, VIPs, like unusual jobs or outstanding people.
- **Possibility of matching the microdata with external information**. People who possess a unique or peculiar combination of characteristics/ variables on the microdata (making JOIN).
- **Existence of a high number of common attributes between the microdata and the external sources**

- **Accuracy or resolution of the data** if I release the precise date of birth, I am more exposed than if I release only the year.
- **Number and richness of outside sources** (growing every moment) not all of which may be known to the agency releasing the microdata.

Factors contributing decrease disclosure risk of microdata

- A microdata table often contains only a **subset** instead of the whole population.
- **Some information change in time**, like number of children, age. This may lead to errors if I want to join data collected in different dates.
- **Noise**/mismatching due to mistakes in data injection, filling forms, intentionally or not.
- **Different forms** of data, same piece of information can change with respect to the ways of representing the value [way of writing dates].

Every time we release a table, we need to measure the risk at which we are exposed. In general, we look at how many subjects are unique.

K-anonymity

= way of protecting data according to a given threshold *k*. To do so, it adopts generalization and suppression techniques: want to release real data, not fake or modified information.

The released data should be indistinguishably related to no less than certain number of respondents.

Each single individual should be related with at least k number of tuples: nobody should identify precisely the person in the "room".

Strictly related to quasi-identifier: set of attributes that can be exploited for linking (whose release must be controlled).

The basic idea is that each release of data must be such that every combination of values of quasi-identifier can be indistinctly matched to at least k respondents.

Requires that each quasi-identifier value appearing in the released table must occur with at least k occurrences.
→ sufficient condition for satisfaction of k-anonymity requirement.

We need to make assumption based only on our knowledge and for security is better to take the worst assumption possible. So, if we guarantee that each quasi-identifier value appears at least k times, it means that there must be at least k people in the real world which correspond to each of them.

→ never have 1:1 correspondence. There should be at least k-1 people that can be confused with a certain subject in the table.

We have different kinds of attributes in the table, mainly 3 kinds:

- Identifiers = attributes that directly identify a person and usually correspond to primary key [name, surname, numerical ID]
- Quasi-identifiers = sets of attributes that in combination with additional external available information can help to restrict the number of people to which each record in the table released refers.
- Sensitive attributes = attributes that in association with the identity of a person should not be released

K-anonymity works on definition of protection of **quasi-identifier**.

For each tuple in the table there are at least *k* subjects/ respondents/ real people which can correspond to that specific tuple. → k-anonymity requires that each quasi-identifier value appearing in the released table must have at least k occurrences.

To obtain the satisfaction of this condition, k-anonymity adopts two conditions in combination: generalization and suppression, which are masking techniques (= not generating any fake data) and not-perturbative (= not adding any piece of not correct/ false information to protect data, but simply removing).

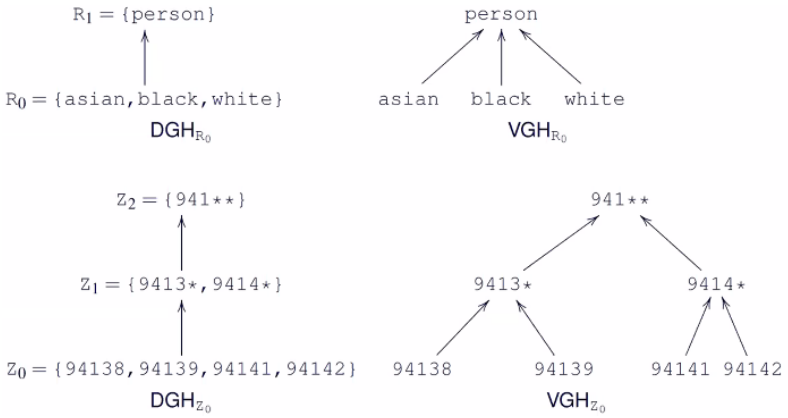
Domain generalization hierarchy

A generalization relationship $\leq_D$ defines a mapping between domain D and its generalizations. Given two domains D_i and D_j belonging to Dom, $D_i \leq_D D_j$ states that the values in domain D_j [only 4 digits in ZIP code] are generalization of values in D_i [all 5 digits in ZIP code]

$\leq_D$ implies the existence, for each domain D, of a domain generalization hierarchy $DGH_D = (Dom, \leq_D)$.

The maximal elements of Dom are singleton.

Given a domain tuple $DT = (D_1, \ldots, D_n)$ such that $D_i \in Dom$, $i = 1, \ldots, n$ the domain generalization hierarchy of *DT* is th $DGH_{DT} = DGH_{D1} \times \ldots \times DGH_{Dn}$



Value generalization hierarchy

A value generalization relationship $\leq_V$ associates with each value in domain D_i a unique value in domain D_j, direct generalization of D_i.

$\leq_V$ implies the existence, for each domain D, of a value generalization hierarchy VGH_D.

VGH is a tree: the leaves are the values on *D*, the root (=most general value) is the value in the maximum element in DGH_D.

Generalized table with suppression

Let T_O and T_G be two tables defined on the same set of attribute. Table T_G is said to be generalization (with tuple suppression) of table T_O if:

1. The cardinality of T_G is at most that of T_O

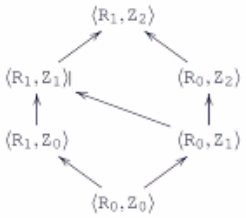
2. The domain of each attribute A in T_G is equal to, or a generalization of, the domain of attribute A in T_O
3. Is possible to define correspondence (an injective function) associating each tuple t_G in T_G with different tuple t_O in T_O, such that the value of each attribute in t_G is equal to, or a generalization of, the value of the corresponding attribute in t_O (some tuples in T_O might not have corresponding tuples in T_G)

Race:R_0	ZIP:Z_0
asian	94142
asian	94141
asian	94139
asian	94139
asian	94139
black	94138
black	94139
white	94139
white	94141

PT

Race:R_1	ZIP:Z_0
person	94141
person	94139
person	94139
person	94139
person	94139
person	94139
person	94141

$GT_{[1,0]}$



Better to suppress or generalize?

Generalization operates on the level of attribute (column) and suppression at the level of cell (value). Generalizing may increase information loss, because it hits all the values in the column.

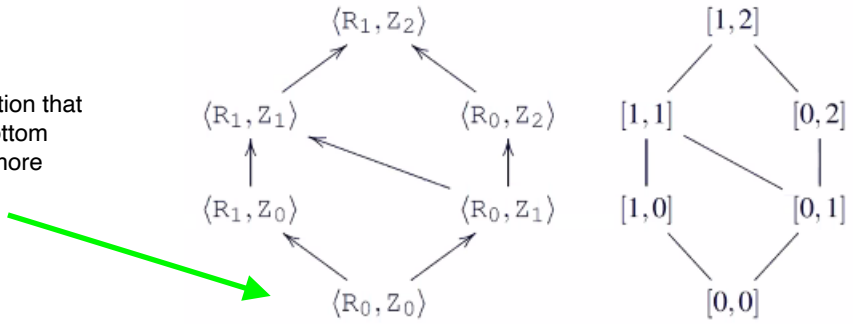
Assume a threshold of suppression, if requires suppression is.

- Below the threshold → suppress
- Above the threshold → generalize

Minimal generalization: suppress and generalize as needed, not more. Not overdoing it, only to guaranty k-anonymity.

Distance vector. Let $T_i(A_1,...,A_n)$ and $T_j(A_1,...,A_n)$ be two tables such that $T_i \preccurlyeq T_j$. The distance vector of T_j from T_i is the vector $DV_{i,j} = [d_1,...,d_n]$, where each d_z , z = 1,...,n, is the length of the unique path between $dom(A_z, T_i)$ and $dom(A_z, T_j)$ in the domain generalization hierarchy DGH_{Dz}.

We prefer a solution that is close to the bottom (since we have more information)



Let MaxSup be the specified threshold of acceptable suppression. T_j is said to be a k-minimal generalization of table T_i iff:

1. T_j satisfies k-anonymity enforcing minimal required suppression, that is, T_j satisfies k-anonymity and $\forall T_z : T_i \preccurlyeq T_z , DV_{i,z} = DV_{i,j}$, T_z satisfies k-anonymity $\Rightarrow |T_j| \geq |T_z|$

2. $|T_i| - |T_j| \leq MaxSup$
3. $\forall T_z : T_i \preccurlyeq T_z$ and T_z satisfies conditions 1 and 2 $\Rightarrow \neg(DV_{i,z} < DV_{i,j})$

Look at less frequent

MaxSup = 2

MaxSup = 2

Race:R_0	ZIP:Z_0
asian	94142
asian	94141
asian	94139
asian	94139
asian	94139
black	94138
black	94139
white	94139
white	94141

PT

Race:R_0	ZIP:Z_0
asian	94142
asian	94141
asian	94139
asian	94139
asian	94139
black	94138
black	94139
white	94139
white	94141

PT

Satisfy only 1-anonymity (if max n. of tuple you can suppress is 0)

Race:R_1	ZIP:Z_1
person	9414*
person	9414*
person	9413*
person	9413*
person	9413*
person	9413*
person	9413*
person	9413*
person	9414*

$GT_{[1,1]}$

3-anonymity

Race:R_0	ZIP:Z_2
asian	941**
asian	941**
asian	941**
asian	941**
asian	941**
black	941**
black	941**
white	941**
white	941**

$GT_{[0,2]}$

2-anonymity

Race:R_1	ZIP:Z_2
person	941**
person	941**
person	941**
person	941**
person	941**
person	941**
person	941**
person	941**
person	941**

$GT_{[1,2]}$

Too Much general!

Race:R_1	ZIP:Z_0
person	94141
person	94139
person	94139
person	94139
person	94139
person	94139
person	94141

$GT_{[1,0]}$

Race:R_0	ZIP:Z_1
asian	9414*
asian	9414*
asian	9413*
asian	9413*
asian	9413*
black	9413*
black	9413*

$GT_{[0,1]}$

We delete White since ZIPCODE is different between the two

Preference criteria: for generalization

- Minimize absolute distance with respect to the original value/table, with the smallest total number of generalization steps (regardless of the hierarchies on which they have been taken) Better in the bottom
- Minimum relative distance prefers the generalization with the smallest relative distance, that minimized the total number of relative steps (a step is made relative by dividing it over the height of the domain hierarchy to which it refers) Better more step in the hierarchy tree
- Maximum distribution prefers the generalizations(s) with the greatest number of distinct tuples
- Minimum suppression prefers the generalization(s) that suppresses less tuples, that is, the one with the greatest cardinality Better less suppressions

Different granularity levels: can have generalization applied for column [all values of the column] of differently for each individual [some subjects have the entire date of birth some only day, some only year...] and can have suppression applied at level of row. Trade off between liability of the data and complexity of the algorithm

Generalization	Suppression: *Tuple*	*Attribute*	*Cell*	*None*
Attribute	**AG_TS**	**AG_AS** ≡ AG_	**AG_CS**	**AG_** ≡ AG_AS
Cell	**CG_TS** not applicable	**CG_AS** not applicable	**CG_CS** ≡ CG_	**CG_** ≡ CG_CS
None	**_TS**	**_AS**	**_CS**	**_** not interesting

Race	DOB	Sex	ZIP
asian	64/04/12	F	94142
asian	64/09/13	F	94141
asian	64/04/15	F	94139
asian	63/03/13	M	94139
asian	63/03/18	M	94139
black	64/09/27	F	94138
black	64/09/27	F	94139
white	64/09/27	F	94139
white	64/09/27	F	94141

PT

Race	DOB	Sex	ZIP
asian	64/04	F	941**
asian	64/04	F	941**
asian	63/03	M	941**
asian	63/03	M	941**
black	64/09	F	941**
black	64/09	F	941**
white	64/09	F	941**
white	64/09	F	941**

AG_TS

Race	DOB	Sex	ZIP
asian		F	
asian		F	
asian		F	
asian	63/03	M	9413*
asian	63/03	M	9413*
black	64/09	F	9413*
black	64/09	F	9413*
white	64/09	F	
white	64/09	F	

AG_CS

Race	DOB	Sex	ZIP
asian	64	F	941**
asian	64	F	941**
asian	64	F	941**
asian	63	M	941**
asian	63	M	941**
black	64	F	941**
black	64	F	941**
white	64	F	941**
white	64	F	941**

AG_≡AG_AS

Race	DOB	Sex	ZIP
asian	64	F	941**
asian	64	F	941**
asian	64	F	941**
asian	63/03	M	94139
asian	63/03	M	94139
black	64/09/27	F	9413*
black	64/09/27	F	9413*
white	64/09/27	F	941**
white	64/09/27	F	941**

CG_≡CG_CS

Race	DOB	Sex	ZIP

_TS

Race	DOB	Sex	ZIP
asian		F	
asian		F	
asian		F	
asian		M	
asian		M	
black		F	
black		F	
white		F	
white		F	

_AS

Race	DOB	Sex	ZIP
asian		F	
asian		F	
asian		F	
asian		M	94139
asian		M	94139
	64/09/27	F	
	64/09/27	F	94139
	64/09/27	F	94139
	64/09/27	F	

_CS

Finding Optimal Generalization and suppressions is NP problem

QI: quasi-identifier
For a table to satisfy 2-anon is necessary if for each combination we have two tuple with the same value. Incognito said that to guarantee it, it needs for each column there is at least two occurrences.

Incognito algorithm

k-anonymity with respect to a proper subset of QI is a necessary (not sufficient) condition for k-anonymity with respect to QI:

- Iteration 1: check k-anonymity for each attribute in QI, discarding generalizations that do not satisfy k-anonymity
- Iteration 2: combine the remaining generalizations in pairs and check k-anonymity for each couple obtained

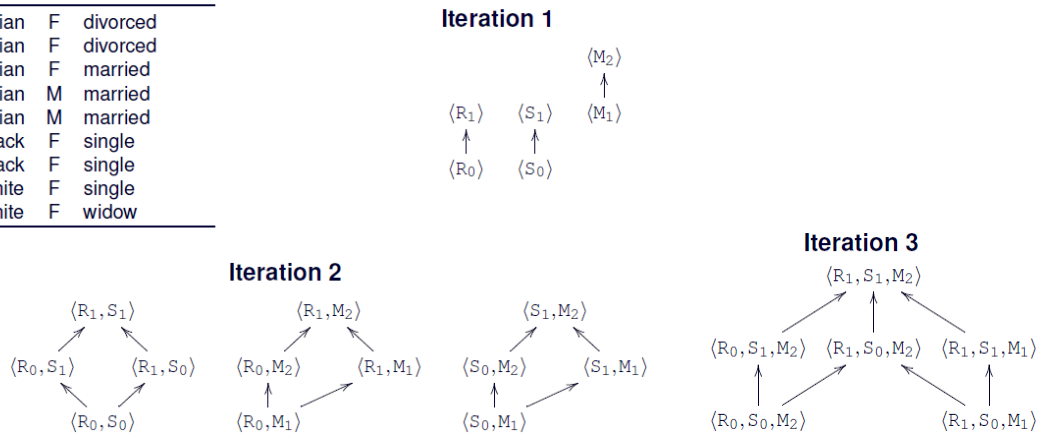
  . . .
- Iteration i: consider all the i-uples of attributes, obtained combining generalizations that satisfied k-anonymity at iteration $i-1$. Discard non k-anonymous solutions

  . . .
- Iteration |QI| returns the final result

Incognito adopts a bottom-up approach for the visit of DGHs.

Race	Sex	Marital status
asian	F	divorced
asian	F	divorced
asian	F	married
asian	M	married
asian	M	married
black	F	single
black	F	single
white	F	single
white	F	widow

We consider k=2



Mondrian multidimensional algorithm

Operates at cell level, in contrast to attribute generalization and tuple suppression we have seen.

To guarantee k-anonymity, the multi-dimensional space is partitioned by splitting dimensions such that each area contains at least k occurrences of point values.

Each attribute in QI represents a dimension.

Each tuple in PT represents a point in the space defined by QI. Tuples with the same QI value are represented by giving a multiplicity value to points.

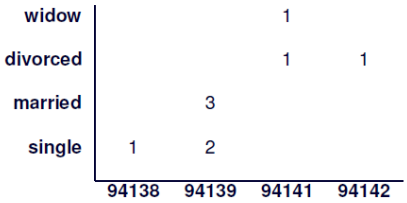
All the points in a region are generalized to a unique value. The corresponding tuples are substituted by the computed generalization.

So can operate on different number of attributes (single or multi-dimension), with different recording (generalization) strategies, with different partitioning strategies (strict or relaxed partitioning, so non-overlapping or potentially overlapping), using different metrics to determine how to split on each dimension.

Both algorithms have good perfomance in terms of information loss but we cannot be sure about optimal solution

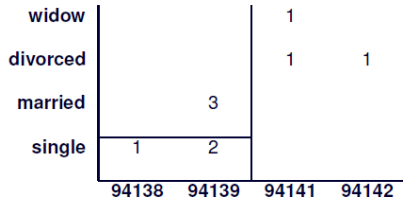
Private table

Marital status	ZIP
divorced	94142
divorced	94141
married	94139
married	94139
married	94139
single	94138
single	94139
single	94139
widow	94141



3-anonymous table

Marital status	ZIP
divorced or widow	9414*
divorced or widow	9414*
married	94139
married	94139
married	94139
single	9413*
single	9413*
single	9413*
divorced or widow	9414*



Since we have 3-anon the area of ZIPCODE for 94141 and 94142 cannot be farder splitted

The working of this algorithm depends a lot to the order we decide to adopt for the values of the domain along the dimensions.

k-anonymity requirement: each release of data must be such that every combination of values of quasi-identifiers can be indistinctly matched to at least k respondents

When generalization is performed at attribute level (**AG**) this is equivalent to require each quasi-identifier *n-uple* to have at least k occurrences.

When generalization is performed at cell level (**CG**) the existence of at least k occurrences is a sufficient but not necessary condition; a less strict requirement would suffice

1. For each sequence of values *pt* in PT[QI] there are at least k tuples in GT[QI] that contain a sequence of values generalizing *pt*
2. For each sequence of values t in GT[QI] there are at least k tuples in PT[QI] that contain a sequence of values for which *t* is a generalization

Race	ZIP
white	94138
black	94139
asian	94141
asian	94141
asian	94142

PT

Race	ZIP
person	9413*
person	9413*
asian	9414*
asian	9414*
asian	9414*

2-anonymity

2-anonymity

Race	ZIP
person	9413*
person	9413*
asian	94141
asian	9414*
asian	9414*

Not protected

Race	ZIP
person	9413*
person	9413*
asian	9414*
asian	9414*
asian	94142

Not protected

Race	ZIP
person	9413*
person	9413*
asian	94141
asian	94141
asian	9414*

Attribute disclosure

K-anonymity protect from identity disclosure, not attribute disclosure.

Race	DOB	Sex	ZIP	Disease
asian	64	F	941**	hypertension
asian	64	F	941**	obesity
asian	64	F	941**	chest pain
asian	63	M	941**	obesity
asian	63	M	941**	obesity
black	64	F	941**	short breath
black	64	F	941**	short breath
white	64	F	941**	chest pain
white	64	F	941**	short breath

The table is 2-anonymous, but still if I can restrict which diseases corresponds to someone I know, staring from some assumptions or some target or external knowledge.

In this case, I can connect "short breath" to black female subjects. Furthermore, if I know Hellen how is a white female, I can infer that the disease is chest pain or short breath: but since she is a runner, I can infer Hellen has chest pain.

l – diversity

A way to solve this problem is *l*-diversity.

A q-block (i.e., set of tuples with the same value for QI) in T is ℓ-diverse if it contains at least ℓ different "well-represented" values for the sensitive attribute in T: "well-represented" has different definitions based on entropy or recursion (e.g., a q-block is ℓ-diverse if removing a sensitive value it remains (ℓ-1)-diverse).

An adversary needs to eliminate at least ℓ-1 possible values to infer that a respondent has a given value.

T is ℓ-diverse if all its q-blocks are ℓ-diverse, so the homogeneity attack is not possible anymore and the background knowledge attack becomes more difficult.

ℓ-diversity is monotonic with respect to the generalization hierarchies considered for k-anonymity purposes.

Any algorithm for k-anonymity can be extended to enforce the ℓ-diverse property BUT ℓ-diversity leaves space to attacks based on the distribution of values inside q-blocks (skewness and similarity attacks).

Skewness attack

Occurs when the distribution of values in a *q*-block (set of tuples) is considerably different than the distribution in the original population.

20% of the population suffers from diabetes; 75% of tuples in a q-block have diabetes
$\Longrightarrow$ people in the q-block have higher probability of suffering from diabetes

Race	DOB	Sex	ZIP	Disease
black	64	F	941**	diabetes
black	64	F	941**	short breath
black	64	F	941**	diabetes
black	64	F	941**	diabetes

Similarity attack

Occurs when a *q*-block has different but semantically similar values for the sensitive attribute.

Race	DOB	Sex	ZIP	Disease
black	64	F	941**	stomach ulcer
black	64	F	941**	stomach ulcer
black	64	F	941**	gastritis

(satisfy 2-diversity and 3-anonymity)

We are not precisely releasing the disease, but we are saying that the kind of problem has something to do with the stomach.

The solution for this problem is the **closeness**.

A *q*-block respects ***t*-closeness** if the distance between the distribution of the values of the sensitive attribute in the *q*-block and in the considered population is lower than *t*. the value *t* respects the closeness if all its *q*-blocks respect *t*-closeness. Any algorithm for *k*-anonymity can be extended to force the *t*-closeness property, which however might be difficult to achieve.

External knowledge → positive inference: a respondent has a given value; negative inference: a respondent does not have a given value. We need to pay attention more on positive than negative inference.

Not possible to know a-priori what kind of external knowledge people have:

- Target individual [neighbor]
- Others
- Same-value families – genomic information

Name	DOB	Sex	ZIP	Disease
Alice	74/04/12	F	94142	aids
Bob	74/04/13	M	94141	flu
Carol	74/09/15	F	94139	flu
David	74/03/13	M	94139	aids
Elen	64/03/18	F	94139	flu
Frank	64/09/27	M	94138	short breath
George	64/09/27	M	94139	flu
Harry	64/09/27	M	94139	aids

Original table

$\Longrightarrow$

DOB	Sex	ZIP	Disease
74		941**	aids
74		941**	flu
74		941**	flu
74		941**	aids
64		941**	flu
64		941**	short breath
64		941**	flu
64		941**	aids

4-anonymized table

DOB	Sex	ZIP	Disease
74		941**	aids
74		941**	flu
74		941**	flu
74		941**	aids
64		941**	flu
64		941**	short breath
64		941**	flu
64		941**	aids

4-anonymized table

$\Longrightarrow$

DOB	Sex	ZIP	Disease
64		941**	flu
64		941**	short breath
64		941**	flu
64		941**	aids

4-anonymized table

An adversary knows that Harry, born in 64 and living in area 94139, is in the table

$\Longrightarrow$ Harry belongs to the second group
$\Longrightarrow$ Harry has aids with confidence 1/4

DOB	Sex	ZIP	Disease
64		941**	short breath
64		941**	flu
64		941**	aids

4-anonymized table

$\Longrightarrow$

DOB	Sex	ZIP	Disease
64		941**	flu
64		941**	aids

4-anonymized table

From personal knowledge, the adversary knows that Harry does not have short breath

$\Longrightarrow$ Harry has aids with confidence 1/2

Multiple Releases

Protection needs to be guaranteed in the different snapshots

T_1

DOB	Sex	ZIP	Disease
74		941**	aids
74		941**	flu
74		941**	flu
74		941**	aids
64		941**	flu
64		941**	short breath
64		941**	flu
64		941**	aids

4-anonymized table at time t_1

T_2

DOB	Sex	ZIP	Disease
[70-80]	F	9414*	hypertension
[70-80]	F	9414*	gastritis
[70-80]	F	9414*	aids
[70-80]	F	9414*	gastritis
[60-70]	M	9413*	flu
[60-70]	M	9413*	aids
[60-70]	M	9413*	flu
[60-70]	M	9413*	gastritis

4-anonymized table at time t_2

T_1

DOB	Sex	ZIP	Disease
74		941**	aids
74		941**	flu
74		941**	flu
74		941**	aids

4-anonymized table at time t_1

T_2

DOB	Sex	ZIP	Disease
[70-80]	F	9414*	hypertension
[70-80]	F	9414*	gastritis
[70-80]	F	9414*	aids
[70-80]	F	9414*	gastritis

4-anonymized table at time t_2

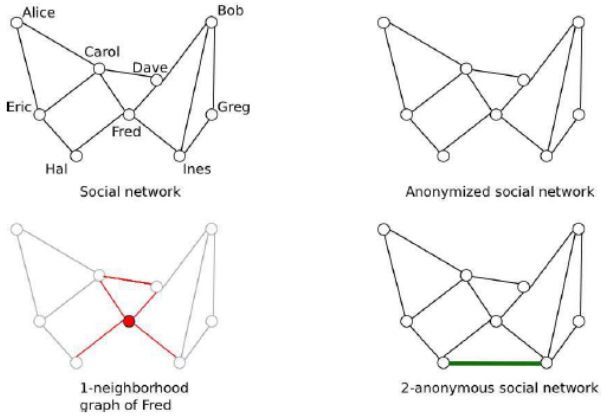
An adversary knows that Alice, born in 1974 and living in area 94142 is in both releases

$\Longrightarrow$ Alice belongs to the first group in T_1
$\Longrightarrow$ Alice belongs to the first group in T_2

Alice suffers from aids (it is the only illness common to both groups)

Assumption for using these techniques in real world: Multiple tuples per respondent, release of multiple tables, multiple QI, Non-predefined QI, Release of data streams, Fine-Grained privacy preference

Neighborhood attack $\Longrightarrow$ given a de-identified graph G' of a social network graph G, exploit knowledge about the neighbors of user u to re-identify the vertex representing u



Privacy preserving data mining techniques depend in the definition of privacy capturing what information is sensitive in the original data and should then be protected. → Association rule mining, Classification mining

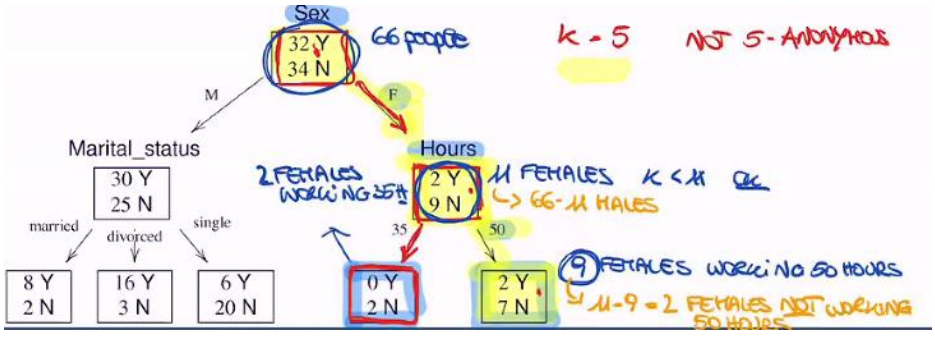
Association and Classification rule mining

Transactions, like supermarket having each purchase for each client.

Marital_status	*Sex*	*Hours*	*#tuples (Hyp. values)*
divorced	M	35	2 (0Y, 2N)
divorced	M	40	17 (16Y, 1N)
divorced	F	35	2 (0Y, 2N)
married	M	35	10 (8Y, 2N)
married	F	50	9 (2Y, 7N)
single	M	40	26 (6Y, 20N)

If QI includes *Marital_status* and *Sex*: {divorced}→ {M}

Violates k-anonymity for k>19 and also violates k-anonymity for k>2.



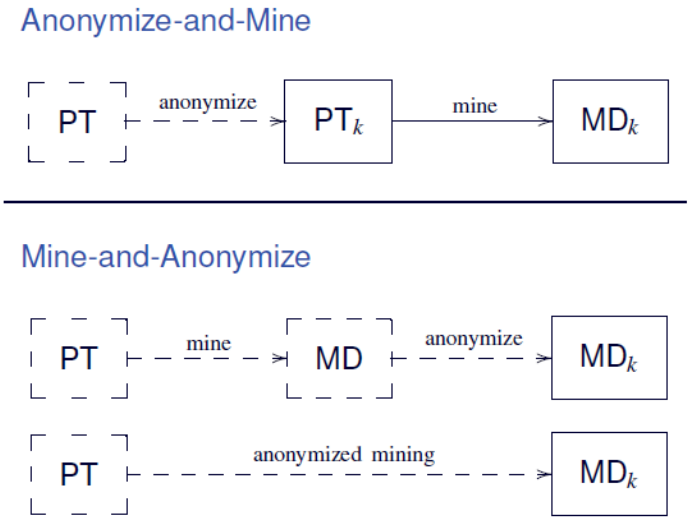
→ Indirectly exposed.

We have 3 different ways to work to guarantee that k anonymity is satisfied.

Anonymize-and-Mine: first anonymize table, then apply the mining technique. If the table I am working on with my mining technique is k-anonymous, the result cannot expose something at a finer granularity → not more precise inferences. Similar to the idea of confidentiality edit: if we protect the microdata, then we can compute the macrodata in an already protected way.

Second way: use a specific anonymization technique which works over the result of mining. To check whether the result of a classification mining of a frequent item is exposed with respect to k-anonymity. Then if the result is not anonymous, apply the protection technique.

Third way: apply transformation which both anonymizes while computing the result of mining, so modify mining algorithms to take k-anonymity into consideration. Adv: may obtain better result in terms of classification trees, because it combines the two techniques and better balance utility-privacy.



Location-based services

= any kind of service based on the location of the subject using it.

You might want to protect identity of people if people are in location that is considered sensitive [hospital].

- hide people in a crowd of k individuals in a given area
- Enlarge the area to have at least k-1 users (k-anonymity)
- Protect the location of the subject, so don't even release the place/ obfuscate the are so to decrease its precision or confidence (location privacy)
- Protect the location path of users, like if you visit the same place every day (or quite often). Block tacking by mixing/modifying trajectories (trajectory privacy)

America OnLine case: internet provider. Suppressed any obviously identifying information such as username and IP address. Even though we hide identifiers, the researches the user did, can be used to identify him/her.

Netflix prize data study (2006): De-identified Netflix data can be re-identified by linking with external sources (e.g., user ratings from IMDb users).

JetBlue case: published travel records of 5 million customers, that combined with other external information created an issue.

Syntactic and Semantic privacy definitions

Syntactic: capture the protection degree enjoyed by data respondents with a numerical value.

→ each release of data must be indistinguishably related to no less than a certain number of individuals in the population (threshold).

Semantic: based on the satisfaction of a semantic privacy requirement by the mechanism chosen for releasing the data, not on data themselves.

→ the result of an analysis carried out on a released dataset must be insensitive to the insertion or deletion of a tuple in the dataset.

Example of semantic technique: **Differential Privacy** which aims at preventing adversaries from being capable to detect the presence or absence of a given individual in a dataset. [count cancer from a medical database]. → Preventing a single individual to make the difference in the result.

K-anonymity:

- Adv: nice capturing or real-world requirement
- Cons: not complete protection

Differential privacy:

- Adv: better protection guarantees
- Cons: not easy to understand/enforce, not guaranteeing complete protection either

Examples:

- Privacy and genomic data. Cheaper to sequence the DNA of people. DNA is a unique identifier and contains a sensitive information [ethnic heritage, predisposition to several diseases...] and discloses information about the relatives and descendants
- Sensitive inference from data mining: target is the second-largest discount retailer in the US. It assigns every consumer to each customer a Guest ID number for discounts: discovered a teenager was pregnant before her parents.
- Inferences from social networks: people tend to connect to people with others with similar interests/ activities/ experiences, so what someone discloses, exposes also others.

Differential privacy

Classic intuition behind the idea of privacy: I can say that it is safe for me to be in a database D, if any kind of computation/algorithm performed over the dataset without my presence does not influence any way the result of the computation.

→ But, if individuals had no impact on the released results, then the results would have no utility.

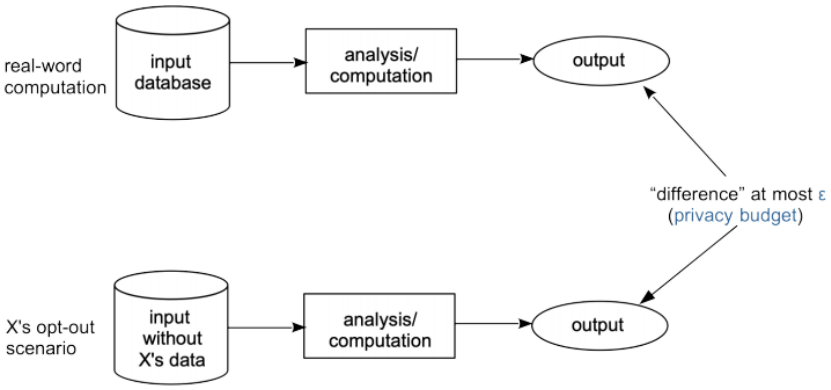
If I knew that the information learned about an individual by the published result R is no more than the information, we can learn about that individual without access to R.

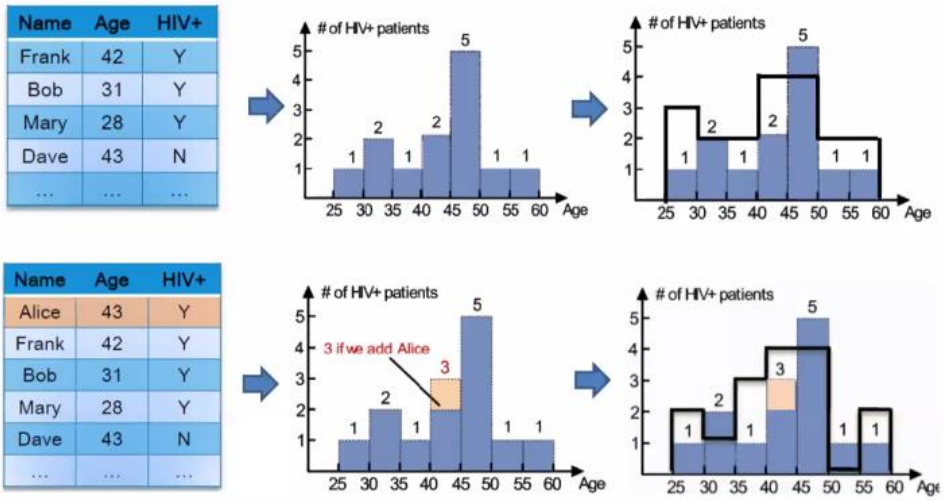
Inferences about an individual from a differentially private computation are essentially limited to what could be inferred form everyone else's data without her own data being included in the computation.

→ the privacy of an individual is protected whenever the result R does not depend on her specific information.

Example: insurance, if the characteristics/info are not released but the premium increases, then we can infer that the risk is increasing for some reason.

If you can identify the singular subject, it would be a problem.





Adding random noise to the result to hide difference between real-world computation and the opt-out scenario of each individual in the database.

The outcome of a differentially private analysis is not exact but an approximation. A differentially private analysis may, if performed twice on the same dataset, return different result: it is often possible to calculate accuracy bounds for the analysis.

Let databases D and D' be two neighbors database. An algorithm A satisfies ε-differential privacy if for all pairs of neighbor databases D, D' and for all outputs *o*:

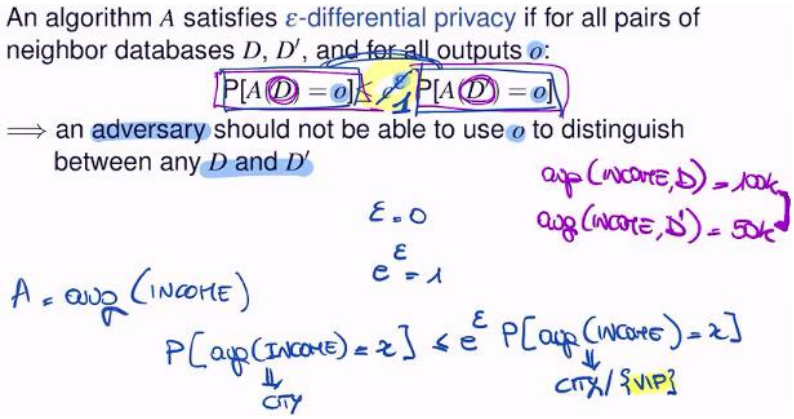
$$P[A(D) = o] \leq e^{\varepsilon} P[A(D') = o]$$

→ an adversary should not be able to use the result *o* to distinguish between any D and D'.

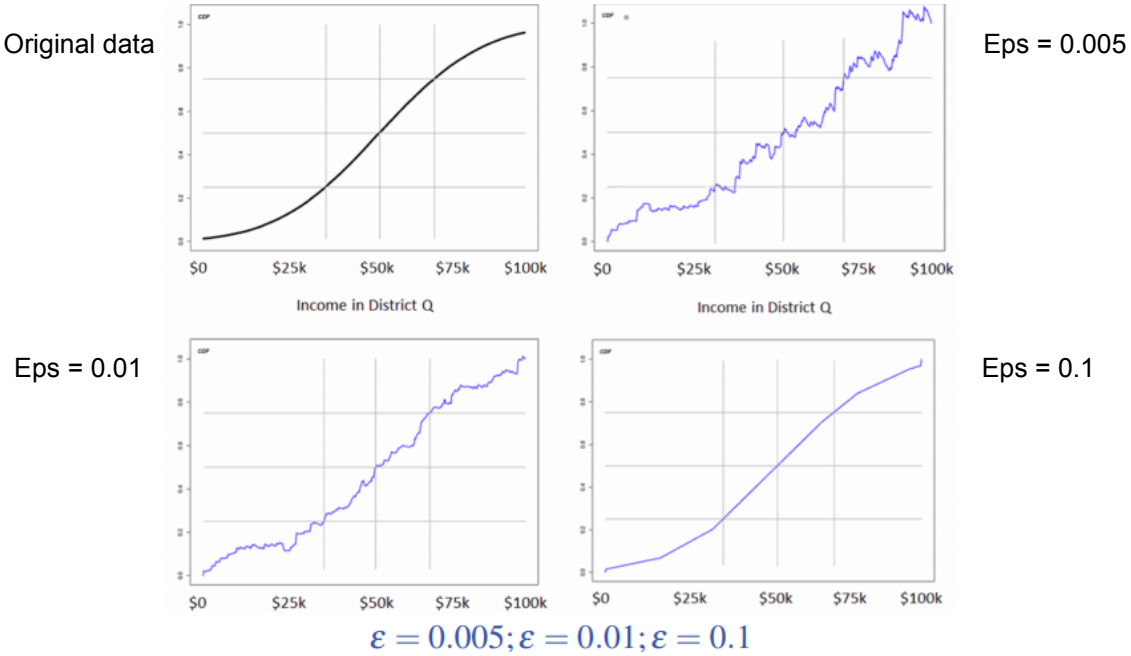
The presence of Alice can be too much impact on the result I obtain.

The **privacy parameter** (privacy budget) used, is **ε**. represents the amount of noise added to the computation, so the trade-off between privacy and accuracy.

The larger ε the less is the noise. The smaller is ε more is the noise



- ε = 0 an analysis could not provide any meaningful output.
- ε = 0,1 it provides strong privacy guarantees and useful statistics
- ε =1 it provides high accuracy but low privacy
- higher values for ε are not suggested



The probability of obtaining *o* applying the algorithm A to the dataset D and to the dataset D' is quite the same.

Example: insurance company and smoking people.

Let's consider the case D'= "Alice opted out".

People smoking +20% of long disease →premium +2%.

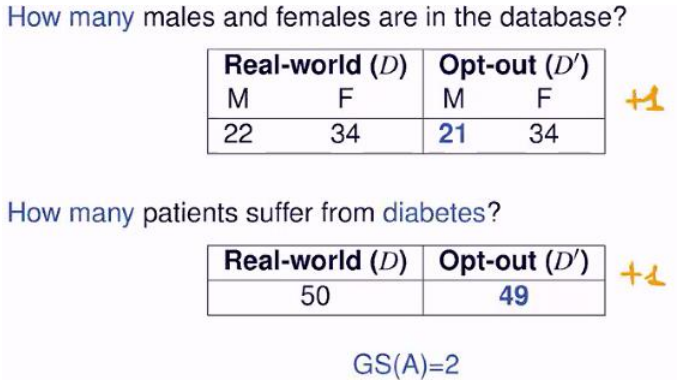
If Alice smokes → she will pay +2% in 2021, independently if she participated or not in the survey.

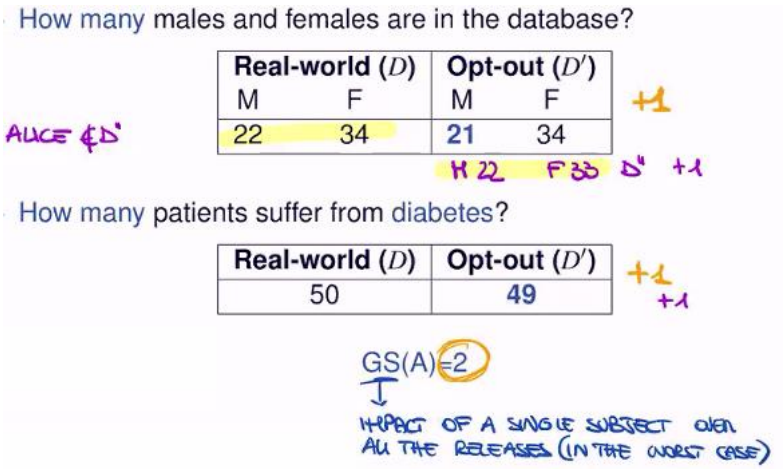
Let's consider the case D= "Alice is represented".

People smoking +22% long disease. → Alice will pay +2,2%

How to achieve differential privacy? It's important to calibrate noise in order to achieve differential privacy.to do so we need to understand which is the influence an individual can have on the result. → **Global sensitivity**:

characterizes the scale of the influence of one individual (worst case), and hence how much noise we must add. → D + Add noise



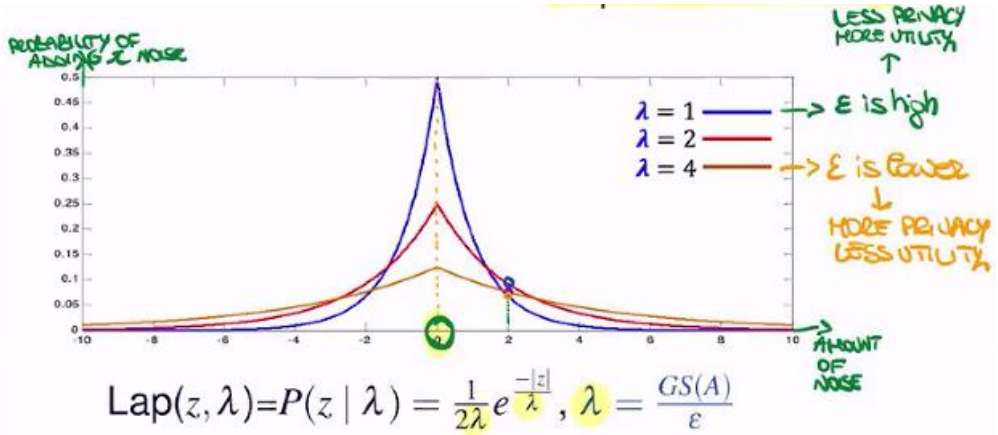


Result R is sampled from a Laplace distribution with mean the true result and some scale λ (determined by ε and the global sensitivity of the computation).

$$R=A(D)+Z$$

Z is a variable drawn from the Laplace distribution.

The mean is the true result, so computed over D (original/real world dataset).



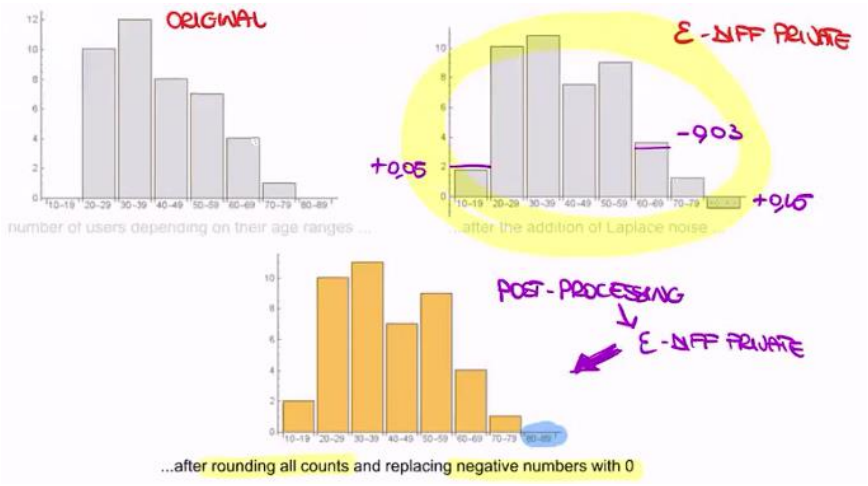
If global sensitivity is high, we need more noise, if we have outlier.

It global sensitivity is low, by design the two neighbors datasets are similar one to the other, so we don't need so much noise to make the two results look similar in case a subject is removed.

Closure under post-processing

Differential privacy is resilient to post-processing → the computation of a function over the result of a differentially private computation cannot make it less differentially private.

If we perform different analysis over the same dataset: if we perform any kind of post-processing over differentially private dataset, we have the guarantee that also the result satisfies at least the same privacy parameter for differential privacy.



(does not make sense to have -80 users – Laplace distribution)

Adding a noise, we make some rounding to make the domain more reliable, similar to what we have in a real-world situation → post-processing over differentially private dataset (second one)

Differential privacy composes well with itself. What does it mean?

Sequential composition: sequence of *m* computations over database *D* with overlapping results. Need to perform different computations over the same dataset and maybe release in a public manner the results.

$\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_m = \varepsilon$ for people in D

1° comp 2°comp m comp.

where ε_1= is the privacy parameter of the first computation…

ε is the threshold provided to people

The protection we guarantee to people is the sum of privacy parameters used for each of the computations you are performing (the εs).

Same subject is possibly involved in all the computations (each revealing something).

Parallel composition: sequence of *m* computations over disjoint subsets of a database D. Different computations operate on disjoint subsets of a dataset → (most affected) a single subject can participate in one computation only the privacy parameter provided to each subject is the one to the computation to which he/she participates, so if I need to find general measure I need to look at the worst case, that corresponds to the maximum epsilon.

for example performing two studies one over Male and one over Female Population

A1 Males - A2 females

Alice app A2
Bon app A1

Nobody in A1 Intersec A2

Protection provided is :

$$max(\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_m)$$

Each individual participates in one computation only (C_i), so impacts only one result and only one result reveals something. → worst case scenario and general measure.

Example: privacy budget ε. I can perform as many releases as long as my privacy budget remains below ε.→ when I reach the threshold I stop releasing information.

Ask for count of female patients and count of patients suffering from diabetes. #Females: 34; #Diabetes: 23.

→ sequential composition because the 2 releases overlap: we can have females suffering from diabetes.

Each count must be released in such a way that ε_1 (first count) + ε_2 (second count) be equal to ε.

Example2: Ask for count of people broken down by handedness, hair color.

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

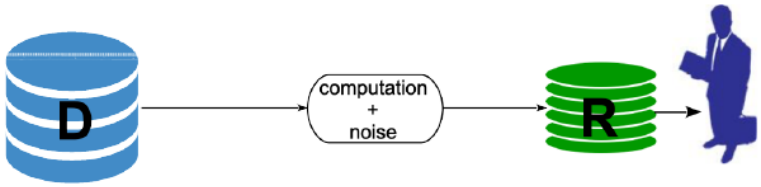
Each cell is a disjoint set of individuals.

Each cell can be released with ε-differential privacy.

Group privacy. Differential privacy has been introduced for reasoning about the privacy of a single individual but allows also reasoning about the privacy of groups. Privacy guarantees that apply to an individual with ε apply to a group of size n with the privacy parameter becoming $n\varepsilon$.

Non-interactive model

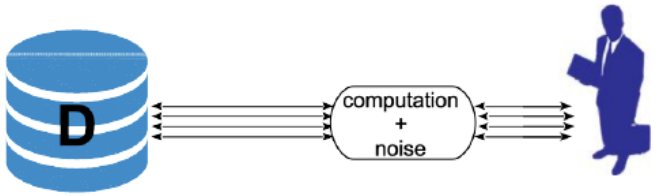
Perform computation + inject noise to protect data + release result to final recipient.



Non-interactive model

Interactive model

The recipient can ask to perform continuously computation over dataset and then every time is requested, need to perform the computation and inject noise.



Interactive model

A1 Males - A2 females

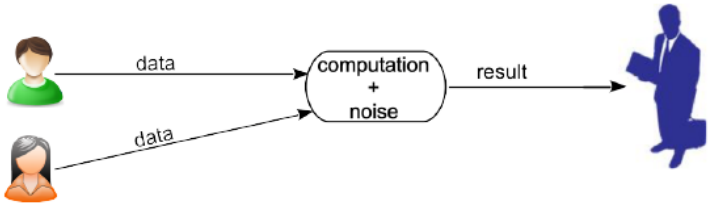
Alice app A2
Bon app A1

Nobody in A1 Intersec A2

In this case, we need to work more on the mechanism, so we could obtain different results. If different people request multiple times the same computation over the data, they might get different results, because we are injecting noise in different manners.

Global differential privacy

If you have multiple subjects that contribute to the dataset; once data have been combined, we add noise to protect.



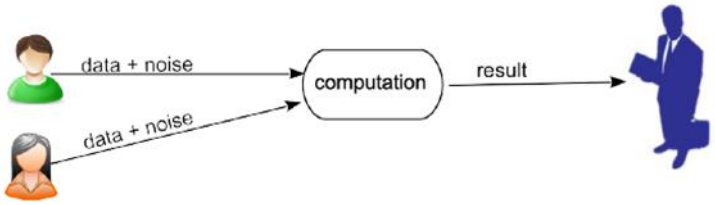
Global differential privacy

Local differential privacy

Data collected over different subjects and each of them inject noise to their data. Each user runs a differential private algorithm on their data.

An external party (not necessary trusted) combines all the (noised) data received from the users to get a final result. Noise can cancel out or be subtracted. True answer plus noise; noise is typically larger than in the global case.

Differential privacy works well if dataset is large



Local differential privacy

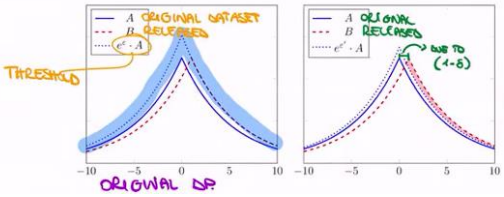
$$P[A(x) = o] \leq e^{\varepsilon} P[A(x') = o]$$ → we consider the inputs x,x' instead of D, D'.

The computation over each single x basically reproduces the same result. In this case, any output should not depend on user's secret.

Example: US Census Bureau deployed OnTheMap (2008), a web-based application that shows where workers are employed and where they live.

Based on a varion of ε-differential privacy, called approximate differential privacy ((ε, δ)-differential privacy):

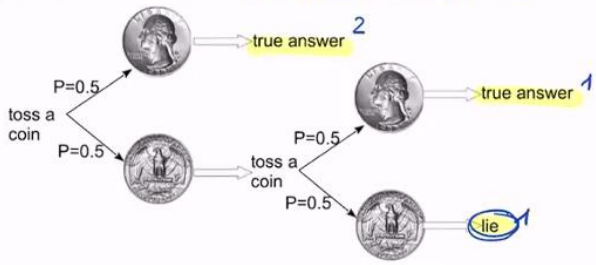
- ε is the privacy budget
- δ is related to the confidence $(1-\delta)$ that the result satisfies ε-differential privacy



Differential privacy based on coin tossing is widely deployed

- Google Chrome browser to collect browsing statistics (Rappor)
- Apple iOS and MacOS to collect typing statistics

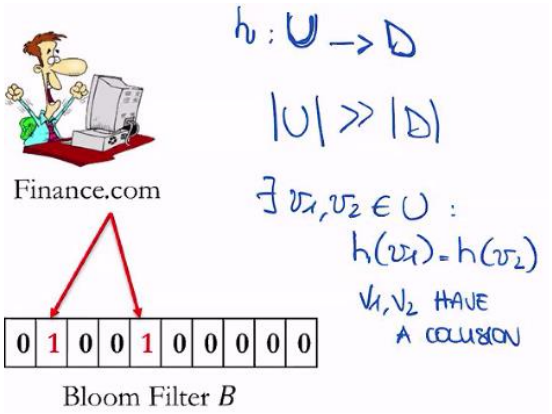
All deployments are based on randomized response



- $P(R = yes \mid Truth = yes) = 3/4 = 1/2 + (1/2 \cdot 1/2)$
- $P(R = yes \mid Truth = no) = 1/4 = 1/2 \cdot 1/2$

Rappor is used by Chrome browser to collect browsing history of people and protecting them. Each user has one value *v* out of a very large set of possibility and is based on the use of two techniques: Bloom Filter and two levels of randomized response: permanent and instantaneous.

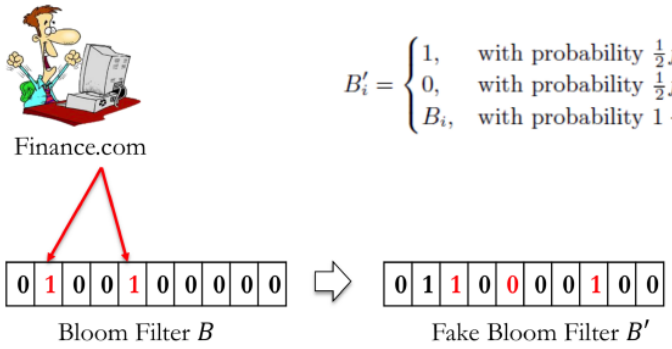
Compression: use *h* hash functions to hash input string to k-bit vector (Bloom Filter).



Can compress really much the representation.

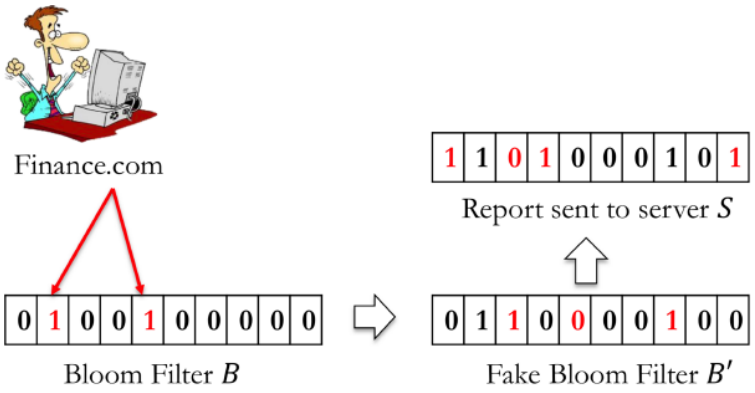
h function over the value v that can give value 0 or 1: h(v)= {0,1}

Permanent randomized response: from B a B' permanent randomized response in created with (user tunable) probability parameter *f*. → is called permanent because it is done once and the result is stored: B' is memorized and will be used for all future reports.



Instantaneous randomized response: send a report to the server of size *k* bit generated from B':

- flip bit value 1 with probability 1-q
- flip bit value 0 with probability p.



- Google
  - ε =2 for particular data that are uploaded
  - ε =8-9 is an upper limit over the lifetime of the user
- Apple
  - ε =6 for macOS
  - ε =14 for iOS 10
  - ε =43 for beta version iOS 11 (version unknown)

Differential privacy works well if we have only a small number of outliers with respect to the general solution: so if we have a huge data collection and only few outliers.

Count, histogram computations: differential privacy works well [presence/absence of a single record ca change the result slightly].

Sum computation: the application of differential privacy can be a problem: what is the total income earned by men/women? A single very high income would cause a lot of noise for this worst-case individual.

What does it happen when the privacy budget ends? No solution.

Authentication and Access Control

Different security strategies:

- **Prevention**= take measures to prevent system from being damaged {[lock the door]
- **Detection**= take measure that detect when, how and whom the system has been damaged [missing items from your house]
- **Reaction**= take measures so that you can recover your system from damages [call the police]

Security has a cost even we don't apply any security measure.

Security objectives (CIA):

- **Confidentiality**= prevent unauthorized disclosure of information

- **Integrity**= prevent unauthorized modification of information. Avoid people to make changes or make use of information.
- **Availability**= guarantee that information (or resources) are always availability to authorized users.

Authentication

= one of the basic for providing systems with the ability of identifying its users and confirming their identity.

- Identification= by parties that need to be authenticated, users declare who they are and present proofs of this
- Authentication= by the system doing the authentication, to certain of the identity presented.

Users authentication is necessary for: access control; security logging.

Cryptography

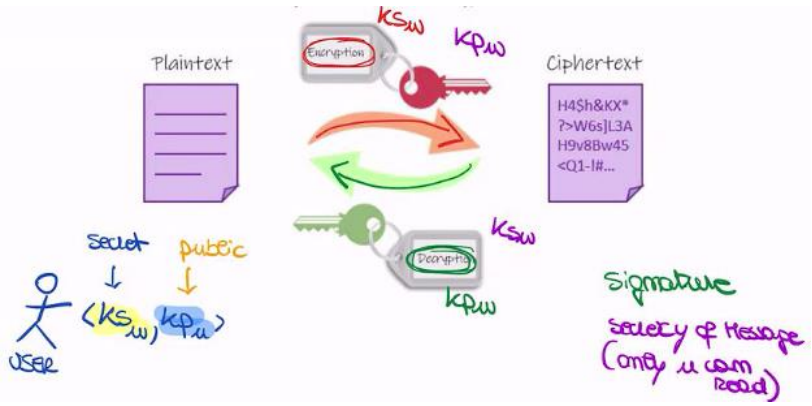
= transforms a cleartext into a non-intelligible (encrypted text or ciphertext) and viceversa.

Is based on the use of a key to encrypt and decrypt messages.

We can classify in two classes the encryption algorithms:

- **Symmetric Encryption**
  Start from the plaintext (= original file that we want to hide) and we obtain a *ciphertext* using an encryption key (= secret tool). The secret piece of information used to pass from one direction to the other and viceversa is the same.
  The same private key is used for encryption and decryption.
  The key is secret and known to both sender and receiver.

- **Asymmetric Encryption**
  Each subject possesses a pair of keys (public & private), one for encryption, the other for decryption.
  Decryption key needs to remain secret, while the encryption key can be known to everyone.



Generally asymmetric encryption technique proved to be a little bit more protected than symmetric technique. But symmetric encryption technique is more efficient. So what happens is that we use asymmetric encryption to establish the connection channel and then move to symmetric encryption the new generated key that we just changed.

Authentication is said to be mutual= I need to make sure that the system I am talking with is the system I am expecting to; there is nobody in the middle between me and the system; Th system has to verify that I am the user I am claiming to be.

Authentication can be considered the primary security service.

Authentication can be based on:

- Something the user knows [password]
- Something the user has [token]
- Something the user is [biometric trait-face recognition]

Or we can combine them [multi-factor authentication].

Password-based authentication: based on pairs username + password. → oldest and most widely used method: simple, cheap, easy to implement but weak. Weakness: password can be guessed, snooped while typing, sniffed while passing over the network, spoofing impersonating the login interface.

Authentication based on possession: based on possession by user of tokens (small in size) each of which has a cryptographic key that can be used to prove the identity of the token to a computed. Usually used in combination with passwords. Weakness→ proves the identity of the token, not the identity of the user, can also be lost, stolen, forgot.

Authentication based on user characteristics: based on biometric characteristics of the user [fingerprints, face recognition, typing cadence, signature]. There is a tolerance errors interval to be properly tuned. It requires an initial enrollment phase that defines a profile.

Access control

= evaluated access requests to the resources by the authenticated users and, based on some access rules, it determines whether they must be granted or denied (the request):

- It may be limited to control only direct access.
- It may be enriched with inference, information flow and non-inference controls.

Correctness of access control rests on:

- Proper user identification/authentication: none should be able to acquire the privileges of someone else.
- Correctness of the authorizations against which access is evaluated (which must be protected from proper modifications).

Authentication is also important for accountability [updates over the files] and establishing responsibility.

Each principal (logged subject into the system) should correspond to a single user → no shared accounts.

In open system it should rely on authenticity of the information, in contrast to authenticity of the identity (authentication) → credential-based access control.

Any kind of access control model has 3 classes of elements:

- **Policy**= defines guidelines/rules that need to be satisfied to be authorized by the system to access [closed= nobody can; open= everybody apart from whom explicitly denied]
- **Model**= formally defines the access control specification and enforcement (rules). Logical way for enforcing policy.
- **Mechanism**= implements the policies through low level functions [software and hardware]

→ not necessary 1:1 relationship between the last 2, may be 1:n.

Can change mechanism while leaving unchanged policy (no change the way the access is controlled, while changing the software used). → flexibility to the system.

The implementation of access control mechanisms based on the definition of **reference monitor** that must be:

- Tamper-proof: cannot be altered, like black-box, unless authorized
- Non-bypassable: mediated all accesses to the system and its resources
- Security kernels: confined in a limited part of the system
- Small enough to be susceptible of rigorous verification methods

The implementation of a correct mechanism is far from being trivial and is complicated by need to cope with:

- Storage channels (residue problem): storage elements such as memory pages and disk sectors must be cleared before being released to a new subject, to prevent data scavenging.
- Covert channels: channels that are not intended for information transfer [program's effect in the system load] that can be exploited to their information

Security policies can be distinguished in:

- Discretionary (DAC)
- Mandatory (MAC)
- Role-based (RBAC)
- Credential-based
- Attribute-based (ABAC)

Administrative policies = define who can specify authorizations/rule governing access control.

DAC

= enforce access control on the basis of:

- The identity of the requestors (or on properties they have)
- And explicit access rules that establish who can or cannot execute which actions on which resources.

They are called discretionary as users can be given the ability of passing on their right to other users (granting and revocation of rights regulated by an administrative policy).

Discretionary policy is the simplest one we can think about. Often reported as HRU (from later formation by Harrison, Ruzzo and Ullman). Can be represented as a 2x2 table, called access matrix since the authorization state (or protection system) is represented as a matrix.

Abstract representation of protection system found in real systems (many subsequent systems may ne classified as access matrix-based).

State of the system defined by a triple SOA:

- S set of subjects (who can exercise privileges)
- O set of objects/resources (on which privileges can be exercised) subjects may be considered as objects, in which case S ⊆ O
- A access matrix, where rows are subjects, columns are objects and *A[s,o]* reports the privileges of *s* on *o*.

Changes of states via command calling **primitive operations**

- Enter *r* into *A[s,o] OR* Delete *r* from *A[s,o]*
- Create subject *s' OR* Destroy subject *s'*
- Create object *o' OR* Destroy object *o'*

The shape of the access matrix is the following:

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

→ inside the cells we have sort of privileges.

Which is the problem/ how should I implement access matrix?

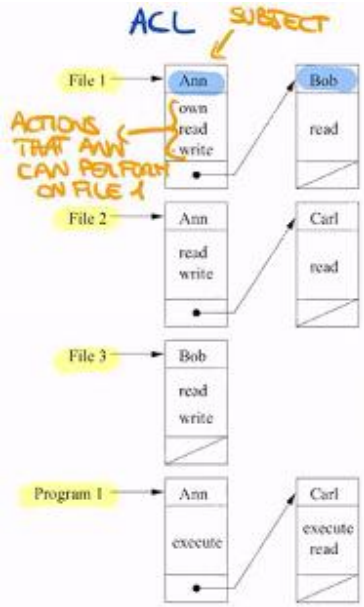
The matrix is usually very large (huge number of subjects and objects) and also very sparse: strong matrix implies also a waste of memory space.

Alternative approaches:

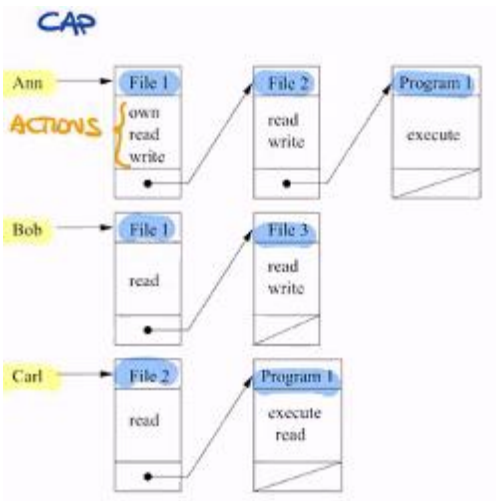
- **Authorization table**: simply keep track of the non-null triples, storing it in a table. Used in DBMS

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

- **Access control lists** (ACLs): read the matrix



- **Capability lists** (tickets): storing the list by row



Usually in systems ACL is used since User tends to leave the system, while files don't

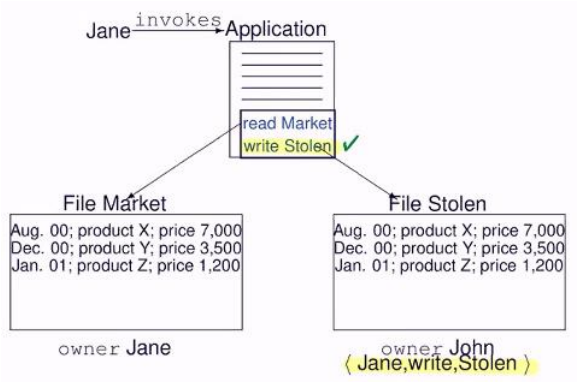
ACL vs Capabilities

If everything is centralized in the system, then in better to have ACL. Otherwise, if it is not centralized, then is better to have something different.

• ACLs require authentication of subjects. • The per-object basis usually wins out so most systems are based on ACLs. • ACLs provide superior for access control and revocation on a per-object basis. • Some systems use abbreviation form of ACLs [Unix 9 bits].	• Capabilities do not require authentication of subjects, but require unforgeability and control of propagation of capability. • Capability provide superior for access control and revocation on per-subject basis.

DAC weaknesses: constraint only **direct access**, so making explicit request of a subject over an object. We have no control on what happens to information once released. DAC is vulnerable from Trojan horses exploiting access privileges of calling subject.

Application running using the privileges of the calling subjects.



A malicious subject may substitute the original version of the *application* including two additional instructions, in order to make Jane write a copy of the file that should remain secret: add a privilege for Jane, to write file *Stolen*.

Jane, subject that is not aware of what there is written in the application (she does not read the listing of instructions written in *application*, inadvertently makes the information about the prices of products sold in the company flow to another file which can be read by John, who should not be authorized to read file *Market*.

Thanks to this Trojan Horse, the same information is copied into a file that John can read. Does not block the flow of information, DAC check only the direct, not indirect flow of information.

MAC

Tries to restrict information flows and prevent the problem caused by Trojan Horses. → Impose restrictions on information flow which cannot be bypassed by Trojan Horses.

Makes restriction between users and subjects operating on their behalf:

- User = human being [Jane]
- Subject = process in the system (program in execution). It operates on behalf of a user. [Application running]

While users may be trusted not to behave improperly, while the programs they execute are not.

The most common form of mandatory policy is multilevel security policy. It is based on classification of subjects and objects. We have two classes of policies: Secrecy-based [Bell La Padula model] and Integrity-based [Biba model]

Security classification

A security class is characterized by two components:

- **Security level** is an element of hierarchical set of elements, like TopSecret TS, Secret S, Confidential C, Unclassified U: TS > S > C > U
  Or like Crucial C, Very Important VI, Important I: C > VI > I
- **Categories** are are labels characterized by non a specific order, set on a non-hierarchical set of elements [Administrative and Financial]. It may partition different area of competence within the system. It allows enforcement "need-to-know" restrictions.

The combination of the two introduces a partial order on security classes, called **dominates ≥**

$$(L_1, C_1) \geq (L_2, C_2) \Leftrightarrow L_1 \geq L_2 \wedge C_1 \supseteq C_2$$

→ A security class which is denoted by security level L and category C dominates another one with the same shape if two conditions are satisfied: the level of the first class is gte the level of the second one (like to say, more secure), the set of security class of the first class is a superset of the set of categories of the second one.

$$(TS, \{Fin, Eco\}) \geq (C, \{Eco\}) \Leftrightarrow TS \geq C \wedge \{Fin, Eco\} \supseteq \{Eco\}$$

Security classes together with dominance ≥ introduce a lattice (SC, ≥):

Reflexivity of $\succeq$ $\forall x \in SC : x \succeq x$

Transitivity of $\succeq$ $\forall x, y, z \in SC : x \succeq y, y \succeq z \Longrightarrow x \succeq z$

Antisymmetry of $\succeq$ $\forall x, y \in SC : x \succeq y, y \succeq x \Longrightarrow x = y$

Least upper bound $\forall x, y \in SC : \exists\, !z \in SC$

- $z \succeq x$ and $z \succeq y$
- $\forall t \in SC : t \succeq x$ and $t \succeq y \Longrightarrow t \succeq z$.

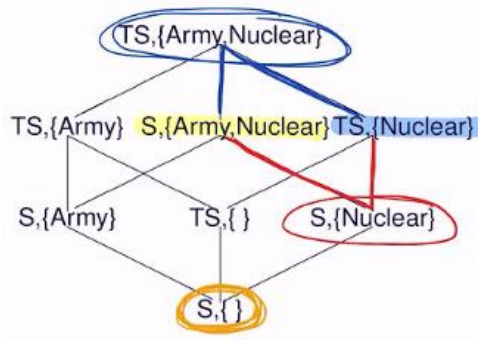
Greatest lower bound $\forall x, y \in SC : \exists\, !z \in SC$

- $x \succeq z$ and $y \succeq z$
- $\forall t \in SC : x \succeq t$ and $y \succeq t \Longrightarrow z \succeq t$.

Least Upper Bound: We can find one SC which dominates both x and y, but which is not dominated by any other SC which dominates both of them.

Greatest Lower Bound: We can find one SC which is dominated by both x and y, but which does not dominate any other SC which is dominated by both of them.

Example: We have 2 security levels: top secret TS, secret S, where TS≥S. We have 2 categories: Army, Nuclear.



Least upper bound (lub): TS, {Army, Nuclear}

Greatest lower bound (glb): S, {Nuclear}

We say that users are associated with a clearance (= assigned security class), so user can connect to the system at any class dominated by his clearance. Subjects activated in a session take on the security class with which the user has connected.

Secrecy classes:

- Assigned to users reflect user's trustworthiness not to disclose sensitive information to individuals who do not hold appropriate clearance.
- Assigned to objects reflect the sensitivity of information contained in the object and the potential damage that could result from their improper leakage.

Categories define the area of competence of users and data.

Bell La Padula Model

Define mandatory policy for secrecy.

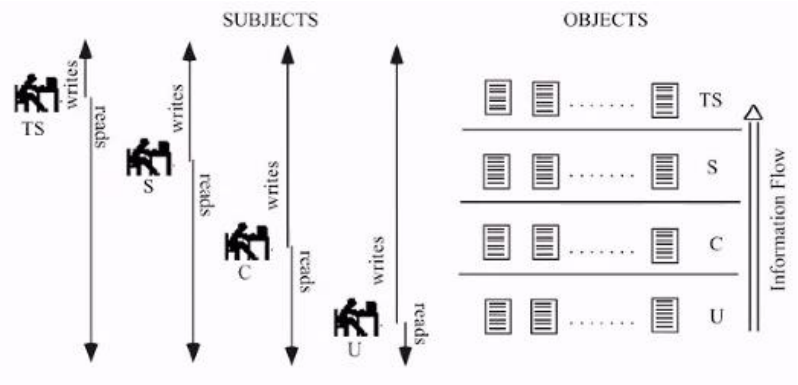
Different versions of the model have been proposed (with small differences or related to specific application environments); but the basic principles remain the same.

Goal: prevent information flow to lower or incompatible security classes.

a. **Simple property** A subject *s* can read object *o* only if $\lambda(s) \geq \lambda(o)$ [for example subject (TS, {Fin}) can read object (S, {Fin})] → Don't want to read something that is more confidential then my level of trustworthiness. **no read up**
b. ***-property** A subject *s* can write object *o* only if $\lambda(o) \geq \lambda(s)$ → I don't want secret information to flow downwards. → **now write down**

Alice has clearance (TS, {FIN, ECO}) connect to run P1 with (S, {ECO}).

Easy to see that the Trojan Horses leaking information through legitimate channels are blocked.



Exception to axioms

Real-world requirements may need mandatory restrictions to be bypassed:

- Data association: A set of values seen together is to be classified higher than the value singularly taken [name and salary]
- Aggregation: An aggregate may have higher classification than its individual items [the location of a singular military ship is unclassified but the location of all the ships of a fleet is secret]
- Sanitization and Downgrading: Data may need to be downgraded after some time [embargo]. A process may produce data less sensitive than those it has read. → trusted process. A trusted subject is allowed to bypass (in a controlled way) some restrictions imposed by the mandatory policy.

DAC and MAC are not mutually exclusive, can work together and combine them. DAC provides discretionarily within the boundaries of MAC.

Like DAC, also MAC has some **limitations**: controls only over channels of information (flow through legitimate channels, explicit channels of information you are regulating). Remains vulnerable to convert channels (= not intended for communicating information, but can be exploited to leak information). Every resource of observable of the system shared by processes of different levels can be exploited to create a cover channel.

Examples:

- Low level subject ask to access a resource [like CPU]. The system returns the file does not exist (if the system creates the file the user may not be aware when necessary.
- Putting together different pieces of information, we can be able to attack the system.
- Timing channel: a high level process can lock shared resources and modify the response times of process at lower levels. With timing channel, the response returned to a low level process is the same, it is the time to return it that changes.

Covert cannel analysis is usually done in the implementation phase (to assure that a system's implementation of the model primitive is not too weak). Interface models attempt to rule such channels in the modelling phase. Non-interference: the activity of high level processes must not have any effect on processes at lower or incomparable levels.

Integrity mandatory policy

Control only improper leakage of information.

Do not safeguard integrity: information can be tampered.

Dual policy can be applied for integrity, based on assignment of (integrity) classification.

Integrity classes:

- Assigned to users reflect users' trustworthiness not to improperly modify information
- Assigned to objects reflect the degree of trust information contained in the objects and the potential damage that could result from its improper modification/deletion. → how much damage I can do if someone not authorized uses/modifies that data?

Categories define the area competence of users and data.

Biba Model

Defines mandatory policy for integrity.

Goal: prevent information to flow to higher or incomparable security classes.

Strict integrity policy. Based on principles dual to those of BLP:

- Simple property A subject s can read object o only if $\lambda(o) \geq \lambda(s)$
- *_property It does not safeguard integrity but simply signals its compromise

Limitations of Biba model: flow restrictions may result too restrictive; it enforces integrity only by preventing information flows from lower to higher access classifications: it captures only a very small part of the integrity problem.

Integrity is a complex concept. Ensuring that really no resources has been modified in an unauthorized or improper way and that data stored in the system correctly reflect the real world they are intended to represent. → we need to prevent flaws and errors.

Any data management system has functionalities for ensuring integrity:

- Concurrency control and recovery techniques: to ensure that no concurrent access can lead to data loss or inconsistency
- Recovery techniques: to recover the state of the system in case of errors or violations
- Integrity constraint: that enforce limitation on the values that can be given to data

RBAC

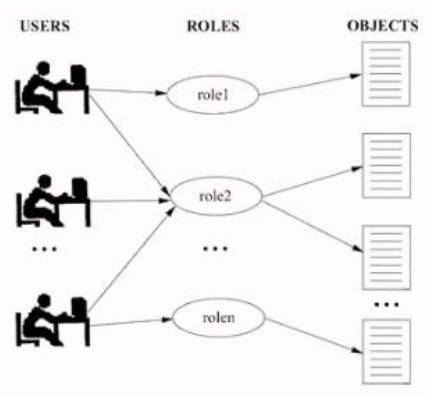
= Role-based access control polices

Role named set of privileges related to execution of a particular activity.

Access of users to objects mediated by roles.

- Roles are granted authorizations to access objects. Depending on the role, we can activate privileges.
- Users granted authorizations to activate roles
- By activating a role *r* a user can execute all access granted to *r*
- The privileges associated with a role are not valid when the role is not active

There is a difference between group (= set of users) and role (= set of privileges).



Sort of indirect passage from user to privileges, only passing through roles.

Role hierarchy defines specialization relationships. Hierarchical relationships define authorization propagation:

- If a role is granted authorization to execute (action,object) → all roles generalization of *r* can execute (action, object)
- If *u* is granted authorization to activate role *r* → *u* can activate a generalization of *r*

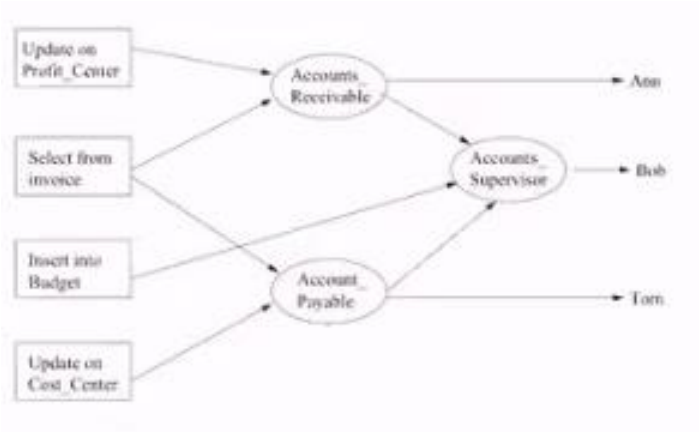
Adv: **Easy to manage** the specification of authorization; **Role hierarchy** makes authorization management easier; **Restrictios** can be associated with roles such as cardinality or mutual exclusions; It allows associating

with each subject the **least set of privileges** the subject needs to execute it works, so limits buses and damages due to violations and errors; Roles allows the enforcement of **separation of duty** (split privileges among different subjects).

Work on role-based models has been addressing also:

- Relationships beyond hierarchical [secretary ca operate on behalf of his manager], sort of elegation of roles
- The hierarchy-based propagation is not always wanted [privileges not propagate to subroles]
- Enriched administartive policies (authority confinement)
- Relationships with user indentifiers [secretary of -]
- Addiutional constraints [dynamical separation of duty, competion of an activity requires particiaption of at least *n* individuals]

In SQL privileges can be grouped in **roles** that can be assigned to users or to other users (nested). Activating a role, we enable for all the privileges in a subset rooted at that role. → Roles can be granted to users with **grant option** (the user can grant it to others)



Administrative polices

Define who can grant and revoke access privileges authorizations:

- Centralized: a privileges authority (system security officer) is in charge of authorization specification.
- Ownership: the creator of an object is its owner and as such can administer access authorization on the object. Ownership not always clear in: hierarchical data models or RBAC framework

Authority to specify authorizations can be delegated. Delegation is often associated with ownership: the owner of an object delegates administrative privileges to others.

Decentralized administration introduces flexibility, but complicates the scenario.

Separation of duty = no user (or restricted set of users) should have enough privileges to be able to abuse the system.

- Static who specifies the authorizations must make sure not to give “too much privileges” to a single user

- Dynamic the control on limiting privileges is enforces at runtime: a user cannot use "too many" privileges but he can choose which one to use. The system will consequently deny other accesses. It is more flexible, because we do not restrict in advance.

Operations: order-goods, send-order, record-invoice, pay

Four employees. Protection requirements:
at least two people must be involved in the process

- static: the administrator assigns tasks to users so that none can execute all the four operations
- dynamic: each user can execute any operation, but cannot complete the process and execute all four

DAC – Expanding authorizations

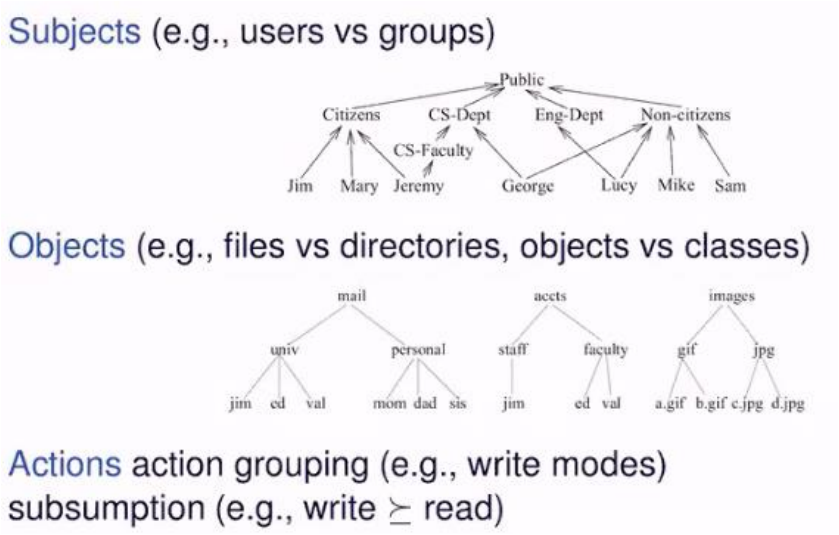
DAC extension of the basic triples when needed.

- User groups. Users collected in groups and authorizations specified for groups.
- Conditional. Validity of authorizations dependent on satisfaction of some conditions.
  - System-dependent: evaluate satisfaction of system predicates location/time [only if you are connected from your office] [only in a specific timing]
  - Content-dependent: depending on value of data [DBMS]
  - History-dependent: dependent on history of requests [when you have access too many of them, you are blocked]

Relatively easy to implement in simple systems. Introduce complications in richer models.

- Support abstractions (grouping them). Usually hierarchical relationship: user/groups; object/classes; files/directories. Authorizations may propagate along the hierarchies.

Support of hierarchies can be applied to all dimensions of authorizations.



Usefulness of abstractions limited if exceptions are not possible [all employess but X can read a file]

→ Support negative authorization: (Employess, read, file, +) (Sam, read, file, -)

Presence of permissions and denials can bring inconsistencies: how should the system deal with them?

→ easy way is via negative authorization.

To support exceptions via negative authorization. Negative authorizations first introduced by themselves are:

- Open policy: whatever is not explicitly denied can be executed as opposed to. → need to add negative authorization
- Closed policy: only accesses explicitly authorized can be executed. → need to add positive authoriation

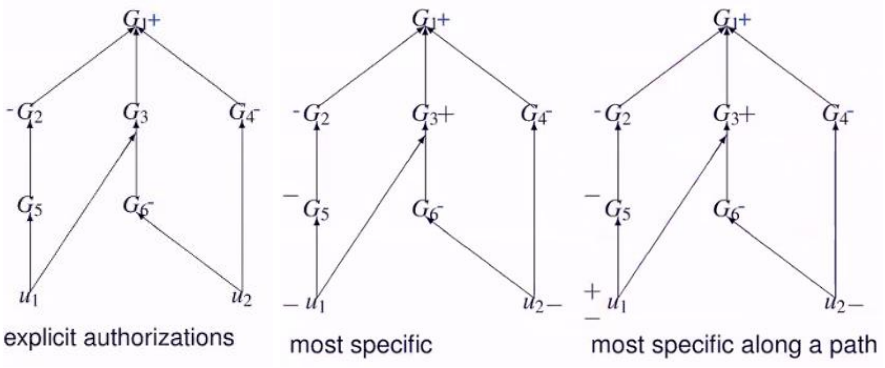
We can also have hybrid approaches, where we have both positive and negative authorization:

- If for an access we have **both** positive and negative → Inconsistency
- If for an access we have **neither** positive not negative → Incompleteness

To solve <u>incompleteness</u> is easy: we can rely on the idea that I am in an open world or close world and I apply as a default the decision take by open policy (permit) or closed policy, so deny. [underlying open or closed policy]

To solve <u>inconsistency</u>: harder to solve because I need to choose between negative and positive authorization. How to solve?

- Denials-take-precedence: negative authorization wins
- Most-specific-takes-precedence: the authorization that is more specific wins, so closer to the subject.
- Most-specific-along-a-path-takes-precedence: the authorization that is "more specific" winly o the paths passing through it. Authorizations propagate until overridden by more specific authorizations.



Authorizations have associated explicit priorities, but it is difficult to manage.

Most important rules are listed before, while going down we find less important rules. Giving the responsibility to specify the importance to security administrator and controlled administration is difficult to enforce.

Grantor-dependent: strength of authorizations depends on who granted them, or can depend on time [more recent rules are more important].

Time-dependent: strength of authorizations depends on time they have been granted [More recent rules, are more important]

The use of conflict resolution policy is not mutual exclusive. We can apply first one then another type.

Sometimes we are allowed to present digital certificate: don't present yourself with your identity, but present yourself with an electronic piece of data that is certified by the server and which combines the public key with the identity or some properties [membership in groups]. The server can use certifiactes to enforce access control.

Recent access control models are:

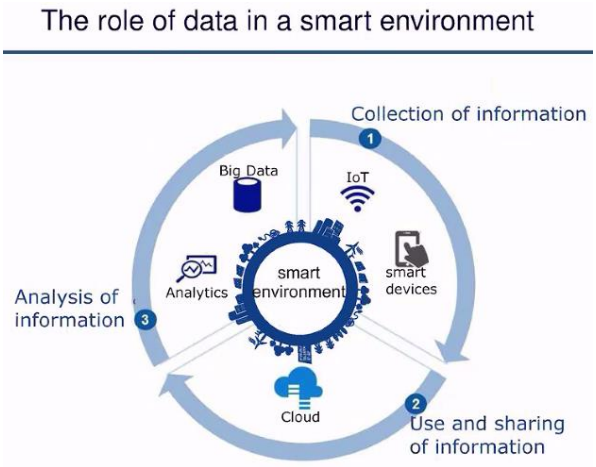
- ABAC = Attribute-based access control. The authorization is defined on attribute/properties of the requester
- CBAC = Credential-based access control

PRIVACY AND DATA PROTECTION IN EMERGING SCENARIOS

ICT is growing, everything is connected. This can lead to better protection mechanism, they are better in providing continuity and disaster recovery and better prevention and response.

On the other hand, the system becomes also weaker, because we have a larger surface of attack, so more exposed to violations. The weakest device in a system is the one that needs much more protection. The risk in case of damage is considerably more … reliable, if you don't have complete control on machines/infrastructures. We can lose control over data and processes.

Protection starts from infrastructure and devices, but we need also to protect in terms of communication protocols. We need to be protected from malware and attacks coming from the net.



External providers in this case play the role of data management and data storage.

The cloud allows users and organizations to rely on external providers for storing, processing and accessing their data.

Adv: high configurability and economy of scale; data and services are always available [infrastructure of physical duplication, good network connection]; scalable infrastructure for applications.

Cons: user lose control over their own data; new security and privacy problems. → We need solutions to protect data.

Cloud Service Providers CSPs apply security measures in the services they offer but these measures protect only the perimeter and storage against outsiders. **Functionality →** Using current cloud solutions, they provide all the functions we need, BUT with no protection, implies full trust in the CSP [Dropbox] that have full access to the data. The encryption key is decided and known by and with the cloud provider.

Another solution: other CSP [Boxcryptor] instead of functionality, there are based on **protection**. The user encrypts data before storing data. They cannot provide the same functionalities because they don't know the key, so cannot perform queries nor analysis, search… so we have limited functionalities.

Escudo-cloud's vision: Solution that provide guarantees giving the data owners both: full control over their data and cloud functionality over them. The boundary of trust is with the data owner, so we don't give trust to the cloud provider to look at sensitive information, but use techniques that support direct processing of encrypted data in the cloud.

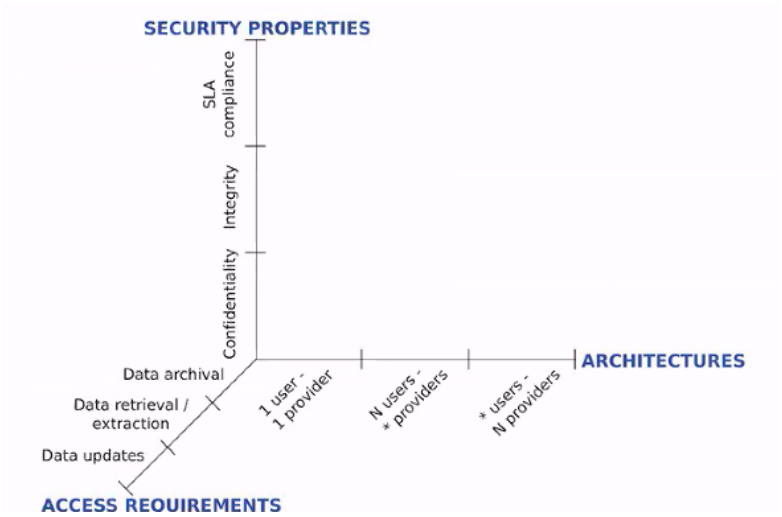
Encryption is one of the most widely used method s for protecting data.

For data protection we need access and usage control. One important aspect is the possibility of sharing data among clouds in a controlled manner.

Confidentiality

Every time we talk about data we need, to maintain confidentiality, to minimize release/expositions. We are releasing a more complex object that is also correlated with possibly other sources of information which are already available or will become available in the future.

We can classify the problems that need to be solved in 3 dimensions: security properties, architectures, access requirements.



Security properties:

- Confidentiality. Data externalities stores; Users' identities; Actions that uses perform on the data may expose identities.
- Integrity. Data externally stored [guarantee that data is exactly the same you stored at the cloud provider and none con modify them]; Computation and query results. [If with a query I ask for the all people living in Milan suffering from diabetes I get the complete and correct list of people]

- SLA compliance. Assurance and certification.

Access requirements

- Data archival: Building simple techniques we can use; upload/download data and protection of data storage.
- Data retrieval/extraction: Need to have support for fine-grained data retrieval [techniques that while protecting data are able to extract information I am looking for]
- Data update: Need to support both retrieval and updates information in a secure manner and protect actions and their effect to the actions of whom are not authorized.

→ Growing level of complexity

Architectures:

- 1 user & 1 provider: Need to protection of data stored at rest, fine-grained retrieval [queries and updates] and query privacy/integrity.
- n users & * providers: Need to have access control [not only one user relying on "Dropbox" but a company with many employees] and need to manage the situation in which multiple subject are writing in the same time or modifying the data.
- * users & n providers: Security shared between different parties, so we need to control data sharing and computation

Combining all the different dimension (number of subject involves) we come up with a complex scenario.

We need to take care of the possibility that the provider can be curious, lazy or malicious.

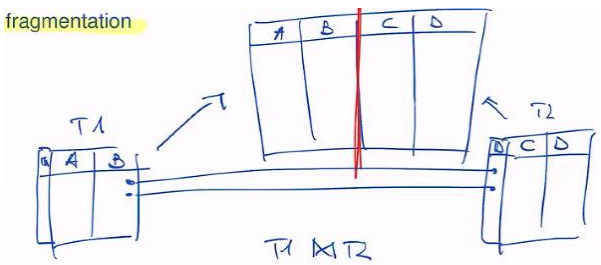
Issues to be addressed: data protection, query execution (searches over the data), private access, data integrity and correctness, access control enforcement, data publication and utility.

The research community has been very active and produced several contributions and advancements: solutions for protecting confidentiality of stored data; indexes supporting different types of queries; inference exposure evaluation; data integrity; selective access to outsourced data….

Protecting data confidentiality

The solutions are: encryption[2]; fragmentation; the two together.

Fragmentation: if you have a table storing different attributes and having different rows, we split the tables in different fragments, that store subset of attributes. We can reconstruct the original table by applying a join on the fragments (this is called vertical fragmentation).



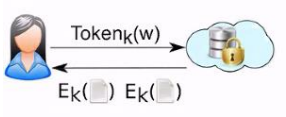
[2] We are assuming symmetric encryption.

The most common technique used is encryption: choose encryption key, encrypt data on your side before sending the data to the cloud provider. → Protected to the eyes of who does not know the encryption key. Data confidentiality is provided by wrapping a layer of encryption around sensitive data. Encryption is typically applied at the tuple level. Cloud provider cannot decrypt data nor process/access it.

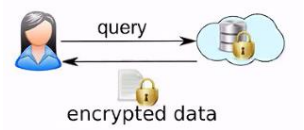
The server can be honest-but-curious and should not have access to the resource content.

There are different approaches (fine-grained access) that have been proposed:

- **Keyword-based searches directly on the encrypted data**: supported by specific cryptographic techniques. Need to know in advance the keywords that we want to search.

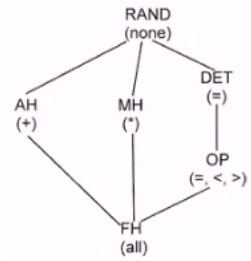


- **Homomorphic encryption**: support the execution of operation directly on the encrypted data. The most common technique support production of operations and some kind of operations dependently on the type of data we have. Problems: quite expensive in computational time to encrypt data; the size of data increase a lot because of the encryption.

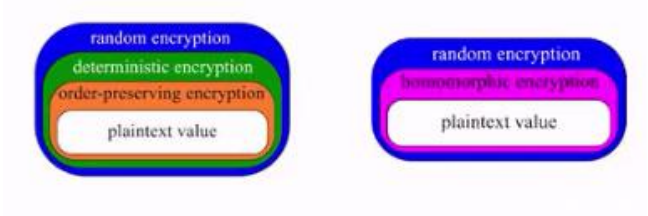


1:1000 times operations w.r.t. operations over encrypted data

- **Encryptions schemas**: to maintain distinction between values, can perform searches (check wheter two values are the same or different. Each column can be encrypted with different encryption schema, depending on the conditions to be evaluated on if [Google encrypted BigQuery]

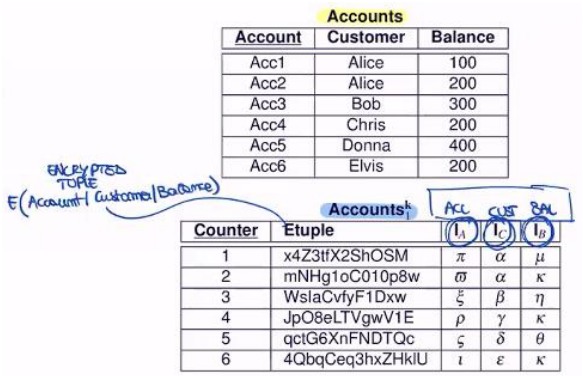


- **Onion encryption**: different onion layers each of which supports the execution of a specific SQL operation [commercial name CryptDB]. So encrypt data several times using different algorithms.
  Looking at the picture/graph, while you go down, you gain in functionalities. Protection is higher at the top of the hierarchy. So peeling out the onion structure we are reducing the level of protection of the data.
  Plaintext value = the more sensitive that is at the center of the onion.
  Order-preserving encryption= preserves the order among values.
  Deterministic encryption = two values are the same or different (but does not depend on order).
  The external layer is the more protected one but does not permit to make any action on data.



→ Consider that once we have removed a level of encryption, because the data have already been exposed.

An alternative solution is **indexes**: metadata attached to the data and used for fine-grained information retrieval and query execution. Can also be complementary to encryption (even with encryption users want to have the ability to perform searches based on metadata). Indexes associated with attributes are used by the server to select data to be returned in response to a query.



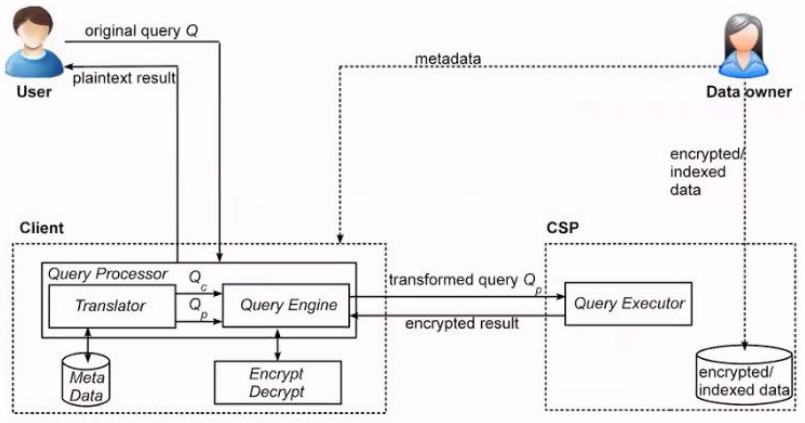
Accounts

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

ENCRYPTED TUPLE E(Account|Customer|Balance)

Accounts^k

Counter	Etuple	I_A	I_C	I_B
1	x4Z3tfX2ShOSM	π	α	μ
2	mNHg1oC010p8w	ϖ	α	κ
3	WslaCvfyF1Dxw	ξ	β	η
4	JpO8eLTVgwV1E	ρ	γ	κ
5	qctG6XnFNDTQc	ς	δ	θ
6	4QbqCeq3hxZHklU	ι	ε	κ



There are different ways of indexing:

- **Direct**. Adv: simple and precise so the result is exactly what you expect to obtain. Disadv: weakest because maintains the distinction between the plaintext values. Data are a little bit more exposed.

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	WslaCvfyF1Dxw	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)**. Partition-based or hash-based. Adv: supports for equality queries and collisions remove plaintext distinguishability. Disadv: Result may contain spurious tuples (post processing query); still vulnerable to inference attacks.

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	WslaCvfyF1Dxw	ξ	λ	α	ν
4	JpO8eLTVgwV1E	ρ	υ	β	γ
5	qctG6XnFNDTQc	ι	μ	α	σ
6	kotG8XnFNDTaW	χ	o	β	ψ

- **Flattened (1:n)**. Make the frequency distribution flat, so that we cannot reconstruct the correspondence between plaintext values and index value. Adv: Deceases exposure to inference attacks. Disadv: remains vulnerable to dynamic observations.

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	WslaCvfyF1Dxw	ξ	λ	α	ν
4	JpO8eLTVgwV1E	ρ	υ	β	γ
5	qctG6XnFNDTQc	ι	μ	α	σ
6	kotG8XnFNDTaW	χ	o	β	ψ

Partition-based index

Partition-based index (kind of bucket indexing) creates a n:1 correspondence, so index function is defined over the domain of the attribute, so multiple plaintext values are mapped to the same index.

Queries should be executed over the index in a simple manner, otherwise there is something not what we expected to be.

Example:

Consider an arbitrary plaintext attribute A_i in relational schema R, with domain D_i

D_i is partitioned in a number of non-overlapping subsets of values, called partitions, containing contiguous values

Given a plaintext tuple t in r, the value of attribute A_i for t belongs to a partition

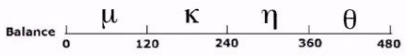
- function $ident_{R.A_i}(p_j)$ assigns to each partition p_j of attribute A_i in R an identifier

The corresponding index value is the unique value associated with the partition to which the plaintext value $t[A_i]$ belongs

- $Map_{R.A_i}(v) = ident_{R.A_i}(p_j)$, where p_j is the partition containing v

$Map_{R.A_i}$ can be order-preserving or random

Random mapping



- $Map_{Balance}(100) = \mu$
- $Map_{Balance}(200) = \kappa$
- $Map_{Balance}(300) = \eta$
- $Map_{Balance}(400) = \theta$

Query conditions supported by partition-based index are the ones where conditions are boolean formulas over terms of the form: Attribute operator Value; Attribute operator Attribute (if both are indexed according to a partition-based index). The operation allowed for *op* include = > < ≥ ≤ (comparison between values).

How can we translate a condition that we have in a query?

A_i=v I want all tuples having value v for the attribute over which I have built the index.

Look into the encrypted database of the index value corresponding. Then, will have back all the tuples that belong to that interval/partition and among them we need to select at the client side only the one of interest.

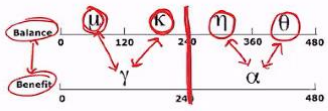
Depend if the order is maintained or not maintained in the domain. If it is maintained, is easier, translating v into the corresponding index value and look for all the index value < > (ecc) wrt the threshold. On the other hand, if the domain is not order preserving, we need to find all the partitions to list all index values of interest.

A_i=A_j When it comes to compare different attributes values, the translation needs to consider that the 2 attributes might have used different labeling/indexing function when they have been mapped. The idea of translation of the conditions is to look at the 2 mapping functions and find all the possible combination of values that overlap.

$$Map_{cond}(A_i = A_j) \Longrightarrow \bigvee_{\varphi}(I_i = ident_{A_i}(p_k) \wedge I_j = ident_{A_j}(p_l))$$

where φ is $p_k \in$partition(A_i), $p_l \in$partition(A_j), $p_k \cap p_l \neq \emptyset$

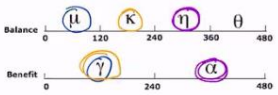
Example



Map_{cond}(Balance=Benefit) $\Longrightarrow$ (Balance=μ $\wedge$ Benefit=γ)
$\vee$ (Balance=κ $\wedge$ Benefit=γ)
$\vee$ (Balance=η $\wedge$ Benefit=α)
$\vee$ (Balance=θ $\wedge$ Benefit=α)

A_i<A_j If one attribute should be lower than the other, we just need to find out the combination of index values that possibly satisfy for at least a value in the partition the condition. Need to look at each combination one by one and extract the ones that could satisfy at least for a value within the extremes the condition you are imposing.

Example



Map_{cond}(Balance<Benefit) $\Longrightarrow$ (Balance=μ $\wedge$ Benefit=γ)
$\vee$ (Balance=μ $\wedge$ Benefit=α)
$\vee$ (Balance=κ $\wedge$ Benefit=γ)
$\vee$ (Balance=κ $\wedge$ Benefit=α)
$\vee$ (Balance=η $\wedge$ Benefit=α)
$\vee$ (Balance=θ $\wedge$ Benefit=α)

How does query execution in this scenario? Each query Q on the plaintext DB is translated into:

- A query Q_s to be executed at the server
- A query Q_c to be executed at client on the result

Accounts

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

Accounts$_2^k$

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	qctG6XnFNDTQe	ς	β	κ
6	4QbqC3hxZHklU	ι	β	κ

we have a collision and we get a Spurious tuple

So we have to remove at the client side row 4

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

→ we have a spurious tuple

Hash-based index

Alternative way with respect to defining partitions.

Based on the concept of one-way hash function. We can compute easily and quickly *h(x)*, but is hard to compute *x:h(x) = y*, so we cannot invert the function, at least not in an easy manner.

We store in the cloud, instead of the original value, the hash value.

Given a plaintext tuple t in r, the index value corresponding to $t[A_i]$ is $h(t[A_i])$

Important properties of any secure hash function h are:

- $\forall x,y \in D_i: x = y \implies h(x) = h(y)$ (determinism)
- given two values $x,y \in D_i$ with $x \neq y$, we may have that $h(x) = h(y)$ (collision)
- given two distinct but near values x,y ($|x-y| < \varepsilon$) chosen randomly in D_i, the discrete probability distribution of the difference $h(x) - h(y)$ is uniform (strong mixing)

Accounts

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

$Accounts_2^k$

Enc_tuple	I_A	I_C	I_B
x4Z3tfX2ShOSM	π	α	μ
mNHg1oC010p8w	ϖ	α	κ
WslaCvfyF1Dxw	ξ	δ	θ
JpO8eLTVgwV1E	ρ	α	κ
qctG6XnFNDTQc	ς	β	κ
4QbqC3hxZHklU	ι	β	κ

So using hashes we cannot say anything about the relative distance/order of the original values, because it guarantees this kind of mixing.

Support queries where conditions are Boolean formulas over terms of the form:

- Attribute = Value → Translate into index = h(value)
- Attribute1 = Attribute 2 → if attribute1 and attribute2 are indexed with the same hash function. → index1=index2 and h(A1)=h(A2)

It does not support range queries/inequality condition (a solution similar to the one adopted for partition-based methods is not viable): > and <. And we still have spurious tuples.

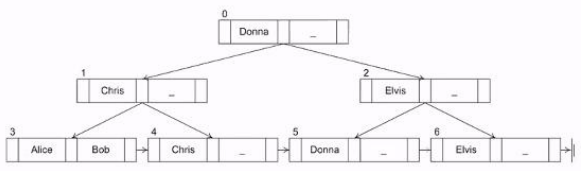
Interval-based queries are not supported by hash-based indexes (does by the interval-based). [18<age<30] Order-preserving indexing techniques: comparing the ordered sequences of plaintext and indexes would lead to reconstruct the correspondence. B+-tree example:

Order-preserving techniques not good because we can get 1 to 1 correspondence between plain text and encrypted text

v1 < v2 < v3

e(v1) < e(v2) < e(v3)

Solution: B+-Tree structure



B+-tree Table

ID	Node
0	(1,Donna,2,_,_)
1	(3,Chris,4,_,_)
2	(5,Elvis,6,_,_)
3	(Alice,Bob,4)
4	(Chris,_,5)
5	(Donna,_,6)
6	(Elvis,_,_)

Encrypted B+-tree Table

ID	Enc_Node
0	/WKu5y8laqK82(
1	AXYaqohgyVObU
2	IUf7R.PK5h5fU
3	uOtdm/HDXNSqU
4	GLDWRnBGlvYBA
5	a9yl36PA3LeLk
6	H6GwdJpXiU8MY

Searchable encryption

Order Preserving Encryption Schema (OPES) takes as input a target distribution of index values and applies an order preserving transformation so that the resulting index values follow the target distribution. The comparison can be applied on encr. data and query evaluation does not produce spurious tuples, but is vulnerable with respect to inference attacks.

Order Preserving Encryption with Splitting and Scaling (OPESS) schema creates index values so that their frequency distribution is flat.

Fully homomorphic encryption schema allows performing specific computation on encrypted data and decryption of the computation result, yields the result of operations performed on the plaintext data. So if I have values *a* and *b* and I want to compute $a+b \rightarrow E(a)+E(b) = E(a+b)$ which property usually does not hold for regular encryption algorithms. But it is computationally too expensive.

Inference exposure

There are two conflicting requirements in indexing data:

- Want effective query execution (correct and quick)
- Want not to permit inference and linking attacks

It is important to measure quantitatively the level of exposure due to the publication of indexes:

ε = Exposure Coefficient {ε=1 completely exposed, server can reconstruct the orig. db}

There are different scenarios we can consider. The computation of the exposure coefficient ε depends on two factors:

- The indexing method adopted
  - Direct encryption 1:1 [1 index : 1 plaintext]
  - Hashing n:1 [n plaintext : 1 index]
- The a-priori knowledge
  - Freq+DB^k the attacker knows completely outsourced db (the sequence distribution of values). The attacker may be interested in: determine the existence of a certain tuple I the original database (plaintext content); determine the correspondence between plaintext values and indexes (indexing function).
  - DB+DB^k (worst case scenario) the attacker wants to reconstruct the indexing function to absorb updates to the dataset. The attacker wants to determine the correspondence between plaintext values and indexes. This scenario is less common than the previous one.

There are different ways to compute the exposure coefficient:

	Direct Encryption	Hashing
Freq+DB^k	Quotient Table	Multiple subset sum problem
DB+DB^k	RCV graph	RCV line graph

Freq-DB

Knowledge

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

Accounts^k

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

Inference

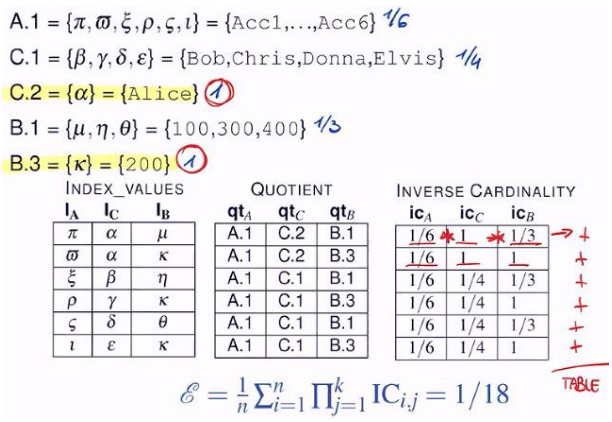
- I_A = Account
- I_C = Customer
- I_B = Balance
- $\kappa = 200$ (indexing inference)
- α =Alice (indexing inference)
- ⟨Alice,200⟩ is in the table (association inference)
- Alice is also associated with a value different from 200 ("100,300,400", all equiprobable)

The fact that the attacker is able to identify Alice and assume she has income 200 is called association inference.

→ Positive and negative inference

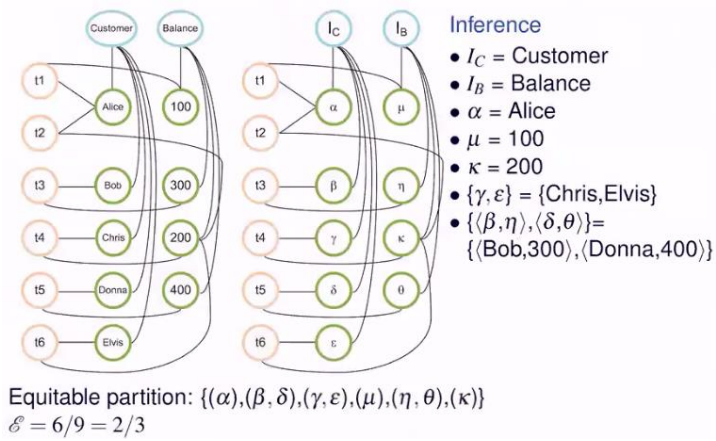
Protection happens when there are multiple values having exactly the same number of occurrences of index/value.

Assessment of index exposure based on equivalence relation where index/plaintext values with values with **same number of occurrences** belong to the same class (can be considered equivalent in terms of occurrence). Exposure of values in equivalence class C is 1/|C|



The higher the number of attributes, the higher the risk of exposure of the plaintext values.

DB+DB^k



Depending on how much correspondences in edges I can make, the coefficient would be higher or lower.

The more the graph is similar with itself, the higher the number of nodes having the same edges, the higher the graph is protected.

What does it happen if we use hashing exposure? We are more protected. Because we have fewer number of values which combine together different original values, so the frequency distribution is obtained by summing up the frequencies of different values that map to the same hash value. If we have a flat distribution of the index, we are fully protected!

Bloom Filter

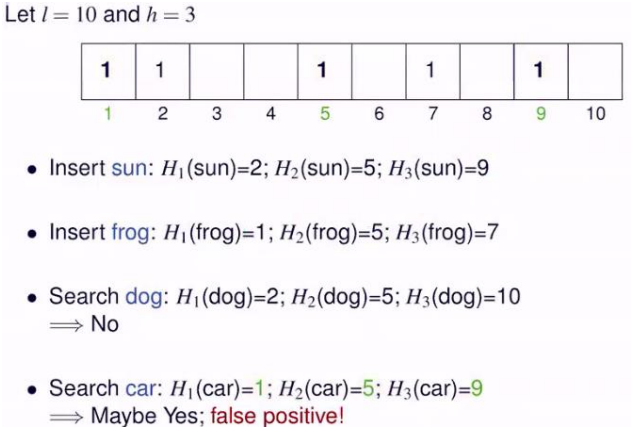
A bloom filter is at the basis of the construction of some indexing techniques. It is an efficient method to encode set membership:

So we have a sort of compression

- Set of n element (where n is large)
- Vector of l bits (l is small)

- H independent hash functions: $H_i : \{0,1\}^* \rightarrow [1,l]$ so demine multiple hash functions that permits to represent a large domain of elements through a small number of bits. → So we have a sort of compression of data representation.
- Represent a value using a vector of bits and each position in the vector is obtained applying one different hash function over the elements I want to insert in the bloom filter.

The hash function obtains a value that is either 0 or 1.



In each position we have a Hash function

Can create a false positive results (spurious). Looking only at the simple sequence of the bits, we cannot say which is the content of the bloom filter.

Generalization of hashing is very space efficient, but elements cannot be removed [cannot put =0 because in this way we may remove also other elements we are not taking into consideration].

Yield a constant false positive probability that is theoretically considered not acceptable but is acceptable in practical applications as fine price to pay for space efficiency.

Data Integrity

Two aspects of integrity in outsourced data needs to be taken into consideration:

- Integrity in storage: data must be protected against improper modifications: unauthorized updates to the data must be detected [opened letter]
  Data integrity in storage relies/is based on digital signatures. Signatures are computed at tuple level. The problem is that the verification time of integrity grows if we download a high number of tuples.
- Integrity in query computation: query results must be correct and complete: server's misbehavior in query evaluation must be detected.

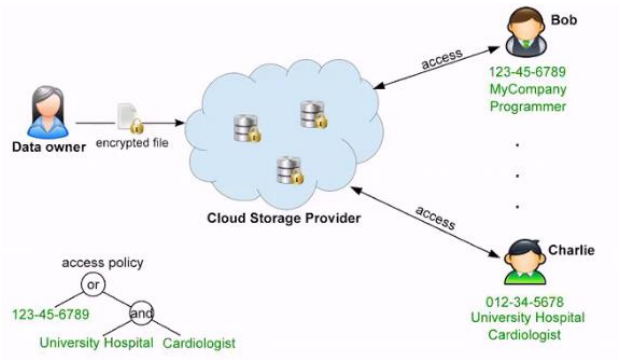
What does it happen if I move all my information system to the cloud and I do not trust the provider for looking at the data? I cannot move the access control directly to the cloud, asking him to perform checks on my behalf.

Authorization enforcement may not be delegated to the provider: data owner should always remain in control.

We need something that is more flexible and more autonomous. → using different types of encryption

Attribute-based encryption (ABE): Each user has his/her own private key, each data item is encrypted with a different key and based on the properties/characteristics/roles of each subject, they can compute the key used

to encrypt data only if the characteristics satisfy the policy/conditions regulating access of data. [Professor can only see the marks in only his/her exam, while secretary can see all exams]



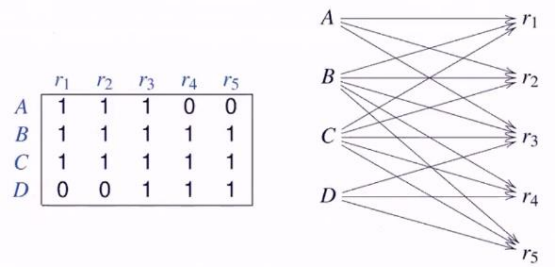
Selective encryption: The authorization policy defined by the data owner is translated into an equivalent encryption policy. Each user has his own key and different files are encrypted sing different keys, so that users can only rear files encrypted with the key they know. I distribute key to subject such that only some subjects can decrypt some specific files. → we translate authorization into knowledge of encryption key.

Permits to enforce access control without the intervention of the data owner and the cloud provider does not have any role in providing access control. → Not prevent download, but prevent opening the downloaded files.

Authorization policy $\mathcal{A}$ is a set of permissions of the form ⟨user, resource⟩. It can be represented as an access matrix or a directed and bipartite graph (having a vertex for each user u and for each resource r and an edge from u to r for each permission $\langle u,r \rangle$.

In the example we have 1 if the subject is authorized to read the file and 0 otherwise.

[A, D] are users; [r1, r5] are resources.



Encryption policy

= the authorization policy defined by the data owner is translated into an equivalent encryption policy.

There are 2 possible solutions:

- Encrypt each resource with a **different key** and give users the keys for the resources they can access. But it requires each user to manage as many keys as the number of resources she is authorized to access.
- Use a **key derivation method** for allowing users to derive/compute trough a mathematical process from their user keys all the keys that are entitled to access. It allows limiting to one the key to be released to each user, not reversible.

Based on a key derivation hierarchy $(\mathcal{K}, \preceq)$: where $\mathcal{K}$ is the set of keys in the system and $\preceq$ is the partial order relation defined on $\mathcal{K}$.

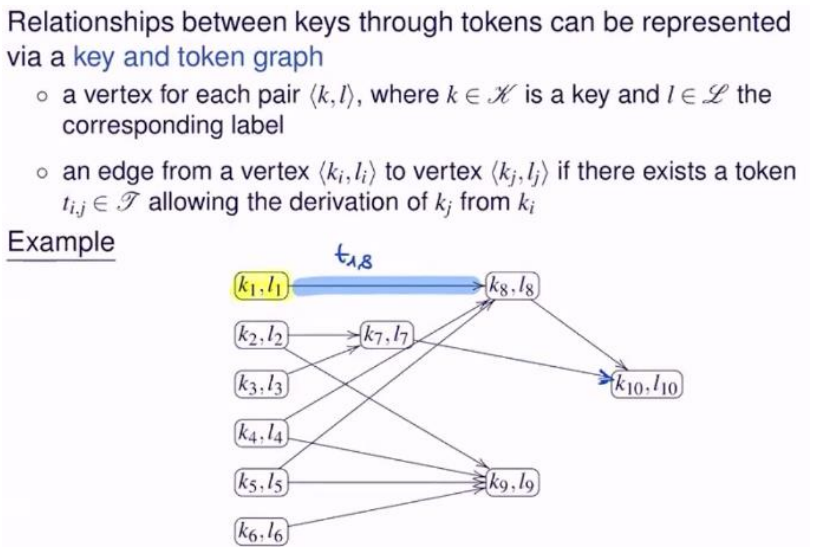
The knowledge of the key of vertex v_1 and of a piece of information publicly available allows the computation of the key of a lower level vertex v_2 such that $v_2 \preceq v_1$.

Each node of the graph corresponds to an encryption key. Keys are arbitrarily assigned to vertices. An edge in the graph means that a subject knowing the key of the starting point of the edge, can compute the key at the arranging point of the edge.

A public label l_i is associated with each key k_i. a piece of public information $t_{i,j}$ called **token** is associated with each edge in the hierarchy.

Given an edge (k_i,k_j) token $t_{i,j}$ is computed as $k_j \oplus h(k_i,k_j)$ where $\oplus$ is the n-ary *xor* operator and h is a secure hash function.

The advantages of tokens are: they are public and allow users to derive multiple encryption keys, while having to worry about a single one; they can be stored on the remote server (just like the encrypted data) so any user can access them.



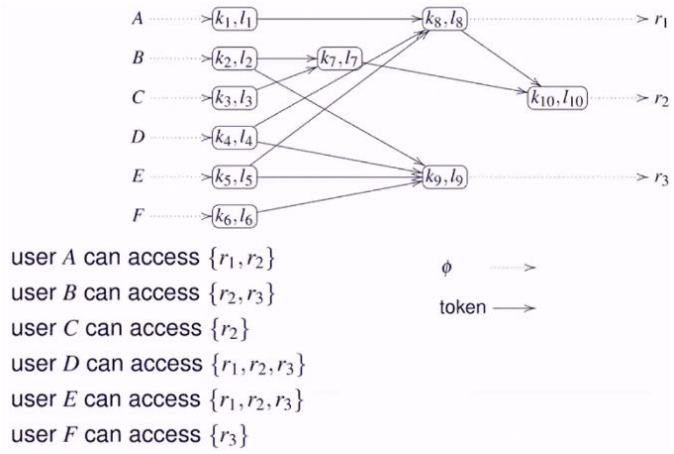
h is a secure hash function because it is not invertible: $h(x)$ is easy to computed, while h^{-1} is hard to compute.

How can we translate authorization policy into the corresponding encryption policy?

Starting form two assumptions (desiderata): each user can be released only a single key and each resource is encrypted only once (with a single key).

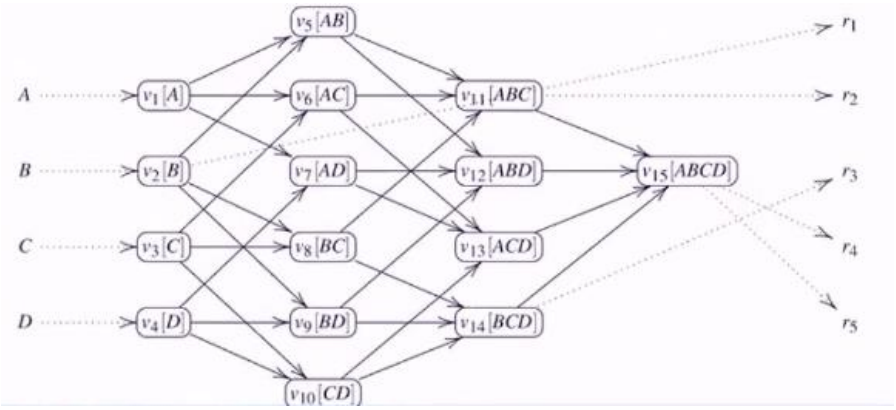
Φ: $\mathcal{U} \cup \mathcal{R} \rightarrow \mathcal{L}$ describes the association between user and the label of her key; the association between a resource and the label of the key used for encrypting it.

Goal: translate an authorization policy into an equivalent encryption policy $\mathcal{E}$.



To translate the authorization policy (0/1 matrix) into a corresponding encryption policy maintaining the correspondences, we can use a simple solution which consists in using a different key for each user, a different key for each resource and a token generated and published for each permission. Producing and managing a token for each single permission can be unfeasible in practice (huge number of tokens). To limit the number of tokens, we can exploit *acls* and user groups: group users with the same access privileges and encrypt each resource with the key associated with the set of users that can access it.

If I build all the possible group of users, the structure/schema becomes very large.



In this manner, using and defining a structure which builds a key for each possible subset of users and that connects vertex in our structure based on self-contained relationship (=starting point is a subset of arriving point). The structure is built considering this subset containing property. The subset in the graph are possible *acls*. → Satisfy equivalence relation between the access control policy and the encryption policy.

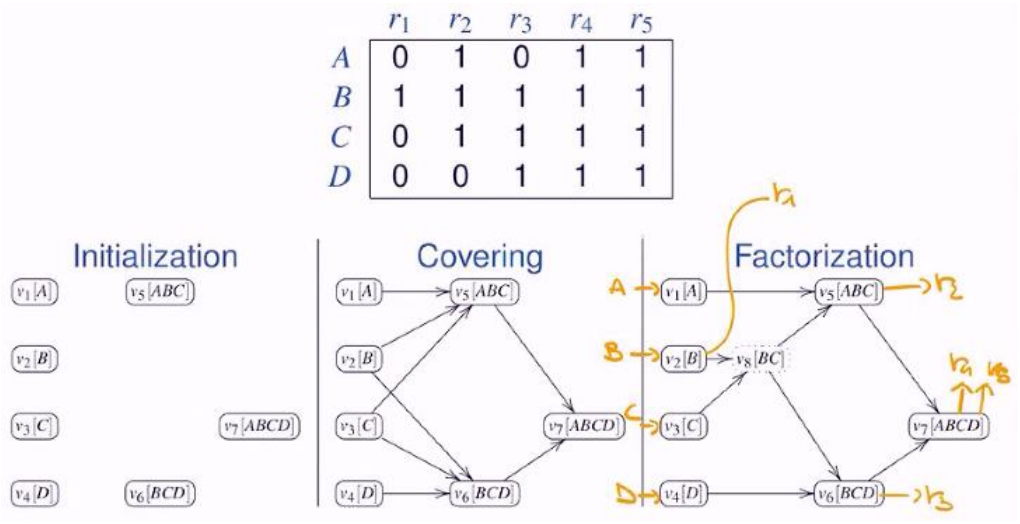
$acl(r_4)$=ABCD $acl(r_2)$=ABC $acl(r_1)$=B

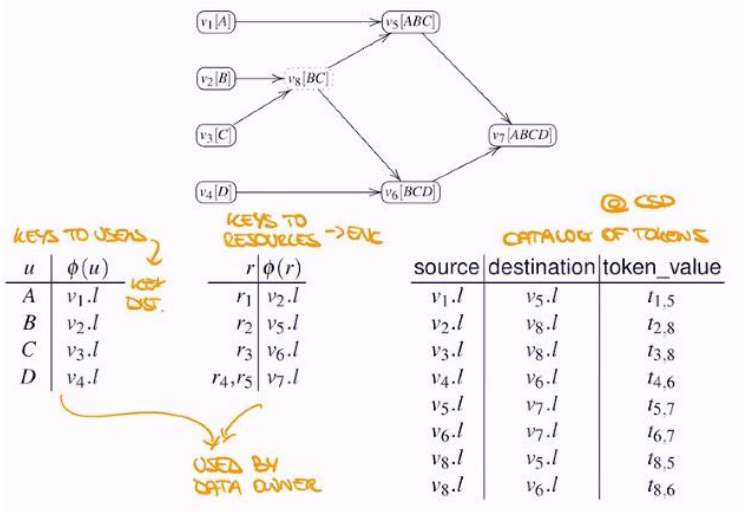
User groups that do not correspond to any acl do not need to have a key. We can shortcut some paths if the node in between are not really used for encryption. To limit the number of tokens in the system, we use only relevant acls.

→ We use only the relevant ones: only vertices associated with user groups corresponding to actual acls need to be associated with a key. The encryption policy graph may include only the vertices that are needed to enforce a given authorization policy, connecting then to ensure a correct key derivability. Other vertices can be included if they are useful for reducing the size of the catalog (add only few additional nodes).

Starting from an authorization policy:

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 $u \in v'.acl$.
3. Factorize common ancestors (factorization)





Over-encryption

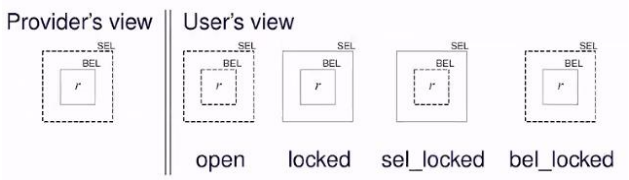
If the authorization and access control list change, the data owner needs to create a new key, re-encrypt the resource with the new key and re-upload resources. But this is inefficient.

A possible solution is: **over-encryption** = outsource to the cloud provider managing updates.

The resources are encrypted twice: by the owner with a key shared with the users and unknown to the provider (BaseEL Level) and by the provider with a key shared with authorized users (SurfaceEL level).

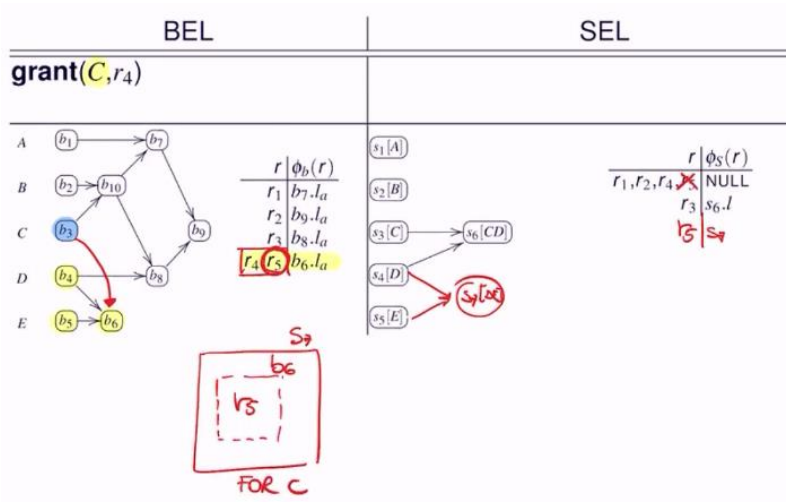
→ a subject need to know both the key of the owner and the key of the provider to open up the resource.

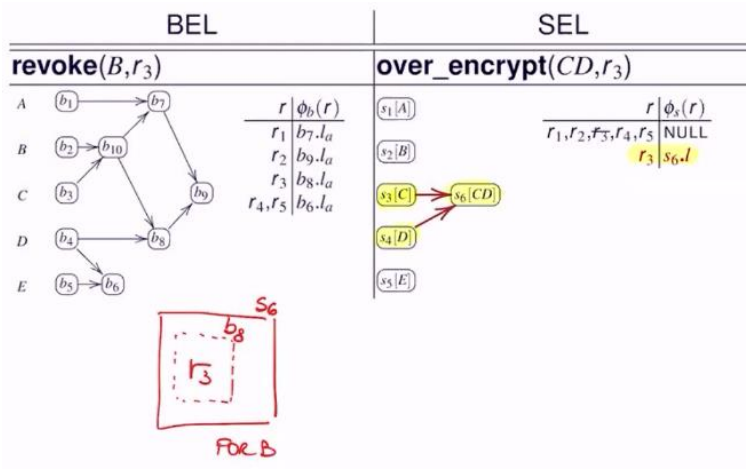
Every grant and revoke operations require the addition of new tokens at the BEL level and the update of the SEL level according to the operations performed.



Revoke operation: need to protect it through the encryption performed by the cloud provider.

Grant operation: if a BEL key protects multiple resources and access is to be granted only to a subset of them, there is the need to protect at SEL level the resources on which access is not being granted.





An alternative solution for enabling authorization in access control policy enforced to selective encryption:

Mix&Slice

= avoiding intervention of cloud provider, consider the resources as different small pieces and encrypt in such a way that there is a complete mixing of all the bits in the encrypted portion of the resource from the original content of the resource itself.

If you remove even a small piece (fragment) of the resource, nobody will be able to perform the decryption operation anymore.

Other issues

Support for write privileges for data collections with multiple owners.

Selective encryption for supporting subscription-based authorized policies. [streaming system: you can look at content for which you pay]

Fragmentation and encryption

We can split data and store information in such a way that only subjects that are authorized can combine their content.

Encryption makes query evaluation and application execution more expensive or not always possible: gives too much protection with respect what we need.

Often what is sensitive is the association between values of different attributes, rather than values themselves [name, salary] → protect associations by breaking them, rather than encrypting when possible.

So we combine encryption and data fragmentation.

We need to understand what needs to be protected: we model this requirement through confidentiality constraint, that is a set of attributes we consider in a relational model whose joint visibility needs to be protected, because is considered sensitive.

Besides this, we can have singular attributes and associations that can be considered sensitive.

Sensitive attributes: the values of some attributes are considered sensitive and should not be visible (singleton constraints).

Sensitive association: the associations among values of given attributes are sensitive and should not be visible (non-singleton constraints).

Confidentiality constraints – Example

R = (Name,DoB,Gender,Zip,Position,Salary,Email,Telephone)

- {Telephone}, {Email}
  - attributes Telephone and Email are sensitive (cannot be stored in the clear)
- {Name,Salary}, {Name,Position}, {Name,DoB}
  - attributes Salary, Position, and DoB are private of an individual and cannot be stored in the clear in association with the name
- {DoB,Gender,Zip,Salary}, {DoB,Gender,Zip,Position}
  - attributes DoB, Gender, Zip can work as quasi-identifier
- {Position,Salary}, {Salary,DoB}
  - association rules between Position and Salary and between Salary and DoB need to be protected from an adversary

The data owner splits information over 2 fragments stored in 2 independent servers/cloud providers that cannot communicate with each other.

The original relation is decomposed/splitted in 3: $\langle R_1, R_2, E \rangle$

- R_1 and R_2 include a unique tuple ID needed to ensure lossless decomposition
- $R_1 \cup R_2 = R$
- E is the set of encrypted attributes and $E \subseteq R_1$, $E \subseteq R_2$
- for each $c \in \mathscr{C}$, $c \not\subseteq (R_1 - E)$ and $c \not\subseteq (R_2 - E)$

$R(A_1, A_2, \ldots A_m)$ ← $R_1 \subseteq A_1 \ldots A_m$ @ CSP1; $R_2 \subseteq A_1 \ldots A_m$ @ CSP2; $E \subseteq A_1 \ldots A_m$ ENCRYPTED ATTRIBUTES

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

tid	Name	YoB	SSN^k	$Disease^k$
1	Alice	1980	jdkis	hyaf4k
2	Bob	1980	u9hs9	j97;qx
3	Carol	1970	j9und	9jp'md
4	David	1970	p0vp8	p;nd92
5	Eva	1970	8nn[	0-mw-n
6	Frank	1960	j9jMK	wqp9[i
7	Gary	1960	87l'D	L0MB2G
8	Hilary	1960	8pm}n	@h8hwu

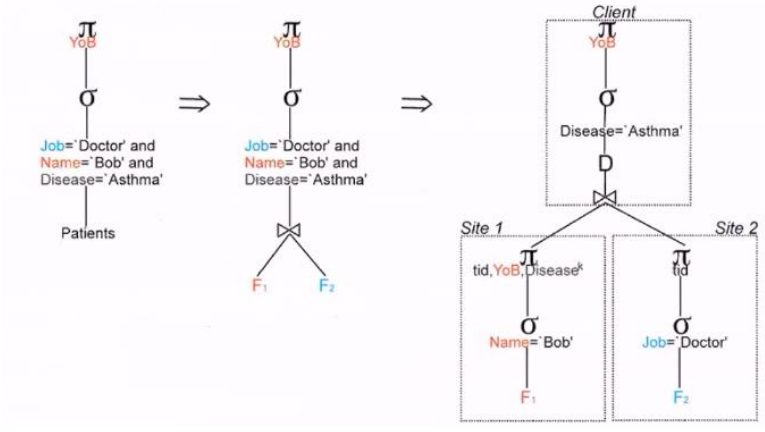
F_2

tid	Job	SSN^k	$Disease^k$
1	Clerk	uwq8hd	jsd7ql
2	Doctor	j-0.dl;	0],nid
3	Nurse	8ojqdkf	j-0/?n
4	Lawyer	j0i12nd	5lkdpq
5	Doctor	mj[9;'s	j0982e
6	Doctor	aQ14l[	jnd%d
7	Teacher	8qsdQW	OP['
8	Nurse	0890UD	UP0D@

At the logical level, we can always replace the original R with the join between R_1 and R_2.

We need to revise the different operations performed in the query in such a way that as much as the complexity of the query as possible is anticipated and moved at the cloud providers.

- F_1: (tid,Name,YoB,SSN^k,$Disease^k$)
- F_2: (tid,Job,SSN^k,$Disease^k$)



You want to extract only with name *Bob* whose job is *Doctor* and suffer from *Asthma*. I can first joint the two F and then I perform the query, but this means that you have to perform it at the client side. It is not efficient because you need to download all the data.

→ we move as much as possible the evaluation of the condition to the side of the cloud provider.

I can try to keep together those attributes that are usually queried together. → affinity matrix

We can transform the problem of fragmenting a relation into an optimization problem.

Safe: if they do not communicate one with the other, it is protected. But too strong and difficult to enforce in real environments and limits the number of associations that can be solved by fragmenting data, often forcing the use of encryption.

Allow for more than two non-linkable fragments.

A fragmentation is formed by an arbitrary number *m* of fragments. Each fragment has not attributes in common (including identifiers) neither the tuples identifier.

On the other hand, if fragments have no attributes in common we need to have guarantees of completeness of information stored in each fragment in such a way that each client can query a single fragment and retrieve all the relation content. The idea is to combine in the physical fragment *F* some attributes in the clear and all the others represented in encrypted form (a salt is applied on each encryption).

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

Enc of F_1enc(SSN, job, disease); Enc of F_2enc(SSN, name, yob , disease); Enc of F_1enc(SSN, name, yob, job)

→ join is not necessary for query evaluation.

The goal is to find a fragmentation that makes query execution efficient.

The fragmentation process can take into consideration different criteria: the number of fragments, the affinity among attributes, the query workload.

Only attributes that appear n singleton constraints (sensitive attributes) are encrypted. All attributes that are not sensitive appear in the clear in one fragment.

One possible solution is to keep the number of fragments and compute when the number of fragments is minimal. → avoid excessive fragmentation.

Example:

MEDICALDATA

SSN	Name	DoB	Zip	Illness	Physician
123-45-6789	Nancy	65/12/07	94142	hypertension	M. White
987-65-4321	Ned	73/01/05	94141	gastritis	D. Warren
963-85-2741	Nell	86/03/31	94139	flu	M. White
147-85-2369	Nick	90/07/19	94139	asthma	D. Warren

Confidentiality constraints
c_0 = {SSN}
c_1 = {Name, DoB}
c_2 = {Name, Zip}
c_3 = {Name, Illness}
c_4 = {Name, Physician}
c_5 = {DoB, Zip, Illness}
c_6 = {DoB, Zip, Physician}

Minimal fragmentation $\mathcal{F}$

- F_1 = {Name}
- F_2 = {DoB,Zip}
- F_3 = {Illness,Physician}

Merging any two fragments would violate at least a constraint

Another idea is to use max affinity.

Query workload: we want to determine a fragmentation that minimizes the query workload cost.

Fragmentation

The basic idea (hybrid scenarios) is to have two storage spaces, one in the cloud (considered not trusted) and another space (that is trusted). Since query execution is made difficult due to the use of encryption and having several fragments requires performing joins (which is expensive), we can avoid encryption by storing something in data owner part.

- Completeness: The combination of attributes should completely cover information.
- Confidentiality: The text cannot be completely represented, but can be stored at owner side.
- Non-redundancy: Don't want to have the same information at owner and cloud provider side.

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_o

tid	SSN	Job	Disease
1	123456789	Clerk	Asthma
2	234567891	Doctor	Asthma
3	345678912	Nurse	Asthma
4	456789123	Lawyer	Bronchitis
5	567891234	Doctor	Bronchitis
6	678912345	Doctor	Gastritis
7	789123456	Teacher	Gastritis
8	891234567	Nurse	Diabetes

F_s

tid	Name	YoB
1	Alice	1980
2	Bob	1980
3	Carol	1970
4	David	1970
5	Eva	1970
6	Frank	1960
7	Gary	1960
8	Hilary	1960

Query evaluation

Queries are formulated on R therefore need to be translated into equivalent queries on F_O and/or F_s.

SELECT A FROM R WHERE C:

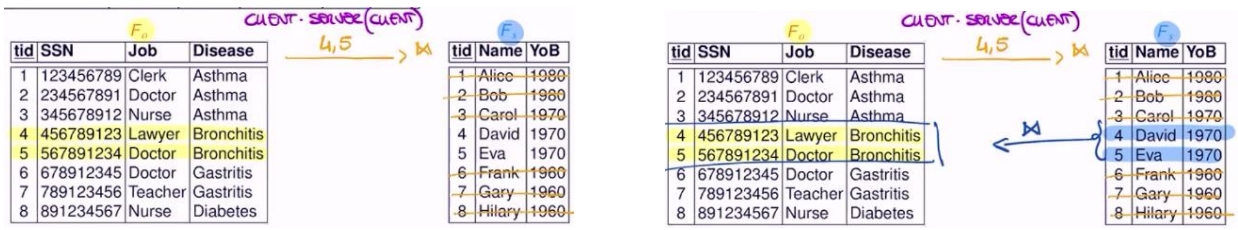
- C_o is the condition that involves only attributes stored at the client
- C_s is the condition that involves only attributes stored at the server
- C_{so} is the condition that involves attributes stored ate the client and attributes stored at the server

- F_o={SSN,Job,Disease}, F_s={Name,YoB}
- q = SELECT SSN, YoB
  FROM Patients
  WHERE (Disease="Bronchitis")
  AND (YoB="1970")
  AND (Name=Job)

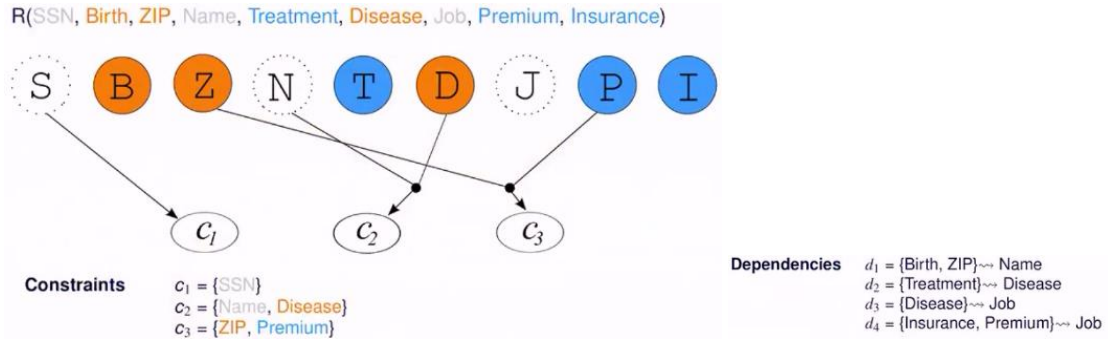
- The conditions in the WHERE clause are split as follows
  - C_o = {Disease = "Bronchitis"}
  - C_s = {YoB = "1970"}
  - C_{so} = {Name = Job}

Server-Client strategy: server (cloud provider) evaluates C_s and returns to client, who receives the result and joins it with F_o and evaluates C_o and C_{so} on the joined relation.→ only 1 flow of information (from S to C)

Client-Server strategy: client evaluates C_o and send tid of tuples in result to server, which joins input with F_s and evaluate C_s and returns to client, who joins result from server with F_o and evaluate CSO. → 2 flows of information.

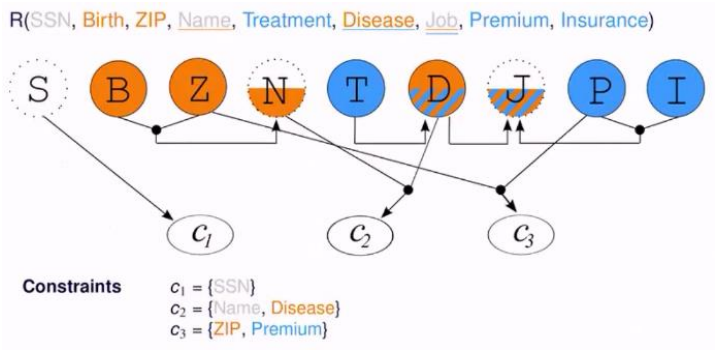


Inference



A fragmentation as the one in the picture (colors), is fine with the respect to the constraints.

Problem: there are dependencies between attributes, because at a logical level, they are strictly related.



This creates a problem for constraint c2: we have N and D converging and both of them have a part that is orange.

So when we perform fragmentation, we need to take into consideration dependences among attributes. So to compute a good fragmentation we may either store the attributes appearing in the independency rule all together in the same fragment or split the head of the independency rule in such a way that if two attributes are not stored together [premium and insurance], no inference can be done.

COMBINING INDEXES, SELECTIVE ENCRYPTION AND FRAGMENTATION

We need to make sure that the cloud provider cannot access data that we consider confidential.

Query evaluation becomes less expensive in case in which we use fragmentation instead of encryption.

If we try to use these solutions combined, what happens is that if each one of them provides a good level of protection, combining them could open the door to new inferences, new risks that were not present while considering the single techniques alone. Adding too much to the protected data, we are still exposed.

What happens if we combine these solutions?

Thy may open the door to inferences by users: indexes and selective encryption; Indexes and fragmentation.

Access control and Indexes

Selective encryption provides access control and efficiency in query is given by indexes. Provides different data views to different users but can open the door to inferences by users.

Example: each user knows the index function used to define indexes in R^e, the plaintext tuples that she is authorized to access, the encrypted relation r^e in its entirely.

AC

	acl
t_1	A
t_2	A,B
t_3	B
t_4	A,C
t_5	C

SHOPS

	Id	City	Year	Sales
t_1	001	NY	2010	600
t_2	002	Rome	2010	700
t_3	003	Rome	2011	600
t_4	004	NY	2011	700
t_5	005	Oslo	2011	700

SHOPS^e

tid	etuple	I_c	I_y	I_s
1	α	ι(NY)	ι(2010)	ι(600)
2	β	ι(Rome)	ι(2010)	ι(700)
3	γ	ι(Rome)	ι(2011)	ι(600)
4	δ	ι(NY)	ι(2011)	ι(700)
5	ε	ι(Oslo)	ι(2011)	ι(700)

K_B → K_{AB} → K_{ABC}; K_B → K_{AC} → K_{ABC}

B knowing the index value that produces *i*(2010), can see that also other tuples have the same *i* so must have the same value. Is not the same for flattened indexes.

How to prevent this kind of inference?

Different index function for each user, generate a table that is a little bit larger.

Since the i_A and i_B appear together, I know that $i_A(700)=i_B(700)$ even if I should not be authorized.

Indexes can release something even if the original plaintext has been hidden in a proper manner. If you use the same index function for different users, you are still exposed.

We need to be careful in case of collusion. Different kinds of indexes are subject to different kinds of inferences.

Indexes and Fragmentation

When we use fragmentation with an arbitrary number of fragments with some of the attributes in plaintext and all the others are encrypted and query operations on fragments, of course we have queries that involve some attributes represented in plaintext and some other in encrypted form.

Improves efficiency of query evaluation, but can cause a leakage of confidential information.

Vertical knowledge: Each observer can see both the fragments; we know exactly which are the values of the attributes represented in the index; the domain is known precisely.

Horizontal knowledge: I know someone and so I know the disease, so I also know the association.

F_1^e

salt	enc	Name	State	i_d
s_{11}	r_{11}^e	Adams	VA	α
s_{12}	r_{12}^e	Brown	MN	α
s_{13}	r_{13}^e	Cooper	CA	α
s_{14}	r_{14}^e	Davis	VA	β
s_{15}	r_{15}^e	Eden	NY	β
s_{16}	r_{16}^e	Falk	CA	γ
s_{17}	r_{17}^e	Green	NY	δ
s_{18}	r_{18}^e	Hack	NY	δ

vertical knowledge

salt	enc	Disease
s_{21}	r_{21}^e	Flu
s_{22}	r_{22}^e	Flu
s_{23}	r_{23}^e	Flu
s_{24}	r_{24}^e	Diabetes
s_{25}	r_{25}^e	Diabetes
s_{26}	r_{26}^e	Gastritis
s_{27}	r_{27}^e	Arthritis
s_{28}	r_{28}^e	Arthritis

horizontal

Name	Disease
Adams	Flu

Risks we can have by using different kinds of indexes: direct index, bucket index.

Direct index

1:1 correspondence between index values and original attribute value.

Considering vertical knowledge, we can learn that α= Flu and γ= Gastritis, because of the frequency correspondence between values, so the individuals are exposed.

Considering horizontal knowledge: I know that Adam has flu and has α, so I learn that α=Flu. Then 3 subjects are exposed.

Bucket index

Bucket indexes n:1 so we have multiple original values mapped to the same index value, so we could have collisions.

F_1^e

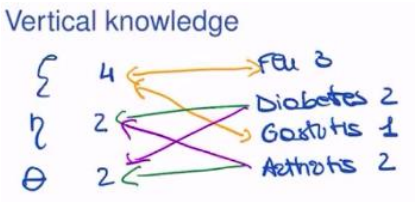
salt	enc	Name	State	i_d
s_{11}	r_{11}^e	Adams	VA	ζ
s_{12}	r_{12}^e	Brown	MN	ζ
s_{13}	r_{13}^e	Cooper	CA	ζ
s_{14}	r_{14}^e	Davis	VA	η
s_{15}	r_{15}^e	Eden	NY	η
s_{16}	r_{16}^e	Falk	CA	ζ
s_{17}	r_{17}^e	Green	NY	θ
s_{18}	r_{18}^e	Hack	NY	θ

vertical knowledge

salt	enc	Disease
s_{21}	r_{21}^e	Flu
s_{22}	r_{22}^e	Flu
s_{23}	r_{23}^e	Flu
s_{24}	r_{24}^e	Diabetes
s_{25}	r_{25}^e	Diabetes
s_{26}	r_{26}^e	Gastritis
s_{27}	r_{27}^e	Arthritis
s_{28}	r_{28}^e	Arthritis

horizontal

Name	Disease
Adams	Flu



For horizontal knowledge, we can make no inference given $i(\zeta)$= Flu, because we don't know the percentage of probability of having the same disease as Adams, because we have only horizontal knowledge.

Considering both vertical and horizontal knowledge: we percentage of probability has changes for Flu and Gastritis.

Not breaking confidentiality constraint completely.

Flatten index

F_1^e

salt	enc	Name	State	i_d
s_{11}	r_{11}^e	Adams	VA	κ
s_{12}	r_{12}^e	Brown	MN	λ
s_{13}	r_{13}^e	Cooper	CA	μ
s_{14}	r_{14}^e	Davis	VA	ν
s_{15}	r_{15}^e	Eden	NY	ξ
s_{16}	r_{16}^e	Falk	CA	π
s_{17}	r_{17}^e	Green	NY	ρ
s_{18}	r_{18}^e	Hack	NY	σ

vertical knowledge

salt	enc	Disease
s_{21}	r_{21}^e	Flu
s_{22}	r_{22}^e	Flu
s_{23}	r_{23}^e	Flu
s_{24}	r_{24}^e	Diabetes
s_{25}	r_{25}^e	Diabetes
s_{26}	r_{26}^e	Gastritis
s_{27}	r_{27}^e	Arthritis
s_{28}	r_{28}^e	Arthritis

horizontal

Name	Disease
Adams	Flu

1:n the same index value can be mapped to multiple index values to guarantee a flat distribution of the indexes. [the frequency is one, only one time for each value]

We cannot exploit anything in vertical knowledge any value is equally likely to the others, and same for horizontal knowledge. ➔ prevents any kind of vertical and horizontal knowledge because it blocks any kind of exposure to make any kind of inference. The Cons is that you can be exposed to the dynamic observation.

Dynamic observation: when you want to formulate a query over your data, you need to take into consideration the fact that you have built indexes in a given manner.

On the other hand, you are protected in the static sense.

We have not considered many issues: protect observation of multiple accesses to fragments; protect against the release of multiple indexes; protect against different types of observer's knowledge; development of flattened index function that generate collisions; definition of metrics for accessing exposures due to indexes….

PRIVACY AND DATA PROTECTION IN EMERGING SCENARIOS

Guaranteeing privacy of outsourced data entails protecting the confidentiality of the data (content confidentiality) as well as pf the accesses to them:

- **Access confidentiality**: confidentiality of queries and computations performed in the cloud. We need to protect the target of each specific query/access. [query extracting disease of person named Sara, I want to keep it secret I am looking for my disease]. Can say something about the person performing it. Even revealing I am making specific searches, may reveal something about me. Confidentiality of the fact that an access aims at a specific data.
- **Pattern confidentiality**: confidentiality of the fact that two accesses aim at the same data. Sequences of accesses/updating many times for the same thing. We might perform different searches/downloads over the same collection of information and we must protect that different searches are looking for the same element.
  The static observation of data: the observer simply looks at the content of the data set/ file collection while nobody is using it. Dynamic observation: related to the interaction between the serve and one or more clients. Not only look at the content of the data, but also at what is returned in response to request. Provide additional information for the observer, that we did not take into consideration before.

There are different techniques in the literature to protect the 2: Private Information Retrieval (PIR); tree-structures; oblivious RAM structures; we will take a took to Path ORAM, Ring ORAM and Shuffle index based on B+-tree structure with dynamic allocation of data.

On one hand, I want to maintain confidential, on the other hand I want also to maintain confidential which is the tuple returned as a response to query.

The problem of providing this type of protection (moving to another physical machine, other cloud provider) is that the cloud provider can observe and keep track of every kind of action I perform over my data, because has physical control over the machine and disks. Even if tuples are encrypted, the cloud provider can precisely pin point the part of information that I have downloaded, decrypted and used. Furthermore, the cloud provider can look at the physical level at the blocks of memory that I have read every time I have downloaded, so can easily understand whether I am downloading twice the same file or different files → full control over the physical storing of the data.

Idea: I need to decuple the physical address where the file is stored from the file itself. To do so, we move the physical location where the file is stored after I have downloaded it.

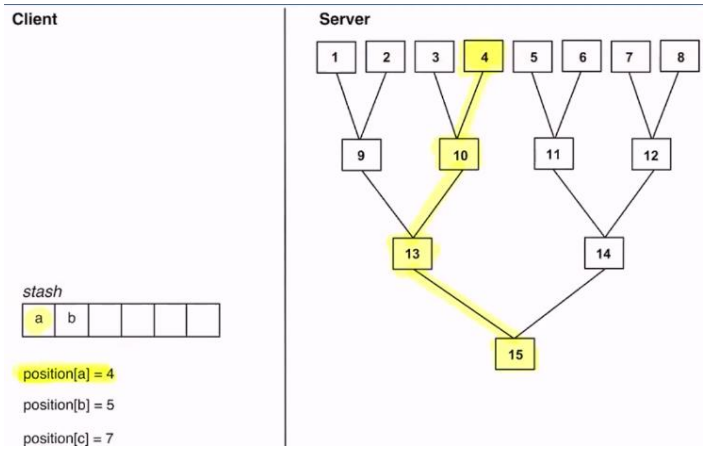
Path ORAM

Two elements that interact one with the other:

1. **Server side**= cloud service provider
   We store a tree-structure with different levels having different nodes, each refers to a page in memory, so each node contains up to Z real blocks. Dummy blocks = do not store any data. The cloud provider cannot get the difference between blocks including data and blocks not including anything.

Any leaf node *x* defines a unique path *P(x)* from *x* to the root.

2. **Client side**= machine used at the user side
   Has a stash (sort of cache), a memory storage place used for the working of the storage and a position map, that is a piece of information which tells to the client to which leaf each data is associated = File name/data identifier in association with the leaf id.
   The position map changes every time we look for something (for every search).



Stash is needed to store temporarly information downloaded form the server before we rewrite them back.

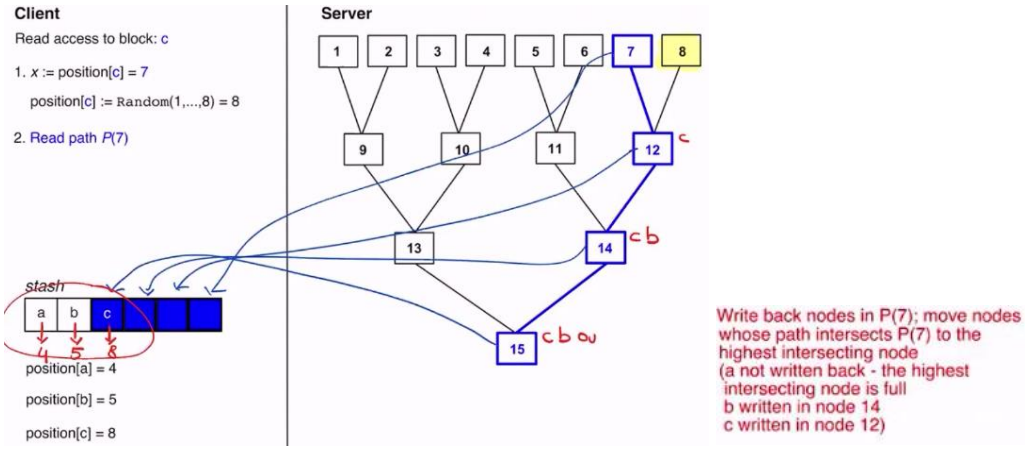
Each block is mapped to a uniformly random leaf bucket in the tree.

Unstashed blocks are always placed in some bucket along the path to the mapped leaf.

How does it work to perform searches over the Path ORAM structure:

1. Remap block: if I know that the position of *a* is *x*, then I need to read all the path from the root to *x*. Then I remap the position of *a* to a new random position (a new leaf node)
2. Read path: I read the nodes in *P(x)* containing now *a*.
3. Write path: I write the nodes in *P(x)* back possibly including some additional blocks [content] from the stash if they can be placed into the path.

Example: looking for *c*, it is in position 7, so can be stored in 12, 14 ore 15. So I change the position: position[c]=8 and I download 7, 12, 14, and 15. Then I put all these blocks into the stash with all its content. Then I change the content of boxes 7, 12,14,15.



Ring ORAM

Ring ORAM tries to reduce data downloaded.

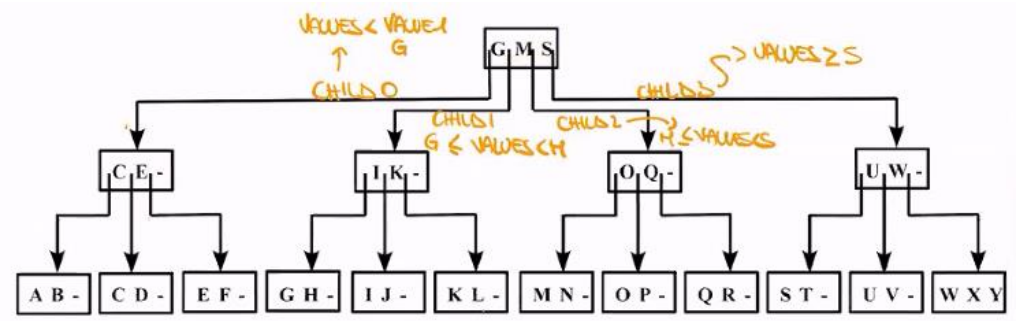
Cons: does not support range queries because data are mapped in a random manner to the leaves; accesses by multiple clients are not supported.

Shuffle Index

Combines different protection methods to reach the same goal as Path ORAM.

Particular kind of tree: B+-tree. Data are indexed over a candidate key K and organized as an unchained B+-tree with fan out F. relational databases for storing data: efficient when you want to perform searches in a limited time over ordered data.

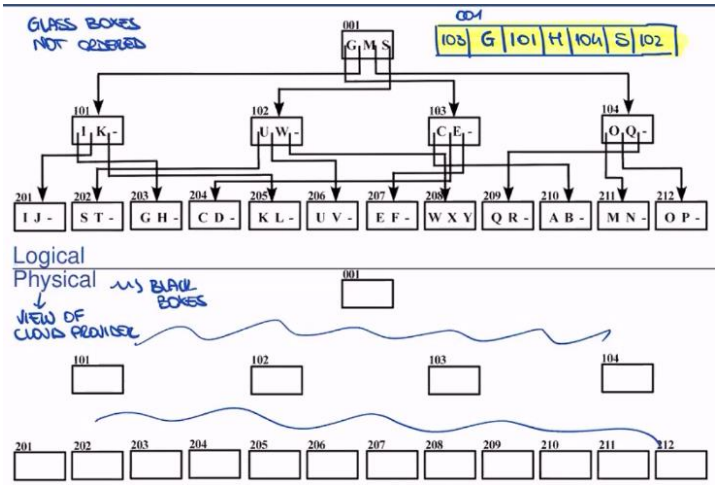
As many children as the number of arrow staring from the node (3 values, 4 children). Structure built at a data owner size.



Reach the target leaf node, making 3 access (number of nodes which need to be read), without looking at the all data. Enables searches in an easier manner. If I want to perform a range search, I can go to the neighbors step by step after I searched for 1 specific value [for example M] until I find the other values I am looking for.

The idea is to organize data in this way and then perform the searches and swapping over the original structure.

The logical representation of the Shuffle index is the number of the box that I associated each element into the structure I am storing at the cloud provider. → I consider each node in the box and I associate a label with it, logical identifier. I need to use a random order to associate identifiers to nodes, as an element of protection.



And this is the order I will use to put data in the cloud provider side, but before doing it I **encrypt** the content.

The physical representation is the view of cloud provider (*black boxes*).

We are interacting with the cloud provider as many times as the levels of the tree.

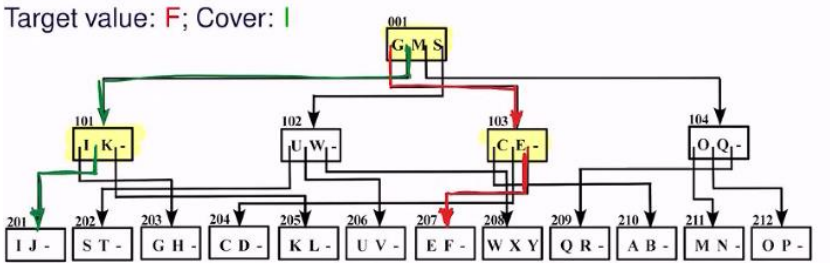
I need to download one at a time because we don't know how these arrows have been defined.

You are protecting the confidentiality of the content, BUT you are not protected against the observation: if the provider observes that you are looking always for the same leaf node, he knows that you are looking many times for the same value.

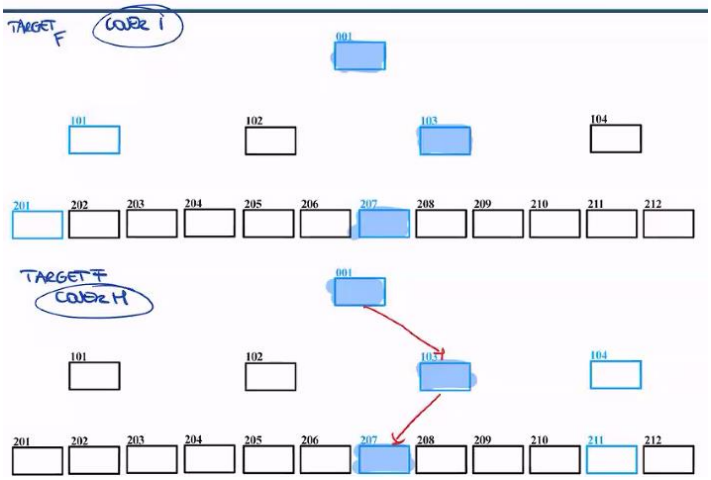
Mixing the boxing is not sufficient to protect the confidentiality of access and of pattern.

What can be do to provide pattern confidentiality? Combined adoption of cover searches, cached searches and shuffling.

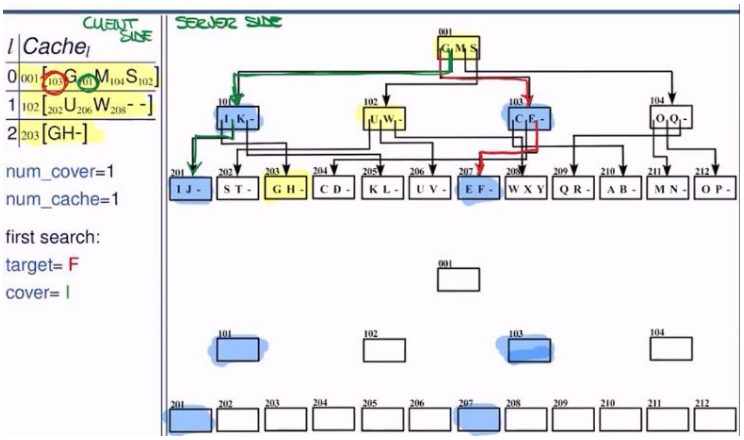
- **Cover searches**
  Provide confusion on the target of an access by hiding it within a group of other requests that act as covers. Add *num_cover* (parameter used for protection). Cover searches must provide block diversity and the search should be indistinguishable [cannot choose C and F] between the real and the fake one.



  Leaf blocks have the same probability of containing the actual target; the parent-child relationship between accessed blocks is confused. BUT the parent-child relationship can be disclosed by intersection attacks.
  Intersection attacks:



- **Cached searches**
  Client maintains a local **cache** (= small piece of memory at the client side that keeps track of the last operations that have been performed) of nodes in the path to the target for counteracting intersection attacks. The cache is stored at the client side: set of blocks downloaded during the last searches only for the target
  Whenever you are looking something that is already in the cache, instead of searching also at the server you look for an additional fake cover. Cover should not clash with the cache, otherwise it would be useless.
  Instead of perform only one cover, I perform two of them. Since I already have the path to F in the cache, I only need to continue with the paths of the 2 fake searches.

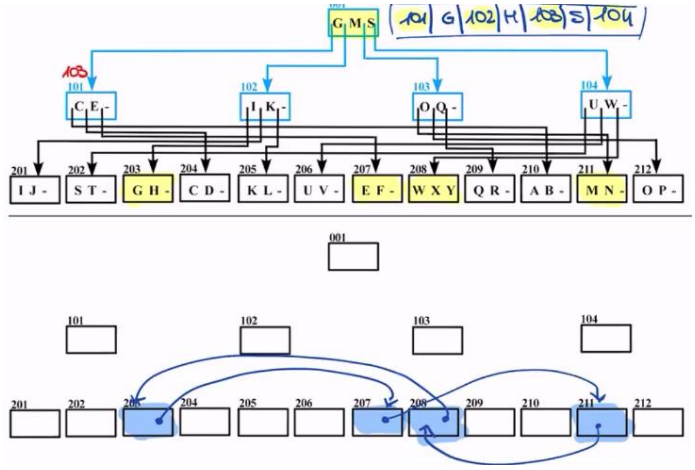


Intersection attacks are covered: the server would not be able to determine whether the two requests aim at the same target.

Problem of the cache is that does not prevent intersection attacks on long-history observation are not protected, on observations that go beyond the size of the cache.

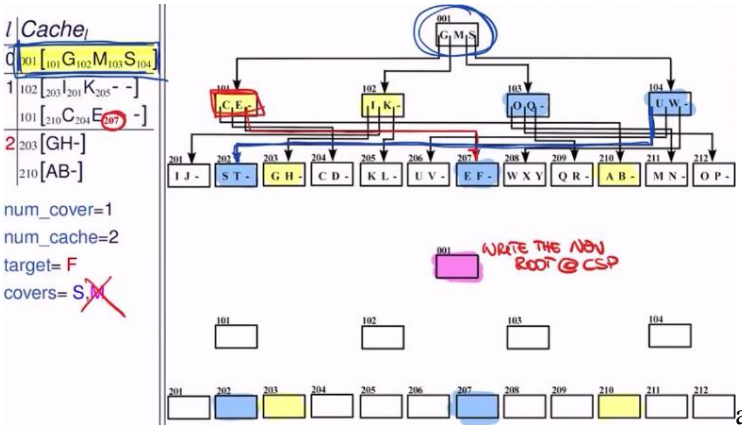
- **Shuffling**

  Operates swapping the content of different boxes among the one that have been downloaded in the operation. The swap should be not deterministic: in a random manner.

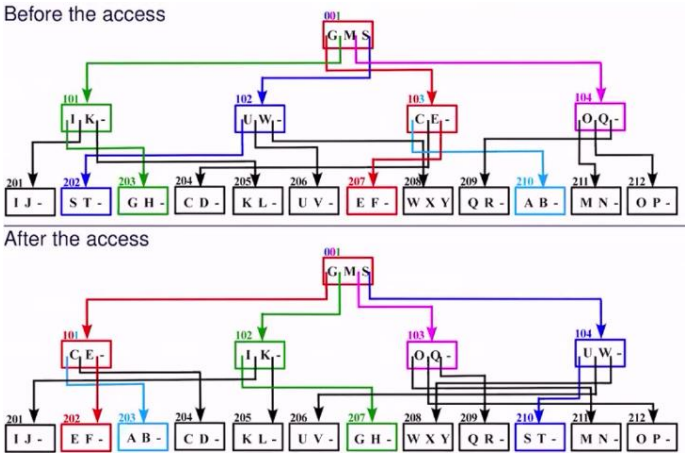


- **Access execution**

  Let *v* be the target value. Determine *num_come+1* (parameter) cover values and for each level *l* of the shuffle index: chose a further cover if is already in the cache; I visit the tree looking at each number of blocks; update the cache; perform the shuffle and rewrite data to the server side.



and then shuffle.



The longer is the history kept in the cache and the higher the number of covers, the more you are protected. But becomes costlier.

Need to find a way for parallel searching to work.

INTEGRITY IN QUERY COMPUTATION

Integrity in general means that we need to have some kind of technique if someone improperly modified data. Data owner and users need mechanisms that prove integrity for query results:

- **Correctness** = computed on genuine data, not modified
- **Completeness** = computed on the whole data collection, not a fraction
- **Freshness** = computed on the most recent version of the data, new data

We can have 2 approaches:

- **Deterministic**: uses authenticated data structured [signature chains] or encryption-based solutions [verifiable homomorphic encryption schema]. If your prof of integrity is fast, we have integrity. Build on top of database, an identity data structure.
- **Probabilistic**: exploits insertion of fake tuples in query results, replication of tuples in query results, pre-computed tokens. Not provide 100% guarantee that data is integrate when you get them from query, but more flexible: independent from the data collection used, the kind of searches performed and from the idea of adding structure on top of data. Injected additional tuples/files into the dataset and the client knows exactly which are the fake ones, then if the result on the fake data is not correct, you are sure that the cloud provider did not operate in a proper manner.

Other approaches consider the verification of the integrity of query results of complex queries: JOINS. Combine 2 datasets.

Merkle Hash Tree

The most widely used technique used to guarantee protection in computation (100% proved). But if the verification of integrity is passed, we are 100% sure that the result of the query is complete and correct. If the verification of integrity fails, we are 100% sure that the cloud provider misbehaved in computation of the result.

It is a binary tree where: each leaf contains the hash of one $h:R \rightarrow D$ not invertible; each node contains the result of the hash of the concatenation of its children.

The hash function used to build the tree is collision-resistant: try to avoid that 2 tuples product the same hash.

The root is signed by the data owner and communicated to authorized users.

Tuples in the leaves are ordered according to the value of the attribute A on which the tree is defined.

The tree is created by the data owner and stored at the server side, apart from the root that is also communicated to server users.

Merkle hash tree over attribute SSN $h(123\text{-}45\text{-}6789, \text{Alice}, \text{Asthma})$

At the second level we compute the hash function of the concatenation of two hashes at the first level.

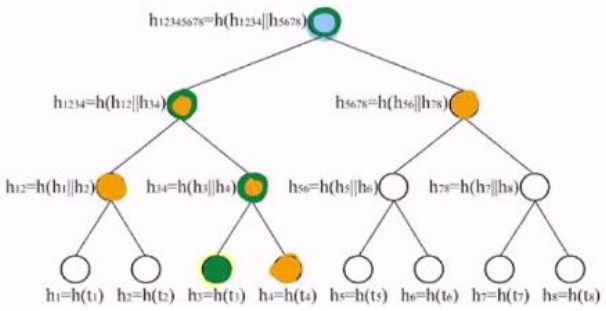
The root node has hash function that depends on all the leaves, all content of the dataset. The idea is to SIGN($h_{12345678}$, K_{owner}). Everybody can verify the signature, so that the root node was really computed by the data owner, but nobody can build it from scratch: if someone changes the table without being authorized, this root would have a different value with respect to the one publicly released by the data owner at the first releasing. So, ideally if I download the whole dataset and I compute again the Markle tree at the client side, I need to obtain exactly the same root node communicated by the data owner, otherwise data has been improperly modified.

We can perform verification only over attributes on which Merkle tree has been defined: the tuples have been ordered in the leaves of the tree according to attributes. The server returns a **verification object**= set of nodes necessary to the client for re-computing the root node.

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

Patients

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



For each node, we can compute its parent only if we have also its brother.

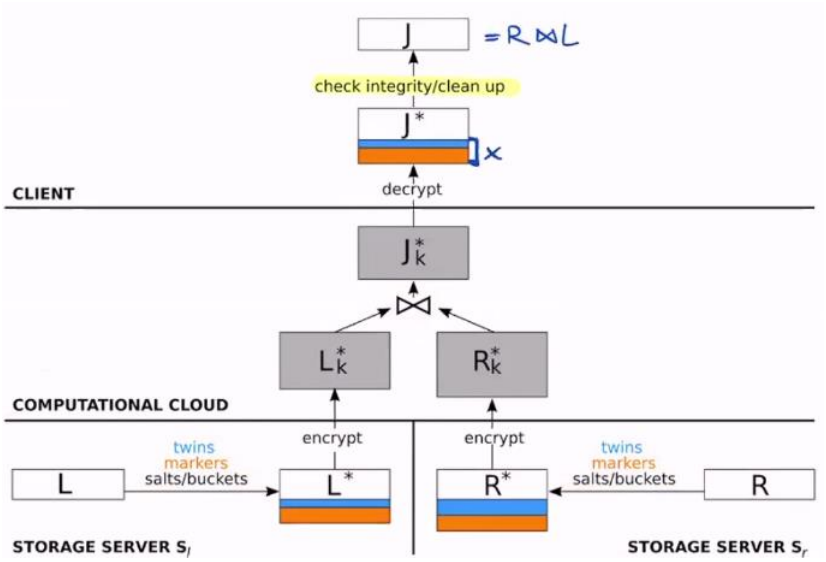
Starting from the node we extracted from the server *h3*, the client will compute the nodes along the path to the root using the verification object returned by the server, which is composed of *h4*, *h12*, *h5678*. → They are called the verification object for tuple *t3*.

Merkle hash tree can be used also for the verification in rang queries: we can be sure there is no object between *t3* and *t4* that is not returned to us.

Probabilistic approach for join queries

Let's consider the situation in which we have multiple cloud providers. Different CSP are available on the market offering a variety of service [storage, computation] at different prices. Multiple CSPs can help enhancing security but we need solutions to verify the correct behavior of these CSPs. We will discuss a probabilistic approach for join queries.

The probabilistic techniques are: encryption, markers, twins, salts/buckets (for flattening frequency distribution of values). → If we see that a marker is missing or a twin appears solo, we are sure that there is an integrity violation.



On-the-fly encryption

Encryption applied on the fly by each of the storage server of the data, so by the owner of the 2 relations or the cloud provider storing them. On-the-fly= data stored in plaintext and is encrypted before being sent to the cloud provider, can use a different key every time we user a different query to prevent from tracking. Over the join attribute we need to evaluate the equality condition, so the 2 tables have to use the **same encryption key** and the **same encryption algorithm** for performing the comparison. While for the rest of the table can use any encryption.

R_l

	I	Attr
l_1	a	Ann
l_2	b	Beth
l_3	c	Cloe

R_r

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

J

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

R_{lk}

I_k	L.Tuple$_k$
α	λ_1
β	λ_2
γ	λ_3

R_{rk}

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

J_k

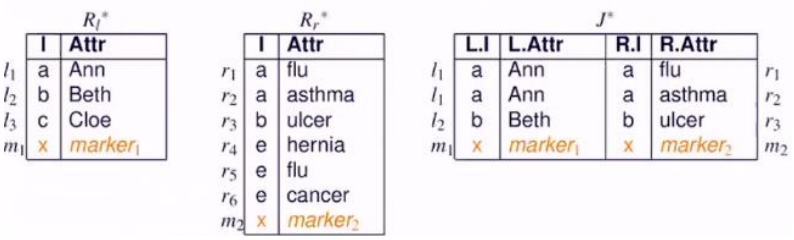
L.I_k	L.Attr$_k$	R.I_k	R.Attr$_k$
α	λ_1	α	ρ_1
α	λ_1	α	ρ_2
β	λ_2	β	ρ_3

So only in the clients knows the encryption keys can actually retreive the original join result.

Markers

= fake tuples (artificial) that cannot be recognized by the computational server. They should not generate spurious tuples. Need also to be inserted in a concerted manner to guarantee that they belong to the join result.

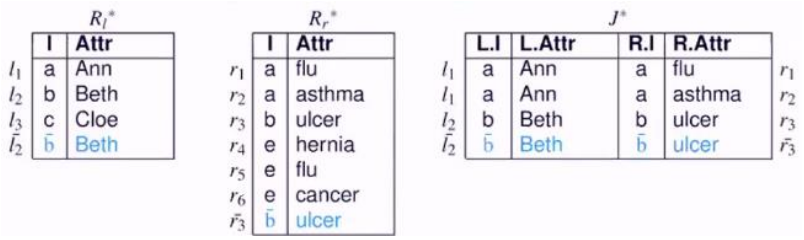
The absence of markers signals incompleteness of the join result.



	R_l^*			R_r^*			J^*				
	I	Attr		I	Attr		L.I	L.Attr	R.I	R.Attr	
l_1	a	Ann	r_1	a	flu	l_1	a	Ann	a	flu	r_1
l_2	b	Beth	r_2	a	asthma	l_1	a	Ann	a	asthma	r_2
l_3	c	Cloe	r_3	b	ulcer	l_2	b	Beth	b	ulcer	r_3
m_1	x	$marker_1$	r_4	e	hernia	m_1	x	$marker_1$	x	$marker_2$	m_2
			r_5	e	flu						
			r_6	e	cancer						
			m_2	x	$marker_2$						

Twins

= duplicate the subset of the tuples that satisfy condition C_{twin} that: is defined on the join attribute I, tunes the percentage p_I of twins, is defined by the client and communicated to S_l and S_r. a twin appearing solo signals incompleteness of the join result.



	R_l^*			R_r^*			J^*				
	I	Attr		I	Attr		L.I	L.Attr	R.I	R.Attr	
l_1	a	Ann	r_1	a	flu	l_1	a	Ann	a	flu	r_1
l_2	b	Beth	r_2	a	asthma	l_1	a	Ann	a	asthma	r_2
l_3	c	Cloe	r_3	b	ulcer	l_2	b	Beth	b	ulcer	r_3
$\bar{l}_2$	$\bar{b}$	Beth	r_4	e	hernia	$\bar{l}_2$	$\bar{b}$	Beth	$\bar{b}$	ulcer	$\bar{r}_3$
			r_5	e	flu						
			r_6	e	cancer						
			$\bar{r}_3$	$\bar{b}$	ulcer						

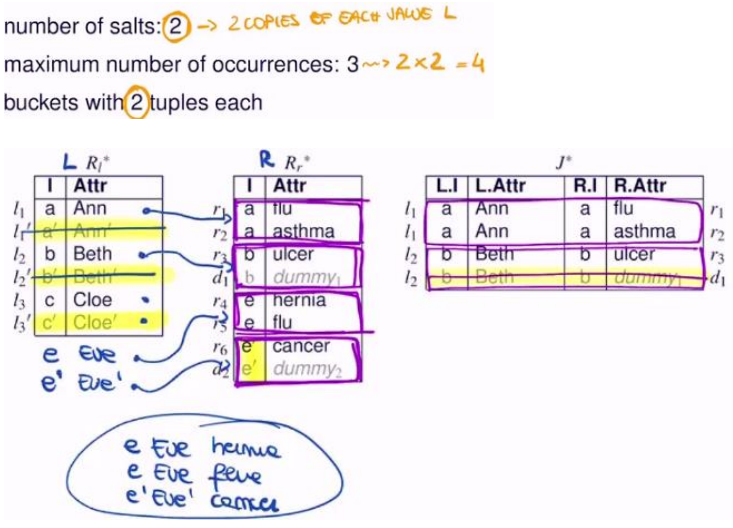
Salts and buckets

Can happen that joins represent one-to-many relationships (1:n), so the distribution of values on the side "many" can cause the cloud provider some kind of inference: can exploit this distribution to identify markers (because they have always frequency 1) and also the twins have the same frequency.

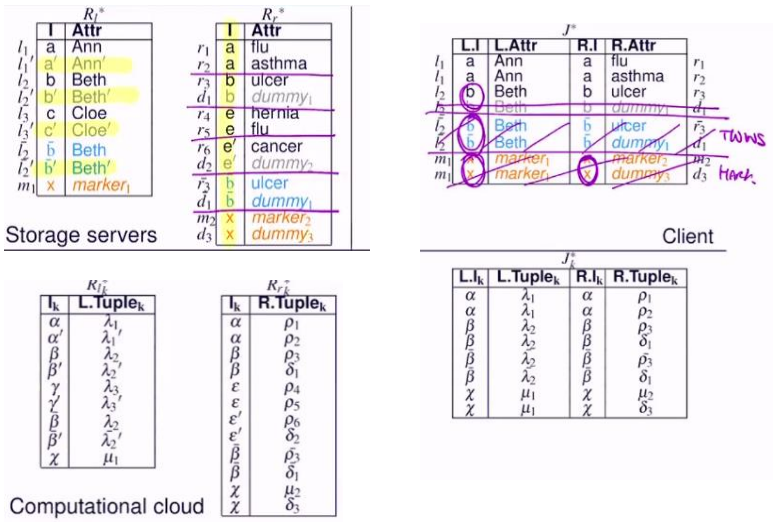
The idea is that of flattening the frequency in such way that all the values of the join attributes have frequency 1:n [1:5].

Salts: used on the Left side, so on the one having frequency 1. We would have table with 1 tuple. When we create a second bucket in R side, we create also another copy in L side [A'] duplicating the tuple to make the join work correctly.

Buckets: used on the Right side, so on the one having frequency >1. Let's consider original frequency value 4: we split the set into two buckets b1 with 3 tuples, b2 will include the remaining 1 + 2 dummy/empty tuples.



Query evaluation: the client needs to share to the 2 client servers the symmetric key k_i: need to communicate the number of markers injected m, the percentage of twins p_t, and the number of salts s.



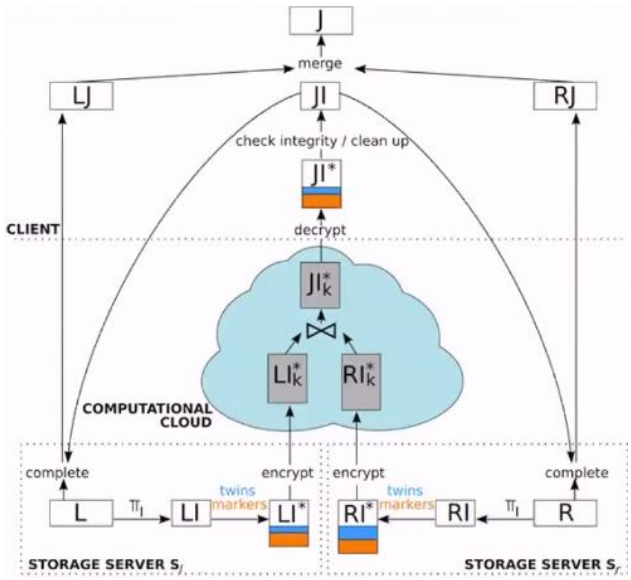
Twins are twice as effective as markers, but lose their effectiveness when the computational cloud omits a large fraction of tuples.[all tuples omitted]

Markers are a little bit less effective, but allow detecting extreme behavior [all tuples omitted] and provide effective when the computational cloud omits a large fraction of tuples.

Semi-join

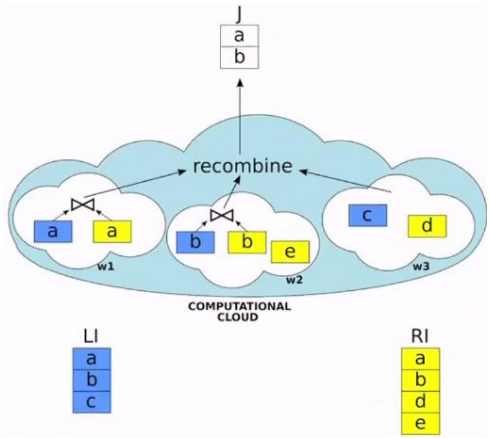
= avoids the use of salts and buckets. Instead of evaluating the join directly between the 2 relations that have to been combined, we first extract all the attributes removing the duplicates and we find out which are the values which are present in both the relations. Then we ask the 2 cloud providers to give us the tuples corresponding to those values only.

Instead of computing the join over the whole relation, we compute it only on the column of interest and recombine/add the remaining attributes when they are needed.



MapReduce

Parallel and distributed scenario: many nodes/machines working in parallel for performing the computation.



Markers should be properly distributed among workers: should guarantee that each worker has at least 1 marker. Can be distributed randomly, deciding a minimum number or balance them.

Twin separation: when we create a copy of a tuple, we give the original version to one worker and the copy to another one. So If they produce the same result, they are working both in a proper manner, otherwise one of the two is not.