Database Security

Security requirements of database systems

- *Physical* database integrity. The data of a database are immune to physical problems, such as power failures, and someone can reconstruct the database if it is destroyed through a catastrophe.
- *Logical database integrity*. The structure of the database is preserved. With logical integrity of a database, a modification to the value of one field does not affect other fields, for example.
- *Element integrity*. The data contained in each element are accurate.
- **Auditability.** It is possible to track who or what has accessed (or modified) the elements in the database.
- *Access control*. A user is allowed to access only authorized data, and different users can be restricted to different modes of access (such as read or write).
- *User authentication*. Every user is positively identified, both for the audit trail and for permission to access certain data.
- *Availability*. Users can access the database in general and all the data for which they are authorized.

Security vs Precision

- To *increase availability* of data, *disclose as much data as possible*
- *Sensitive data* in the database should be *protected* from disclosure as they are confidential
- **Precision:** aims to *protect all sensitive* data while *revealing* as much *non-sensitive* data as possible

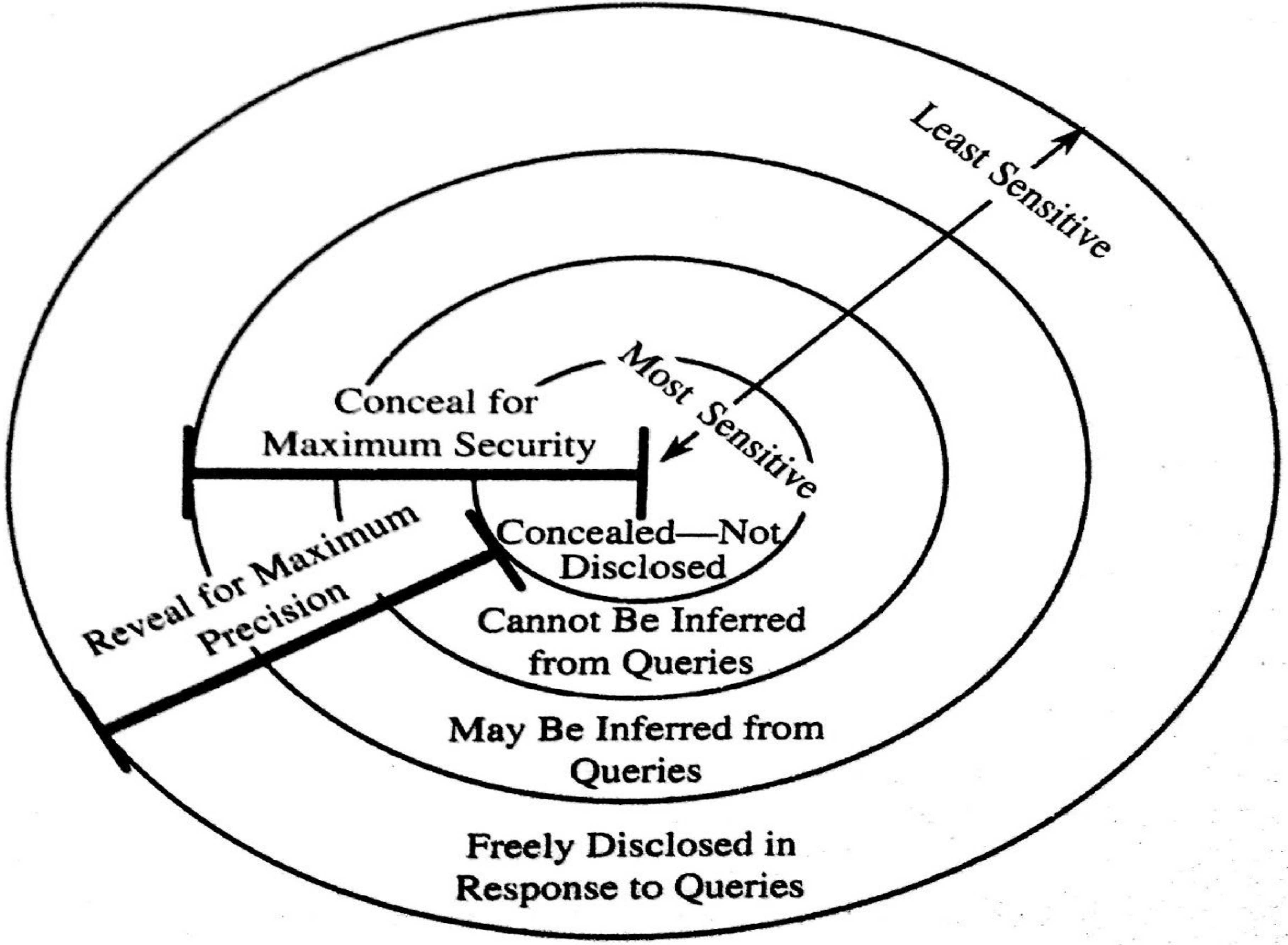
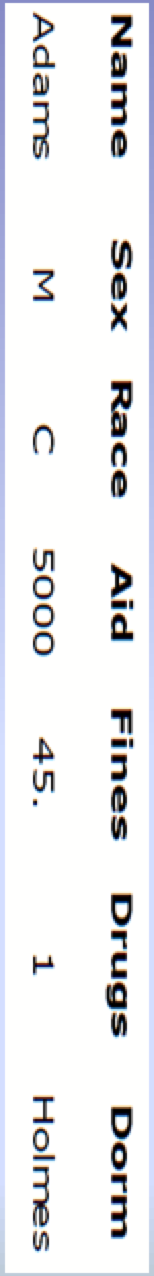

Least Sensitive
Most Sensitive
Conceal for Maximum Security
Concealed—Not Disclosed
Reveal for Maximum Precision
Cannot Be Inferred from Queries
May Be Inferred from Queries
Freely Disclosed in Response to Queries

FIGURE 6-3 Security versus Precision.

Remark

- Rule: “n items over k percent” means that data should be withheld if n items represent over k percent of the result reported.
  - i.e., do not reveal results when a small number of people make up a large proportion of a category
- So DBMS may conceal data when a small number of entries make up a large proportion of data revealed

- Consider the following university students' database containing fields:
  - name, sex, race, financial aid, parking fines, drug use, dormitory.
  - Non-sensitive/least sensitive fields are: name, dormitory
  - Most sensitive fields: financial aid, parking fines, drug use
  - Medium sensitive fields: sex, race
- Few people will have access to each of most sensitive fields, but *no one will have access to all "most sensitive fields*"



Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes

Table 6-7. Sample Database

Drugs: 0 means never used
3 means frequent user

Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes
Bailey	M	B	0	0.	0	Grey
Chin	F	A	3000	20.	0	West
Dewitt	M	B	1000	35.	3	Grey
Earhart	F	C	2000	95.	1	Holmes
Fein	F	C	1000	15.	0	West
Groff	M	C	4000	0.	3	West
Hill	F	B	5000	10.	2	Holmes
Koch	F	C	0	0.	1	West
Liu	F	A	0	10.	2	Grey
Majors	M	C	2000	0.	2	Grey

Inference Problem

- Inference problem is a way to infer or **derive sensitive data from non-sensitive** data
- Inference problem can be solved by
  - Direct attack
  - Indirect attacks
  - Tracker attacks

Direct Attack

- Determine values of sensitive fields by directly writing specific queries to get the data from the database.
- Eg., to know male students who use drug rarely, we can write direct query:
  - List NAME where SEX=M ^ DRUGS=1
- DBMS will not give the result as per rule “n items over k percent”, since query result has 1 record and 1 percent selected has 100 percent of the data reported.
- This is obvious attack

- The same result can be produced by following less obvious query:
  - List NAME where

    (SEX=M ^ DRUGS=1) v

    (SEX≠M ^ SEX≠F) v

    (DORM=AYERS)

    This query does not appear to be specific but achieves same result.

Indirect attack

- To *protect sensitive* data, an organization will *publish statistics* such as count, sum and mean by *suppressing individual names*, addresses, or other fields by which single person/individuals can not be identified.
- Indirect attack is a statistical attack to *infer individual data/sensitive data* from one or more *intermediate statistical results*.

sum

- From the following report, we can infer that no female living in Grey receives financial aid. This approach is used to determine negative results

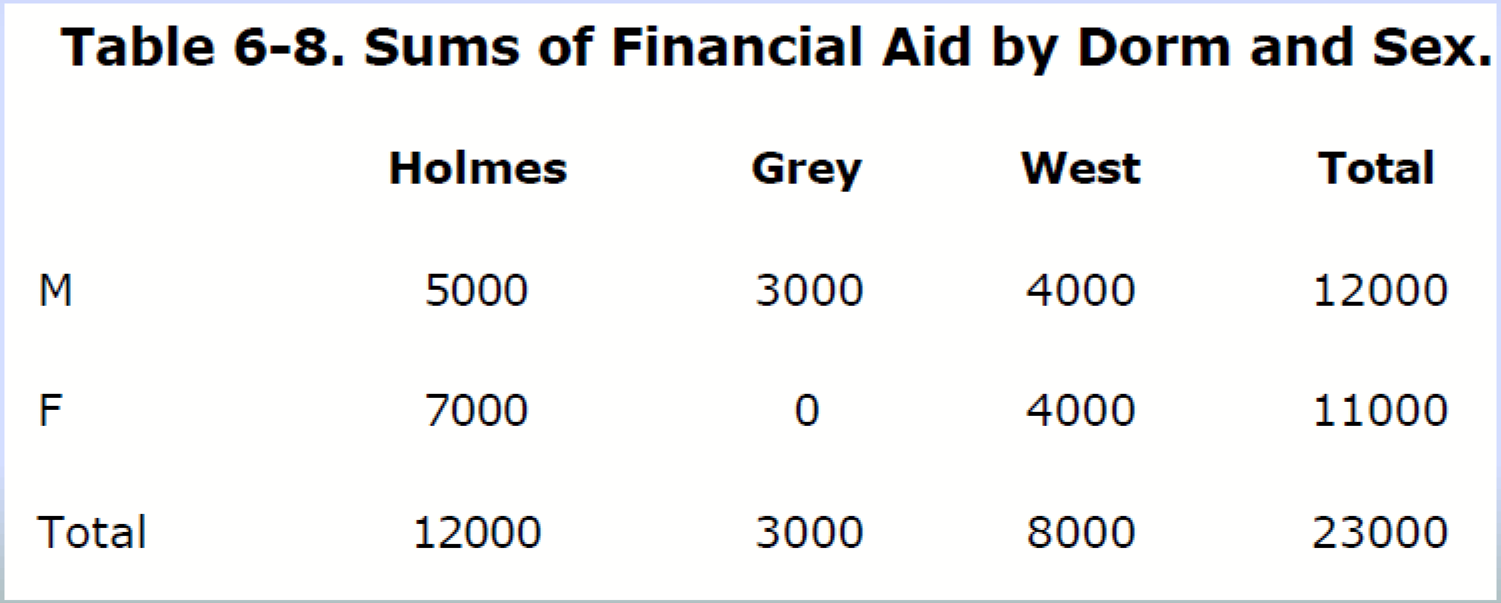


Table 6-8. Sums of Financial Aid by Dorm and Sex.

	Holmes	Grey	West	Total
M	5000	3000	4000	12000
F	7000	0	4000	11000
Total	12000	3000	8000	23000

count

- By using preceding sum table and following count table, we can infer that
  - 1 male from Holmes receives aid of 5000
  - 1 male from West receives aid of 4000
  - 1 female from Grey receives no aid
  - We can obtain Names by selecting subschema NAME, DORM that are not sensitive data :
    - Eg List NAME where SEX=M ^ DORM=HOLMES

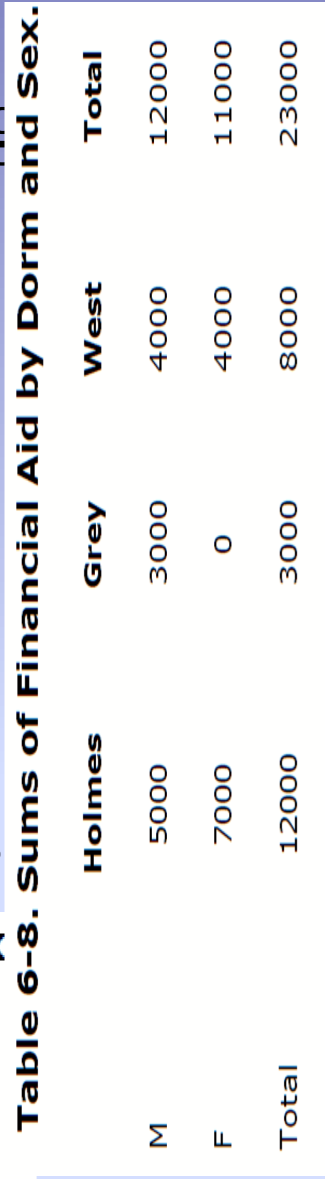


Table 6-8. Sums of Financial Aid by Dorm and Sex.

	Holmes	Grey	West	Total
M	5000	3000	4000	12000
F	7000	0	4000	11000
Total	12000	3000	8000	23000

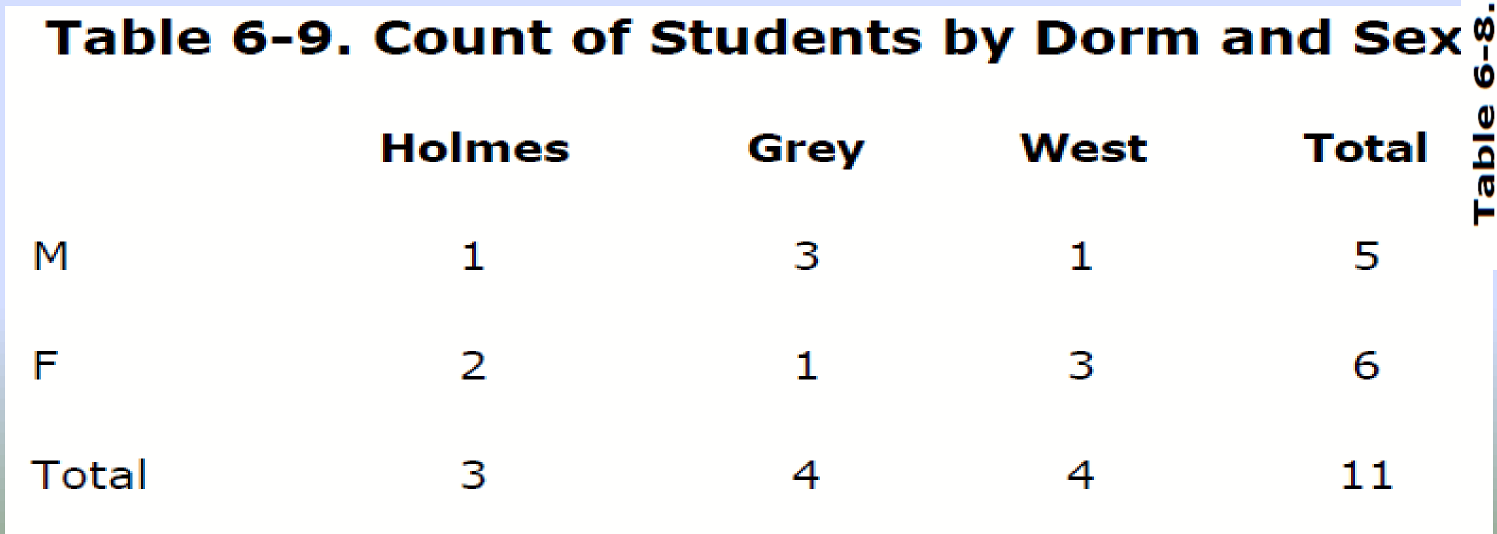


Table 6-9. Count of Students by Dorm and Sex

	Holmes	Grey	West	Total
M	1	3	1	5
F	2	1	3	6
Total	3	4	4	11

Tracker Attacks

- If there are n values in the database, then Tracker attack will identify unique value by querying n-1 other values.
  - Given n and n-1, we can easily identify 1 element by eliminating n-1 from n.
- Tracker attack uses additional queries in such a way that two sets of records cancel each other leaving only desired data.

Example

- How many female Caucasians live in Holmes?
- The sensitive query:
  - Count ((SEX=F) ^ (RACE=C) ^ (DORM=Holmes))
  - Will fetch one record but it will not be displayed as per "n items over k percent rule"
- Note:
  - $n(a \wedge b \wedge c) = n(a) - n(a \wedge \neg (b \wedge c))$ (Venn diagram)
  - $n(\neg (b \wedge c)) = n(\neg b \vee \neg c)$

- The same result can be obtained by following non-sensitive equivalent query:
- Let a be “SEX=F”, b be “Race=C” and

  c be “DORM=Holmes”

Then count (a^b^c)=count(a)-count(a^(not b v not c))

= 6 – 5

= 1 (sensitive data)

Multilevel database security

Database security

- Database security is provided by following mechanisms:
  - Partitioning
  - Encryption
  - Integrity lock
  - Sensitivity lock

Partitioning

- Database is divided into separate databases based on the sensitivity levels of data
  - Eg., Most sensitive data in level 1, so in DB 1

    ………

    Least sensitive data in level n, so in DB n
  - Define/restrict access to these databases to various users based on their privileges.
  - So partitioning provides security based on access control

Encryption

- Store the sensitive data in an encrypted form.
    - Identify the sensitive data
    - classify them into various levels based on sensitivity.
    - Create table for each level of data
    - Use separate key for each sensitive level to encrypt the data in the table.

Integrity lock

- Lock is created to provide *integrity* and *limited access* for a database
- Integrity lock has 3 parts: actual data, sensitivity label, and checksum
  - Each data can be figuratively *painted with a colour* that denotes its *sensitivity*
  - Calculate *checksum* on both *data and its sensitivity* label

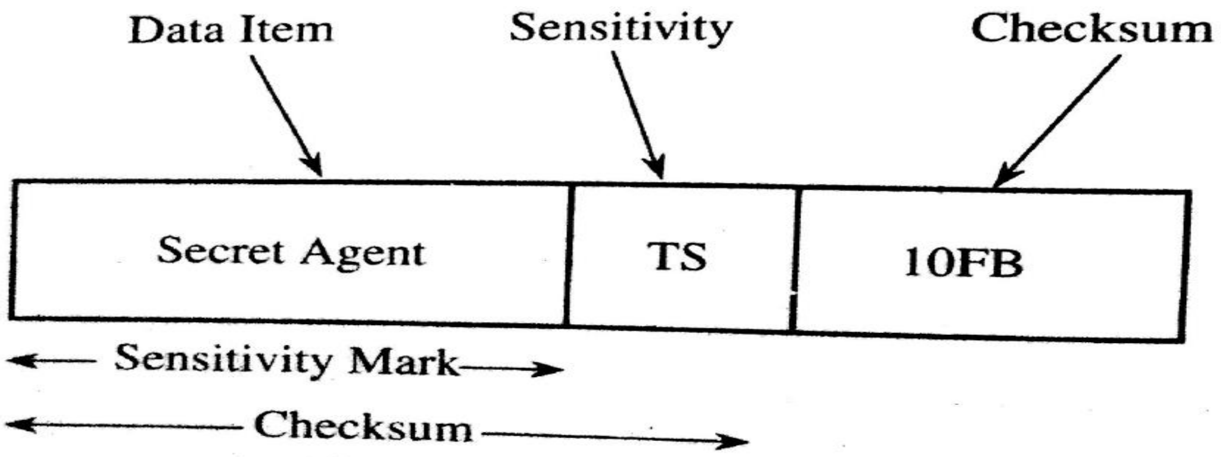
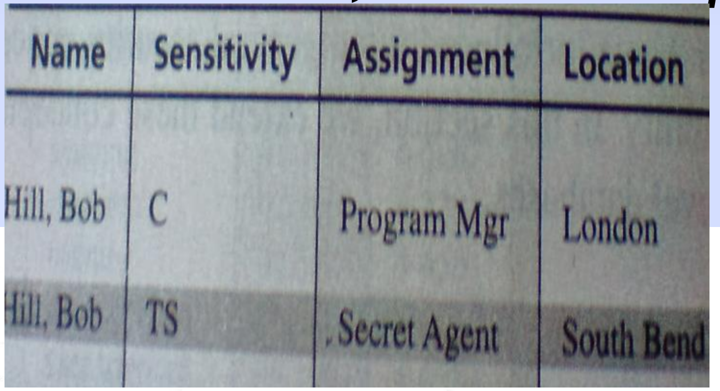


FIGURE 6-7 *Integrity Lock.*

- Sensitivity label must be unique, concealed &unforgeable
- Checksum is an error-detecting part generated by cryptographic algorithm(eg., DAA), hence it is called as cryptographic checksum.
- To get *unique checksum* value for each element we can calculate the checksum on 4 parts: *record number, attribute field name, data, sensitivity label*.



Name	Sensitivity	Assignment	Location
Hill, Bob	C	Program Mgr	London
Hill, Bob	TS	Secret Agent	South Bend

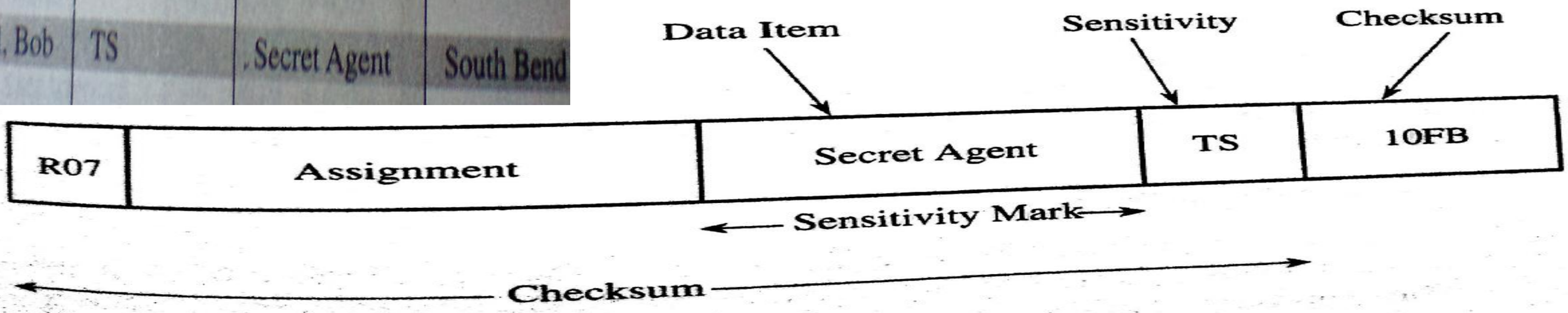


FIGURE 6-8 Cryptographic Checksum.

Sensitivity lock

- It protects sensitivity level of an element from un-authorised changes by encrypting record no and sensitivity label of the element as shown below:

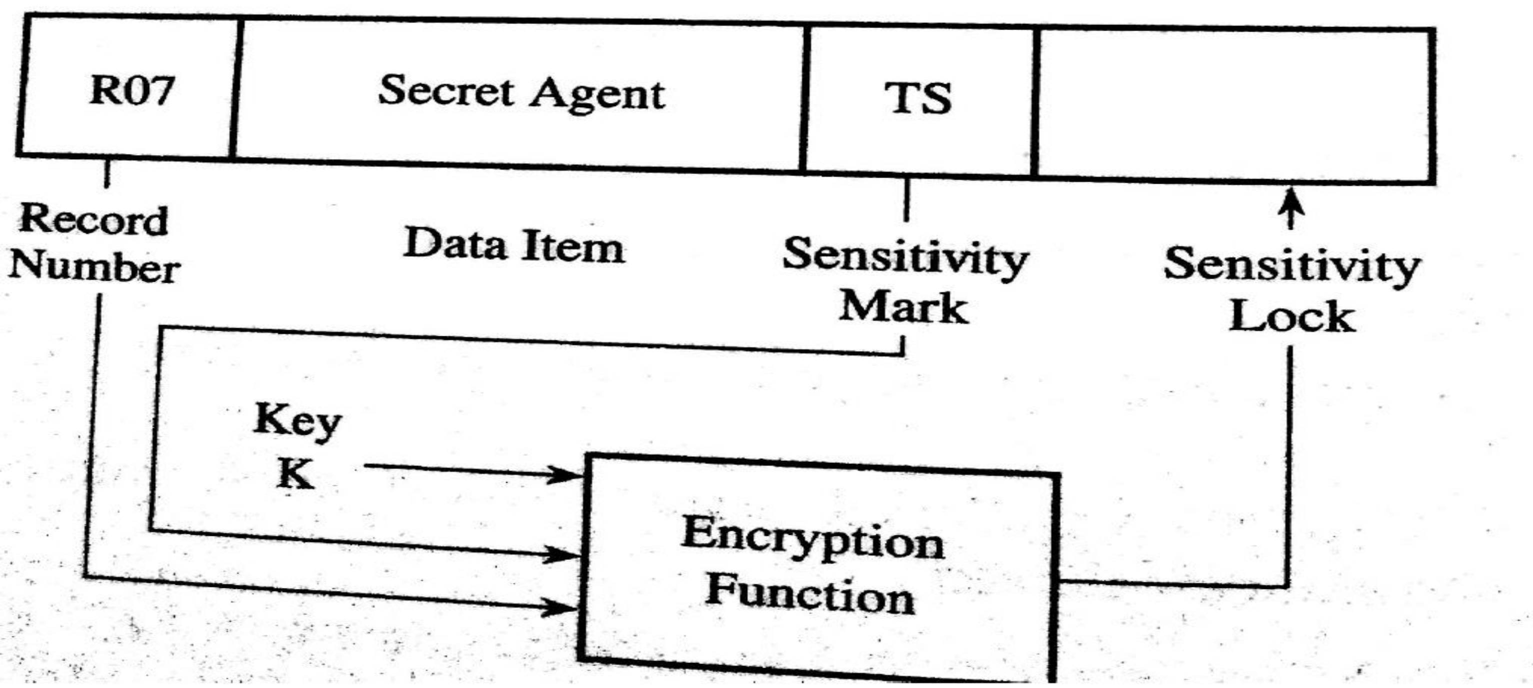


FIGURE 6-9 Sen
Lock

- Design of multilevel secure databases – approaches used are:
  - Integrity lock
  - Trusted front end
  - Commutative filters
  - Distributed databases
  - Windows / views

Integrity lock

- Uses any (untrusted) database manager with trusted procedure that handles access control as shown in following fig.
- Sensitive data with sensitivity label are stored in encrypted form.

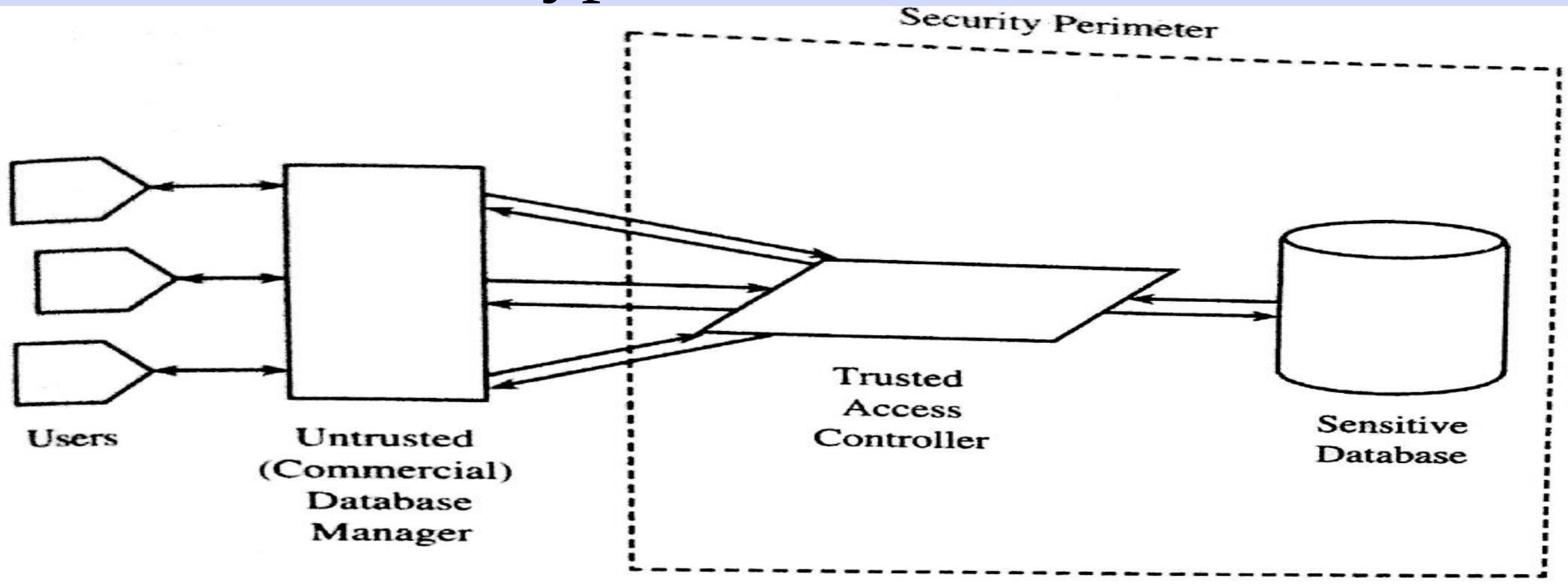

Security Perimeter
Users
Untrusted (Commercial) Database Manager
Trusted Access Controller
Sensitive Database

FIGURE 6-10 Trusted Database Manager.

Trusted front end

- It uses trusted front that acts as a guard between users and DBMS. Steps involved are:

1. A user identifies himself or herself to the front end; the front end authenticates the user's identity.
2. The user issues a query to the front end.
3. The front end verifies the user's authorization to data.
4. The front end issues a query to the database manager.
5. The database manager performs I/O access, interacting with low-level access control to achieve access to actual data.
6. The database manager returns the result of the query to the trusted front end.
7. The front end analyzes the sensitivity levels of the data items in the result and selects those items consistent with the user's security level.
8. The front end transmits selected data to the untrusted front end for formatting.
9. The untrusted front end transmits formatted data to the user.

The trusted front end serves as a one-way filter, screening out results the user should not be able to access. But the scheme is inefficient because potentially much data is retrieved and then discarded as inappropriate for the user.

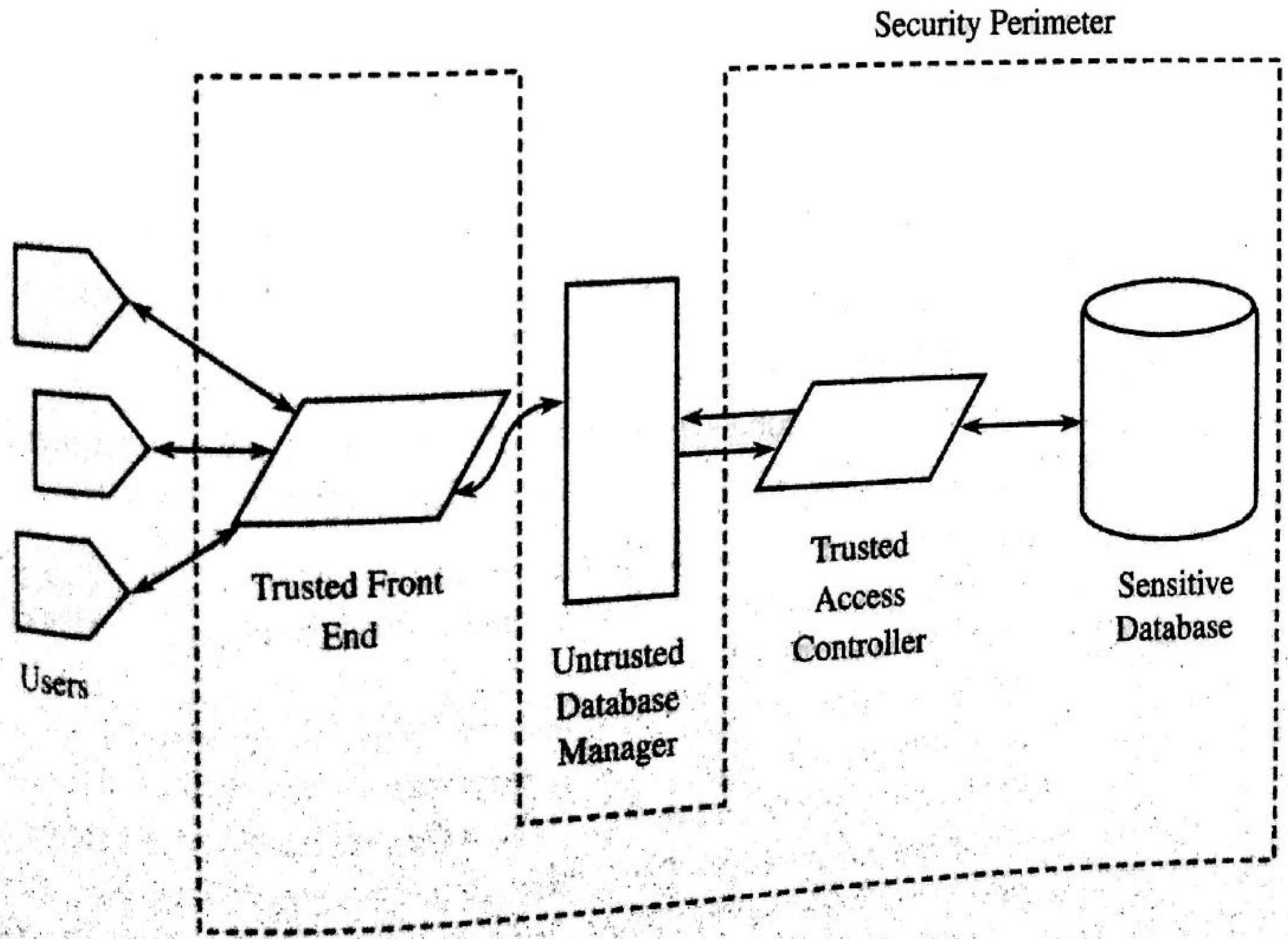


FIGURE 6-11 Trusted Front End.

Commutative filters

- Filter is an *interface* between user and DBMS.
- It *screens* user's query and *reformats* it, so that only data with permitted sensitivity level are returned to user.
- Query reformatting helps the database manager to fetch only the permitted data which is again screened by filter and then passed to user

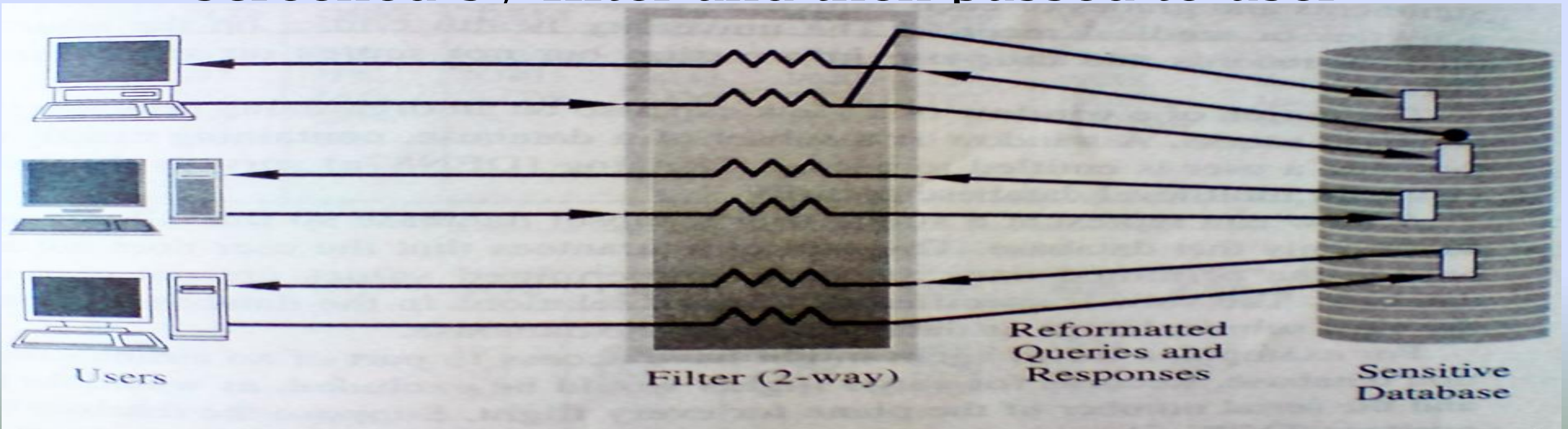


FIGURE 6-12 Commutative Filters.

Example.

- Suppose a group of physicists in Washington works on sensitive project and current user is not allowed to access the names of physicists in the db.
- If the user's query is
  - List NAME where ((OCCUP=PHYSICIST) ^ (CITY=WASHDC))
- Then filter reformats the query: (So that sensitive data is not extracted)
  - List NAME where ((OCCUP=PHYSICIST) ^ (CITY=WASHDC)) from all records R where

(NAME-SECRECY-LEVEL(R) <= USER-SECRECY-LEVEL)^

(OCCUP-SECRECY-LEVEL(R) <= USER-SECRECY-LEVEL)^

(CITY-SECRECY-LEVEL(R) <= USER-SECRECY-LEVEL)