TDDE49
Databases and Information Security for Bioinformatics

Topic: Database Security

Olaf Hartig

olaf.hartig@liu.se

Acknowledgement:
Several of the slides in this slide set are adaptations from slides made available for the database textbook by Elmasri and Navathe.

LINKÖPING UNIVERSITY

Threads and Control Measures

What are the threads?

- Loss of confidentiality: unauthorized disclosure of data
  - e.g., student learns other students' grades
- Loss of integrity: improper modification of data
  - e.g., students changing their grades
- Loss of availability: unavailability of database objects to authorized programs and people
  - e.g., students are denied seeing their own grades
  - “denial of service attack”

Control Measures to Provide DB Security

- Access control
  - Restricting the access to the database (or parts thereof)
  - Requires authentication (e.g., through login and password)
  - Usually with auditing (i.e., logging DB operations by each user)
- Inference control
  - Preventing deductions about database content
  - Summary data without possibility to determine individuals' data
- Flow control
  - Preventing information from reaching unauthorized users
- Data encryption
  - Protecting sensitive data (e.g., when transmitted over network)
  - Making information unintelligible unless authorized
  - Making changes traceable to source

Access Control

Access Control in a Database System

- Security policy specifies who is authorized to do what in the system
- DBMS provides access control mechanisms to help implement a security policy
- Two complementary types of such mechanisms:
  - Discretionary access control
  - Mandatory access control

Access Control

Discretionary Access Control

Idea and Related Concepts

- Idea: achieve access control based on
  1. privileges (specific rights for tables, columns, etc.), and
  2. a mechanism for granting and revoking such privileges
- Authorization administration policy specifies how granting and revoking is organized
  - i.e., who may grant / revoke
  - *Centralized administration*: only some privileged users
  - *Ownership-based administration*: creator of the object
- Administration delegation: if authorized to do so, a user may assign others the right to grant / revoke

Discretionary Access Control in SQL

- Simple examples:
  - to allow user Alice to query the table called Student
    GRANT SELECT **ON** Student **TO** Alice
  - to allow Alice to delete from the Student table
    GRANT DELETE **ON** Student **TO** Alice
  - revoke the previous privilege
    REVOKE DELETE **ON** Student **FROM** Alice
  - to allow Alice to modify any value in Employee
    GRANT UPDATE **ON** Employee **TO** Alice
  - to allow Bob to modify Salary values in Employee
    GRANT UPDATE **ON** Employee(Salary) **TO** Bob

Discretionary Access Control in SQL (cont'd)

GRANT *privileges* **ON** *objects* **TO** *users*

REVOKE *privileges* **ON** *objects* **FROM** *users*

- Possible privileges:
  - SELECT
  - INSERT (may be restricted to specific attributes)
  - UPDATE (may be restricted to specific attributes)
  - DELETE
  - REFERENCES (may be restricted to specific attributes)
- Possible objects:
  - Tables
  - Views
  - Specific attributes (for INSERT, UPDATE, REFERENCES)

Revisiting the Related Concepts

- Idea: achieve access control based on
  1. privileges (specific rights for tables, columns, etc.), and
  2. a mechanism for granting and revoking such privileges
- Authorization administration policy specifies how granting and revoking is organized
  - i.e., who may grant / revoke
  - *Centralized administration*: only some privileged users
  - *Ownership-based administration*: creator of the object
- Administration delegation: if authorized to do so, a user may assign others the right to grant / revoke

Discretionary Access Control in SQL (cont'd)

GRANT *privileges* **ON** *objects* **TO** *users* **[WITH GRANT OPTION]**

REVOKE [GRANT OPTION FOR] *privileges* **ON** *objects* **FROM** *users*

- WITH GRANT OPTION allows users to pass on a privilege (with or without passing on the grant option)
  - When a privilege is revoked from user *X*, it is also revoked from all users who were granted this privilege *solely* from *X*

Example

- Assume we do

GRANT UPDATE **ON** Emp **TO** Alice

GRANT UPDATE **ON** Emp **TO** Bob **WITH GRANT OPTION**

- Next, Bob does

GRANT UPDATE **ON** Emp **TO** Alice, Eve

- Now, Bob, Alice, and Eve have the privilege
- Assume we now do

REVOKE UPDATE **ON** Emp **FROM** Alice

- Alice still has the privilege (thanks to Bob)
- Let's do

REVOKE UPDATE **ON** Emp **FROM** Bob

- Now, neither of them has the privilege anymore

What are Views?

- A **virtual** table **derived** from other (possibly virtual) tables
- Defined by means of a query; view is the up-to-date result of the query (up to date regarding current data in base tables)

```
CREATE VIEW dept_view AS
  SELECT DeptNo, COUNT(*) AS C, AVG(Salary) AS S
   FROM    EMPLOYEE
   GROUP BY DeptNo;
```

- Example of usage in queries:

```
SELECT DeptNo  FROM dept_view  WHERE S > 25000 ;
```

- Why?
  - Simplify query commands; enhance programming productivity
  - Means to implement data security policies with access control

Example: Views in Access Control

```
CREATE VIEW research_colleagues_view AS
  SELECT Fname, Lname, Email
   FROM    EMPLOYEE
   WHERE Dept = 'Research';
```

```
GRANT SELECT ON research_colleagues_view TO Bob;
```

```
SELECT Fname FROM research_colleagues_view;
```

~~SELECT Fname, Salary FROM EMPLOYEE;~~

Granularity of Privileges in SQL

- Seen so far, object-level privileges
  - Objects: tables, views, attributes
  - SQL does not support privileges per tuple
  - When creating an object, the creator has all (object-level) privileges on that object

- System-level privileges
  - CREATE / ALTER / DROP tables or views
  - Not supported by standard SQL but by DBMS-specific extensions of SQL

Trojan Horse Attack

- Assume **discretionary access control**
- Suppose user *Bob* has privileges to read a secret table *T*
- User *Mallory* wants to see the data in *T* (but does not have the privileges to do so)
- *Mallory* creates a table *T'* and gives INSERT privileges to *Bob*
- *Mallory* tricks *Bob* into copying data from *T* to *T'* (e.g., by extending the “functionality” of a program used by *Bob*)
- *Mallory* can then see the data that comes from *T*

Access Control

Mandatory Access Control

Idea

- Achieve access control based on system-wide policies that cannot be changed by individual users
- Basis: partially ordered set of security classes
  - e.g., TopSecrect > Secret > Confidential > Unclassified
- Each database object (e.g., tables, columns, rows) is assigned such a class
- Each subject (users, programs) is assigned a clearance for such a class
- Subject's clearance must match class of object
  - e.g., Bell-LaPadula model

Bell-LaPadula Model

- Rule 1 (no read-up): subject *S* can read object *O* only if clearance(*S*) ≥ class(*O*)
  - e.g., reading *secret* data requires at least *secret* clearance
  - Goal: protect classified data

- Rule 2 (no write-down): subject *S* can write object *O* only if clearance(*S*) ≤ class(*O*)
  - e.g., person with *confidential* clearance cannot write *unclassified* object
  - Goal: flow control (information never flows from a higher to a lower class)

Trojan Horse Attack revisited

- Let's try to use **mandatory access control** instead
- Suppose user *Bob* has privileges to read a secret table *T*

  clearance(*Bob*) = *secret* class(*T*) = *secret*
- User *Mallory* wants to see the data in *T*
  (but does not have the privileges to do so)

  clearance(*Mallory*) < *secret*
- *Mallory* creates a table *T'* and gives INSERT privileges to *Bob*

  class(*T'*) := clearance(*Mallory*), i.e., class(*T'*) < *secret*
- *Mallory* tricks *Bob* into copying data from *T* to *T'* (e.g., by extending the "functionality" of a program used by *Bob*)

  → Writing to T' fails because clearance(*Bob*) $\not\leq$ class(*T'*)
- ~~*Mallory* can then see the data that comes from *T*~~

Multilevel Relations

- Incorporate multilevel security into the relational data model
- Each attribute (column) of a multilevel relation is associated with a corresponding *classification attribute* to denote the security class of each attribute value
- Additionally, a *tuple classification attribute* is added
  - Value of this attribute in a tuple is the highest of the classification attribute values in that tuple
- Example of a multilevel relation:

EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

Example from "Fundamentals of Database Systems" by Elmasri and Navathe, Addison Wesley.

Multilevel Relations (cont'd)

- Appearance of such a relation depends on clearance of user
- Example:
  - For a user with *Confidential* clearance:

EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	NULL C	C
Brown C	NULL C	Good C	C

EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

Example from "Fundamentals of Database Systems" by Elmasri and Navathe, Addison Wesley.

Database Encryption

Limitations of Access Control

… as a means to achieve the objectives of DB security
(in particular, confidentiality and integrity)

- Authorizations enforced by DBMS may be bypassed
  - Intruder can try to mine the database footprint on disk
  - DB administrator has enough privileges to tamper the access control definitions and gain access
- Management of databases outsourced
  - "Database as a service" / cloud services
  - No other choice than trusting the service provider

Database Encryption!

- Complements and reinforces access control by resorting to cryptographic techniques
- Ensures confidentiality of DBs by keeping data hidden from unauthorized persons

Relevant Factors for Database Encryption

- Where should the encryption be performed?

  …in the storage layer? …in the DBMS?

  …in the application that produces the data?

- How much data should be encrypted and exactly which?
- What encryption algorithm and mode of operation?
- Who should have access to the encryption keys?
- How to minimize the impact on performance?

Data Structures for Databases

A *very* brief overview before we continue ...

Database Files

- File is a sequence of records
  - Record is a set of fields that contain values
  - For instance, file = relation / table
    record = tuple / row
    field = attribute value / cell

Data File

ID#	SSN	Dept.	Salary
1	4945864	12	2000
2	7000111	13	4000
3	…		
4	…		

Database Files

- File is a sequence of records
  - Record is a set of fields that contain values
  - For instance, file = relation / table
    record = tuple / row
    field = attribute value / cell

- Files may consist of multiple blocks
  - Block is the unit of data transfer between disk and main memory
  - Each record is allocated to a block

- DBMS maintains not only data files
  - index files (to speed up the search over data files)
  - log files (to be able to recover from failures/crashes)

Data File

ID#	SSN	Dept.	Salary	
1	4945864	12	2000	Block 1
2	7000111	13	4000	
3	…			
4	…			
5	6487539			Block 2
6	7299990			
7	3452626			
8	9000013			
9	8232333			Block 3
10	…			
11	5012128			
12	…			

Encryption Granularity

How much data should be encrypted and exactly which?

Encryption Granularity

- Common levels of encryption granularity:
  - field
  - record
  - file
  - whole database
- Finer granularity has advantages:
  - allows for encryption of only the sensitive data
  - only relevant data need to be decrypted for query execution
  - different encryption keys may be used for different parts
- However, finer granularity is not always possible (see later)
- Note: sensitive data may not only be in the data file,
  but also in temporary files, log files, indexes, etc.

Encryption Layer

Where should the encryption be performed?

Storage-Level Encryption

- Use the storage subsystem to encrypt database files
  - i.e., file pages are encrypted/decrypted by the operating system when written/read from disk
- Advantages:
  - Transparent from the DB perspective, i.e., no changes to the DBMS or the applications necessary
- Disadvantages:
  - Limited to file granularity
  - Cannot be related with user privileges or data sensitivity (because storage subsystem has no knowledge of DB objects or structure)

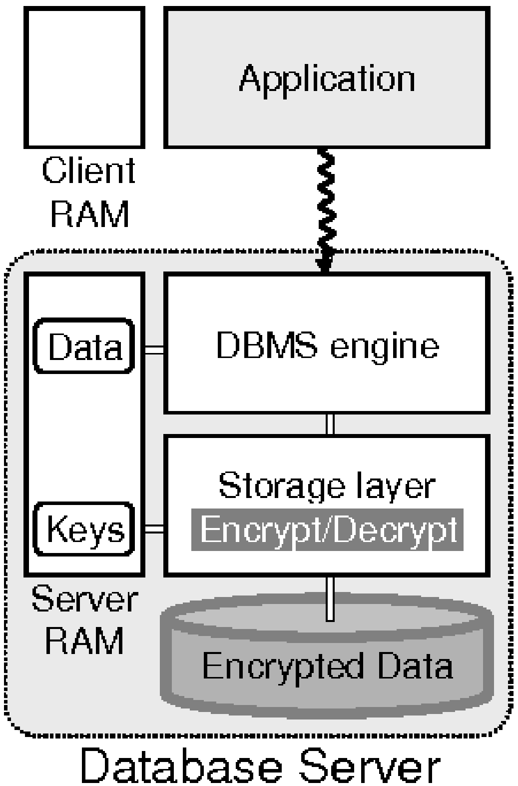


Figure from “Database Encryption” by Bouganim and Guo (2009).

Database-Level Encryption

- DBMS encrypts data when it is inserted into the database
- Advantage: Encryption strategy can be part of the database design (i.e., selective encryption possible, various granularities possible)
- Disadvantage: Performance degradation possible (e.g., encryption may make indexes useless)

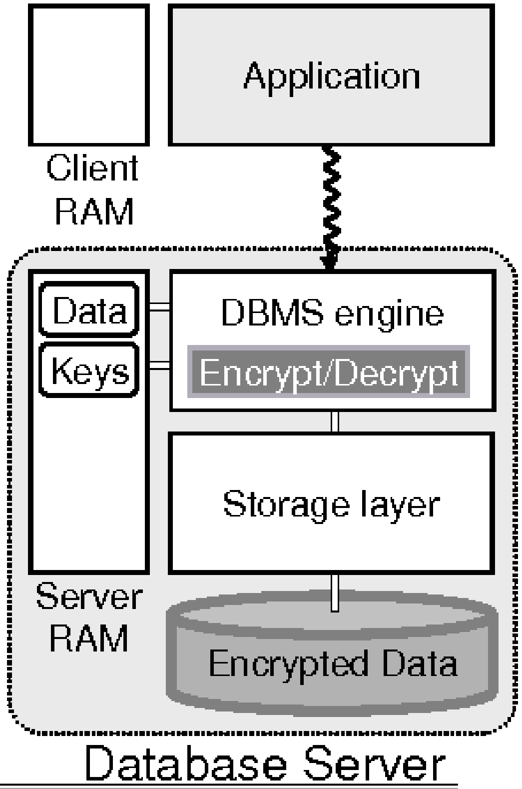


Figure from "Database Encryption" by Bouganim and Guo (2009).

Application-Level Encryption

- Application encrypts sensitive data before sending it to the DBS and decrypts data returned by the DBS
- Advantages:
  - Encryption keys separated from the encrypted data (i.e., no need to trust the DB administrator or cloud provider)
  - Highest flexibility in terms of granularity and key management
- Disadvantages:
  - Applications need to be modified
  - Performance overhead possible (e.g., prevents indexes for range queries)
  - No stored procedures and triggers

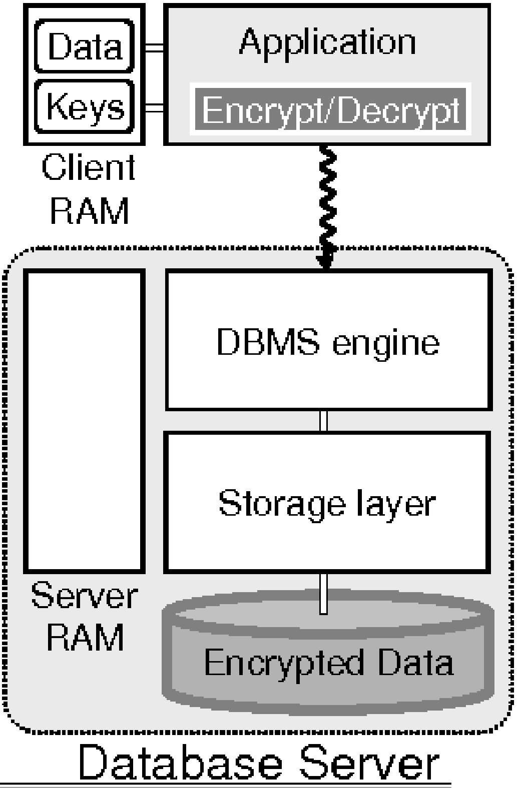


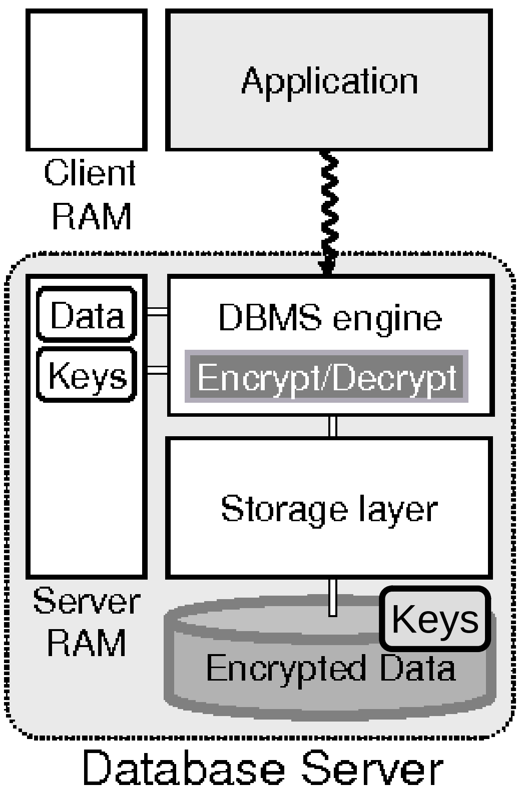
Figure from "Database Encryption" by Bouganim and Guo (2009).

Key Management

Who should have access
to the encryption keys?

Naive Solution (for DB-Level Encryption)

- Store keys in a restricted database table or file
- Potentially encrypt this table/file with a master key
  – Master key must also be stored on the database server

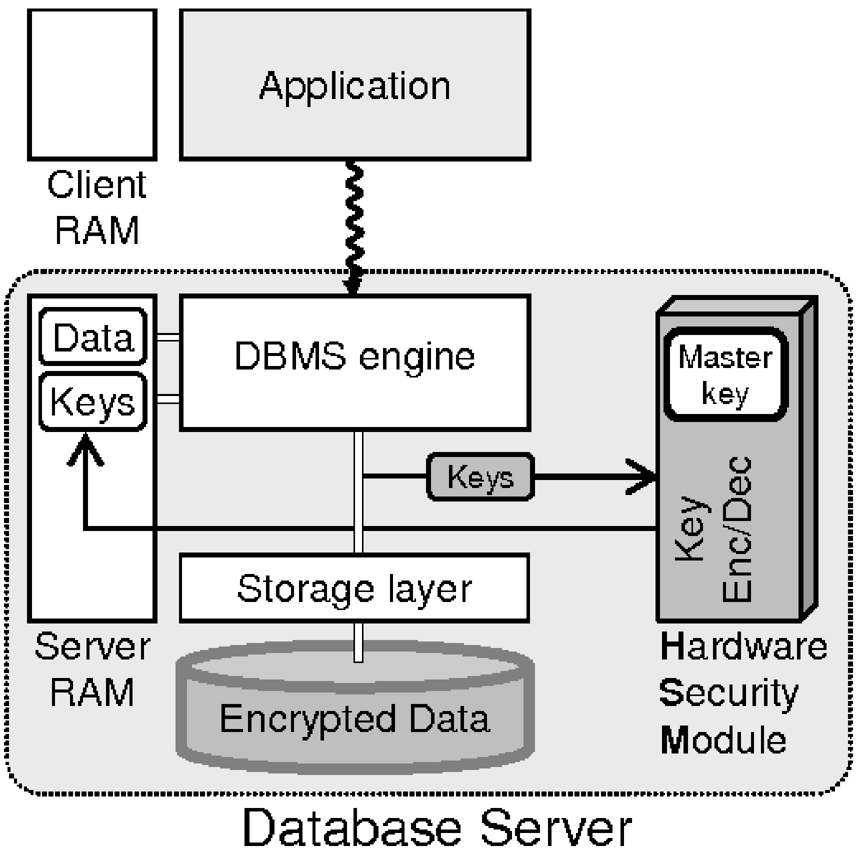


- Disadvantage:
  – Administrators with privileged access may use the keys to see and/or modify the data without being detected

Figure from "Database Encryption" by Bouganim and Guo (2009).

HSM Approach

- Use a hardware security module (HSM)
  - Specialized, tamper-resistant cryptographic chipsets

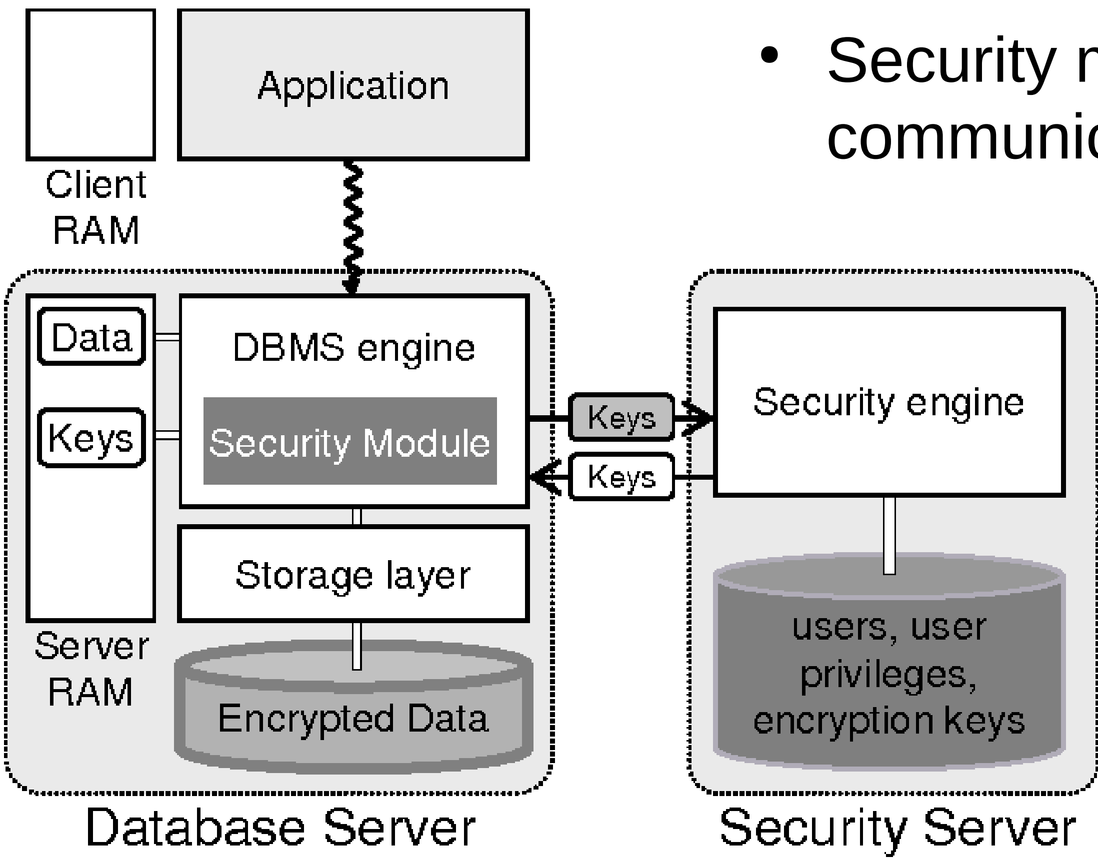


- Keys are stored encrypted in a restricted database table
- To encrypt/decrypt data the needed keys are decrypted by the HSM using the master key
- Decrypted keys are removed from main memory as soon as encryption/decryption of data has been performed

Figure from “Database Encryption” by Bouganim and Guo (2009).

Security Server Approach

- Move security-related tasks to distinct software on a distinct server that manages users, roles, privileges, encryption policies, and keys (potentially using an HSM)



- Security module within the DBMS communicates with the security server
- Clear distinction between DB administrator and security administrator

Figure from "Database Encryption" by Bouganim and Guo (2009).

www.liu.se