

Database and Data Center Security

Computer Security – Principles and Practice (Pearson, fourth edition)
W. Stallings, L. Brown

* These slides are an adaptation of the original slides of the authors of the book



Learning objectives

- Understand the unique need for database security, separate from ordinary computer security measures.
- basic elements of a database management system and of relational database systems.
- Define and explain SQL injection attacks.
- Compare and contrast different approaches to database access control.
- Explain how inference poses a security threat in database systems.
- Discuss the use of encryption in a database system.
- Discuss security issues related to data centers.

↑ Org. are using DB to concentrate the info they manage. They are efficient. Concentrating all this info DB, means that those IS are vulnerable for attacks. Defines a critical perimeter of defense. Might include a number of confidential info: info about your products, employees etc. The access to this DB should be provided to a lot of people for the diversity!

The need for database security

Even external entities, customers, partners...
Not all info should be made available to everybody. So we need security tailored to DBs. M/T is vital for an org.

- Organizational databases tend to concentrate sensitive information in a single logical system.

Examples include:

- Corporate financial data
- Confidential phone records
- Customer and employee information, such as name, Social Security number, bank account information, credit card information
- Proprietary product information
- Health care information and medical records

- Need to provide customers, partners, and employees with access to this information.
- Information in DB can be targeted by internal and external threats of misuse or unauthorized change.
- Security specifically tailored to databases is an increasingly important component of an overall organizational security strategy.

Sec. is complex for a number of reasons: ① People have gigantic databases and IS (data collections). Managing them is extremely complex and measures are not going as fast. ② DB are something you can program with an overall complex system: you should focus on securing the entire system, not only data. ③ You should take into account the full potential vulnerabilities around the system. ④ An org. is not willing to invest for DB specialists. ⑤ Org. have an uncontrolled growth of the IS in a very weird way. Servers here and there, DB, different OSes here and there. Difficult. From time to time there's an effort to standardize.

Database Security

⑥ Makes things complicated: cloud systems are an external org.

The increasing reliance on cloud technology to host part or all of the corporate database

Most enterprise environments consist of a heterogeneous mixture of database platforms, enterprise platforms, and OS platforms, creating an additional complexity hurdle for security personnel

①

Dramatic imbalance between the complexity of modern database management systems (DBMS) and the security technique used to protect them

②

Databases have a sophisticated interaction protocol, Structured Query Language (SQL), which is complex

Reasons database security has not kept pace with the increased reliance on databases are:

③

Effective database security requires a strategy based on a full understanding of the security vulnerabilities of SQL

④

The typical organization lacks full-time database security personnel

⑤

Data base management systems

Databases

= collection of data with a structure

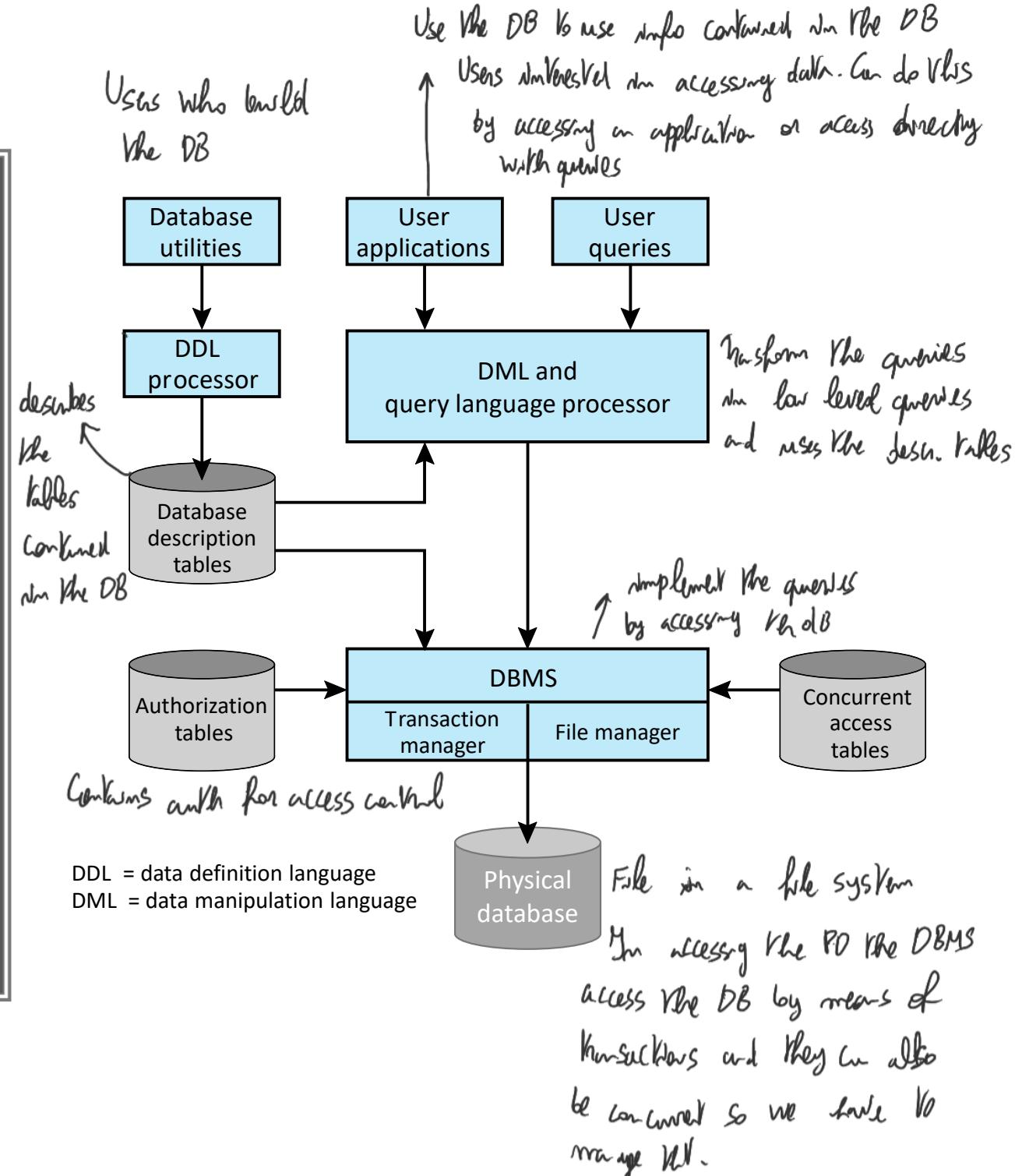
- Structured collection of data stored for use by one or more applications
- Contain the relationships between data items and groups of data items
- Can sometimes contain sensitive data that needs to be secured
- Query language:
• Provides a uniform interface to the database for users and applications

Database management system
(DBMS)

- Suite of programs for constructing ^{build} and maintaining ^{manage} the database
- Offers ad hoc query facilities to multiple users and applications

High level architecture.

DBMS architecture



DBMS relies on Databases, which are files stored in File System. There we already have a level of protection, but for access control we need a different granularity. We want to protect a specific piece of data. 1. Granularity 2. Type of operations: we have many ops like select, insert, update..., so we need more perms. We need something specific for DBs.

DBMS security requirements

- Database systems generate security requirements that are beyond the capability of typical OS-based security mechanisms or stand-alone security packages:
 - operating system security mechanisms typically control read and write access to entire files.
 - they could be used to allow a user to read or to write any information in, for example, a personnel file.
 - they could not be used to limit access to specific records or fields in that file.
- DBMS typically requires such a detailed (granular) access control.
- DBMS usually enable access controls over a wider range of commands
 - such as to select, insert, update, or delete specified items in the database.
- Need for specifically designed security services and mechanisms integrated with DBMS.

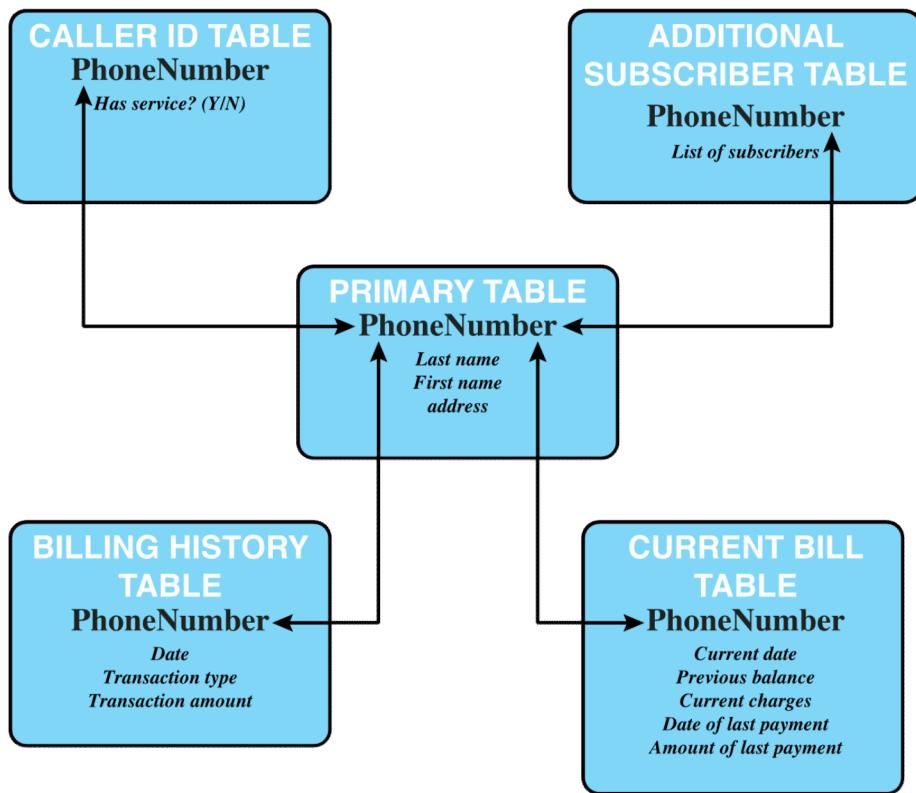
Relational Databases

- Table of data consisting of rows and columns
 - Each column holds a particular type of data
 - Each row contains a specific value for each column
 - Ideally has one column where all values are unique, forming an identifier/key for that row
- Enables the creation of multiple tables linked together by a unique identifier that is present in all tables
- Use a relational query language to access the database
 - Allows the user to request data that fit a given set of criteria
 - It's a declarative language (not a procedural one)

Made of

Kuple that contains attribute values.

attribute



Relational Databases

Example of a relational database model:

- Several tables related to each other
- Linked by a “primary key”
- In this case the primary key is the telephone number

Relational Database Elements

- Relation
 - a flat table in a file
- Tuple
 - rows (records) of the table
- Attribute
 - columns (field) of the table

Primary key

- Uniquely identifies a row
- Consists of one or more column names

Foreign key

- Links one table to attributes in another
- It's a primary key in another table

View/virtual table

- Result of a query that returns selected rows and columns from one or more tables
- Views are often used for security purposes: provide restricted access to a certain rows or columns.

| Formal Name | Common Name | Also Known As |
|-------------|-------------|---------------|
| Relation | Table | File |
| Tuple | Row | Record |
| Attribute | Column | Field |

| | | Attributes | | | | | |
|---------|-----|------------|-------|----------|-------|----------|--|
| | | A_I | • • • | A_j | • • • | A_M | |
| Records | 1 | x_{I1} | • • • | x_{Ij} | • • • | x_{IM} | |
| | • | • | | • | | • | |
| | • | • | | • | | • | |
| | • | • | | • | | • | |
| | i | x_{i1} | • • • | x_{ij} | • • • | x_{iM} | |
| | • | • | | • | | • | |
| | • | • | | • | | • | |
| | • | • | | • | | • | |
| | N | x_{N1} | • • • | x_{Nj} | • • • | x_{NM} | |

Basic Terminology for Relational Databases and
an abstract model of a relational database

A relational database example

| Department Table | | | Employee Table | | |
|------------------|------------------|---------|----------------|-----|------------|
| Did | Dname | Dacctno | Ename | Did | Salarycode |
| 4 | human resources | 528221 | Robin | 15 | 23 |
| 8 | education | 202035 | Neil | 13 | 12 |
| 9 | accounts | 709257 | Jasmine | 4 | 26 |
| 13 | public relations | 755827 | Cody | 15 | 22 |
| 15 | services | 223945 | Holly | 8 | 23 |

primary key
 foreign key
 primary key

(a) Two tables in a relational database

| Dname | Ename | Eid | Ephone |
|------------------|---------|------|------------|
| human resources | Jasmine | 7712 | 6127099348 |
| education | Holly | 3054 | 6127092729 |
| education | Robin | 2976 | 6127091945 |
| accounts | Smith | 4490 | 6127099380 |
| public relations | Neil | 5088 | 6127092246 |
| services | Robin | 2345 | 6127092485 |
| services | Cody | 9664 | 6127093148 |

(b) A view derived from the database

Structured Query Language (SQL)

- Standardized language to define schema, manipulate, and query data in a relational database
- Several similar versions of ANSI/ISO standard
- All follow the same basic syntax and semantics

SQL statements can be used to:

- Create tables
- Insert and delete data in tables
- Create views
- Retrieve data with query statements

Example

This creates the two tables:

```
CREATE TABLE department (
    Did INTEGER PRIMARY KEY,
    Dname CHAR (30),
    Dacctno CHAR (6)
)
CREATE TABLE employee (
    Ename CHAR (30),
    Did INTEGER,
    SalaryCode INTEGER,
    Eid INTEGER PRIMARY KEY,
    Ephone CHAR (10),
    FOREIGN KEY (Did) REFERENCES department (Did)
)
```

Department Table

| Did | Dname | Dacctno |
|-----|------------------|---------|
| 4 | human resources | 528221 |
| 8 | education | 202035 |
| 9 | accounts | 709257 |
| 13 | public relations | 755827 |
| 15 | services | 223945 |

primary
key

Employee Table

| Ename | Did | Salarycode | Eid | Ephone |
|---------|-----|------------|------|------------|
| Robin | 15 | 23 | 2345 | 6127092485 |
| Neil | 13 | 12 | 5088 | 6127092246 |
| Jasmine | 4 | 26 | 7712 | 6127099348 |
| Cody | 15 | 22 | 9664 | 6127093148 |
| Holly | 8 | 23 | 3054 | 6127092729 |
| Robin | 8 | 24 | 2976 | 6127091945 |
| Smith | 9 | 21 | 4490 | 6127099380 |

foreign
key primary
key

This produces a view with only ...?:

```
SELECT Ename, Eid, Ephone  
FROM Employee  
WHERE Did = 15
```

The following query creates the view of the DB in the table

```
CREATE VIEW newtable (Dname, Ename, Eid, Ephone)  
AS SELECT D.Dname E.Ename, E.Eid, E.Ephone  
FROM Department D Employee E  
WHERE E.Did = D.Did
```

Example

| Dname | Ename | Eid | Ephone |
|------------------|---------|------|------------|
| human resources | Jasmine | 7712 | 6127099348 |
| education | Holly | 3054 | 6127092729 |
| education | Robin | 2976 | 6127091945 |
| accounts | Smith | 4490 | 6127099380 |
| public relations | Neil | 5088 | 6127092246 |
| services | Robin | 2345 | 6127092485 |
| services | Cody | 9664 | 6127093148 |

(b) A view derived from the database



Question

The following table provides information about members of a golf club:

| Member-ID | Name | Skill Level | Age |
|-----------|--------|-------------|-----|
| 99 | Jimmy | Beginner | 20 |
| 36 | David | Experienced | 22 |
| 82 | Oliver | Medium | 21 |
| 23 | Alice | Experienced | 21 |

... where the primary key is Member-ID.

Which one of the following rows can be added to the table?

| Member-ID | Name | Skill Level | Age |
|-----------|------|-------------|-----|
| 91 | Tom | Experienced | 22 |
| 36 | Dave | Experienced | 21 |
| | Bob | Beginner | 20 |

SQL injection attacks

If AC is working, only the authorized individuals can perform ops. But those bypass the attack. You inject a query that is accepted by this entry and executed as this was their query. So it's not a matter of AC.

SQL Injection Attacks (SQLi)

- One of the most prevalent and dangerous network-based security threats
 - Sends malicious SQL commands to the database server
 - Most common attack goal is bulk extraction of data
- Depending on the environment SQL injection can also be exploited to:
 - Modify or delete data
 - Execute arbitrary operating system commands
 - Launch denial-of-service (DoS) attacks

Objective: make them believe that it's their command.

Examples of SQL injection attacks

- The July 2013 Imperva Web Application Attack Report:
 - surveyed a cross section of Web application servers in industry
 - monitored eight different types of common attacks.
 - The report found that SQLi attacks ranked first or second in:
 - total number of attack incidents
 - the number of attack requests per attack incident
 - average number of days per month that an application experienced at least one attack incident.
 - Imperva observed a single website that received 94,057 SQL injection attack requests in one day.
- The Open Web Application Security Project's 2013 report:
 - on the 10 most critical Web application security risks listed injection attacks, especially SQLi attacks, as the top risk.
 - This ranking is unchanged from its 2010 report.
- The Veracode 2016 State of Software Security Report:
 - the percentage of applications affected by SQLi attacks is around 35%.
- The Trustwave 2016 Global Security Report:
 - SQLi attacks are one of the top two intrusion techniques
 - SQLi can pose a significant threat to sensitive data such as personally identifiable information (PII) and credit card data;
 - it can be hard to prevent and relatively easy to exploit these attacks.

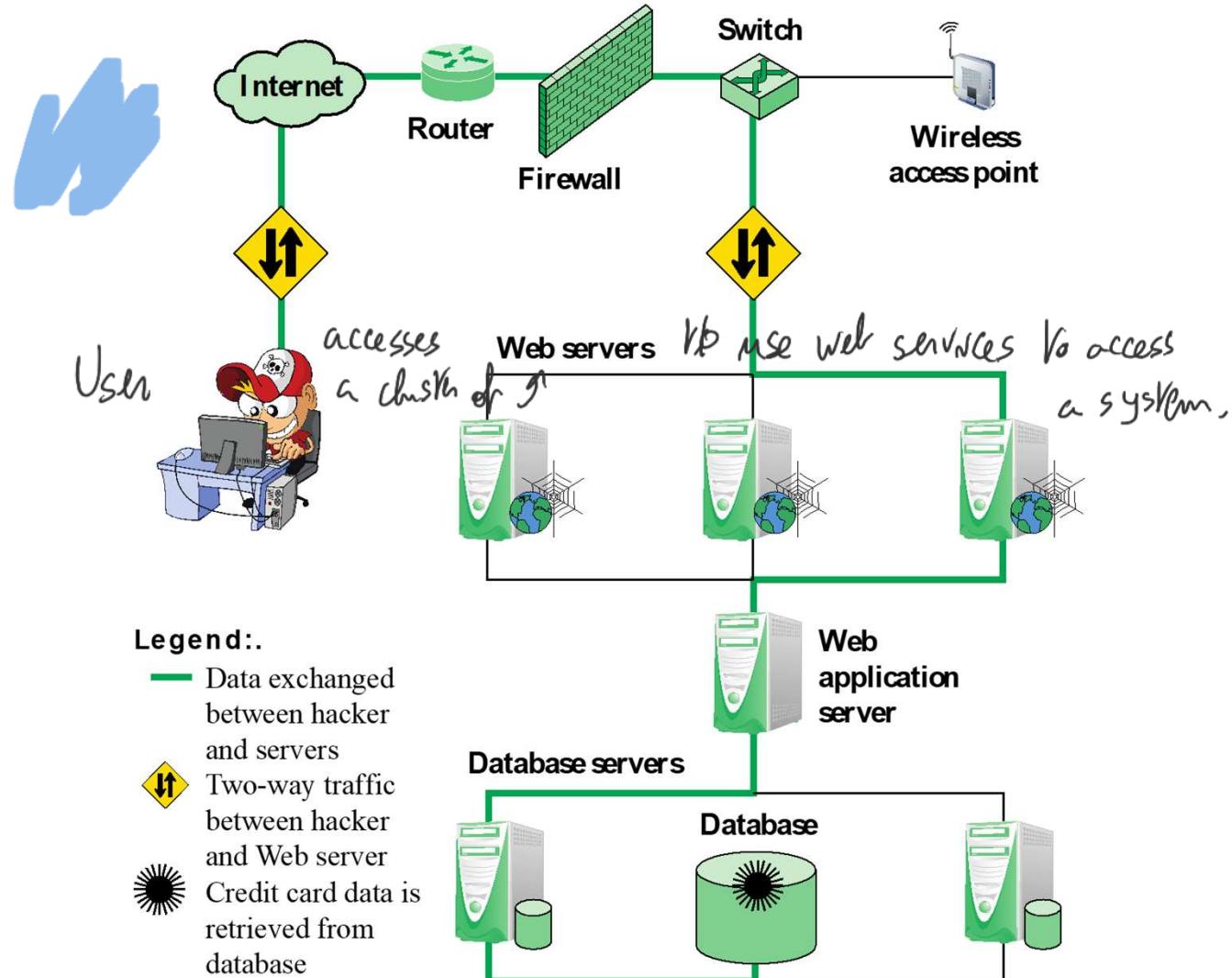
Several reports regarding most popular attacks. Very popular, in many contexts.

DBMS is complex. Users access DB by many apps or queries. So you provide data to the app which produces SQL to get the info. In the recent years, it's natural to implement those apps as web apps. Take user data and send a query. We provide to these interfaces an input that is interpreted wrongly by the application. Usually we have data that is finally interpreted as part of the query itself. You can do whatever, because the query is executed with the app privileges.

SQL Injection Attacks (SQLi)

- In general terms, an SQLi attack is designed to exploit the nature of Web application pages
 - No longer static pages
 - Nowadays web pages have dynamic components and content
 - Many pages query DB in the servers, also to access sensitive data (e.g. when you buy something...)
- SQLi attack is designed to send malicious SQL commands to the database server.
 - Extract bulk of data
 - Modify or delete data
 - Execute arbitrary operating system commands
 - Launch DoS attacks, ...

A typical SQL injection attack

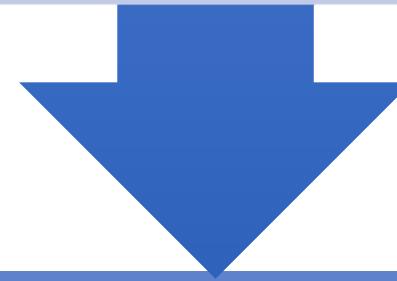


Input should contain special characters.

Injection Technique

The SQLi attack typically works by prematurely terminating a text string and appending a new command

Because the inserted command may have additional strings appended to it before it is executed the attacker terminates the injected string with a comment mark “--”



Subsequent text is ignored at execution time

Example of SQLi

As a simple example, consider a script that builds an SQL query by combining predefined strings with text entered by a user:

```
var ShipCity;  
ShipCity = Request.form ("ShipCity");  
var sql = "SELECT * FROM OrdersTable  
WHERE ShipCity = '" + ShipCity + "'";
```

The intention of the script's designer is that a user will enter the name of a city. For example, if the user enters Pisa, then the following SQL query is generated:

```
SELECT * FROM OrdersTable WHERE ShipCity  
= 'Pisa';
```

Example of SQLi

Suppose, however, the user enters the following:

Boston'; DROP table OrdersTable--

This results in the following SQL query:

```
SELECT * FROM OrdersTable WHERE ShipCity  
= 'Boston'; DROP table OrdersTable--';
```

The semicolon ';' separates two SQL statements, '--' indicates that the rest of the text (if any) is just a comment

The consequence is:

- the server executes the DROP request right after the query...
- ... and deletes the table.



if the table name is correct and the app is authorized to destroy the table.

SQLi Attack Avenues

Let you inject queries.

User input

- Attackers inject SQL commands by providing suitable crafted user input

Server variables

- Attackers can forge the values that are placed in HTTP and network headers and exploit this vulnerability by placing data directly into the headers *Pass values to those header variables*

Second-order injection

- A malicious user could rely on data already present in the system or database to trigger an SQL injection attack... *(Activate other parts of the code at one time)*
- ... when the attack occurs, the input that modifies the query to cause an attack does not come from the user, but from within the system itself

Cookies

- An attacker could alter cookies such that when the application server builds an SQL query based on the cookie's content, the structure and function of the query is modified

Physical user input

- Applying user input that constructs an attack outside the realm of web requests

→ executed using QR codes. For example scan a QR code that provides access to db, so you can modify the QR.

Inband Attacks

If the channel we are using
↑ for injection will get the data is the same.

Uses the same communication channel for injecting SQL code and retrieving results

- The retrieved data are presented directly in application Web page
- Include:

Tautology

This form of attack injects code in one or more conditional statements so that they always evaluate to true

End-of-line comment

After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments

Piggybacked queries

The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request

force the boolean expression in the where clause to always be true.

We do not need the name of the table!

but we limited on the attributes selected by the query.



Question - inband attack – 1

Consider a script whose intent is to require the user to enter a valid name and password:

```
$query = "SELECT info FROM user  
WHERE name = '$_GET[\"name\"]'  
AND pwd = '$_GET[\"pwd\"]'";
```

What does the user may insert to implement a tautology attack?
... for example to read the entire content of the table ...

$$_GET[\"pwd\"] = \"a\" \text{ OR } 1=1;--$

$(. .) \text{ OR } 1=1$



Solution - inband attack – 1

```
$query = "SELECT info FROM user WHERE name =
'$_GET[\"name\"]' AND pwd = '$_GET[\"pwd\"]'";
```

The attacker may submit:

The resulting query would look like this:



Solution - inband attack – 1





Question - inband attack – 2

Consider an SQL statement:

```
SELECT id, firstname, department FROM authors  
WHERE forename = 'David' AND id = 939
```

1. What is this statement trying to search from the database?
2. Assume that the `forename` and `id` fields are being gathered from user-supplied input, and suppose the user responds with:
`forename: David'; drop table employees --`
What will be the effect?
3. What if the user inserts:
`forename: ' OR 9=9 --`



Solution - inband attack – 2



Inferential Attack

There is no real transfer of data.

1. Inject legal queries and combine info and find something that is confidential.

Simple way: combine yes or no answers to get info.

- There is no actual transfer of data, but the attacker is able to reconstruct the information by sending particular requests and observing the resulting behavior of the Website/database server
- Include:
 - Illegal/logically incorrect queries
 - This attack lets an attacker gather important information about the type and structure of the backend database of a Web application
 - The attack is considered a preliminary, information-gathering step for other attacks
 - For example, it may exploit over-informative error messages
 - Blind SQL injection
 - Allows attackers to infer the data present in a database system even when the system is sufficiently secure to not display any erroneous information back to the attacker
 - For example, inject simple true/false queries to infer something from the answers

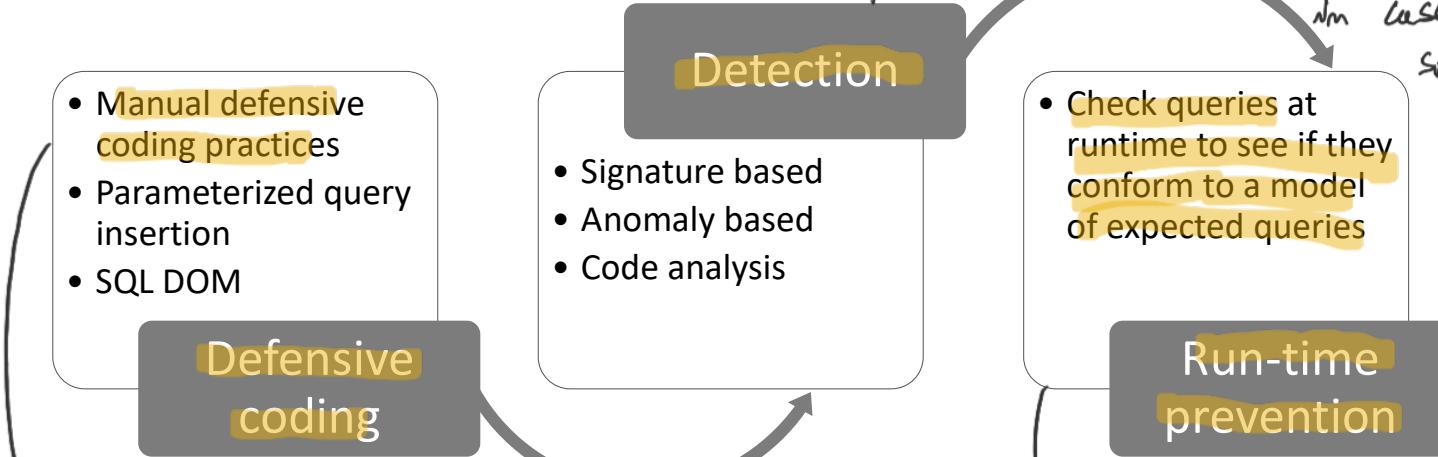
Out-of-Band Attack

Maybe vulnerabilities that allow you
to get info from another channel like sending yourself an email.

- Data are retrieved using a different channel
- This can be used when there are limitations on information retrieval, but outbound connectivity from the database server is lax

SQLi Countermeasures

- Three types: not mutually exclusive:



Adopt strategies to avoid problem from the beginning.

Programming practices that sanitize input. Or use tools that form the query in a parameterized way.

SQL DOM is a set of classes you can use to generate well formed queries.

Observing the incoming and outgoing traffic and adopt strategies to detect SQL attacks. One method is ①: description of the patterns of attacks and we check them. ② You use AI to train to learn normal behavior and in case of deviation it is stopped.

- Check queries at runtime to see if they conform to a model of expected queries

Run-time prevention

In ② we are analyzing input data and output data, we are here analyzing the syntax of queries.

SQLi Countermeasures – defensive coding

- **Manual defensive coding practices:**

- A common vulnerability exploited by SQLi attacks is insufficient input validation.
- apply suitable defensive coding practices.
- An example is input type checking, to check that inputs that are supposed to be numeric contain no characters other than digits.
- Another type of coding practice is one that performs pattern matching to try to distinguish normal input from abnormal input.

- **Parameterized query insertion:**

- allow the developer to accurately specify the structure of an SQL query...
- ... and pass the value parameters to it separately
- ... such that any unsanitary user input is not allowed to modify the query structure.

- **SQL DOM (Domain Object Model):**

- is a set of classes that enables automated data type validation and escaping
- uses encapsulation of database queries to provide a safe and reliable way to access databases.
- changes the query-building process from an unregulated one that uses string concatenation to a systematic one that uses a type-checked API.
- Within the API, developers are able to systematically apply coding best practices such as input filtering and rigorous type checking of user input.

Exercise D-1

Referring to the data base that contains the table Employee, consider the script:

```
$query = "SELECT Ephone FROM Employee  
WHERE Ename ='$_GET["name"]' AND  
Salarycode<24";
```

Consider the following approaches to an injection attack:

- Tautology
- Piggibacked query

Discuss the possibility and the assumption under which each of these attacks is possible and write the attack.

| Employee Table | | | | |
|----------------|-----|------------|------|------------|
| Ename | Did | Salarycode | Eid | Ephone |
| Robin | 15 | 23 | 2345 | 6127092485 |
| Neil | 13 | 12 | 5088 | 6127092246 |
| Jasmine | 4 | 26 | 7712 | 6127099348 |
| Cody | 15 | 22 | 9664 | 6127093148 |
| Holly | 8 | 23 | 3054 | 6127092729 |
| Robin | 8 | 24 | 2976 | 6127091945 |
| Smith | 9 | 21 | 4490 | 6127099380 |

$\overbrace{\text{foreign key}}$ $\overbrace{\text{primary key}}$



Solution D-1

```
$query = "SELECT Ephone FROM Employee  
WHERE Ename ='$_GET["name"]'" AND  
Salarycode<24";
```

Employee Table

| Ename | Did | Salarycode | Eid | Ephone |
|---------|-----|------------|------|------------|
| Robin | 15 | 23 | 2345 | 6127092485 |
| Neil | 13 | 12 | 5088 | 6127092246 |
| Jasmine | 4 | 26 | 7712 | 6127099348 |
| Cody | 15 | 22 | 9664 | 6127093148 |
| Holly | 8 | 23 | 3054 | 6127092729 |
| Robin | 8 | 24 | 2976 | 6127091945 |
| Smith | 9 | 21 | 4490 | 6127099380 |

foreign
key

primary
key

DBMS Access control

Database Access Control

Database access control system determines:

If the user has access to the entire database or just portions of it

what access rights the user has
(create, insert, delete, update, read, write)

Can support a range of administrative policies

Centralized administration

- Small number of privileged users may grant and revoke access rights

Ownership-based administration

- The creator of a table may grant and revoke access rights to the table

Decentralized administration

- The owner of the table may grant and revoke authorization rights to other users, allowing them to grant and revoke access rights to the table

There is a general administration of course but: also letting other roles to give privileges: combination of RBAC and DAC

SQL Access Controls

- Two commands for managing access rights:
 - Grant
 - Used to grant one or more access rights or can be used to assign a user to a role
 - Revoke
 - Revokes the access rights
- Typical access rights are:
 - Select
 - Insert
 - Update
 - Delete
 - References

SQL Access Controls – example

An example of SQL syntax to grant and revoke access rights:

Grant:

```
GRANT          {privileges | role}
              [ON           table]
              TO           {user | role | PUBLIC}
              [IDENTIFIED BY password] associating a PW.
              [WITH         GRANT OPTION]
```

example:

```
GRANT SELECT ON ANY TABLE TO ricflair
```

SQL Access Controls – example

An example of SQL syntax to grant and revoke access rights:

Revoke:

```
REVOKE {privileges | role}
[ON table]
FROM {user | role | PUBLIC}
```

*Also write a role. The system can
become a mess like]*

Example:

```
REVOKE SELECT ON ANY TABLE FROM ricflair
```

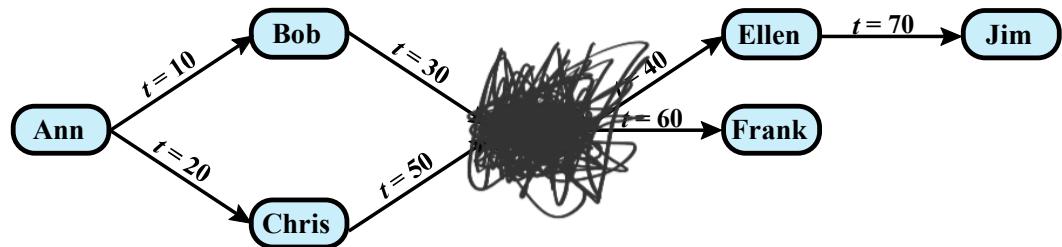
Example: Bob revokes privilege from David

Revocation cascades:

- If a user revokes the right of one another, all the other rights that had been granted meanwhile should be revoked

Consider the grants cascade phenomenon in the figure:

- What should happen if Ann revokes both Bob and Chris?
- What should happen if Bob revokes David?



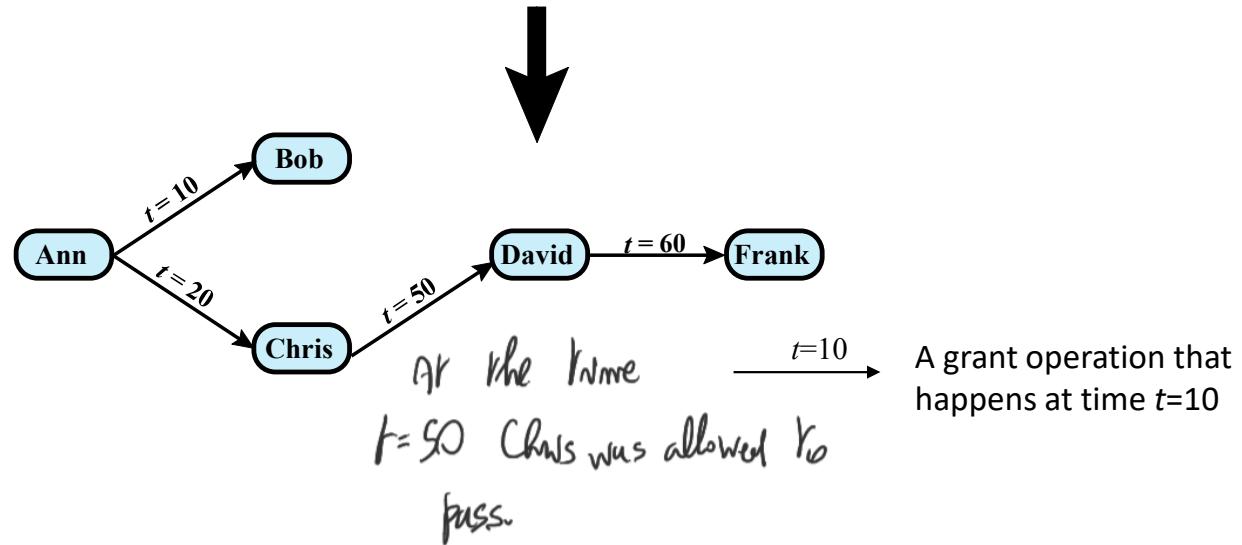
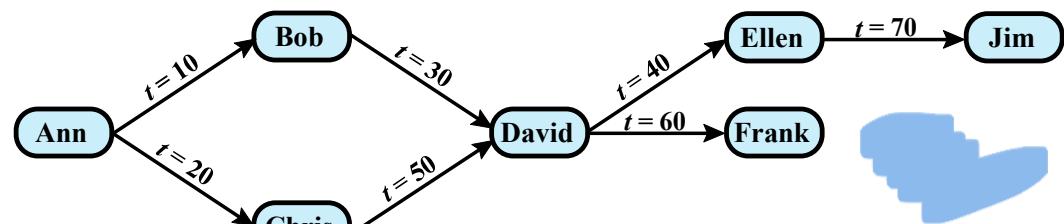
POINT: You can decide how to implement and how to standardize in a sense.

$t=10 \rightarrow$ A grant operation that happens at time $t=10$

If I revoke someone else's privilege, we rollback to a situation where I never gave that privilege to the user.

Convention: When user A revokes an access right, any cascaded access right is also revoked, unless that access right would exist even if the original grant from A had never occurred.

Example:
Bob revokes
privilege
from David



In a FS we have a less fine granularity than in DB, in which we have a larger number of operations.
It violates the principle of minimum privilege.

Role-Based Access Control (RBAC)

- Access control in DBMS different than in file systems:
 - FS with a (relatively) few applications and privileges, coarse-grained access control
 - DBMS with a large number of applications, a user associated to a number of different tasks
- Bad practice to simply give a user all the rights he needs for all tasks he performs
 - Need for fine-grained access control
- Role-based access control a natural fit for DB access control
 - eases administrative burden and improves security
- An application performs a number of tasks, We also have to keep tasks
 - Each task requires specific access rights to portions of the database.
 - For each task: one or more roles specify the needed access rights.

The roles depend on the context in which you are operating. We have classes easily identifiable: groups of users with diff. responsibilities. Users that interact with database might not need to interact directly with the DB.

Owner can modify struct of DB. The Admin manage the physical accesses to the DB, files, the def. of roles etc.
(broad) categories of database users

| Application owner | End user | Administrator |
|---|--|--|
| <ul style="list-style-type: none">An end user who owns database objects as part of an application | <ul style="list-style-type: none">An end user who operates on database objects via a particular application but does not own any of the database objects | <ul style="list-style-type: none">User who has administrative responsibility for part or all of the database |

Role-Based Access Control (RBAC)

- **Assigning roles and rights:**
 - The application owner may assign roles to end users.
 - Administrators are responsible for more sensitive or general roles:
 - those having to do with managing physical and logical database components, such as data files, users, and security mechanisms.
 - ... hence they needs to be set up with certain administrators certain privileges.
 - ... they can, in turn, assign users to administrative-related roles.
- A database RBAC facility needs to provide the following capabilities:
 - Create and delete roles.
 - Define permissions for a role.
 - Assign and cancel assignment of users to roles.

Roles are fixed
 and predefined.
 In addition,
 user defined roles
 for end users.
Example:
 fixed roles in Microsoft SQL Server

- Note:
- users may also create new roles and grant rights to such roles
 - Implement a customizable and complex security management

| Role Permissions | |
|--|--|
| Fixed Server Roles <i>Administrations</i> | |
| sysadmin | Can perform any activity in SQL Server and have complete control over all database functions |
| serveradmin | Can set server-wide configuration options and shut down the server |
| setupadmin | Can manage linked servers and startup procedures |
| securityadmin | Can manage logins and CREATE DATABASE permissions, also read error logs and change passwords |
| processadmin | Can manage processes running in SQL Server |
| Dbcreator | Can create, alter, and drop databases |
| diskadmin | Can manage disk files |
| bulkadmin | Can execute BULK INSERT statements |
| Fixed Database Roles <i>Owners of Tables</i> | |
| db_owner | Has all permissions in the database |
| db_accessadmin | Can add or remove user IDs |
| db_datareader | Can select all data from any user table in the database |
| db_datawriter | Can modify any data in any user table in the database |
| db_ddladmin | Can issue all data definition language statements |
| db_securityadmin | Can manage all permissions, object ownerships, roles and role memberships |
| db_backupoperator | Can issue DBCC, CHECKPOINT, and BACKUP statements |
| db_denydatareader | Can deny permission to select data in the database |
| db_denydatawriter | Can deny permission to change data in the database |



Question – access revoking – 1

Assume A, B, and C grant certain privileges on the employee table to X, who in turn grants them to Y, as shown in the following table, with the numerical entries indicating the time of granting:

| UserID | Table | Grantor | READ | INSERT | DELETE |
|--------|----------|---------|------|--------|--------|
| Y | Employee | A | 15 | 15 | — |
| X | Employee | B | 20 | — | 20 |
| Y | Employee | X | 25 | 25 | 25 |
| X | Employee | C | 30 | — | 30 |

At time $t = 35$, B gives the command

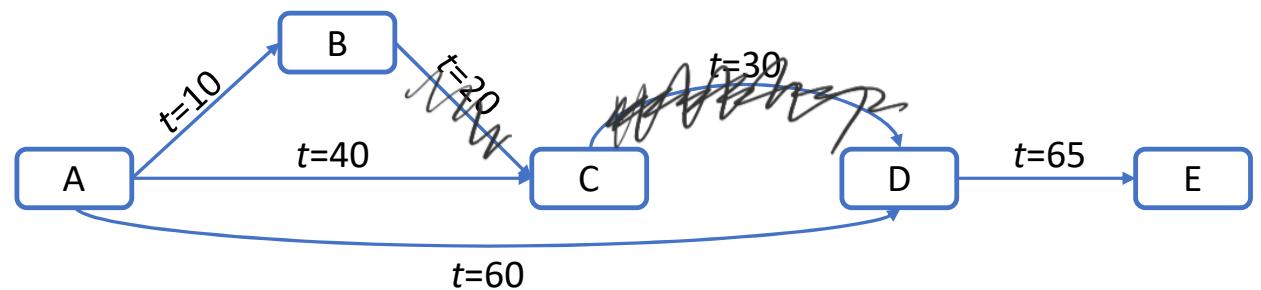
```
REVOKE ALL RIGHTS ON Employee FROM X
```

Which access rights, if any, of X and Y must be revoked?



Question – access revoking – 2

Consider the sequence of grant operations for a specific access right on a table.
Assume at $t = 70$, B revokes the access right from C.
Show the resulting diagram of access right dependencies.





Question – access granting

GRANT SELECT
ON ITEM
TO ...

- Users *hulkhogan* and *undertaker* do not have the SELECT access right to the `Inventory` table and the `Item` table.
- These tables were created by and are owned by user *bruno-s*.
- Write the SQL commands that would enable *bruno-s* to grant SELECT access to these tables to *hulkhogan* and *undertaker*.

Inference

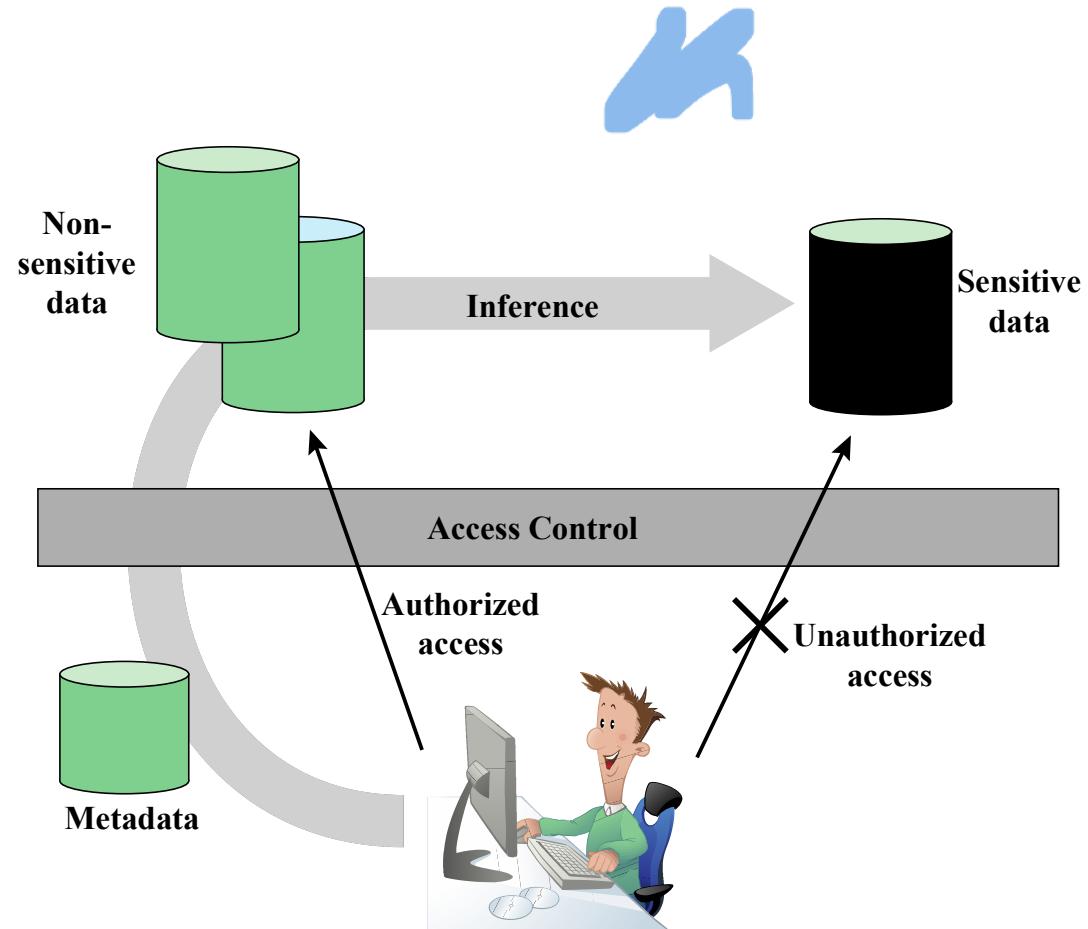
A user is legitimately accessing a database and gets access to confidential info. Combining data into larger chunks are more impactful.

Inference

- Inference is the process of performing authorized queries and deducing unauthorized information from the legitimate responses received.
- The inference problem arises when:
 - the combination of a number of data items is more sensitive than the individual items
 - or when a combination of data items can be used to infer data of higher sensitivity.
- How it works:
 - The attacker may make use of non-sensitive data as well as metadata.
 - Metadata refers to knowledge about correlations or dependencies among data items that can be used to deduce information not otherwise available to a particular user.
 - The information transfer path by which unauthorized data is obtained is referred to as an inference channel

Metadata: common knowledge about the correlation of items that is not displayed directly. If system is vulnerable, an inference channel is said to exist.

Inference:
indirect
information
access via
inference
channel





Inference: example

For example we have metadata that
the output is ordered in the same
way.

| Item | Availability | Cost (€) | Department |
|--------------------|-----------------|----------|------------|
| Shelf support | in-store/online | 7.99 | hardware |
| Lid support | online only | 5.49 | hardware |
| Decorative chain | in-store/online | 104.99 | hardware |
| Cake pan | online only | 12.99 | housewares |
| Shower/tub cleaner | in-store/online | 11.99 | housewares |
| Rolling pin | in-store/online | 10.99 | housewares |

(a) Inventory table

| Availability | Cost (€) |
|-----------------|----------|
| in-store/online | 7.99 |
| online only | 5.49 |
| in-store/online | 104.99 |

Available in store or online (b) Two views

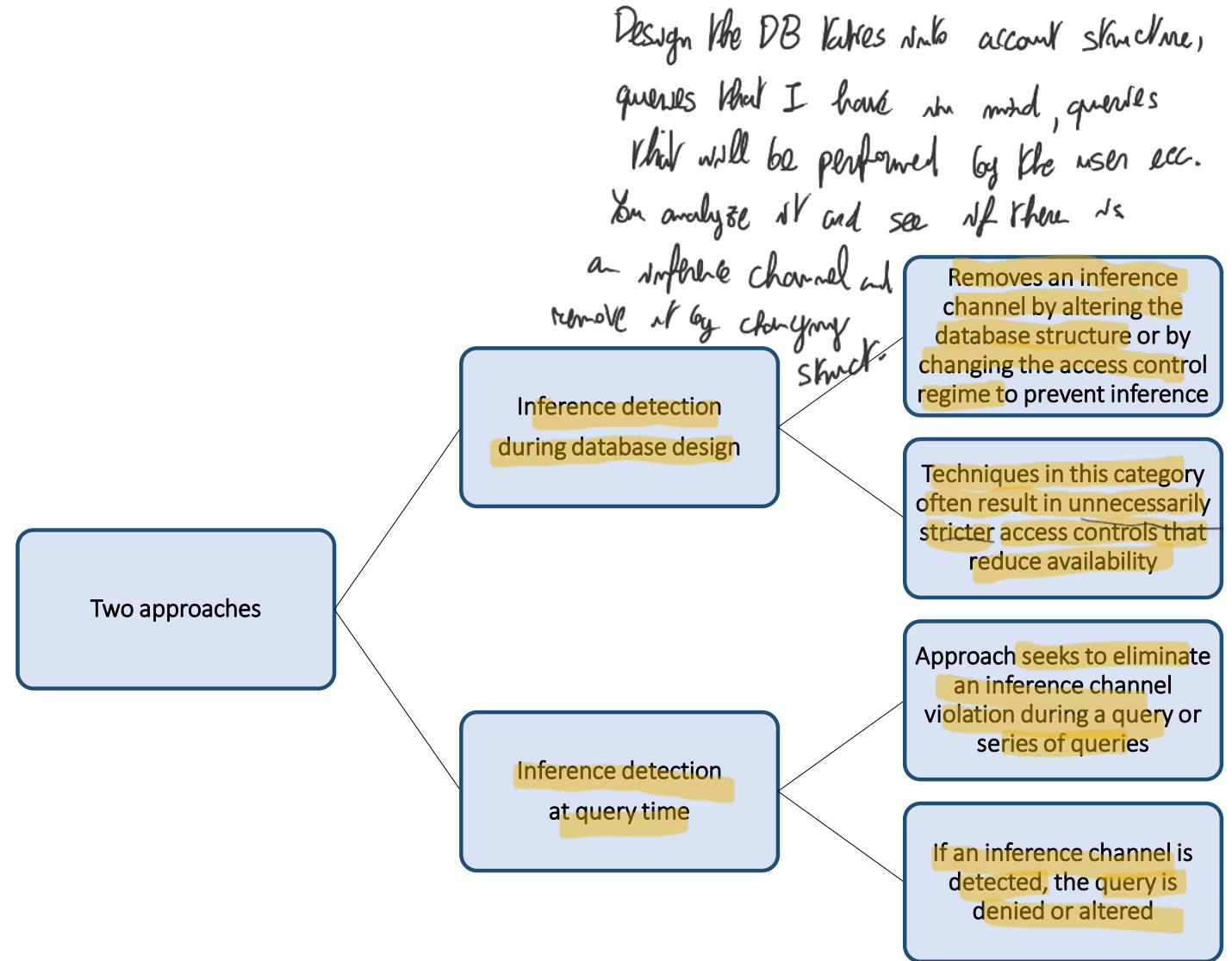
| Item | Department |
|------------------|------------|
| Shelf support | hardware |
| Lid support | hardware |
| Decorative chain | hardware |

Association between cost and view is sensitive.

| Item | Availability | Cost (€) | Department |
|------------------|-----------------|----------|------------|
| Shelf support | in-store/online | 7.99 | hardware |
| Lid support | online only | 5.49 | hardware |
| Decorative chain | in-store/online | 104.99 | hardware |

(c) Table derived from combining query answers

Dealing with the threat of disclosure by inference



- Some inference detection algorithm is needed for either of these approaches
- Progress has been made in devising specific inference detection techniques for multilevel secure databases and statistical databases

Inference

—

example (I)

Not easy to deal with inference, consider the following example:

- A company keeps a DB about its employees, and wishes to keep confidential the association employee-salary.
- Any clerk may access the database to take notice of employees and of salaries, but not of their association.
- The DB can be structured with three tables:
 - Employees (Emp#, Name, Address) // Emp# primary key for Employee
 - Salaries (S#, Salary) // S# primary key for Salaries
 - Emp-Salary (Emp#, S#)
- Where a clerk cannot access Emp-Salary
- ... and the problem is solved (?)

Inference

example (II)

Now assume that we need to associate to the salary a new non-sensitive attribute, which is the employee Employment-Date.

We may add the Employment-Date attribute to the salary table:

- Employees (Emp#, Name, Address) // *Emp# primary key for Employee*
- Salaries (S#, Salary, Employment-Date) // *S# primary key for Salaries*
- Emp-Salary (Emp#, S#)

↓ This is something that we can use
with other individuals

Again, the DB does not provide any info about the association employee name – salary

Is the problem still solved?

Inference — example (III)

- ... the DB does not provide any info about the association employee name – salary, however...
- ... the employment date of an employee may be an easily discoverable attribute of an employee... he may even tell
- A clerk in this way may infer his salary



Question

Any idea on how to solve this inference issue?

Tables:

- Employees (Emp#, Name, Address) // *Emp# primary key for Employee*
- Salaries (S#, Salary, Employment-Date) // *S# primary key for Salaries*
- Emp-Salary (Emp#, S#)

Inference

example (III)

Lessons learned:

1. The first security problem (how to keep confidential the salary of an employee) can be solved by looking at the DB structure
2. The second security problem (a violation of the confidentiality due to the introduction of a non-sensitive attribute) cannot be detected just by looking at the DB!

You use knowledge of the world outside
the technical knowledge of the table.

When it's not just your DB vulnerable to inference...

- Happened in 2018
- Sensitive information about the location and staffing of military bases and spy outposts around the world has been revealed by a fitness tracking company.
- The details in a data visualisation map that shows all the activity tracked by users of its app, while doing their exercises, and share it with others.
- The map is incredibly accurate (with more than 3 trillion individual GPS data points)
 - Gives extremely sensitive information about a subset of Strava users: military personnel on active service.
 - A comment of an analyst: The heatmap looks very pretty, but is not amazing for Op-Sec (short for operational security). US Bases are clearly identifiable and mappable.”

Fitness tracking app Strava reveals location of secret US army base

Data about exercise routes shared online by Strava users can pinpoint overseas facilities

● Latest: Strava suggests military users ‘opt in’ to sharing GPS data



A military base in Helmand Province, Afghanistan with a heatmap overlay from Strava. Photograph: Strava Heatmap



Question

Considering the previous example, is there any other way, beyond putting the attribute Employment-Date in the Employee table?

DB encryption

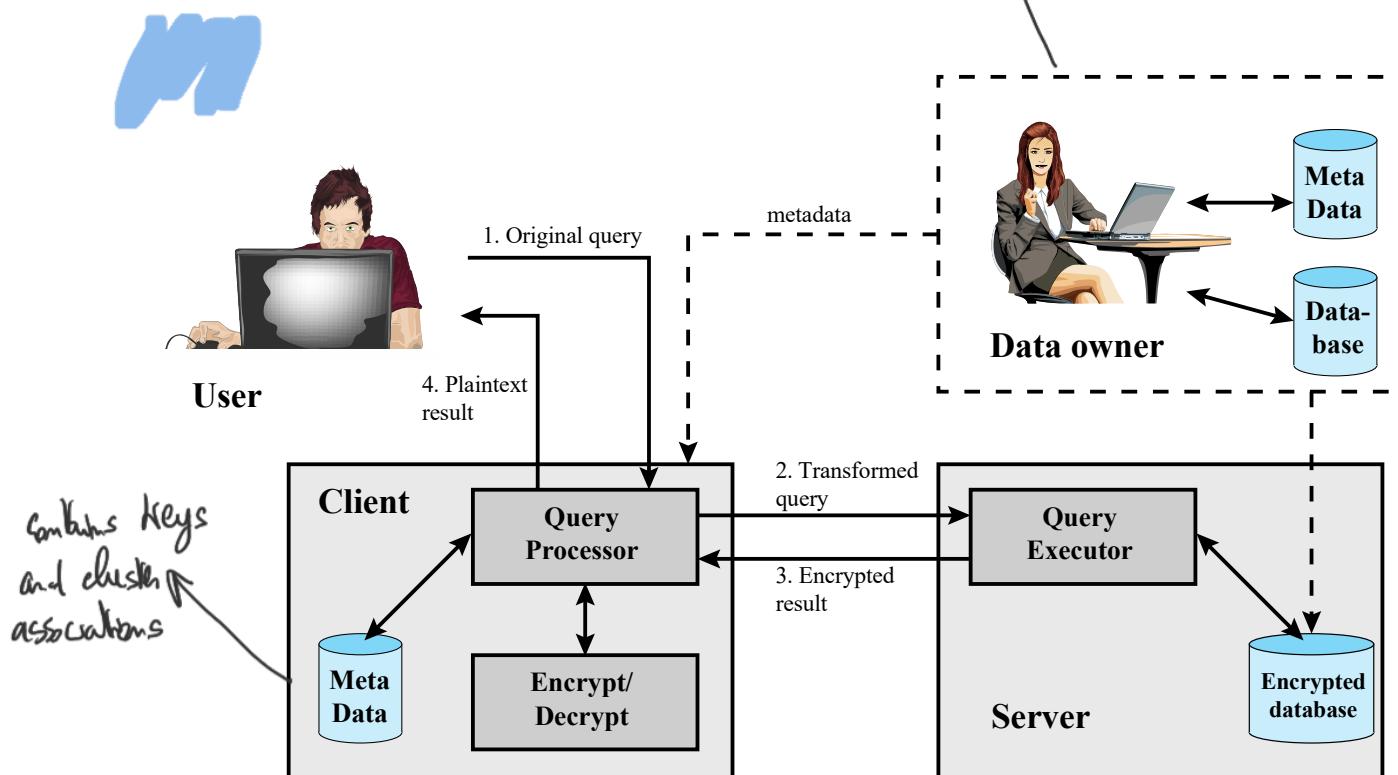
They are critical assets! They should be very well protected.

Database Encryption

- The database is typically the most valuable information resource for any organization
- Protected by multiple layers of security
 - Firewalls, authentication, general access control systems, DB access control systems, database encryption ADDITIONAL LAYER
- Encryption becomes the last line of defense in database security
 - Can be applied to the entire database, at the record level, the attribute level, or level of the individual field
when all other layers fail &
- Disadvantages to encryption:
 - Key management
 - Authorized users must have access to the decryption key for the data for which they have access
 - Lack of flexibility
 - When part or all the database is encrypted, it becomes more difficult to perform record searching
- Encryption at different levels:
 - Entire DB, table, record, attribute etc.

* Can be important for example when we use Cloud Computing service: The protection we already have might not be enough. Encryption could be a solution.

A database encryption scheme



owns database and metadata
that describes the database
itself. What happens if the
owner needs to grant access to a
user?

2 avenues of attack: no encryption. An attacker breaks the server and access data. Or if the server is managed by a service provider, the SP could access the data. So encryption. The ext. user submits a query to the DB, that must be translated by a query processor that gets in touch with a query executor that physically access the DB. The query processor transforms the query to work with encrypted data. We separate the two, because it is more secure. A breach on the server, without key not stored on server

can't access info. If there is a breach in the client, the attacker can obtain the key but not the data. PROBLEM: the structure of the encrypted db is different from the real one. You need metadata to understand the prop association. By using metadata in client (also key) the data is encrypted and interpreted and result is sent back to user.

- For example, consider this query for the table below:

```
SELECT Ename, Eid, Ephone
FROM Employee
WHERE Did = 15
```

- Assume the encryption key k is used and the encrypted value of the department Did 15 is $E(k, 15) = 1000110111001110$.
- Then, the query processor at the client could transform the preceding query into:

```
SELECT Encrypted_rows
FROM Encrypted_Employee
WHERE Did = 1000110111001110
```

Database Encryption



Employee Table

| Ename | Did | Salarycode | Eid | Ephone |
|---------|-----|------------|------|------------|
| Robin | 15 | 23 | 2345 | 6127092485 |
| Neil | 13 | 12 | 5088 | 6127092246 |
| Jasmine | 4 | 26 | 7712 | 6127099348 |
| Cody | 15 | 22 | 9664 | 6127093148 |
| Holly | 8 | 23 | 3054 | 6127092729 |
| Robin | 8 | 24 | 2976 | 6127091945 |
| Smith | 9 | 21 | 4490 | 6127099380 |

Encrypted_employee Table

| Encrypted_rows | Did |
|-----------------------------|------------------|
| 11010011110101000011010... | 1010111010001111 |
| 101111010100100010111010... | 1000110111001110 |
| 001010001110100110101001... | 0000101110011000 |
| 10101011110111010000101... | 0001110101110001 |
| 110101010100111000100101... | 1110100001010101 |
| 001000110010101000101000... | 0111000101100011 |
| 010110101000011001001010... | 1101111100101111 |

POSSIBLE EXAM QUESTION

Consider the following table, and assume the queries have the form:

```
SELECT grade FROM students WHERE date=xxxxx
```

Design an encrypted version of the table and the corresponding version of the transformed query

| Id (primary key) | name | exam | grade | date |
|------------------|-----------------|----------------------|-------|------------|
| 40110 | Ada Lovelace | Computer programming | 20 | 17/02/2024 |
| 40811 | Alan Turing | Information Theory | 27 | 07/01/2024 |
| 40916 | Linus Torvalds | Operating Systems | 24 | 10/01/2024 |
| 41022 | Tim Berners-Lee | Markup languages | 22 | 12/02/2024 |

Encrypted_students Table

| encrypted_rows | encrypted_date |
|----------------|----------------|
| 01001001 | 0011000 |
| 010001110 | 0111110 |
| 0111100010 | 000001 |

```
SELECT encrypted_rows FROM  
Encrypted_students WHERE  
encrypted_date = "0011000"
```

Possible second solution: use 3 attributes, date, rest of the row, grade.

Best practice? Depends on the balance between security and efficiency. And you won't have 1 query.



Question

- In the previous scheme, each data is encrypted and the operations on the DB are based on encrypted keys
- It is simple but not flexible: how to deal with a query that selects users with salary <70K\$?
- Any idea? Encryption produces almost a random number.
1. We add salary in clear text. 2. Give the encryption key to the server, but it's a problem. 3. Send entire table to the query processor. In any case all of these solutions are bad: either for security or performance.

Implementation of ranged queries

- Say that we have to store this record in a table:

$$B_i = (x_{i1} | x_{i2} | \dots | x_{iM})$$

- Let's consider its encryption with key k :

$$E(k, B_i) = E(k, (x_{i1} | x_{i2} | \dots | x_{iM}))$$

- Let's now assume that, for each attribute, we can define classes in which it may range

- For example: if x_{i2} is a salary, we can define ranges of salaries: $\{[0, 59K], [60K, 79K], \dots\}$, and hence x_{i2} would fit in one of those classes, denoted I_{i2}
- For example: if x_{i3} is a name, we can define ranges of names based on the first letter: $\{[A, B], [C, D], \dots\}$, and hence x_{i3} would fit in one of those classes, denoted I_{i3}

- We then may store in the DB the tuple:

$$[E(k, B_i), I_{i1}, I_{i2}, \dots, I_{iM}]$$

Database Encryption

Encrypted database example

| $E(k, B_1)$ | I_{11} | \dots | I_{1j} | \dots | I_{1M} |
|-------------------------------|-------------------------------|---------|-------------------------------|---------|-------------------------------|
| \cdot \cdot \cdot | \cdot \cdot \cdot | | \cdot \cdot \cdot | | \cdot \cdot \cdot |
| $E(k, B_i)$ | I_{i1} | \dots | I_{ij} | \dots | I_{iM} |
| \cdot \cdot \cdot | \cdot \cdot \cdot | | \cdot \cdot \cdot | | \cdot \cdot \cdot |
| $E(k, B_N)$ | I_{N1} | \dots | I_{Nj} | \dots | I_{NM} |

Where $B_i = (x_{i1} | x_{i2} | \dots | x_{iM})$

Encrypted database example

Employee table

| Eid | Ename | Salary | Addr | Did |
|-----|-------|--------|----------|-----|
| 23 | Tom | 70K | Maple | 45 |
| 860 | Mary | 60K | Main | 83 |
| 320 | John | 50K | River | 50 |
| 875 | Jerry | 55K | Hopewell | 92 |

Encrypted employee table with indexes

| $E(k, B)$ | I(Eid) | I(Ename) | I(salary) | I(Addr) | I(Did) |
|--------------------------|--------|----------|-----------|---------|--------|
| 110100111101000011010... | 1 | 10 | 3 | 7 | 4 |
| 111010100100010111010... | 5 | 7 | 2 | 7 | 8 |
| 000001110100110101001... | 2 | 5 | 1 | 9 | 5 |
| 100111110111010000101... | 5 | 5 | 2 | 4 | 9 |

Database Encryption

You query for the rows that have a specific notation.

- A query looking for salaries <70K would look then for all records in which:
 $I_{i1} \in \{[0,59K], [60K, 79K]\}$
- The result would include also records with salary in [70,80], which can be easily filtered out after decryption

- Similarly, a query looking for employees from Abram to Elizabeth would look for all records in which:
 $I_{i3} \in \{[A,B], [C,D], [E,F]\}$
- The result would include also records with names before Abram and beyond Elizabeth, that can also be easily filtered out after the decryption

Database Encryption

Some considerations about the use of ranges of values:

- The range queries are less selective : the client obtains more data than necessary. After decryption it has to filter out to return the result to the user
- A potential vulnerability (for example due to the implicit ordering of the ranges) *with how many employees have the max salary, min salary ecc.*
 - What kind of inference may be possible?
 - Countermeasures? *Randomise the classes associations. In the metadata table we need the right association*
- Metadata: should include the association ranges values on the client
- Can still be combined with a separate encrypted primary key for selective queries
- Can make use of different keys for different parts of the DB to enforce RBAC

Question



Consider this encrypted table.

Would it be possible to increase the strength of the encryption by using a salt?

Employee Table

| Ename | Did | Salarycode | Eid | Ephone |
|---------|-----|------------|------|------------|
| Robin | 15 | 23 | 2345 | 6127092485 |
| Neil | 13 | 12 | 5088 | 6127092246 |
| Jasmine | 4 | 26 | 7712 | 6127099348 |
| Cody | 15 | 22 | 9664 | 6127093148 |
| Holly | 8 | 23 | 3054 | 6127092729 |
| Robin | 8 | 24 | 2976 | 6127091945 |
| Smith | 9 | 21 | 4490 | 6127099380 |

Encrypted_employee Table

| Encrypted_rows | Did |
|-----------------------------|------------------|
| 110100111110101000011010... | 101011010001111 |
| 10111010100100010111010... | 1000110111001110 |
| 001010001110100110101001... | 0000101110011000 |
| 10101011111011010000101... | 1010111010001111 |
| 110101010100111000100101... | 1110100001010101 |
| 001000110010101000010100... | 1110100001010101 |
| 010110101000011001001010... | 1101111110010111 |

With salt it becomes difficult to implement a query in an efficient way.

Datacenter security

Attacks directed to destruction, threat to confidentiality, availability... They are apparently

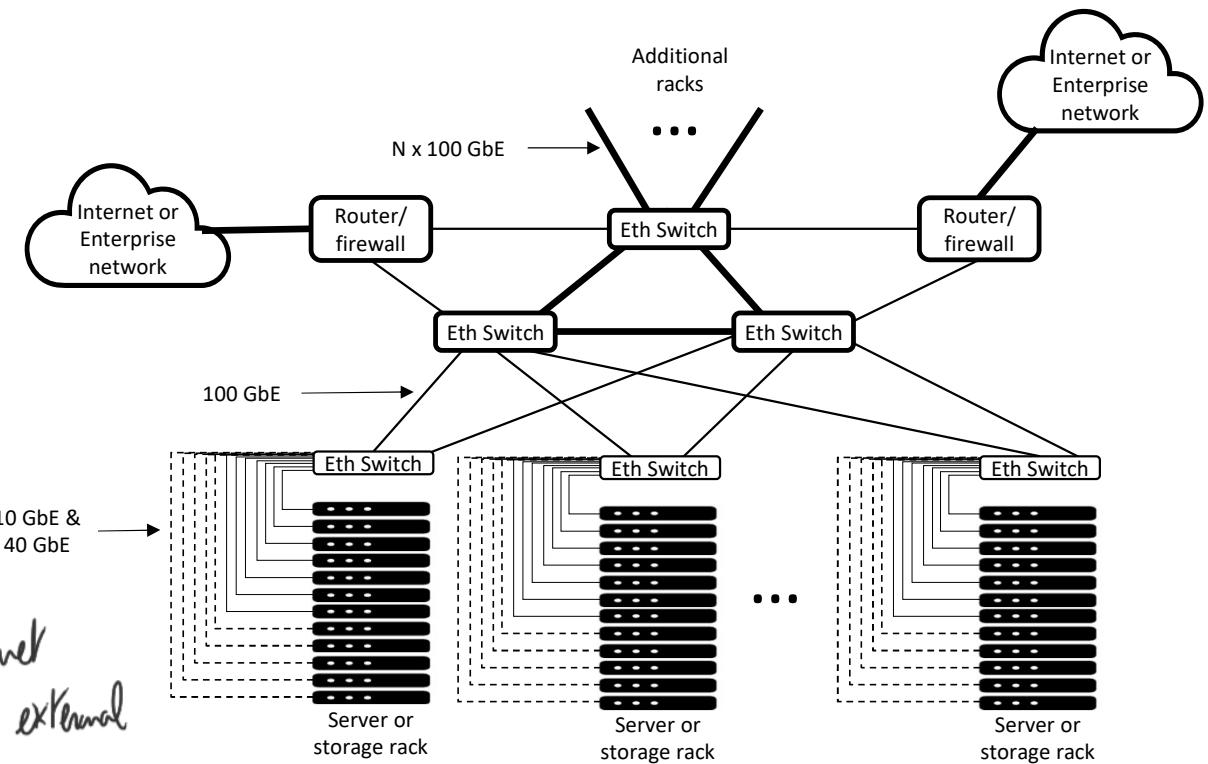
Data Center Security

- Data center:

- An enterprise facility that houses a large number of servers, storage devices, and network switches and equipment
- The number of servers and storage devices can run into the tens of thousands in one facility
- Generally, includes redundant or backup power supplies, redundant network connections, environmental controls, and various security devices
- Can occupy one room of a building, one or more floors, or an entire building
- Examples of uses include:
 - Cloud service providers
 - Search engines
 - Large scientific research facilities
 - IT facilities for large enterprises

Key data center elements

Allocated servers in racks organised in rows connected to powerful ethernet networks that communicate with the external world.



Data center security

- The servers are physically close, so get physical access is very appealing.
- Consider that the data center houses massive amounts of data that are:
 - located in a confined physical space.
 - interconnected with direct-connect cabling.
 - accessible through external network connections, so once past the boundary, a threat is posed to the entire complex.
 - typically, representative of the greatest single asset of the enterprise.
 - Thus, data center security is a top priority for any enterprise with a large data center. Some important threats are:
 - Denial of service
 - Advanced persistent threats from targeted attacks
 - Privacy breaches
 - Application exploits such as SQL injection
 - Malware
 - Physical security threats

Data center security model

| | |
|--|--|
| Data security | Encryption, Password policy, secure IDs, Data Protection (ISO 27002), Data masking, Data retention, etc. |
| Network security | Firewalls, Anti-virus, Intrusion detection/prevention, authentication, etc. |
| Physical security <i>Security of the entire site and single rooms.</i> | Surveillance, Mantraps, Two/three factor authentication, Security zones, ISO 27001/27002, etc. |
| Site security | Setbacks, Redundant utilities, Landscaping, Buffer zones, Crash barriers, Entry points, etc. |

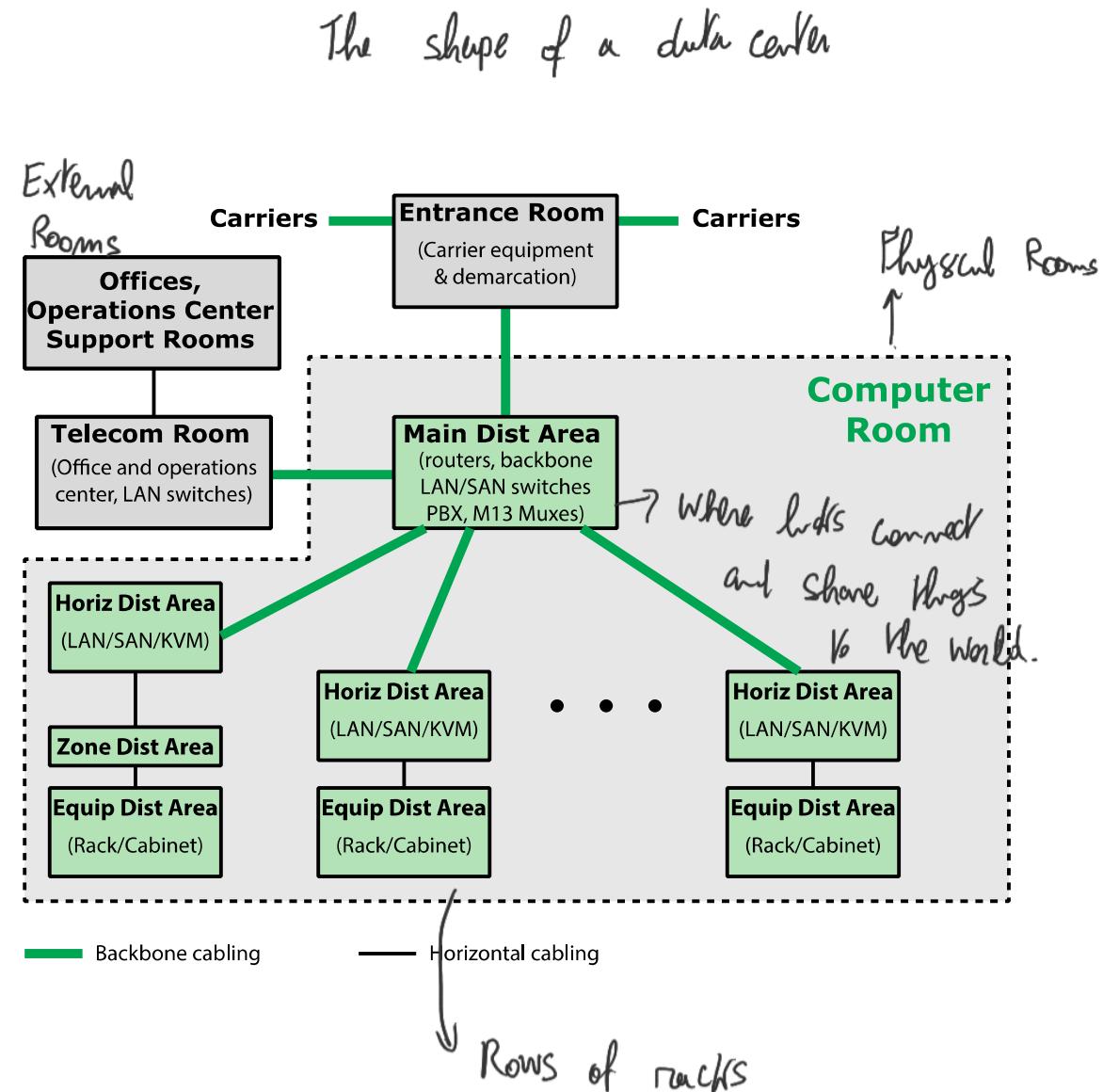
TIA-942

Standards by

- The Telecommunications Industry Association (TIA)
- TIA-942 (*Telecommunications Infrastructure Standard for Data Centers*) specifies the minimum requirements for telecommunications infrastructure of data centers
- Includes topics such as:
 - Network architecture
 - Electrical design
 - File storage, backup, and archiving
 - System redundancy
 - Network access control and security
 - Database management
 - Web hosting
 - Application hosting
 - Content distribution
 - Environmental control
 - Protection against physical hazards
 - Power management

↓
are very wide and include
things that go towards the
electrical parts too.

TIA-942 compliant data center showing key functional areas



May accept

From lowest to highest: objective is increasing
of Availability

Difference between lvl 1-4

is small but the difference
in investment is huge.

Data center tiers defined in TIA-942

| Tier | System design | Annual downtime |
|------|---|-------------------------|
| 1 | <ul style="list-style-type: none">Susceptible to disruptions from both planned and unplanned activitySingle path for power and cooling distribution, no redundant componentsMay or may not have raised floor, UPS, or generatorTakes 3 months to implementMust be shut down completely to perform preventive maintenance | 99.671% / 28.8 hours |
| 2 | <ul style="list-style-type: none">Less susceptible to disruptions from both planned and unplanned activitySingle path for power and cooling distribution, includes redundant componentsIncludes raised floor, UPS, and generatorTakes 3 to 6 months to implementMaintenance of power path and other parts of the infrastructure require a processing shutdown | 99.741% / 22.0 hours |
| 3 | <ul style="list-style-type: none">Enables planned activity without disrupting computer hardware operation but unplanned events will still cause disruptionMultiple power and cooling distribution paths but with only one path active, includes redundant componentsTakes 15 to 20 months to implementIncludes raised floor and sufficient capacity and distribution to carry load on one path while performing maintenance on the other | 99.982% / 1.6 hours |
| 4 | <ul style="list-style-type: none">Planned activity does not disrupt critical load and data center can sustain at least one worst-case unplanned event with no critical load impact even with one major breach. The data center can keep providing service without apparent impact.Multiple active power and cooling distribution paths, includes redundant componentsTakes 15 to 20 months to implement | 99.995% / 0.4 hours |

Summary

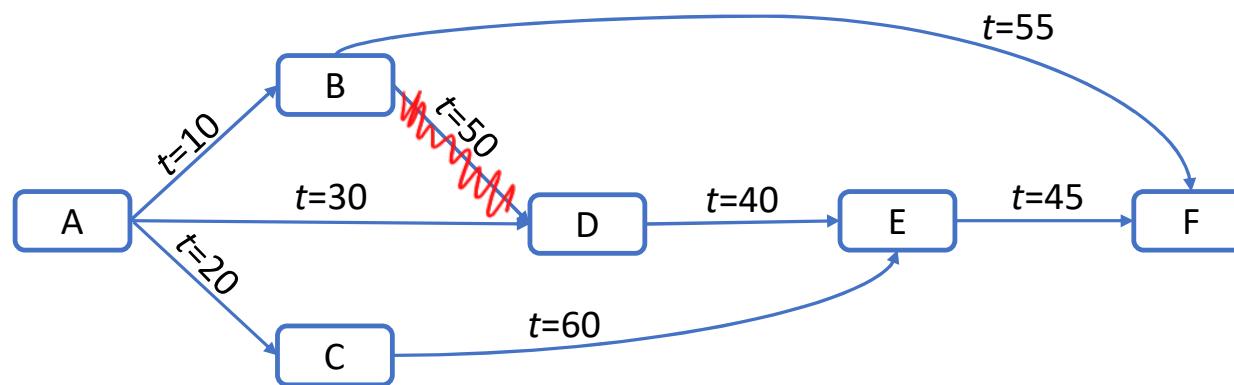
- Need for database security
- Database management systems
- Relational databases
 - Elements of a relational database system
 - Structured Query Language
- SQL injection attacks
 - A typical SQLi attack
 - The injection technique
 - SQLi attack avenues and types
 - SQLi countermeasures
- Database access control
 - SQL-based access definition
 - Cascading authorizations
 - Role-based access control
- Inference
- Database encryption
- Data center security
 - Data center elements
 - Data center security considerations
 - TIA-492

Exercise 1

Consider the sequence of grant operations for a specific access right on a table.

Show the resulting diagram of access right dependencies in the following cases (in alternative):

1. at $t = 70$, B revokes the access right from D
2. at $t = 70$, A revokes the access right from C
3. at $t = 70$, A revokes the access right from D

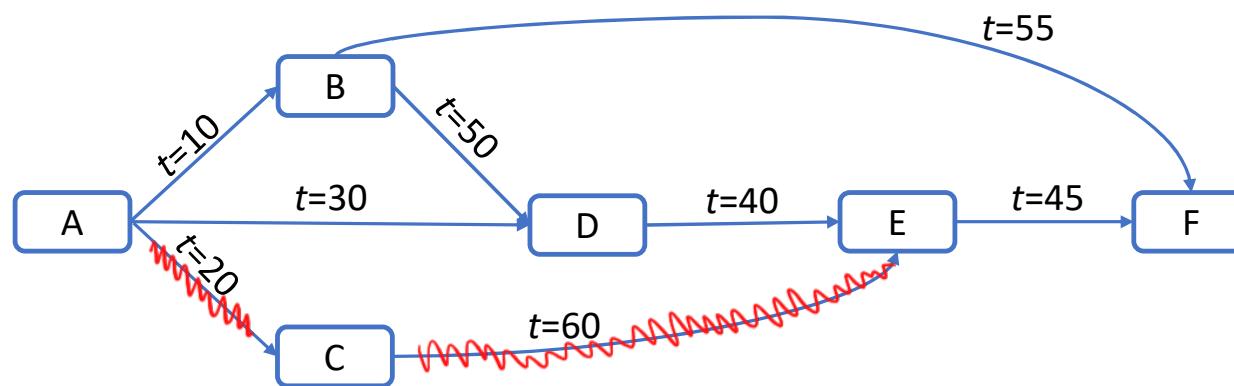


Exercise 1

Consider the sequence of grant operations for a specific access right on a table.

Show the resulting diagram of access right dependencies in the following cases (in alternative):

1. at $t = 70$, B revokes the access right from D
2. at $t = 70$, A revokes the access right from C
3. at $t = 70$, A revokes the access right from D

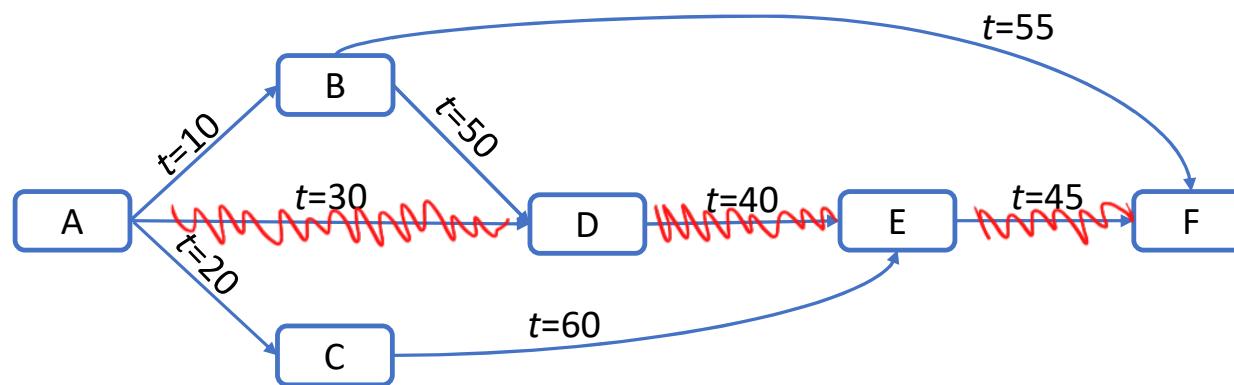


Exercise 1

Consider the sequence of grant operations for a specific access right on a table.

Show the resulting diagram of access right dependencies in the following cases (in alternative):

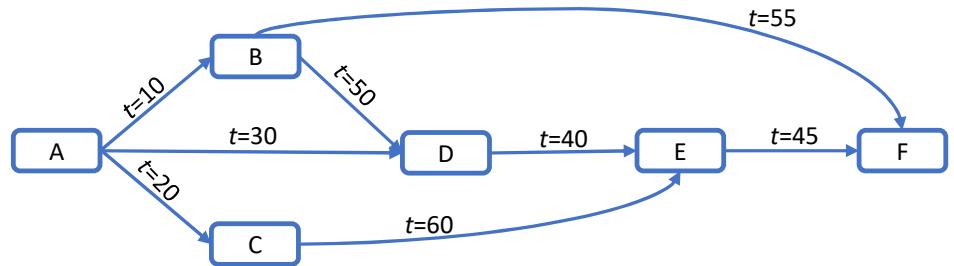
1. at $t = 70$, B revokes the access right from D
2. at $t = 70$, A revokes the access right from C
3. at $t = 70$, A revokes the access right from D





Solution 1.1

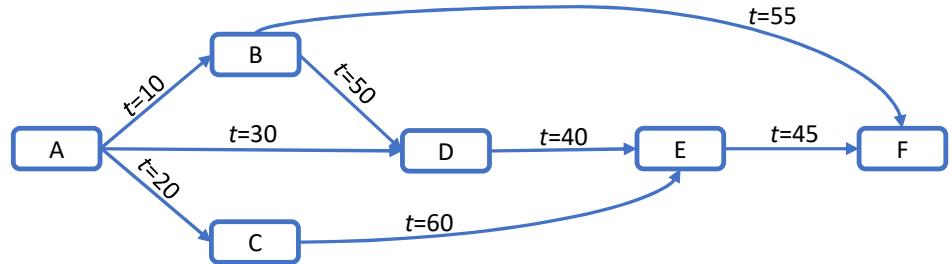
1. at $t = 70$, B revokes the access right from D





Solution 1.2

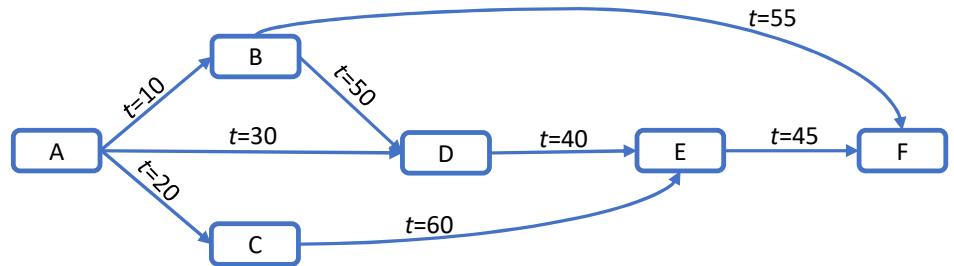
2. at $t = 70$, A revokes the access right from C





Solution 1.3

3. at $t = 70$, A revokes the access right from D



Exercise 2

Some DBMS system allow the use of a `sleep (x)` command in the WHERE clause of a SQL query (that's the case of MySQL for example). The sleep command introduces a delay of x seconds in the execution of the query.

For example, consider a table Employee, the following query:

```
SELECT name FROM Employee WHERE id=525 AND sleep(1)
```

Returns the name of the employee with `id=525` with a delay of 1 second.

Assume that direct queries that return the number of records in the Employee table is forbidden, as any query that reports the entire content of the table.

... but, for example, it is admitted a selective query:

```
SELECT name FROM Employee WHERE id=525
```

Devise a strategy (and a query) to infer the number of records in the table.

Exercise 2

Some DBMS system allow the use of a `sleep(x)` command in a SQL query (that's the case of MySQL for example). The sleep command introduces a delay of x seconds in the execution of the query. Note that `sleep(x)` returns 0 (false) unless it is interrupted.

For example, consider a table Employee, the following query:

```
SELECT name FROM Employee WHERE id=525 AND sleep(1000)
```

Terminates after 1 second (assuming id is a primary key): no need to evaluate `sleep()` if id is not 525...

Assume that, through a web interface, a query like the following can be executed:

```
SELECT * FROM Employee WHERE attr='xxxx'
```

Devise a strategy (and a query) to infer the number of records in the table, by injecting a piggybacked query (assume that the string `xxxx` is taken from an input).

Solution 2

`sleep(x)` command in a SQL query: introduces a delay of x seconds in the execution of the query.

Consider a table Employee, the following query:

```
SELECT name FROM Employee WHERE id=525 AND sleep(1000)
```

Terminates after 1 second (assuming id is a primary key)

Assume that, through a web interface, a query like the following can be executed:

```
SELECT * FROM Employee WHERE attr='xxxx'
```

Devise a strategy (based on query injection) to infer the number of records in the table.

You can inject this string: "abc' AND 1=2 OR sleep(10)" thus the query would become:

```
SELECT * FROM Employee WHERE attr='abc' AND 1=2 OR sleep(10)
```

Assume there are 5000 rows in a table, the pause of 10 milliseconds in the WHERE clause will result in a 50 seconds delay before the server response is sent.

If the table had only 300 records, then only 3 seconds would be added to the query execution time.

Note the use of a false condition (`1=2`). This is necessary to force SQL to compute the `sleep(x)` command at each row of the table. Without it the `sleep` command may be executed only when the first test (`attr='abc'`) is false.



Solution 2

`sleep(x)` command in the WHERE clause of a SQL query: introduces a delay of x seconds in the execution of the query.

Consider a table Employee, the following query:

```
SELECT name FROM Employee WHERE id=525 AND sleep(1)
```

Returns the name of the employee with `id=525` with a delay of 1 second.

Assume that a direct query that returns the number of records in the Employee table is forbidden.

Devise a strategy (and a query) to infer the number of records in the table.

Exercise 3

Consider the table *Attacked_companies* kept by an anti-virus company that contains information about the attacks conducted against client companies.

The table has the columns:

attackID, company_name, attack_code, date_of_attack, severity_level, damage.

The typical query executed on this table is:

```
SELECT attackID, company_name, date_of_attack, attack_code  
FROM Attacked_companies  
WHERE severity_level > X AND damage > Y
```

Where the *severity_level* ranges in [1,100] and damage is expressed in €. On the other hand, the query aims at finding the order of magnitude of the damage, hence Y can take the values 1K, 10K, 100K, 1M, 10M, >100M.

Design the encrypted table *E_Attacked_companies* to store this information, using key *k*.

= Ranges of severity-level = $\{(0, 10], (10, 20], (20, 30], \dots, (90, 100]\}$

8 6 5 ... 3

Ranges of damage = $\{(0, 10K], (10K, 1M], (10M, +\infty)\}$

0 2 1

E_Attacked_companies

| $E_encrypted_rows$ | $I(\text{security-level})$ | $I(\text{damage})$ |
|----------------------|----------------------------|--------------------|
| 010000111000... | 0 | 8 |
| 111111100000... | 0 | 1 |
| 0001011111001... | 2 | 7 |

SELECT $E_encrypted_rows$ FROM E_Attacked_companies

WHERE $I(\text{security-level}) \in (\{a_1, a_2, \dots\})$ AND
 $I(\text{damage}) \in (\{b_1, b_2, \dots\})$



Solution 3

table *Attacked_companies* the columns:

- *attackID, company_name, attack_code, date_of_attack, severity_level, damage.*

The typical query executed on this table is:

```
SELECT attackID, company_name, date_of_attack, attack_code  
FROM Attacked_companies  
WHERE severity_level > X AND damage > Y
```

severity_level ranges in [1,100]

damage is expressed in \$.

Y can take the values 1K, 10K, 100K, 1M, 10M, >100M.

Design the encrypted table *E_Attacked_companies* to store this information, using key *k*.

Exercise 3.B

Still considering the query:

```
SELECT attackID, company_name, date_of_attack, attack_code  
FROM Attacked_companies  
WHERE severity_level > X AND damage > Y
```

and the encrypted DB you designed in the previous step, transform the previous query into the query to be sent to the query executor over the encrypted DB