Introduction to Computer Security

Chapter 5: Database and Cloud Security

Chi-Yu Li (2019 Spring)
Computer Science Department
National Chiao Tung University

Needs for Database Security

- Reasons Why database security has not kept pace with the increased reliance on databases
  - Imbalance between the complexity of DBMS and security techniques
    - DBMS: complex, many new features and services
  - Sophisticated interaction protocol: SQL
    - Much more complex than HTTP
  - Mismatch between requirements and capabilities
    - Administrators: limited knowledge of security or limited understanding of DBMS
  - Heterogeneous mixture of database, enterprise, and OS platforms
    - Database: Oracle, IBM, Microsoft, etc.
    - Enterprise: Oracle E-Business Suite, Siebel, etc.
    - OS: UNIX, Linux, Windows, etc.

Outline

- Database Management Systems
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Cloud Computing
- Cloud Security Risks and Countermeasures
- Cloud Security as a Service

Database

- Structured collection of data stored for use by one or more apps
- Contains 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

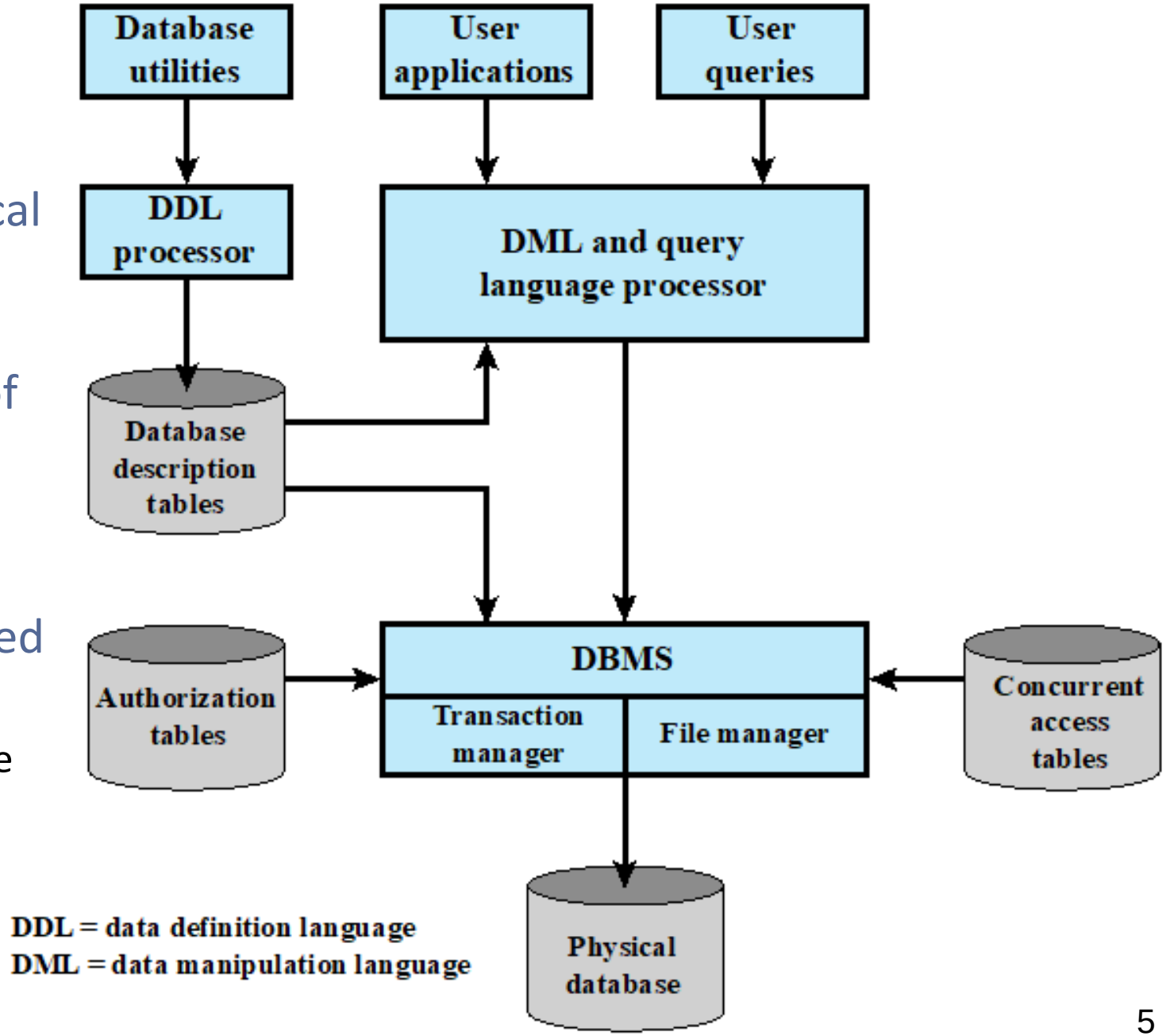
Database management system (DBMS)

- Suite of programs for constructing and maintaining the database
- Offers ad hoc query facilities to multiple users and applications

DBMS Architecture

- DDL: defines the database logical structure and procedural properties
- DML: provides a powerful set of tools for app developers

- Security requirements: beyond the capability of typical OS-based security **Why?**
  - OS: typically control read and write access to entire files
  - Database: ?

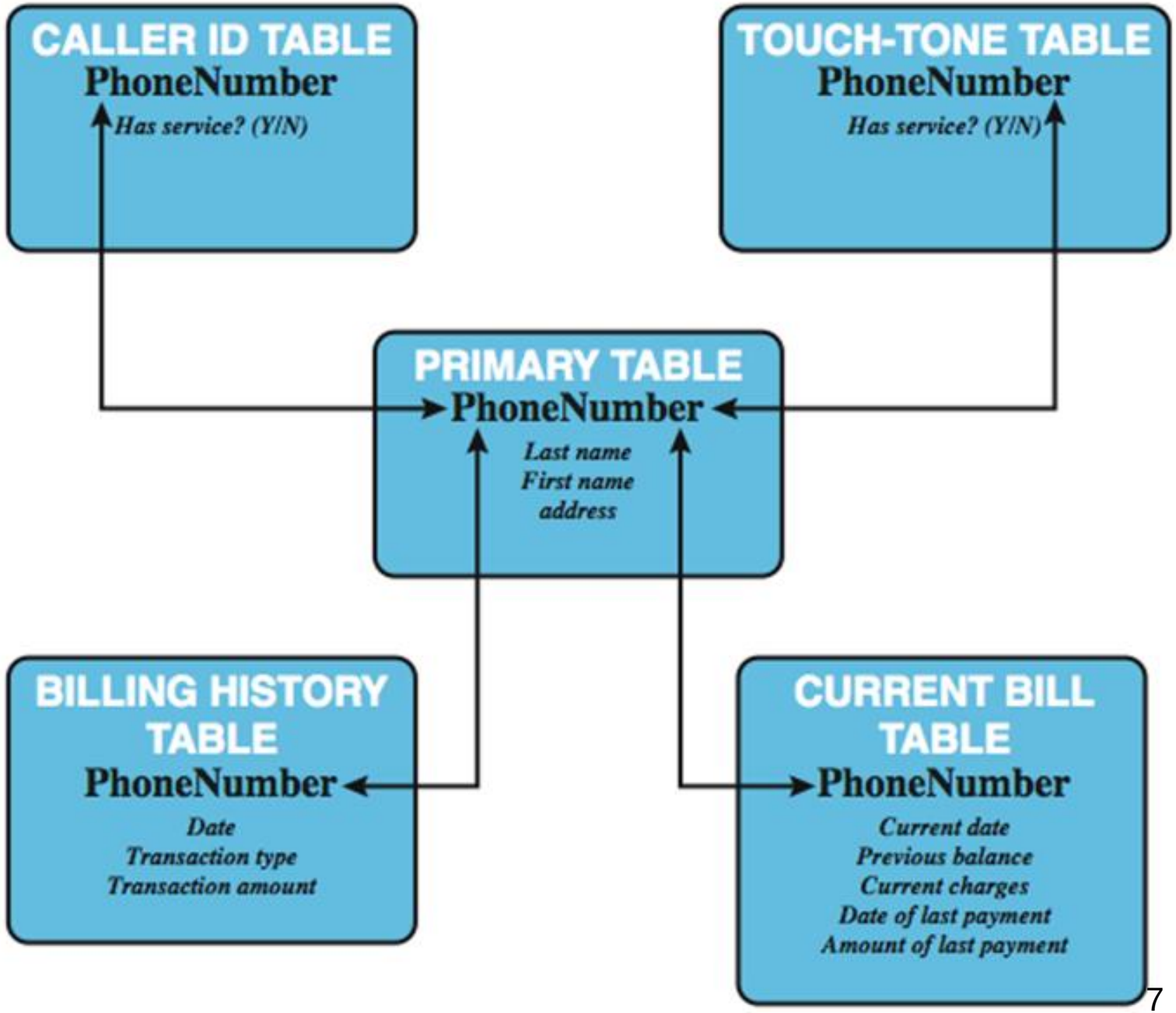


Relational Databases

- Constructed from tables of data
  - Each column holds a particular type of data
  - Each row contains a specific value of each column
  - Ideally has one column where all values are unique, forming an identifier/key for that row
    - Used as the primary key
- Have multiple tables linked by identifiers
- Use a query language to access data items meeting specified criteria

Relational Database Example

- Using multiple tables related to one another by a designated key
  - PhoneNumber serves as a primary key



Relational Database Elements

- Relation/table/file
- Tuple/row/record
- Attribute/column/field

Primary key

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

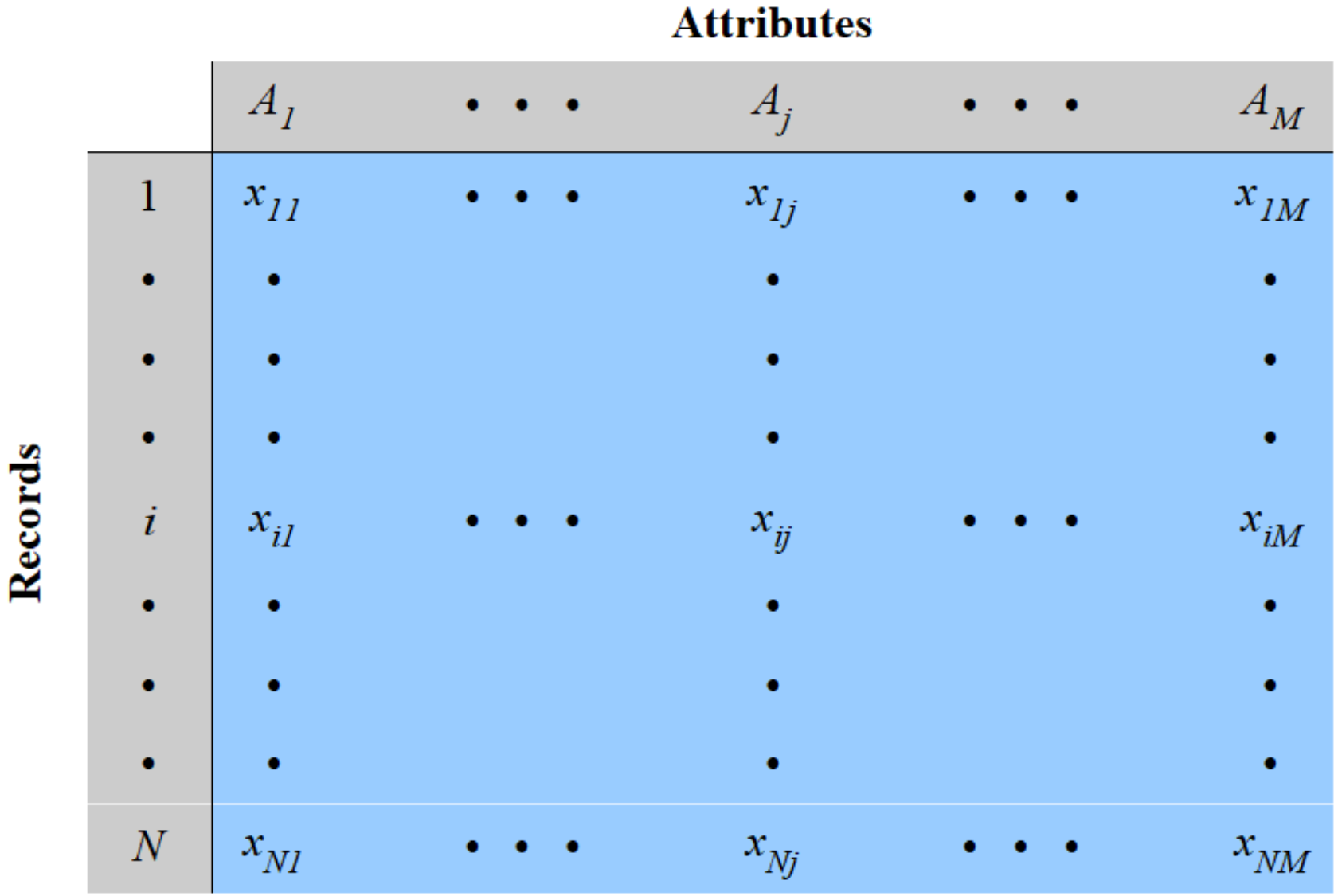
Foreign key

- Links one table to attributes in another

View/virtual table

- Result of a query that returns selected rows and columns from one or more tables

Abstract Model of a Relational Database



Relational Database Example

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

A view derived from the 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

Structured Query Language (SQL)

- Originally developed by IBM in the mid-1970s
- Standardized language to define, 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

SQL Injection Attacks (SQLi)

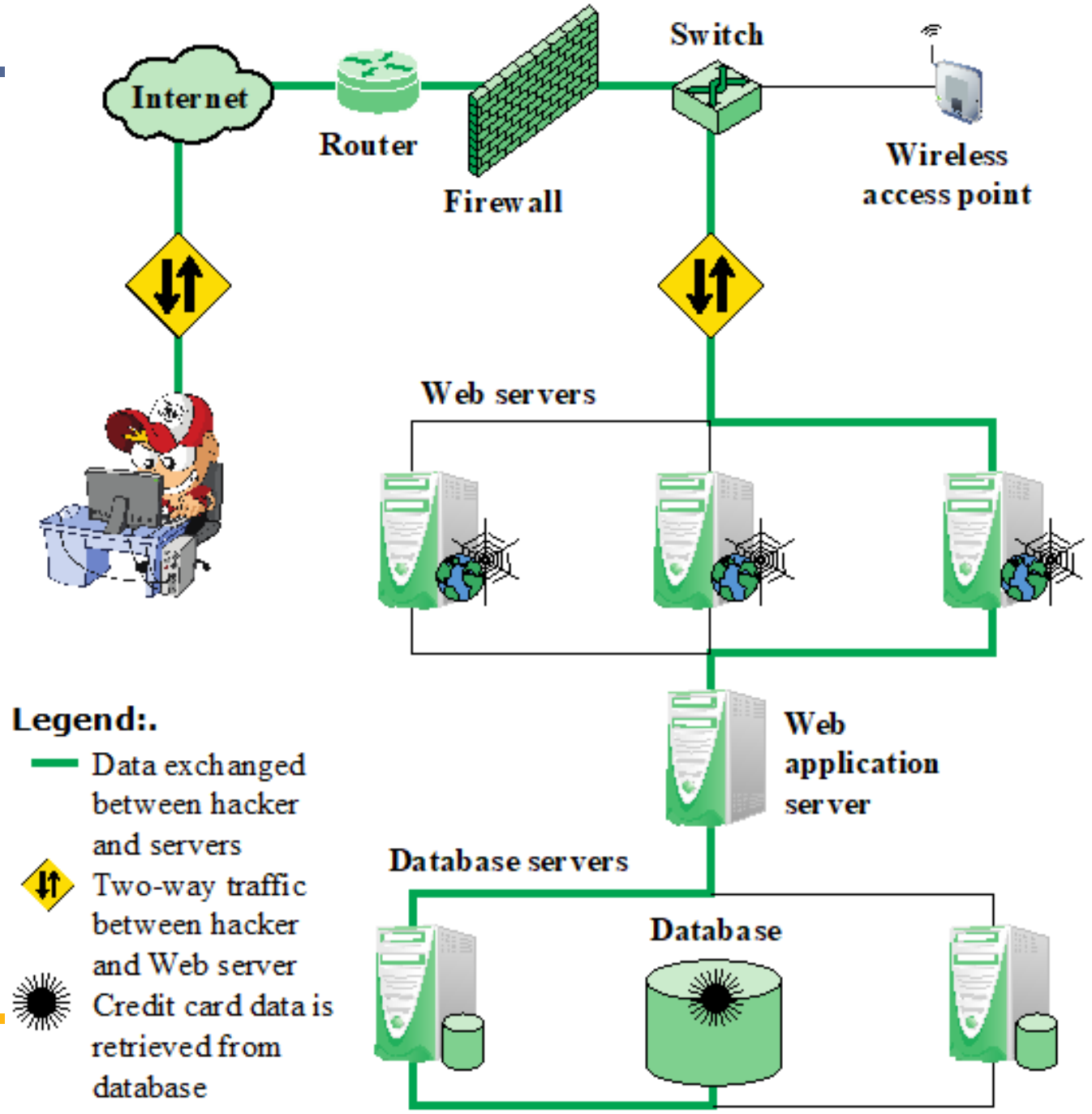
- One of the most prevalent and dangerous network-based security threats
  - Imperva Web Application Attack Report, July 2013 [IMPE13]
    - SQLi attacks ranked 1st or 2nd in total number of attack incidents, …
    - Observed that a single Web site: 94,057 SQL injection attack requests in one day
  - Open Web Application Security Project's 2013 report [OWAS13]
    - Top risk from ten most critical Web app security risks
  - Veracode 2013 State of Software Security Report [VERA13]
    - 32% apps affected by SQLi attacks
    - Account for 26% of all reported breaches
  - Trustwave 2013 Global Security Report [TRUS13]
    - One of the top two intrusion techniques
    - Poor coding practices → SQLi attacks remain more than 15 years

SQLi (Cont.)

- Designed to exploit the nature of Web app pages
- Sends malicious SQL commands to the database server
- Most common attack goal is bulk extraction of data
- Depending on the environment, SQLi can also be exploited to
  - Modify or delete data
  - Execute arbitrary OS commands
  - Launch DoS attacks

Typical SQL Injection Attack Scenario

- Hacker injects an SQL command to a database: sending the command to the Web server
  - Database server executes the malicious command and returns data
  - Web app dynamically generates a page with the data



Injection Technique

- Typical SQLi attacks: prematurely terminating a text string and appending a new command
  - Comment mark "--": subsequent text is ignored at execution time
  - e.g., consider a script that build 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 + "'";
```

How to launch an SQLi attack with "--"?

SQLi Attack Avenues

- User input
  - Injects SQL commands by providing suitably crafted user input
- Physical user input
  - Attacks outside the realm of Web requests
  - e.g., conventional barcodes and RFID tags
- Server variables (e.g., HTTP headers and network protocol headers)
- Second-order injection
- Cookies

SQLi Example I: from User Input

- Consider a general verification SQL command
  - strSQL = "SELECT * FROM users WHERE (name = '" + username + "') and (pw = '" + password + "');

How to bypass the verification check?

- Target: "SELECT * FROM users;"
- Try it online: https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_single_2

SQLi Example II: from Server Variables

- Web apps use the variables in a variety of ways, such as logging usage statistics
- An SQL injection vulnerability: they are logged to a database without sanitization
  - Attackers can forge the values that are placed in HTTP and network headers

- Example: the header of a request HTTP
  - How is it used for SQL?

```
"SELECT user.password FROM admins
WHERE
user='".sanitize($_POST['user']."'
AND
password='".md5($_POST['password'])."'
AND
ip_adr='".ip_adr()."'"
```

```
GET / HTTP/1.1
Connection: Keep-Alive
Keep-Alive: 300
Accept:*/*
Host: host
X_Forwarded_For: 140.113.117.23
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16 ( .NET CLR 3.5.30729;
.NET4.0E)
Cookie: guest_id=v1%3A1328019064; pid=v1%3A1328839311134
```

Used for identifying the originating IP address of the connected client

SQLi Example II: from Server Variables (Cont.)

- Example: the header of a request HTTP
  - How is it used for SQL?

```
"SELECT user.password FROM admins WHERE
user='".sanitize($_POST['user']."'
AND password='".md5($_POST['password'])."' AND
ip_adr='".ip_adr()."'"
```

HTTP_X_FORWARDED_FOR is not properly sanitized.

```
function ip_adr() {
  if (isset($_SERVER['HTTP_X_FORWARDED_FOR'])) {
     $ip_adr = $_SERVER['HTTP_X_FORWARDED_FOR']; }
  else { $ip_adr = $_SERVER["REMOTE_ADDR"]; }
  return $ip_adr;
}
```

How to launch an SQLi attack with "HTTP_X_FORWARDED_FOR"?

SQLi Example III: from Second-order Injection

- Relying on data already present in the system or database to trigger an SQLi attack
- Consider a Web-based app which stores usernames alongside other session information
- Given a session identifier such as a cookie
  - Seek to retrieve the current username and use it in turn to receive SSN

```
“SELECT username FROM sessiontable WHERE session=‘”$_POST[‘sessionid’].”’”

“SELECT ssn FROM users WHERE username=‘”$_POST[‘username’].”’”
```

How to launch an SQLi attack with existing data?

Three Categories of SQLi Attacks

- In-band attacks
  - use the same communication channel for injecting SQL codes and retrieving results
- Out-of-band attacks
  - use a different channel
- Inferential attacks
  - No actual transfer of data, but reconstructing the information by sending particular request and observing results

In-band Attacks

- The retrieved data are presented directly in application web pages

Tautology

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

Out-of-band Attacks

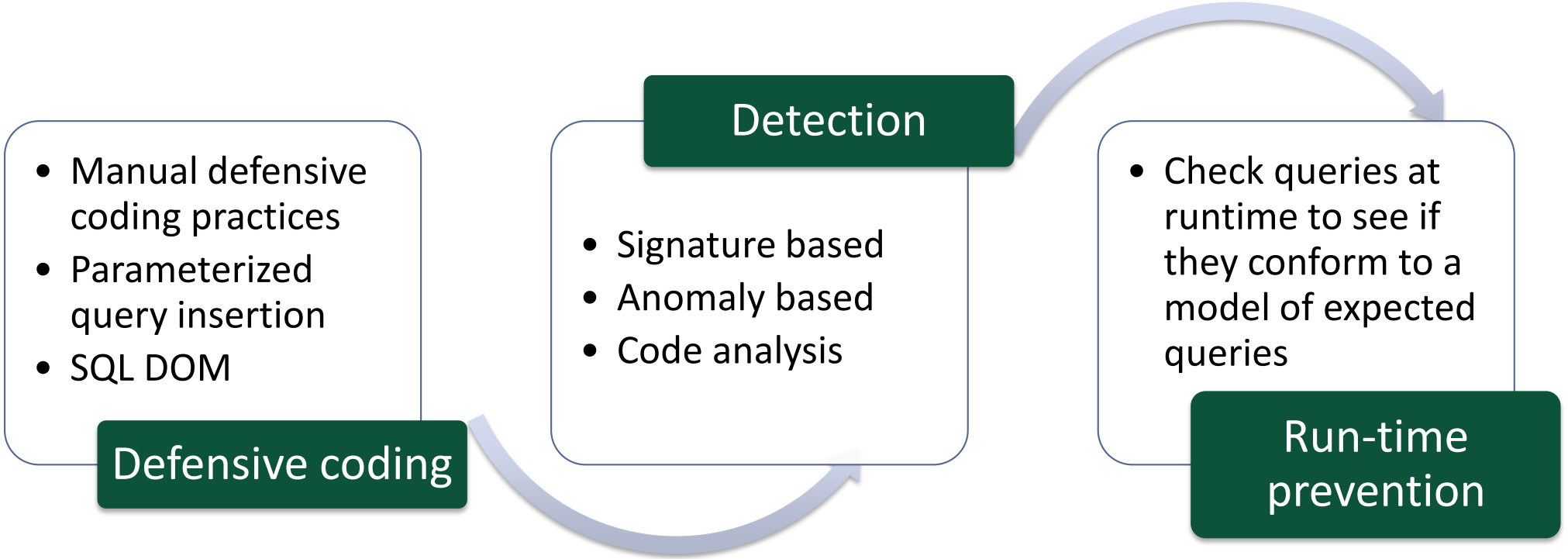
- Data are retrieved using a different channel, e.g., email instead of web pages

- Used when there are limitations on information retrieval
  - But, outbound connectivity from the data server is lax

Inferential Attacks

- Reconstruct the information by sending particular requests and observing the resulting behavior of the Website/database server
  - Illegal/logically incorrect queries
    - Default error page is overly descriptive
    - Collect important information about the type and structure of the backend database of a Web application
    - Considered as a preliminary, information-gathering step for other attacks
  - Blind SQL injection
    - Like asking true/false questions
    - Infers the data present in a database system, even when the system is sufficiently secure to not display any erroneous information back to the attacker

SQLi Countermeasures



DOM: Domain Object Model
Signature based: signatures of attack patterns
Anomaly based: differ from normal behaviors

SQLi Countermeasures

- Manual defensive coding practices
- Parameterized query insertion
- SQL DOM -codegen

Defensive coding

```
using (SqlConnection conn = new SqlConnection(NorthwindConnectionString))
{
  string query = "SELECT * FROM Products WHERE ProductID = @Id";
 SqlCommand cmd = new SqlCommand(query, conn);
 cmd.Parameters.AddWithValue("@Id", Request.QueryString["Id"]);
 conn.Open();
 using (SqlDataReader rdr = cmd.ExecuteReader())
 {
    DetailsView1.DataSource = rdr;
    DetailsView1.DataBind();
 }
}
```

DOM: Domain Object Model
Signature based: signatures of attack patterns
Anomaly based: differ from normal behaviors

Outline

- Database Management Systems
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Cloud Computing
- Cloud Security Risks and Countermeasures
- Cloud Security as a Service

Database Access Control

- Assumption: users have been authenticated
  - They have access to the entire database or just portions of it
- Commercial and open-source DBMSs: DAC or RBAC

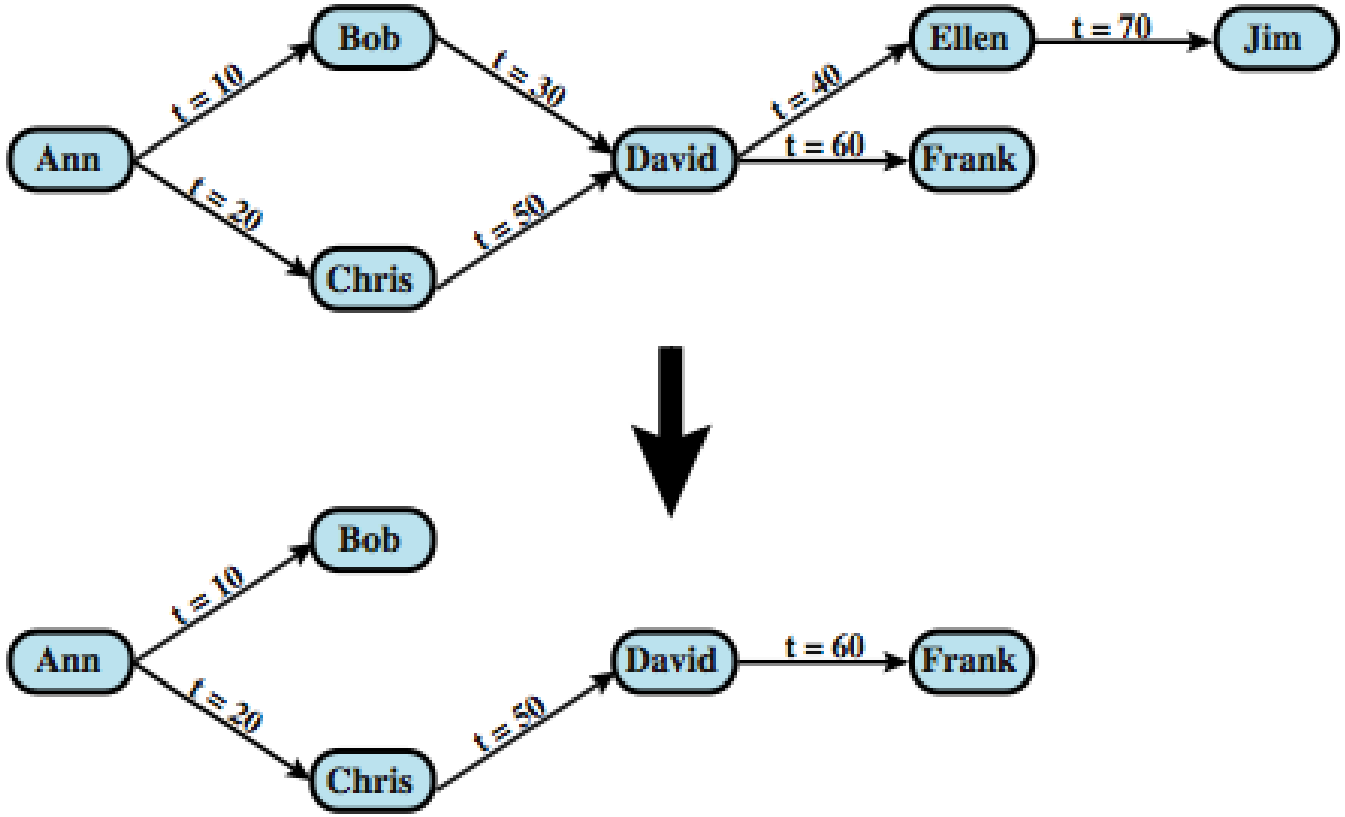
- Typically support a range of administrative policies
  - Centralized administration
    - Small number of privileged users may grant and revoke access rights
  - Ownership-based administration
    - A table's creator may grant and revoke access rights to the table
  - Decentralized administration
    - A table's owner may grant and revoke authorization rights to other users
    - Other users are allowed to grant and revoke access rights to the table

SQL Access Control

- Two commands for managing access rights:
  - `GRANT { privileges | role } [ON table] TO { user | role | PUBLIC } [IDENTIFIED BY password] [WITH GRANT OPTION]`
    - e.g. GRANT SELECT ON ANY TABLE TO Bob
  - `REVOKE { privileges | role } [ON table] FROM { user | role | PUBLIC }`
    - e.g. REVOKE SELECT ON ANY TABLE FROM Bob
- Typical access rights are:
  - `SELECT, INSERT, UPDATE, DELETE, REFERENCES`

Cascading Authorizations

- Grant/Revoke options: enable/disable an access right to cascade through a number of users
- Revoke 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



Role-Based Access Control (RBAC)

- RBAC eases administrative burden and improves security
- A database RBAC needs to provide the following capabilities
  - Create and delete roles
  - Define permissions for a role
  - Assign and cancel assignment of users to roles
- Categories of database users

Application owner	End user	Administrator
• An end user who owns database objects as part of an application	• An end user who operates on database objects via a particular application but does not own any of the database objects	• User who has administrative responsibility for part or all of the database

Example: Microsoft SQL Server

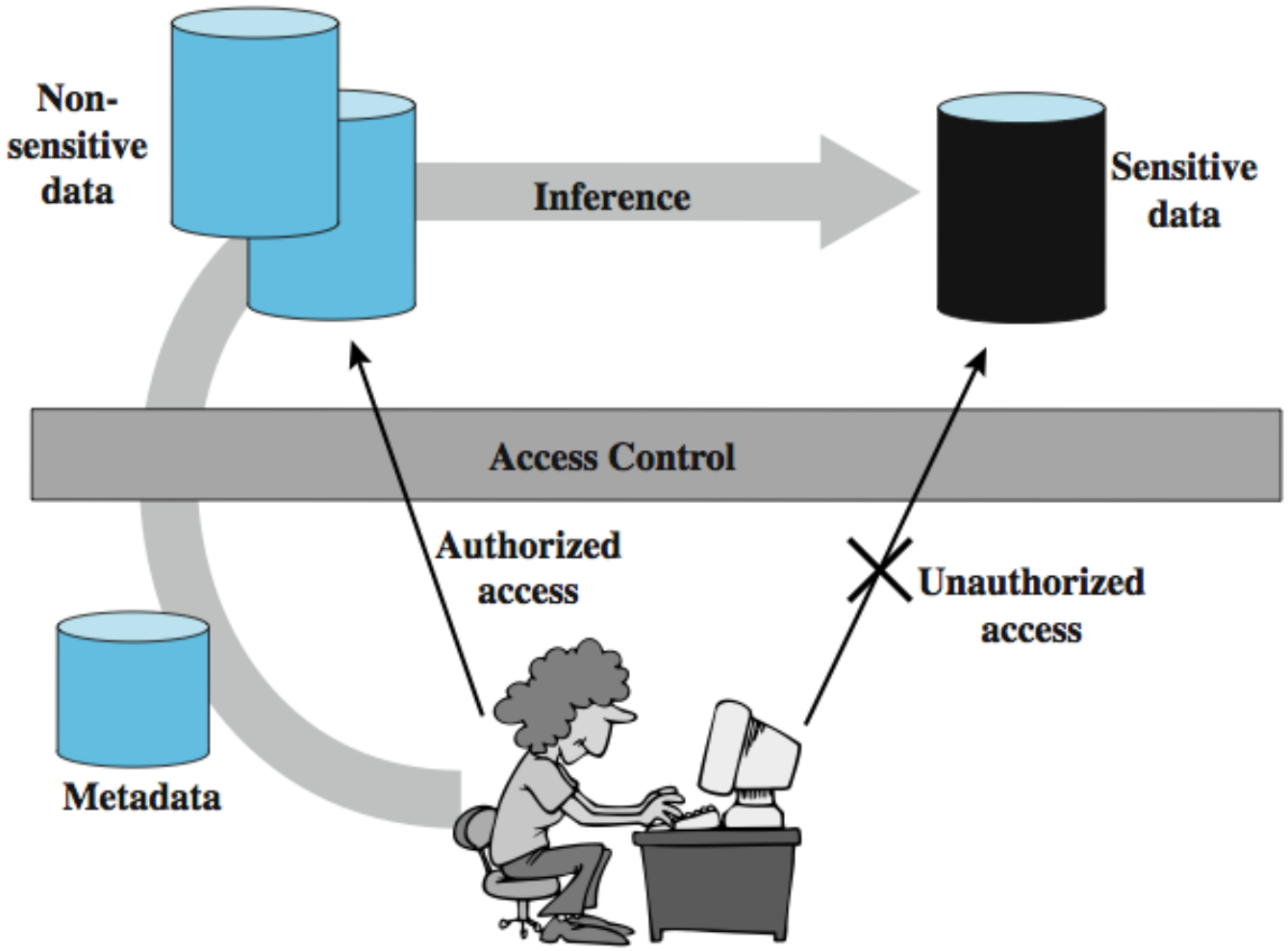
Role	Permissions
Fixed Server Roles	
sysadmin	Can perform any activity in SQL Server and have complete control over all database functions
serveradmin	Can set server-wide configuration options, shut down the server
setupadmin	Can manage linked servers and startup procedures
securityadmin	Can manage logins 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	
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 (DDL) 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

Inference

- The process of performing authorized queries and deducing unauthorized information from the legitimate responses
  - Combination of a number of data items: more sensitive than individual items

Metadata: knowledge about correlations or dependencies among data items

Inference channel: information transfer path by which unauthorized data is obtained



Example

- Users of the views are not authorized to access the relationship between Salary and Name
- However, it can be referred by the combination of the views
  - Knowledge of the table structure is needed

Name	Position	Salary ($)	Department	Dept. Manager
Andy	senior	43,000	strip	Cathy
Calvin	junior	35,000	strip	Cathy
Cathy	senior	48,000	strip	Cathy
Dennis	junior	38,000	panel	Herman
Herman	senior	55,000	panel	Herman
Ziggy	senior	67,000	panel	Herman

(a) Employee table

Position	Salary ($)
senior	43, 000
junior	35,000
senior	48,000

Name	Department
Andy	strip
Calvin	strip
Cathy	strip

(b) Two views

Name	Position	Salary ($)	Department
Andy	senior	43,000	strip
Calvin	junior	35,000	strip
Cathy	senior	48,000	strip

(c) Table derived from combining query answers

Inference Detection: Two Approaches

- Inference detection during database design
  - Removes an inference channel by altering the database structure
    - E.g., splitting a table into multiple tables or more fine-grained access control
  - Availability reduction: unnecessarily stricter access controls
- Inference detection at query time
  - Eliminate an inference channel violation during a query or series of queries
- For either of them, some inference detection algorithm is needed
  - Difficult problem and ongoing research

Example: Inference Problem and Solution

- Consider a database containing personnel information, including names, addresses, and salaries of employees
  - Clerk: name, address, and salary information
  - Administrator: name, address, salary information, and association of names/salaries

- Solution: construct three tables

Employees (Emp#, Name, Address)
Salaries (S#, Salary)
Emp-Salary (Emp#, S#)

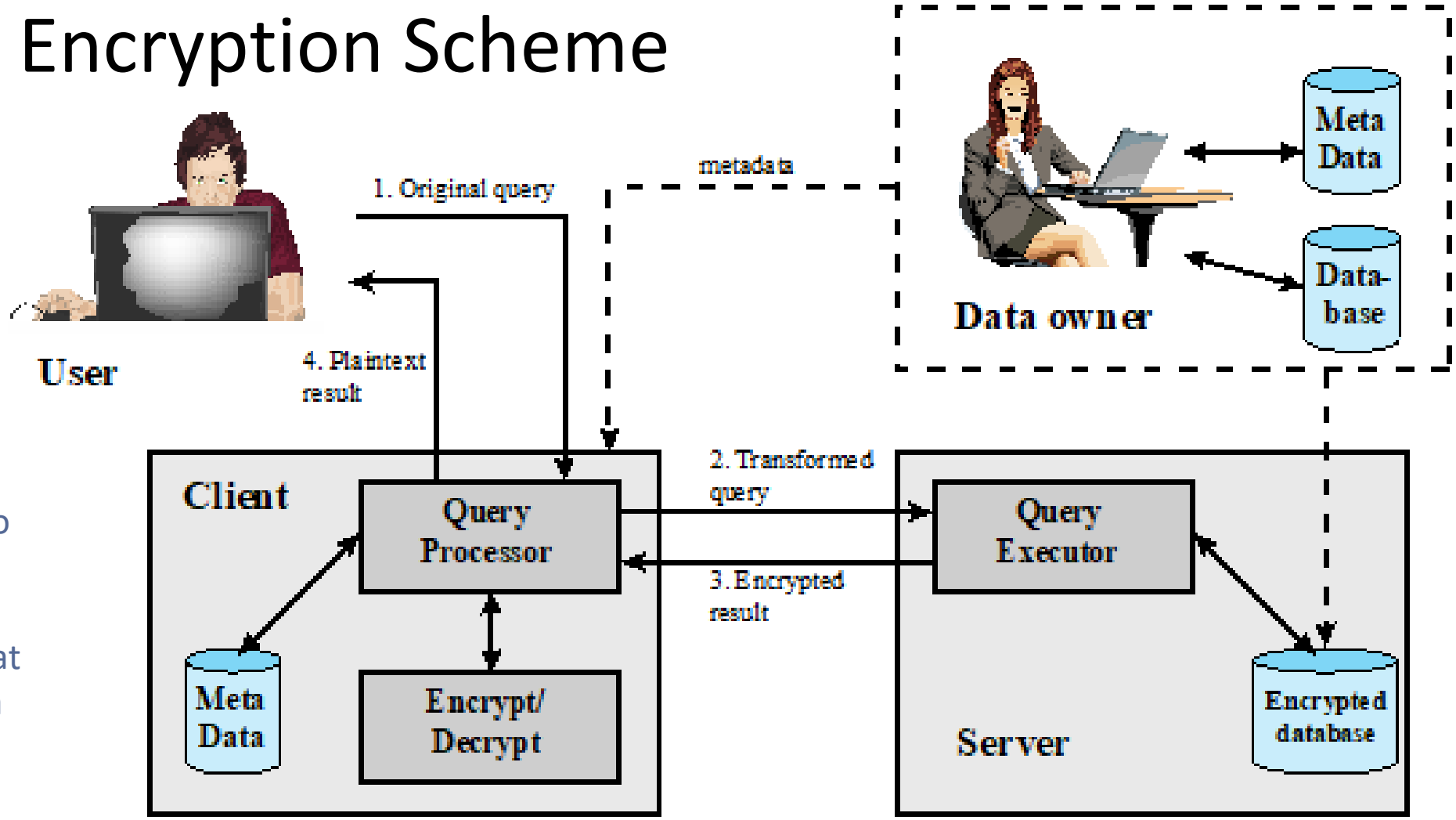
What if a new attribute, employee start date, is needed?
Where should it be added?

Database Encryption

- 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, etc.
    - Encryption becomes the last line of defense in database security
- Can be applied to the entire database, at the record level (rows), the attribute level (columns), or level of the individual field (specific fields)
- Disadvantages
  - Inflexibility: difficult to perform record searching
  - Key management: authorized users must have access to the decryption key for the data for which they have access

A Database Encryption Scheme

- **Data owner** – organization that produces data to be made available for controlled release
- **User** – human entity that presents queries to the system
- **Client** – frontend that transforms user queries into queries on the encrypted data stored on the server
- **Server** – an organization that receives the encrypted data from a data owner and makes them available for distribution to clients



Example for a Straightforward Approach

- Consider a query:

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

  - Assume the encryption key k is used
  - the encrypted value of the Did 15 is $E(k, 15) = 1000110111001110$

- The query processing at the client could transform the query to

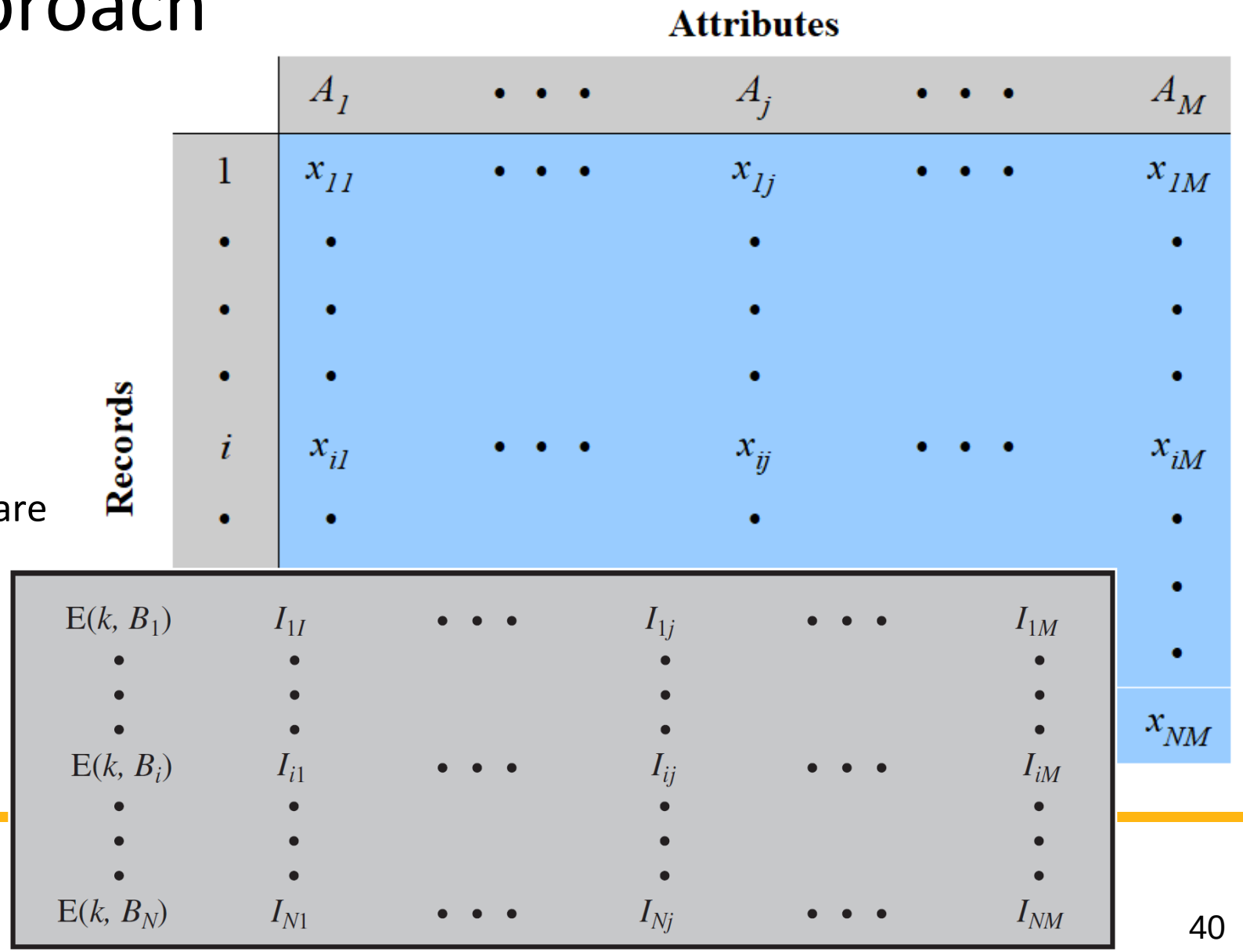
```
SELECT Ename, Eid, Ephone
        FROM Employee
        WHERE Did = 1000110111001110
```

It seems straightforward and sufficient, but what if the query is to retrieve all records with the Dids smaller than 100?

More Flexible Approach

- Each record (row) of a table is encrypted as a block
  - R_i: a continuous block $B_i = (x_{i1}||x_{i2}||\cdots||x_{iM})$
  - All of the attribute values: concatenated together to form a single binary block
  - Data retrieval: attribute indexes are associated with each table

How to assign indexes?
Can we prevent the access of the whole records for the query?



More Flexible Approach (Cont.)

- For any attribute, the range of attribute values is divided into a set of non-overlapping partitions
  - e.g., employee ID (eid): [1, 1000]
    - Index 1: [1, 200], 2: [201, 400], 3: [401, 600], 4: [601, 800], 5: [801, 1000]
  - Consider that a request: all employees with $eid < 300$

(a) 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

(b) Encrypted Employee Table with Indexes

E(k, B)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100110011001011…	1	10	3	7	4
0111000111001010…	5	7	2	7	8
1100010010001101…	2	5	1	9	5
0011010011111101…	5	5	2	4	9

Outline

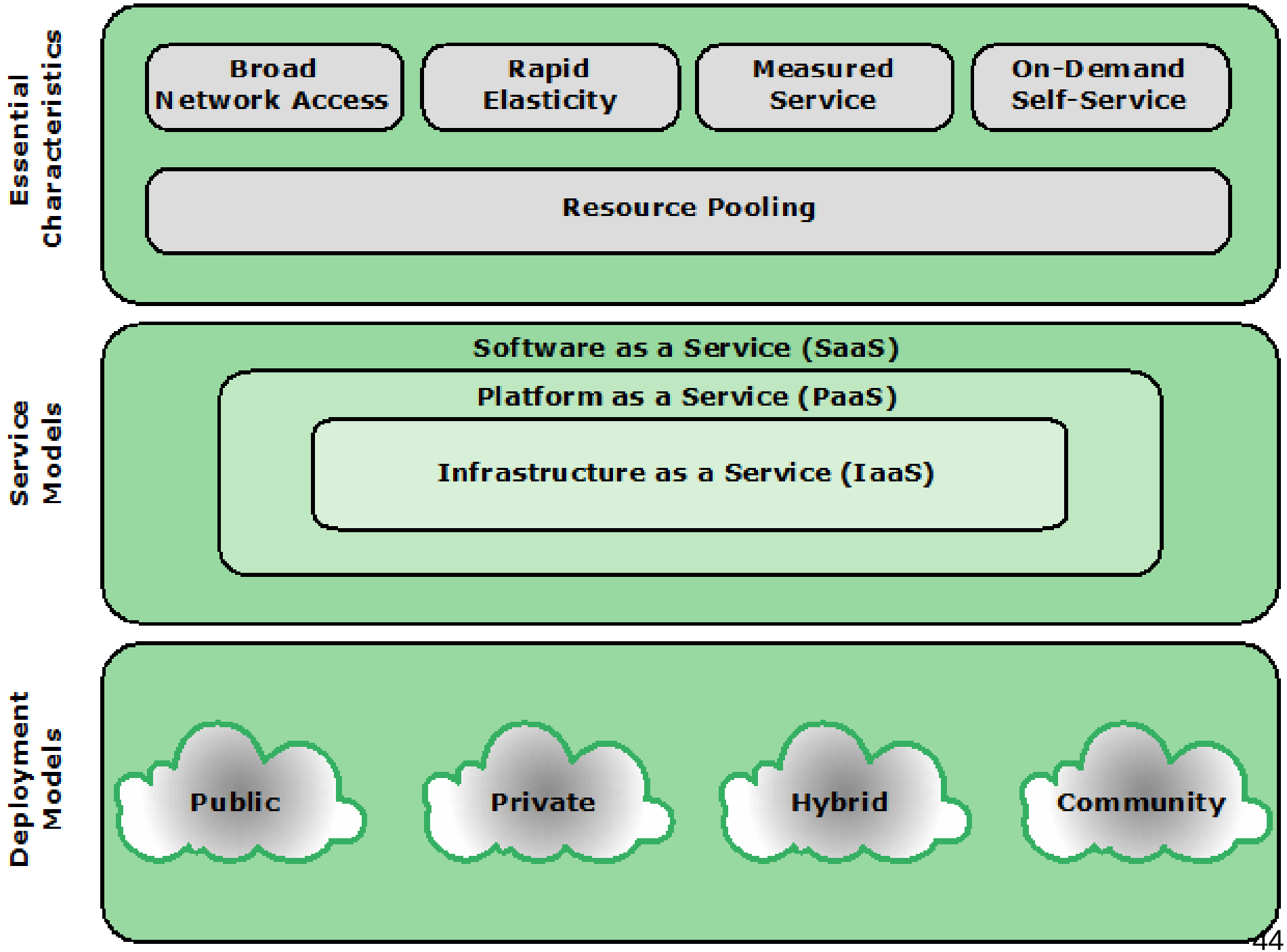
- Database Management Systems
- Relational Databases
- SQL Injection Attacks
- Database Access Control
- Inference
- Database Encryption
- Cloud Computing
- Cloud Security Risks and Countermeasures
- Cloud Security as a Service

Cloud Security

- NIST SP-800-145 defines cloud computing as:

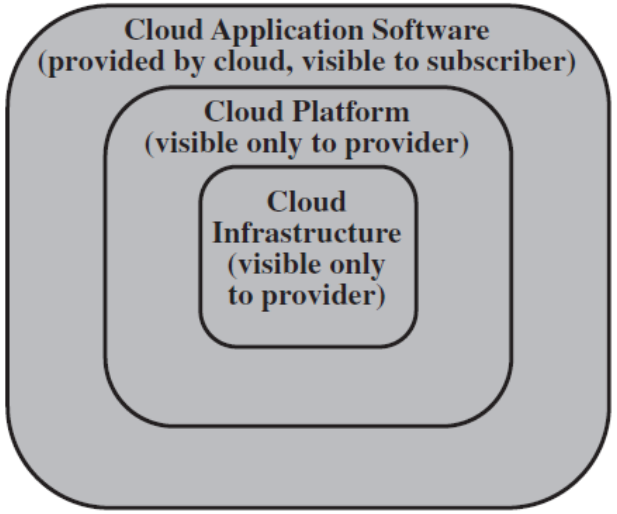
"A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Cloud Computing Elements

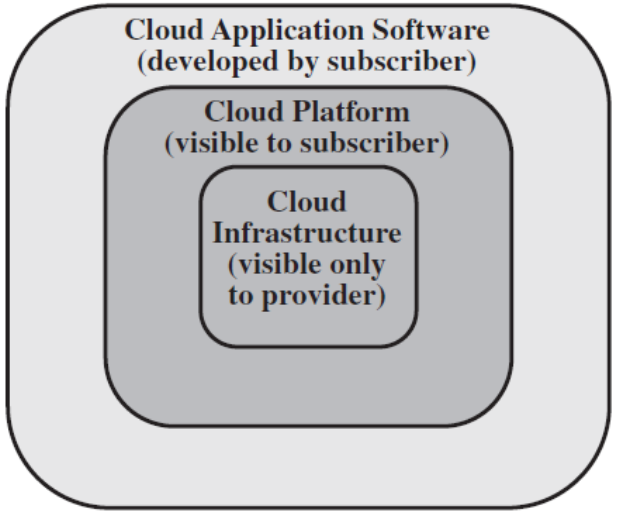


Cloud Service Models

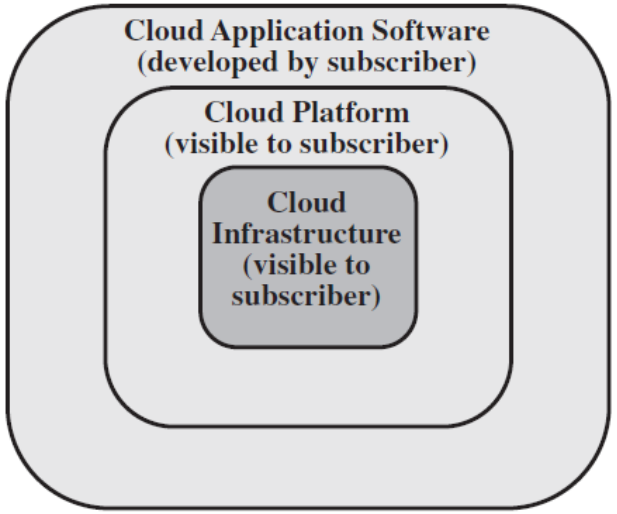
- Software as a service (SaaS)
  - e.g., using software installed on clouds via web browsers
- Platform as a service (PaaS)
  - e.g., enabling customers to developing their own applications running on operating systems provided by clouds
- Infrastructure as a service (IaaS)
  - e.g., enabling customers to install their own operating systems (Amazon EC2 and Windows Azure)
  - Clouds provide hardware (virtualization of hardware)



(a) SaaS

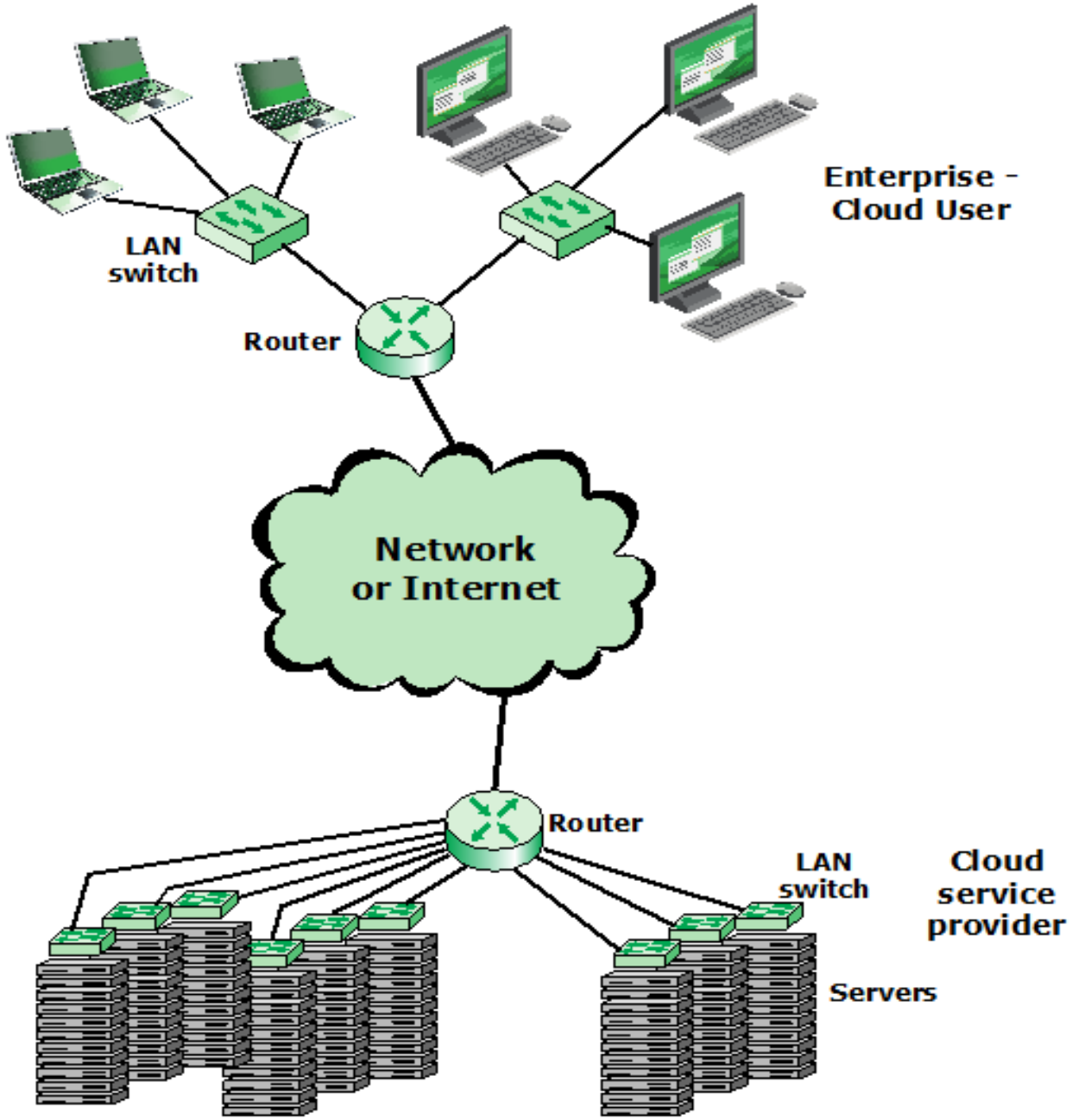


(b) PaaS

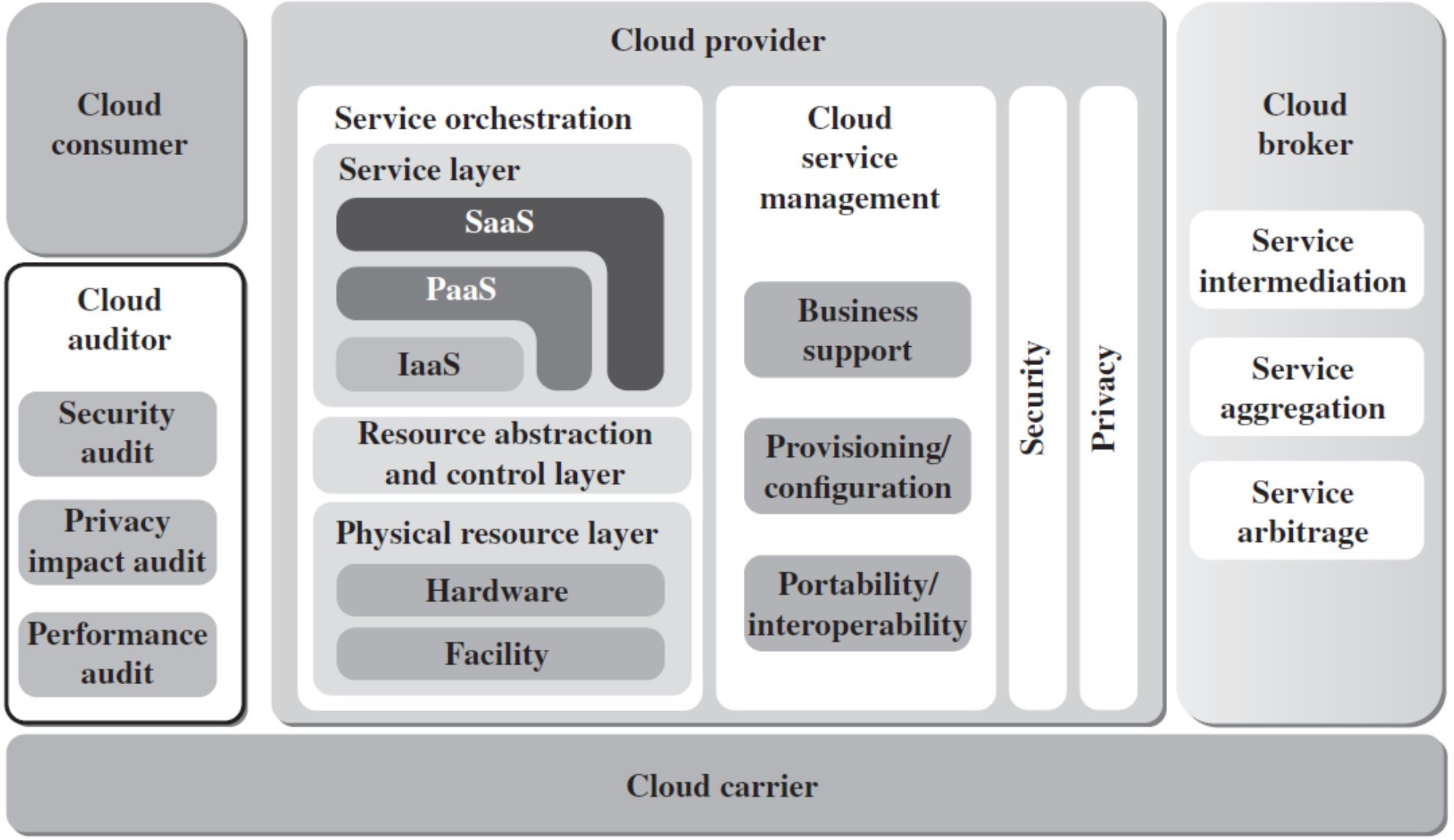


(c) IaaS

Typical Cloud Computing Context



NIST Cloud Computing Reference Architecture



Cloud Security Risks and Countermeasures

- Abuse and nefarious use of cloud computing
  - **Causes**: easy to register; free limited trial periods
  - **Countermeasures**: (1) stricter initial registration process; (2) monitoring fraud and traffic
- Insecure interfaces and APIs
  - **Causes**: a set of interfaces and APIs are exposed to customers
  - **Countermeasures**: (1) analyzing security model; (2) ensuring strong authentication and access control; (3) understanding the dependency chain
- Malicious insiders
  - **Causes**: certain necessary roles are extremely high-risk
  - **Countermeasures**: (1) strengthen management; (2) transparency; (3) security breach notification

Cloud Security Risks and Countermeasures (Cont.)

- Shared technology issues
  - **Causes**: sharing infrastructure in IaaS; the underlying components were not designed to offer strong isolation properties for a multi-tenant architecture
  - Typical solution: using isolated VM for clients, but still vulnerable
  - **Countermeasures**: (1) best security practices for installation/configuration; (2) monitoring, scanning, and auditing; (3) strong authentication and access control

- Data loss or leakage
  - **Countermeasures**: (1) Strong API access control; (2) data integrity protection; (3) data protection analysis; (4) strengthen management

Cloud Security Risks and Countermeasures (Cont.)

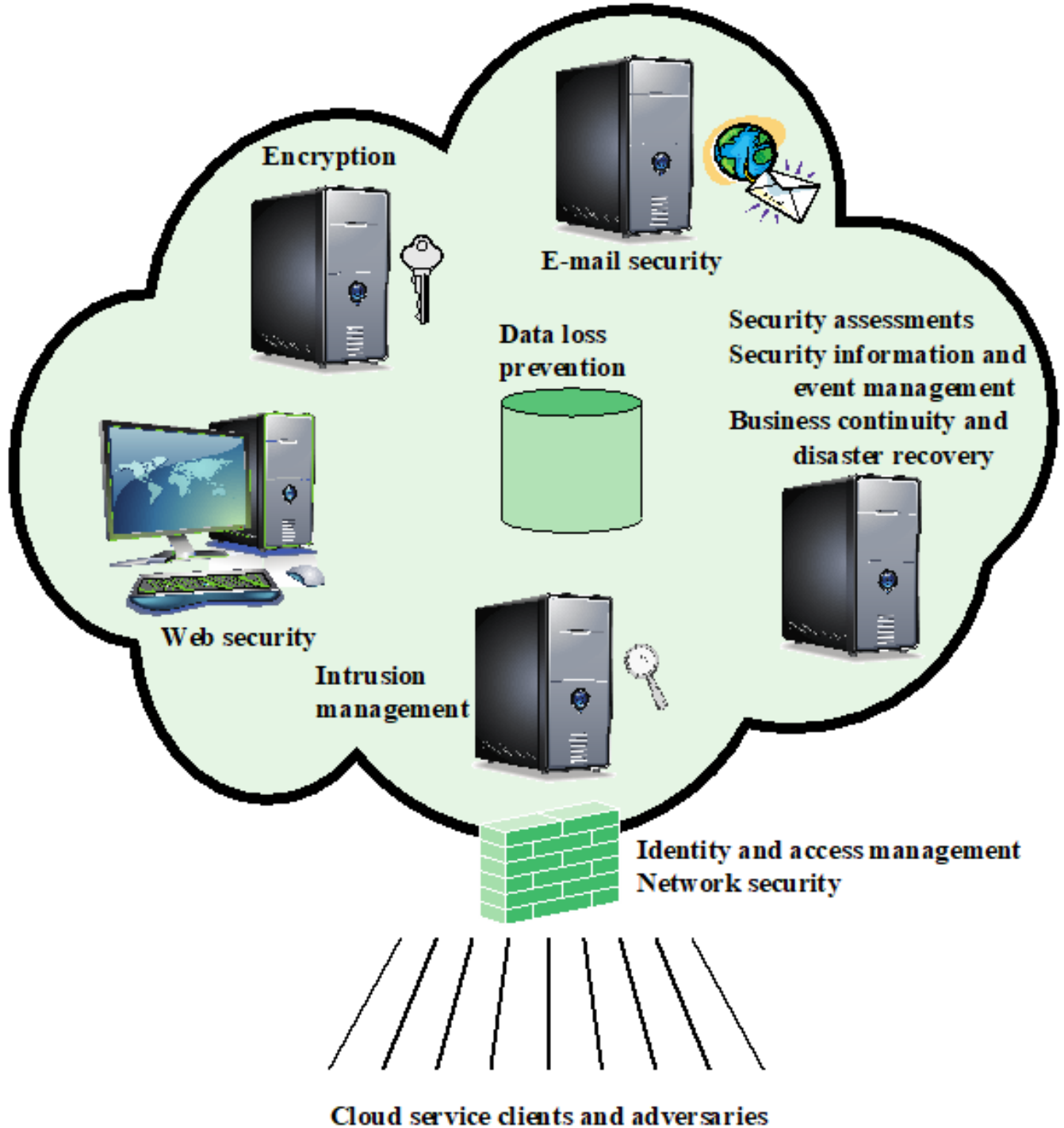
- Account or service hijacking
  - **Causes**: stolen credentials
  - **Countermeasures**: (1) better management: prohibit the sharing of account credentials; (2) strong two-factor authentication; (3) monitoring

- Unknown risk profile
  - **Countermeasures**: (1) disclosure of applicable logs and data; (2) partial/full disclosure of infrastructure details; (3) monitoring and alerting on necessary information

Cloud Security as a Service (SecaaS)

- Offloading of security functions
  - Authentication, anti-virus, antimalware/spyware, intrusion detection, security event management, etc.
- Definition by Cloud Security Alliance (CSA)
  - the provision of security apps and services via the cloud either to cloud-based infrastructure and software or from the cloud to the customers' systems

Elements of Cloud Security as a Service



Questions?