MONASH University
Information Technology

# FIT2093 INTRODUCTION TO CYBER SECURITY



## Week 10 Lecture



## Database Security, Privacy & Blockchain Security



**S1 2022**


# Outline

Database Security & Privacy

- Security issues in relational databases
  - SQL injection attacks
- **Privacy**: inference threats
  - Security issues in statistical databases
- Security against cloud exposure
  - Searchable encryption
- Security against insider attacks
  - Blockchain systems

MONASH University

# Security of Relational Databases

# Databases

## Database Security & Privacy

- **structured collection** of (sensitive) **data**
  - at a **single** location/system
  - **many** parties allowed (different types of) **access**
  - need **Database Security**

# Relational Databases

Database Security & Privacy

- contain **relationships** between data **items** & **groups** of data items



- database management system (**DBMS**)



  - suite of programs to **construct** & **maintain** the database
  - allows to do ad hoc **query** facilities to multiple users & applications



  - **access** control to records/fields, for different commands

- vs OS: access control to database **files**, not records/field

# Databases

## Database Security & Privacy

- why **database security** is **hard**
  - DBMS complex, **many options**
  - **different types** of databases
  - complicated **interaction protocol**:
    - Structured Query Language (**SQL**)
  - **mismatch**: database admin vs security
  - **outsourcing** to clouds

# Relational Databases

Database Security & Privacy

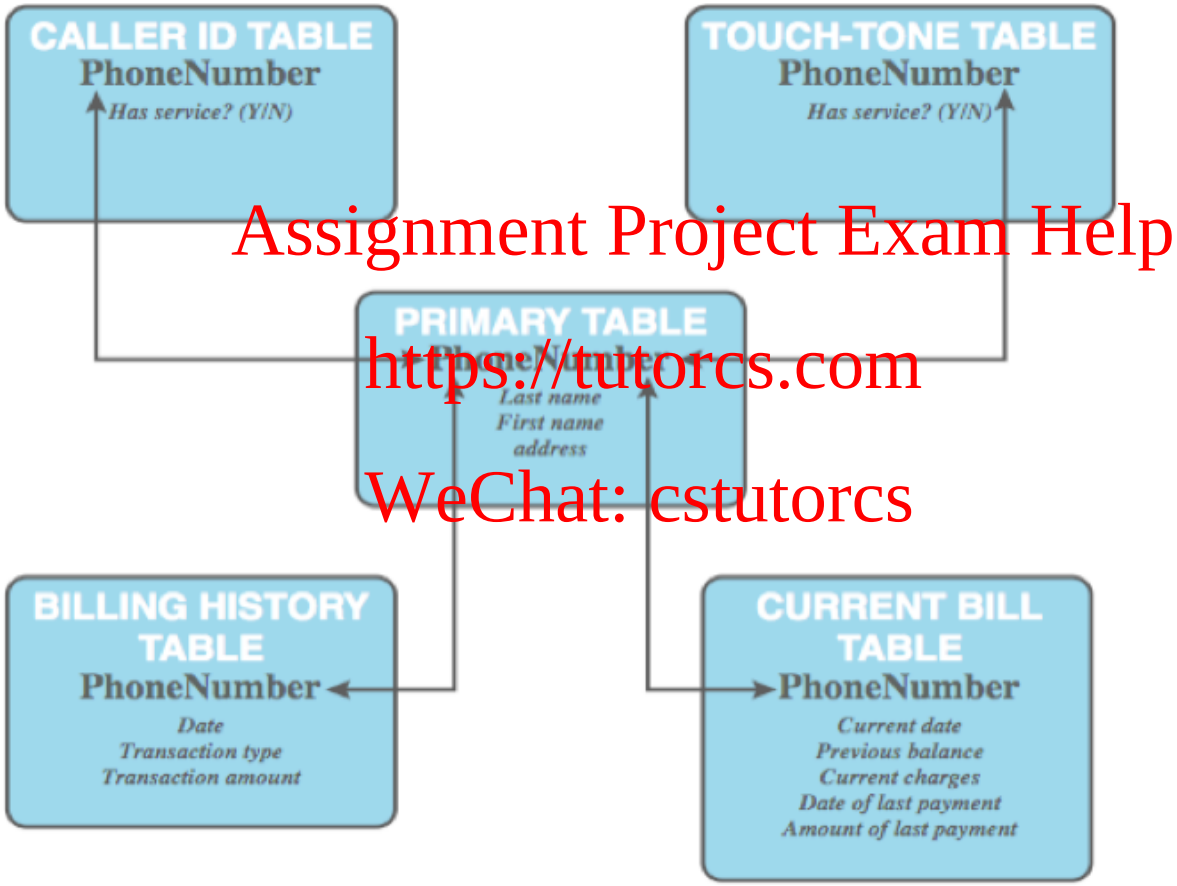
- **table** of data consisting of rows and columns
  - each **column** holds a particular **type** of data
  - each **row** contains a specific **value** for each column
  - ideally has **one column** where **all values** are **unique**, forming an **identifier/key** for that row



- enables the creation of **multiple tables linked** together by a unique **identifier** that is present in **all** tables

# Relational Databases: Example

Database Security & Privacy



# Relational Databases

Database Security & Privacy

- use a **relational query language** to **access** the database
  - allows the user to request data that fit a given set of criteria
  - e.g. MySQL



- view
  - **virtual** table (unifies results from many tables)
  - result of a query, returning selected rows/columns

# Relational Databases: Example

Database Security & Privacy

**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 | 20 | 3054 | 6127092729 |
| Robin | 8 | 24 | 2976 | 6127091945 |
| Smith | 9 | 21 | 4490 | 6127099380 |

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

# SQL: Structured Query Language

Database Security & Privacy

- **SQL: most popular DBMS query language**
  - Many **variants: MySQL, Microsoft, Oracle, …**
    - minor syntax variations b/w variants
    - we use **MySQL** in our examples/lab
- **List of important SQL commands**
  - **SELECT** - extracts data from a database
  - **UPDATE** - updates data in a database
  - **DELETE** - deletes data from a database
  - **INSERT INTO** - inserts new data into a database
  - **CREATE DATABASE** - creates a new database
  - **ALTER DATABASE** - modifies a database
  - **CREATE TABLE** - creates a new table
  - **ALTER TABLE** - modifies a table
  - **DROP TABLE** - deletes a table
  - **CREATE INDEX** - creates an index (search key)
  - **DROP INDEX** - deletes an index

# SQL: Structured Query Language

Database Security & Privacy

- Most used query statement: **SELECT**
  - Retrieves data from a database table
  - **Syntax:**
    - SELECT *column1*, *column2*, ...
      FROM *table_name*; (select specific columns)
    - SELECT * FROM *table_name*; (select all columns)



- e.g. SELECT CustomerName,City FROM Customers;

| CustomerName | City |
|---|---|
| Alfreds Futterkiste | Berlin |
| Ana Trujillo Emparedados y helados | México D.F. |
| Antonio Moreno Taquería | México D.F. |

- Reference / tutorial on SQL: https://www.w3schools.com/sql/default.asp

# Database Security Requirements

Database Security & Privacy

- **Physical** database **integrity**
  - The data are **immune** to physical problems
- Logical database **integrity**
  - The **structure** of the database is **preserved**
- Element **integrity**
  - The **data** contained in each element are **accurate**
- **Auditability**
  - possible to **track** who has accessed the elements
- **Access control /** User **Authentication**
- **Confidentiality / Privacy** of data / private info
- **Avail**ability

# SQL Injection Attacks

Database Security & Privacy

- One of the most prevalent & dangerous network-based security threats

- Designed to exploit the nature of Web application pages
- Sends malicious SQL commands to the database server
  - A type of command injection vulnerability



- 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

# SQL: Idea

Database Security & Privacy

- Exploits:
  - server connected to database
  - server MySQL queries database, based on query from user
    - assumed: user input is value for variable
    - variable used in MySQL query to database
  - input not checked

# SQL: Idea

Database Security & Privacy

- SQL commands that return # of rows that contain a combination of UserName & Password input by user:



```
SELECT Count(*) FROM UsersTable
        WHERE UserName='Joanne'
        AND Password= 'JoannePassword'
```

# SQL: Idea

Database Security & Privacy

- Insert other SQL commands & (optionally) terminate with --



*SELECT Count(*) FROM UsersTable*

*WHERE UserName= '***Joanne' OR 1=1 --'**

*AND Password= ''*



- the ***OR 1=1*** always returns TRUE, so the query will always return a count greater than zero, resulting in a successful login

**Q:** What does -- symbol mean in SQL and why does attacker use it here?

# SQL: Idea

Database Security & Privacy

- SQLi attack typically works by **prematurely terminating** a text string & **appending** a new command
  - because inserted command may have additional strings appended to it before it is executed, the attacker terminates the injected string with a comment mark "--"

# SQL: Example

Database Security & Privacy

- SQL Injection Vulnerabilities
  - **Demo:** authenticated web site login
  - How users are supposed to behave
    - **Honest** Authorised user's "proper" behaviour
      - Name: `john`
      - Password: `monkey`
    - Misbehaviour by **ignorant hacker**
      - Name: `john`
      - Password: `guess`
  - Site rejects incorrect login password

# SQL: Example

Database Security & Privacy

- SQL Injection Vulnerabilities (cont.)
  - **Clever hacker**'s misbehaviour (example 1)
    - Name: `john' --`
    - Password:
  - Why does this work?
    - Exploits knowledge of system "under the hood":
      - login verification checked by server by passing a command string to an SQL database interpreter
      - Command string consists of a concatanation of SQL language commands and user input data
      - Attacker's password input string contains both data `john` and commands (`' --`)!
      - Attacker's command string passed to SQL interpreter and executed!
    - Vulnerable due to invalid assumption by developer: assumed users will only enter alphabetic characters!

# SQL: Example

Database Security & Privacy

- SQL injection (cont.)

"Under the Hood" : Vulnerable Server php code



```
$lUsername = $_REQUEST["username"];

$lPassword = $_REQUEST["password"];

$lQueryString  = " SELECT * FROM accounts WHERE
username=' " . $lUsername . " ' AND
password='" . $lPassword . "'" ;

$lQueryStringResult = $MySQLHandler->executeQuery($lQueryString);
```

- Command string $lQueryString passed to SQL interpreter for **honest** user login (user=john, pwd=monkey):

```
SELECT * FROM accounts WHERE username='john' AND password='monkey'
```

- Command string $lQueryString passed to SQL interpreter for **hacker** login (user=john' -- , pwd=guess):

```
SELECT * FROM accounts WHERE username='john' -- ' AND password='guess'
```

"--" interpreted by SQL as "begin comment": ignores following "AND..." password check!

# SQL: Example

Database Security & Privacy

- SQL Injection (cont.)
  - Control of SQL commands in the hands of the hacker can lead to even more devastating attacks
  - What if the hacker doesn't know any user's name? Wants to get ALL users info from Database?
    - **Clever hacker** misbehaviour (example 2)
      - Name: bob' OR 5=5 --
      - Password:
    - Command string $lQueryString passed to SQL interpreter:

```
SELECT * FROM accounts WHERE username='bob' OR '5'='5' -- ' AND
password='guess'
```

    - All DB entries satisfy `username='bob' OR '5'='5'`!
- **Lesson**: Server application **must** validate (check) user input before using it!
  - Make sure even malicious user input can not be interpreted as commands

# SQL: Countermeasures

Database Security & Privacy

- **defensive coding**
  - e.g. in code: **input type checking, input validation**
- **detection**
  - signature based: match attack patterns
  - anomaly based: detect behaviour beyond norm
- **code analysis**
  - test suite to detect SQLi vulnerabilities
  - check queries at run time

MONASH University

# Privacy of Databases

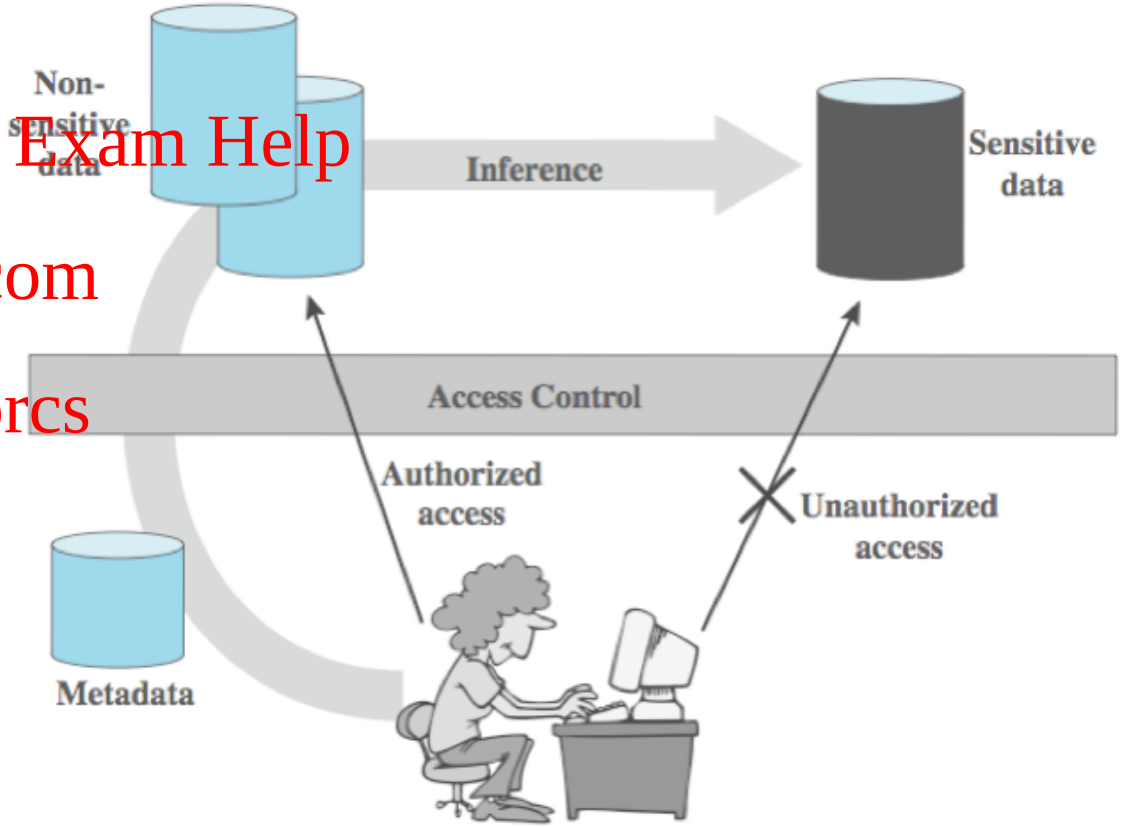
# Databases: Inferences vs Privacy

Database Security & Privacy

- process of performing queries & **deducing unauthorized information** from legitimate responses received

- **inference channel**
  - the **information transfer path** by which unauthorized data is obtained





# Database Inference: Example

Database Security & Privacy

- Analyzing functional dependencies
- Merging views with same constraints
- individual with unique value for Position attribute
  - Position is a “Quasi-ID”



| Name | Position | Salary ($) | Department | Dep Man. |
|---|---|---|---|---|
| Andy | Senior | 48,000 | Accounting | Cathy |
| Chris | Junior | 35,000 | Accounting | Cathy |
| Cathy | Management | 62,000 | Accounting | Cathy |
| Julia | Management | 58,000 | R&D | Julia |
| Maria | Junior | 35,000 | R&D | Julia |

**Q:** In the example database, which employee salaries are uniquely revealed by the published views (Position-Salary and Name-Position)?

| Position | Salary ($) |
|---|---|
| Senior | 48,000 |
| Junior | 35,000 |
| Management | 58,000 |
| Management | 62,000 |
| Junior | 35,000 |



| Name | Position |
|---|---|
| Andy | Senior |
| Chris | Junior |
| Cathy | Management |
| Julia | Management |
| Maria | Junior |

Position Quasi ID used to link b/w published views

# Inference Attacks: Countermeasures

Database Security & Privacy

- To protect a database from inference attacks, following techniques to be used prior to making the database public
  - Cell **suppression**
    - some of the cells in a database are **removed** and left blank in the published version.
  - **Generalization / Averaging**
    - some values in a published database are replaced with more general/averaged values
  - **Noise addition**
    - values in a published database have **random** values added to them, so that the noise across all records for the same attribute averages out to zero

# Inference Attacks: Example

Database Security & Privacy

| Num | Age1 | Age2 |
|---|---|---|
| 11 | 49.3 | 53.6 |
| 18 | 46.9 | 63.2 |
| 20 | 49.3 | 49.8 |
| 35 | 43.6 | 46.5 |
| 42 | 46.4 | |
| 44 | 47.5 | |

(a)

| Num | Age1 | Age2 |
|---|---|---|
| 11 | 49.3 | |
| 18 | 46.9 | 63.2 |
| 20 | 49.3 | |
| 35 | | |
| 42 | 46.4 | |
| 44 | 47.5 | |

(b)

| Num | Age1 | Age2 |
|---|---|---|
| 11 | 45–50 | 50–60 |
| 18 | 45–50 | 60–75 |
| 20 | 45–50 | 45–50 |
| 35 | 40–45 | 45–50 |
| 42 | 45–50 | |
| 44 | 45–50 | |

(c)

| Num | Age1 | Age2 |
|---|---|---|
| 11 | 47.7 | 55.2 |
| 18 | 49.2 | 64.3 |
| 20 | 51.6 | 52.8 |
| 35 | 42.3 | 47.3 |
| 42 | 47.1 | |
| 44 | 48.0 | |

(d)

**Figure 10.4:** Obfuscation techniques for protecting the privacy of individuals included in a public database: (a) A table with individual names removed. (b) A table anonymized using cell suppression. (c) A table anonymized using generalization. (d) A table anonymized using noise.

# Privacy Definitions


Database Security & Privacy

- ***k*-anonymity** [Sweeney 2002]
  - *averaging…* each is indistinguishable from *k*–1 others
  - On the values of ”Quasi-ID” attributes that could be linked with other released data (e.g. ZIP Code, Age below)



Original

| | ZIP Code | Age | Disease |
|---|---|---|---|
| 1 | 47677 | 29 | Heart Disease |
| 2 | 47602 | 22 | Heart Disease |
| 3 | 47678 | 27 | Heart Disease |
| 4 | 47905 | 43 | Flu |
| 5 | 47909 | 52 | Heart Disease |
| 6 | 47906 | 47 | Cancer |
| 7 | 47605 | 30 | Heart Disease |
| 8 | 47673 | 36 | Cancer |
| 9 | 47607 | 32 | Cancer |

*k*-anonymized

| | ZIP Code | Age | Disease |
|---|---|---|---|
| 1 | 476** | 2* | Heart Disease |
| 2 | 476** | 2* | Heart Disease |
| 3 | 476** | 2* | Heart Disease |
| 4 | 4790* | ≥ 40 | Flu |
| 5 | 4790* | ≥ 40 | Heart Disease |
| 6 | 4790* | ≥ 40 | Cancer |
| 7 | 476** | 3* | Heart Disease |
| 8 | 476** | 3* | Cancer |
| 9 | 476** | 3* | Cancer |

# Privacy Definitions

Database Security & Privacy

- *k*-anonymity applied to **faces** [Gross et al. 2009]
  - *averaging…*
  - each is indistinguishable from *k*-1 others



# Privacy Definitions


Database Security & Privacy

- ***k*-anonymity** [Sweeney 2002]
  - but other private attributes still leak (if low diversity / range of possibilities),
  - i.e. though don't know which row/image links to which person, class-specific attribute leaks e.g. heart disease, fatal disease, ...



Original

| | Zip Code | Age | Nationality | Condition |
|---|---|---|---|---|
| 1 | 13053 | 28 | Russian | Heart Disease |
| 2 | 13068 | 29 | American | Heart Disease |
| 3 | 13068 | 21 | Japanese | Viral Infection |
| 4 | 13053 | 23 | American | Viral Infection |
| 5 | 14853 | 50 | Indian | Cancer |
| 6 | 14853 | 55 | Russian | Heart Disease |
| 7 | 14850 | 47 | American | Viral Infection |
| 8 | 14850 | 49 | American | Viral Infection |
| 9 | 13053 | 31 | American | Cancer |
| 10 | 13053 | 37 | Indian | Cancer |
| 11 | 13068 | 36 | Japanese | Cancer |
| 12 | 13068 | 35 | American | Cancer |

| | Zip Code | Age | Nationality | Condition |
|---|---|---|---|---|
| 1 | 130** | < 30 | * | Heart Disease |
| 2 | 130** | < 30 | * | Heart Disease |
| 3 | 130** | < 30 | * | Viral Infection |
| 4 | 130** | < 30 | * | Viral Infection |
| 5 | 1485* | ≥ 40 | * | Cancer |
| 6 | 1485* | ≥ 40 | * | Heart Disease |
| 7 | 1485* | ≥ 40 | * | Viral Infection |
| 8 | 1485* | ≥ 40 | * | Viral Infection |
| 9 | 130** | 3* | * | Cancer |
| 10 | 130** | 3* | * | Cancer |
| 11 | 130** | 3* | * | Cancer |
| 12 | 130** | 3* | * | Cancer |

***4-anonymized***

# Privacy Definitions

Database Security & Privacy

- ℓ-diversity [Machanavajjhala et al. 2007]
  - ensure sufficient diversity within each equivalence class
  - privacy in the sense of non-linkability to certain specific attribute values

e.g. more diverse mix of faces



| | Zip Code | Age | Nationality | Condition |
|---|---|---|---|---|
| 1 | 1305* | ≤ 40 | * | Heart Disease |
| 4 | 1305* | ≤ 40 | * | Viral Infection |
| 9 | 1305* | ≤ 40 | * | Cancer |
| 10 | 1305* | ≤ 40 | * | Cancer |
| 5 | 1485* | > 40 | * | Cancer |
| 6 | 1485* | > 40 | * | Heart Disease |
| 7 | 1485* | > 40 | * | Viral Infection |
| 8 | 1485* | > 40 | * | Viral Infection |
| 2 | 1306* | ≤ 40 | * | Heart Disease |
| 3 | 1306* | ≤ 40 | * | Viral Infection |
| 11 | 1306* | ≤ 40 | * | Cancer |
| 12 | 1306* | ≤ 40 | * | Cancer |

ℓ-diverse

# Inference: General Countermeasures



- inference detection at database **design**
  - alter database **structure** or access controls
- inference detection at **query** time
  - by **monitoring** & altering or rejecting queries
- need some **inference detection** algorithm
  - a difficult problem
  - on-going research

MONASH University

# Security of Statistical Databases

# Statistical Databases

Database Security & Privacy

- data of statistical nature e.g. counts, averages
- **two types:**
  - pure statistical database
  - ordinary database with statistical access
    - contains individual entries
- access control objective is to provide users with needed information without compromising **CONF** of database
- security problem is one of inference

# Statistical Databases: Example

Database Security & Privacy

(a) Database with statistical access with $N = 13$ students

| Name | Sex | Major | Class | SAT | GP |
|---|---|---|---|---|---|
| Allen | Female | CS | 1980 | 600 | 3.4 |
| Baker | Female | EE | 1980 | 520 | 2.5 |
| Cook | Male | EE | 1978 | 630 | 3.5 |
| Davis | Female | CS | 1978 | 800 | 4.0 |
| Evans | Male | Bio | 1979 | 500 | 2.2 |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | 3.0 |
| Good | Male | CS | 1978 | 700 | 3.8 |
| Hall | Female | Psy | 1979 | 580 | 2.8 |
| Iles | Male | CS | 1981 | 600 | 3.2 |
| Jones | Female | Bio | 1979 | 750 | 3.8 |
| Kline | Female | Psy | 1981 | 500 | 2.5 |
| Lane | Male | EE | 1978 | 600 | 3.0 |
| Moore | Male | CS | 1979 | 650 | 3.5 |



(b) Attribute values and counts

| Attribute $A_j$ | Possible Values | $\|A_j\|$ |
|---|---|---|
| Sex | Male, Female | 2 |
| Major | Bio, CS, EE, Psy, … | 50 |
| Class | 1978, 1979, 1980, 1981 | 4 |
| SAT | 310, 320, 330, …, 790, 800 | 50 |
| GP | 0.0, 0.1, 0.2, …, 3.9, 4.0 | 41 |

# Statistical Database Security

Database Security & Privacy

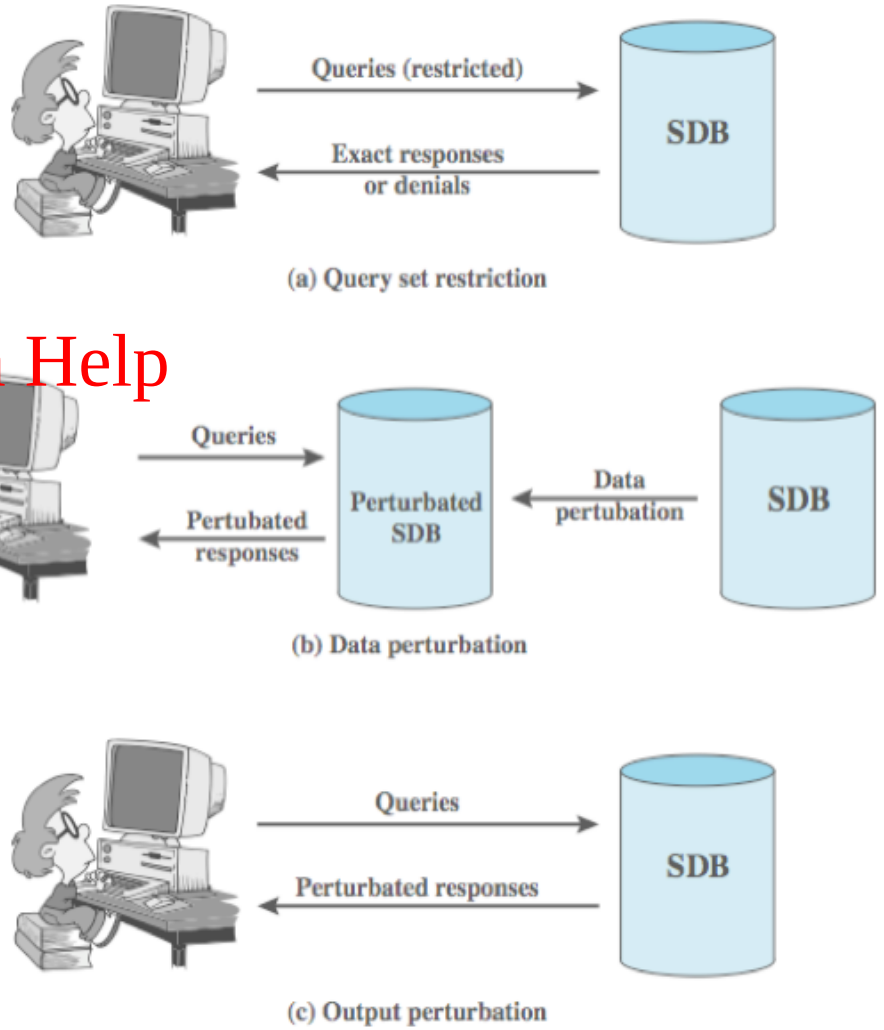
- **use a characteristic formula C**
  - a logical formula over the values of attributes
  - e.g. **(*Sex*=Male) AND ((*Major*=CS) OR (*Major*=EE))**
- query set X(***C***) of characteristic formula ***C***, is the set of records matching ***C***
- a statistical query: produces a value calculated over a **query set**
  - e.g. statistics are count, sum, average, median, max, min

# Statistical Databases Security

Database Security & Privacy

- Security against inference





# # Statistical Database Security

Database Security & Privacy

- Some query restrictions
- **query set overlap control**
  - **limit** overlap between new & previous queries
  - has problems & overheads
- **partitioning**
  - cluster records into number of **mutually exclusive** groups
  - query the statistical properties of each group as a whole
- **query denial & information leakage**
  - denials can leak information
  - to counter must **track queries** from user

# Perturbation

Database Security & Privacy

- add **noise to statistics** generated from data
- data perturbation techniques
  - data modified to produce statistics that **cannot infer** values for **individual** records
- output perturbation techniques
  - **random**-sample query
  - system generates **statistics** that are **modified** from those that the original database would provide
- goal is to minimize differences between original results & perturbed results
- main challenge: to determine the average size of the error/difference to be used

MONASH University

# Searchable Encryption

# Database Security: Searchable Encryption


Database Security & Privacy

- encryption for particularly sensitive data
  - can be applied to the **entire** database at the record level, the **attribute** level, or level of the individual **field**
- special encryption technique for database encryption, to allow server to efficiently **search encrypted database without decrypting**

# Database Security: Searchable Encryption


Database Security & Privacy

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



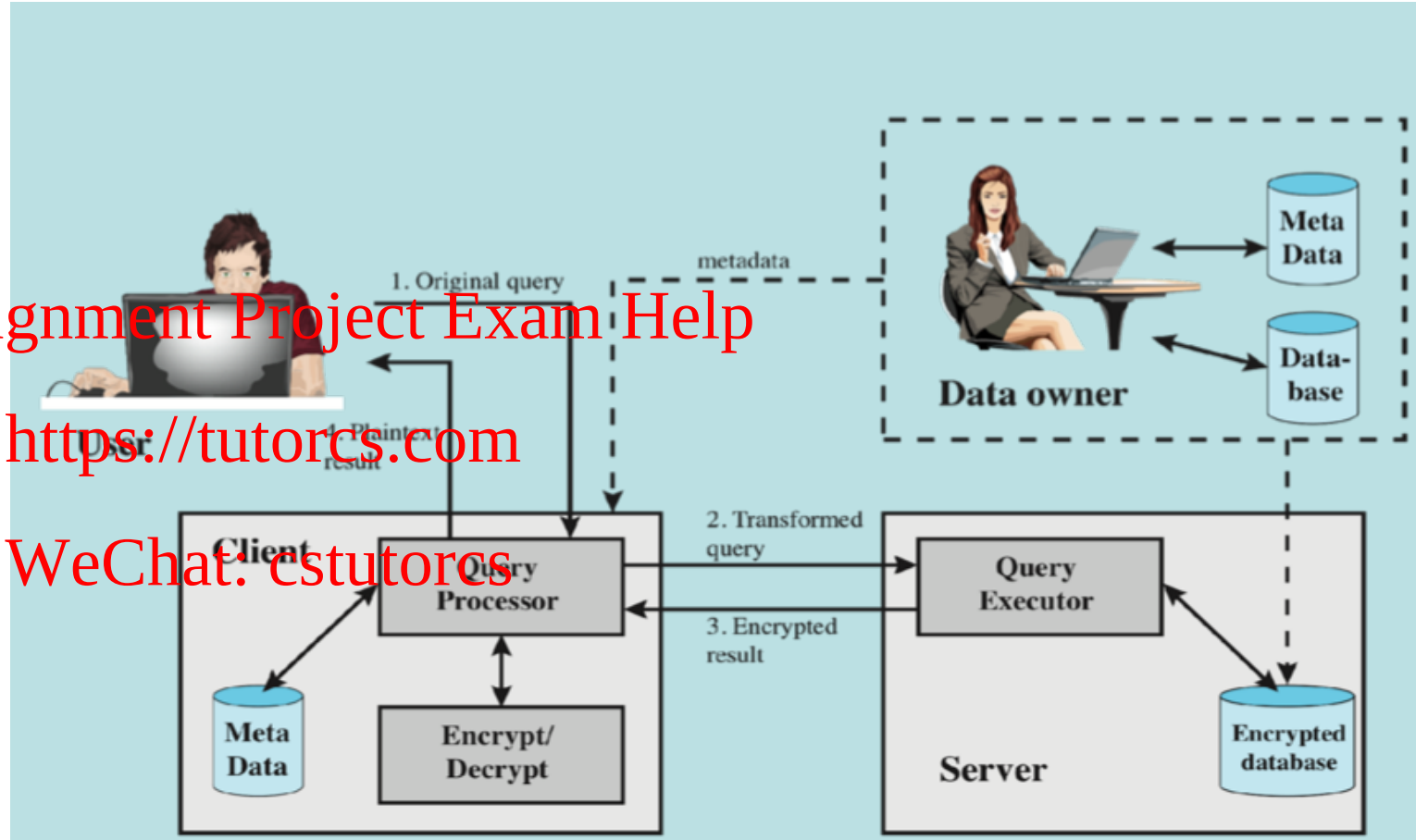



Figure 5.10 A Database Encryption Scheme

# Searchable Encryption

Database Security & Privacy

- Use **special encryption technique** for database encryption, to allow **server** to efficiently **search encrypted database without decrypting**
- **e.g.1: use encrypted index based on deterministic encryption (no randomness) to encrypt search values**
  - Search token for “50000” is C = Enc(K, “50000”)
  - Server returns encrypted rows where encrypted salary = C
  - Efficiency Advantage: fast server search lookup
  - Security Drawback: deterministic encryption leaks frequency statistics (recall ECB mode security problem)
- **e.g.2: to support range query (e.g. “salary <60k”) store/query only index of salary interval on server**
  - E.g. 1=50k-59k, 2=60k-69k, 3=70k-79,… in plain form, actual salary values encrypted.
  - Still info. leakage to server!
- Active research area to improve efficiency-security tradeoff and allow flexible queries

# Encrypted Table: Example



Table 5.7 Encrypted Database Example

(a) Employee Table

| eid | ename | salary | addr | did |
|---|---|---|---|---|
| 23 | Tom | 70K | Maple | 45 |
| 860 | Mary | 60K | Main | 83 |
| 320 | John | 50K | River | 50 |
| 870 | Jerry | 55K | Hopewell | 92 |



(b) Encrypted Employee Table with Indexes

| E($k$, $B$) | I(eid) | I(ename) | I(salary) | I(addr) | I(did) |
|---|---|---|---|---|---|
| 11001100110010 11… | 1 | 10 | 3 | 7 | 4 |
| 01110001110010 10… | 5 | 7 | 2 | 7 | 8 |
| 11000100100011 01… | 2 | 5 | 1 | 9 | 5 |
| 00110100111111 01… | 5 | 5 | 2 | 4 | 9 |

MONASH University

# Blockchain



# Database Security: Blockchain

Database Security & Privacy

What if the DB insider (DB administrator) is compromised?

- To avoid trust in a centralised database administrator, want a **distributed database administrator**
  - many **servers act together as database administrators**
  - Goal: Preserve database integrity as long as a **"majority"** of administrators are honest
    - **Reduced risk of compromise compared to single DB administrator (no single point of failure)**
    - Different distributed database models interpret "majority" in a different way
- Possible solution: a **blockchain** database system
  - Most popular example: Bitcoin cryptocurrency

# Blockchain: Overview

Database Security & Privacy

- Blockchain administrators usually called miners
- Users submit transactions to blockchain miner
  - e.g. transfer of bitcoin currency from one user to another
- Blockchain database records a sequence of blocks
  - Each block contains a header and a sequence of valid transactions
    - Valid transaction meaning:
      - Correct formatting (e.g. input account not already spent)
      - Authentic (digitally signed by user)
    - Block header consists of :
      - Block number
      - Previous block header's cryptographic hash value
      - Cryptographic hash of the block transactions
      - nonce

# Blockchain: Overview

Database Security & Privacy

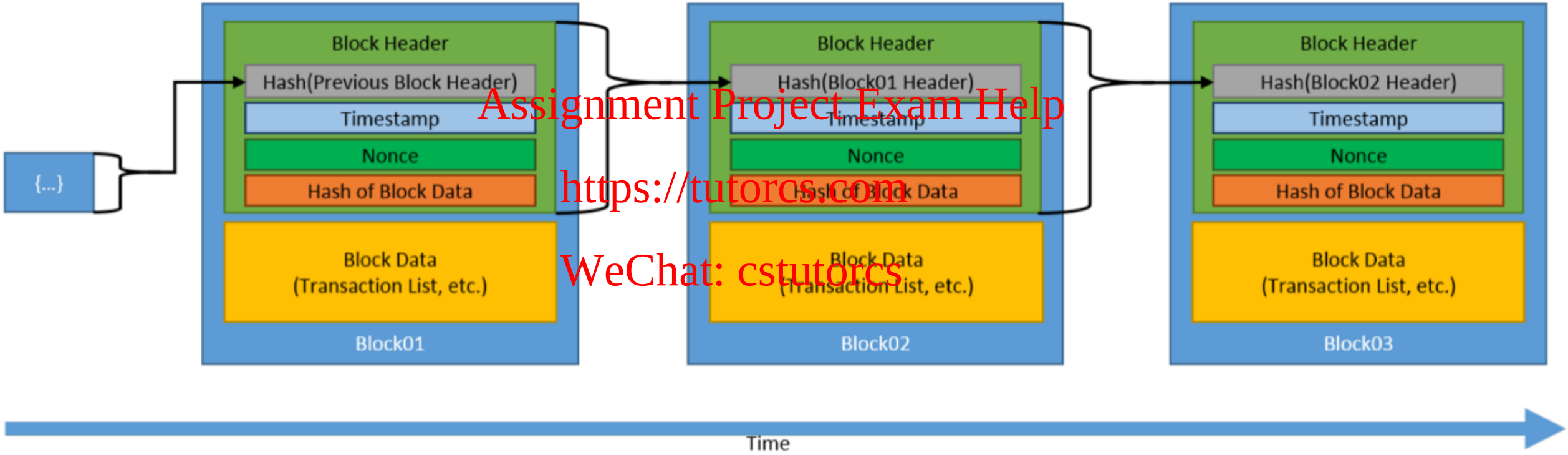







*Image source*: NIST R8202 document

# Blockchain: Overview

Database Security & Privacy

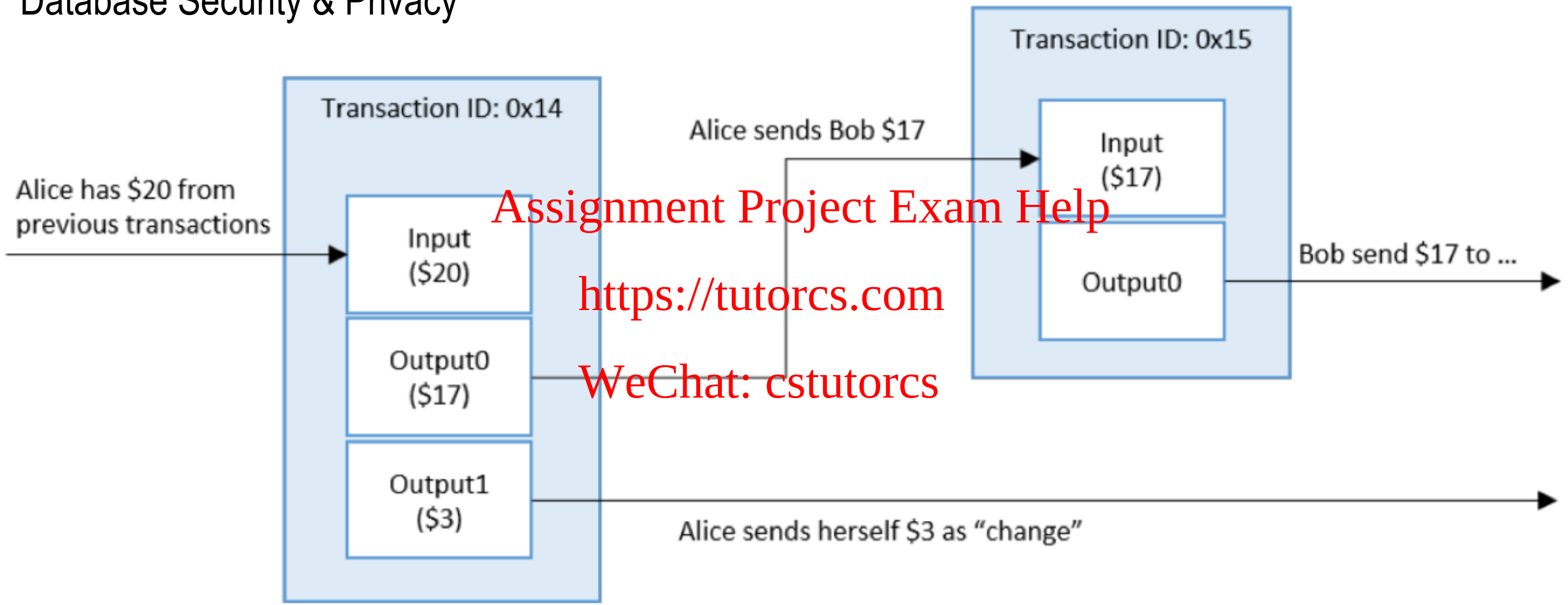





**Figure 1 - Example Cryptocurrency Transaction**



*Image source*: NIST R8202 document

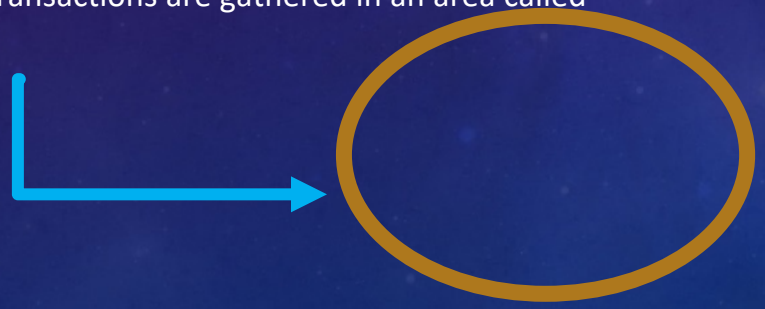
# HOW TRANSACTION WORKS AT THE BACK OF BLOCK CHAIN?

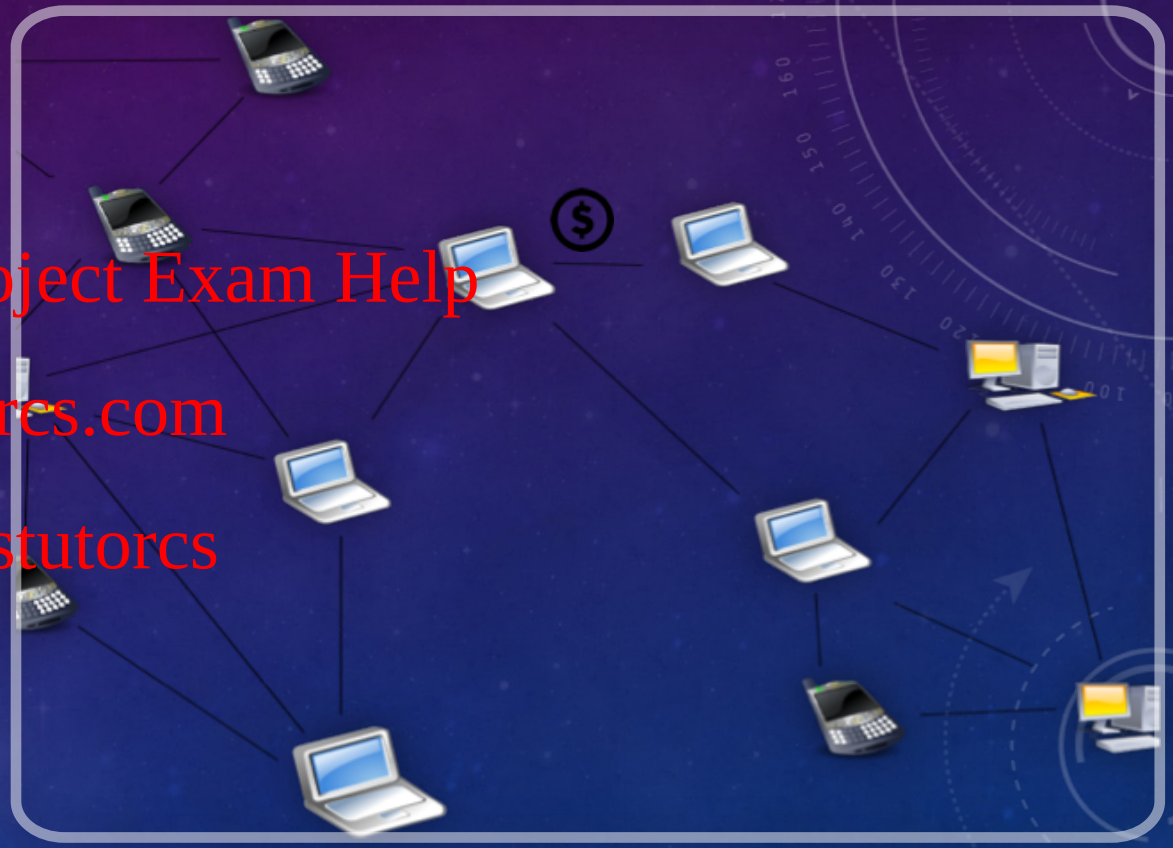
1. When someone does the transaction exchange, the transaction message is being sent to the network

2. The message is then passed around all the network participants which is called nodes

3. The transaction now is currently having 'unconfirmed' status

4. All the unconfirmed transactions are gathered in an area called **transaction pool**





# Blockchain: Security

Database Security & Privacy

Central Blockchain Security against Insiders Goal:

- **Tamper resistance** (”**append only ledger**”) property: Past transactions recorded in blockchain cannot be modified/deleted by blockchain miners
  - As long as “majority” of miners are honest



- Enforced by a protocol between miners to agree on blockchain state (previous and new blocks): called a **consensus protocol**

# Blockchain: Security


Database Security & Privacy

- **Types of blockchain consensus protocols:**
  - **Proof of Work (PoW)**: miner publishes new block by solving a computationally intensive "puzzle"
    - Used in bitcoin and many other cryptocurrencies: called mining
    - Puzzle: find a nonce for new block such that hash(nonce + header + new block) has N leftmost 0 bits.
    - In Bitcoin: N is adjusted so that it takes on average 10 mins for first miner to find a puzzle solution
    - Once a solution is found, solving miner sends it to all other miners and they can easily verify the solution and accept the new block
    - Purpose: attacker must control > 50% of miner computational power to change past blocks and consistently solve the puzzles first (to get honest miners to accept modified blockchain)



**Q:** What do you think is one practical drawback of the PoW consensus mechanism?

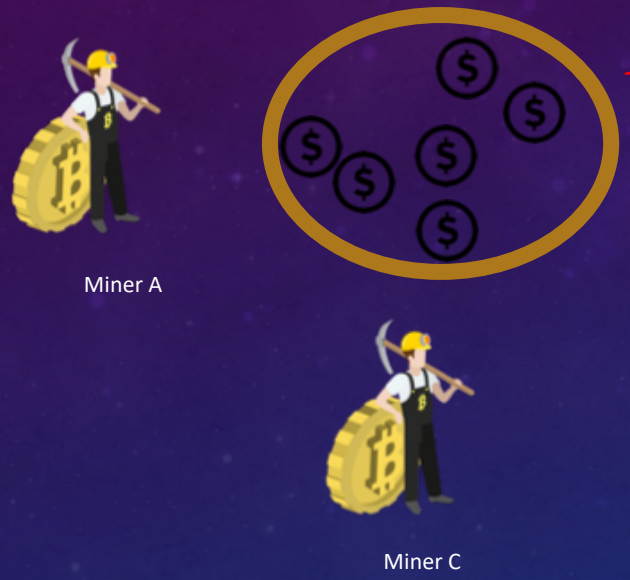
# Blockchain: Security

Database Security & Privacy

- **Other types of consensus protocols exist, e.g.:**
  - **Proof of Stake (PoS):** contributing miner chosen for each block with probability proportional to percentage of ownership (wealth) of the miner
    - Purpose: attacker must control > 50% of blockchain stake (wealth) to change past blocks and consistently get honest miners to accept modified blockchain
  - **Byzantine Fault Tolerance (BFT):** protocol involving communication among all miners to agree on new block
    - Purpose: attacker must control > 33% of participating miners to change past blocks and get honest miners to accept modified blockchain

# MINING AND PROOF OF WORK (POW)

- *Mining* is the process of validating new transactions and record them on the global ledger (block chain itself)




Generally miners will choose the transaction with high transaction fees (need more confirmations) from the transaction pool. For example:


Where each confirmation represents adding each block

When the miners identify that the 998th block is a valid block in the whole block chain, they will try to add a *candidate block*

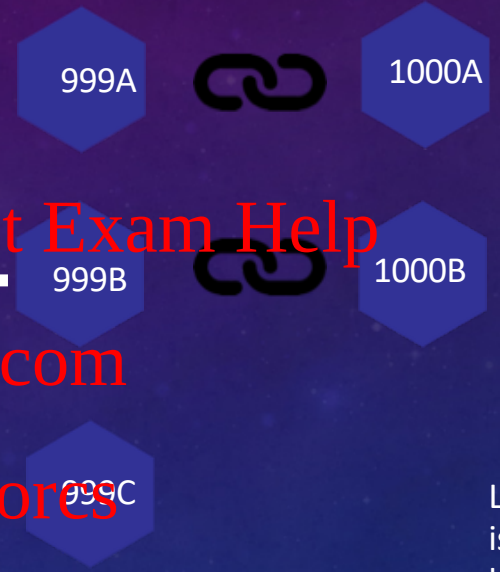
The miners need to solve an intensive computational problem (”puzzle”) in order to add a block after the 998th block – this puzzle is is known as a *proof of work*

# POW CONSENSUS: LONGEST CHAIN RULE

In case of a “fork” in the blockchain with several candidate new blocks, the block accepted as valid is the one in the longest chain (with largest number of solved POW puzzle blocks following it) – called the longest chain rule.




Assume we have 3 miners working on the same block chain, each of them has the proof of work, with candidate block each added after 998th block, which are 999A, 999B, 999C.





The miner with computer having higher processing speed will solve the puzzles to create subsequent blocks faster (on average). This allows the miners in control of the majority of computing power to produce the longest chain (with blocks numbered 1000, 1001… added after the 999th block) first, which means this chain will be the one accepted by the honest miners.


Looking at current situation, the longest chain is the one created by 999B, thus only 999B will be considered valid by miners, among the three candidate blocks.

When there are enough (say 6) subsequent blocks in longest chain following 999B, the chance of a competing chain with slower puzzle solving rate to overtake it is very small. We say that 999B status is transformed from unconfirmed to confirmed

# DIGITAL SIGNATURES IN BLOCK CHAIN

Now you understand how the basic transactions work in block chain.
Next, you need to know what are the roles of digital signatures and hashing in block chain.




| | | | |
|---|---|---|---|
| Verification Key (Public) | 010001… | 010010… | 010011… |
| Signing Key (Private) | 110110… | 110001… | 111100… |

A will sign the block A with its private key which ensures that *only A* can produce that signature.

{ Sign(message, private key) }

If someone wants to verify the block (transaction), they need to use A's public key.

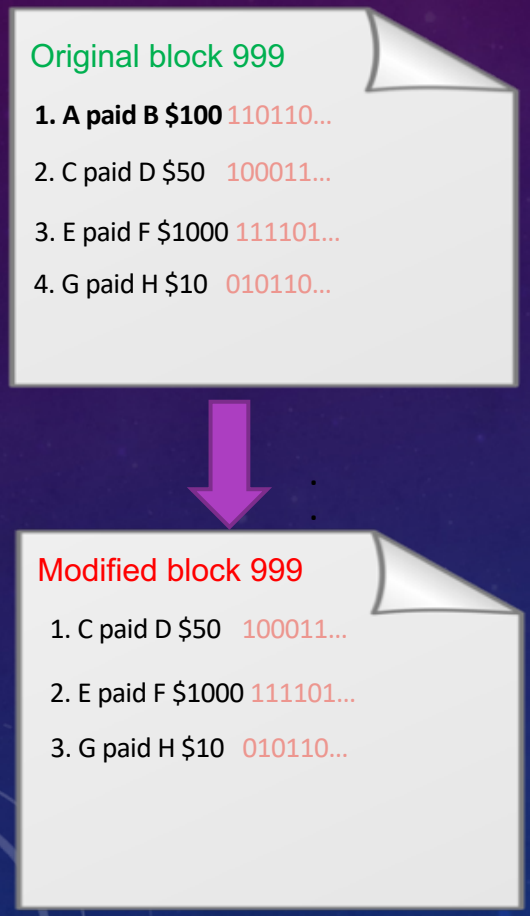
= YES / NO

= relatively short number sequence, $S_M$

By successfully validating the signature, the transaction done by A turned from unconfirmed to confirmed transaction and new blocks can be added behind it.

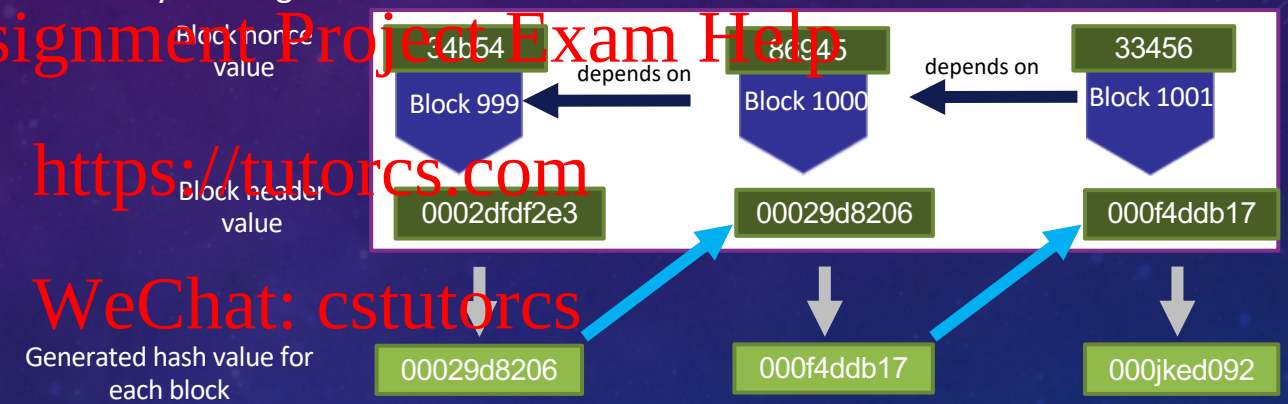
# HASHING IN BLOCK CHAIN

Now we have ensured that no one can forge digital signature to create fake transaction record. However, attacker is still able to delete a past transaction as shown, so he can spend it again later, or copy it multiple times to increase the amount of money spent.




To get around with this, we need to add some sort of unique number associated with the transaction message, and that will affect all subsequent blocks – we use collision-resistant one-way hashing for this.






Since each of the transaction block can only create new hashes based on the previous block, an attacker that modified some block will modify the hash of that block

- Due to the collision-resistant hash chain, all subsequent block hashes will change and the all subsequent block puzzles will need to be re-solved by the attacker
- The attacker will not be able to catch up with the honest miner longest chain length, assuming attacker total computational power for solving block puzzles is less than the honest miner total computation power
- Hash is a one-way function ⍰ no computational shortcut for solving puzzles!

# Further Reading

## Database Security & Privacy

- Chapter 5 of the textbook: *Computer Security: Principles and Practice" by William Stallings & Lawrie Brown,* Prentice Hall, 2015

*Optional*:



- **[Sweeney 2002]** L. Sweeney. "k-anonymity: a model for protecting privacy". International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10 (5), 2002; 557-570. Available at https://epic.org/privacy/reidentification/Sweeney_Article.pdf
- **[Gross 2007]** R. Gross and L. Sweeney. "Towards Real-World Face De-Identification", 2007 First IEEE International Conference on Biometrics: Theory, Applications, and Systems. Available at http://www.casos.cs.cmu.edu/publications/protected/2007-2008/gross_btas_02.pdf
- **[Machanavajjhala 2007]** A. Machanavajjhala, D. Kifer, J. Gherke and M. Venkitasubramaniam. "L-diversity: Privacy beyond k-anonymity", ACM Transactions on Knowledge Discovery from Data, Vol. 1, No. 1, Article 3, March 2007. Available at https://dl.acm.org/doi/pdf/10.1145/1217299.1217302
- G. Poh et al., "Searchable Symmetric Encryption: Designs and Challenges", ACM Computing Surveys, Volume 50 Issue 3, October 2017.
- D. Yaga et al., "Blockchain Technology Overview", NIST document NISTIR 8202, October 2018, available at https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf