DATABASE SECURITY & PRIVACY-PRESERVING TECHNOLOGIES

UNIVERSITY OF SOUTHERN MAINE - COS 457

MEET OUR AWESOME TEAM

Aubin Mugisha

Yunlong Li

Nikki Gorski

Kristina Zbinden

Sarah Kayembe

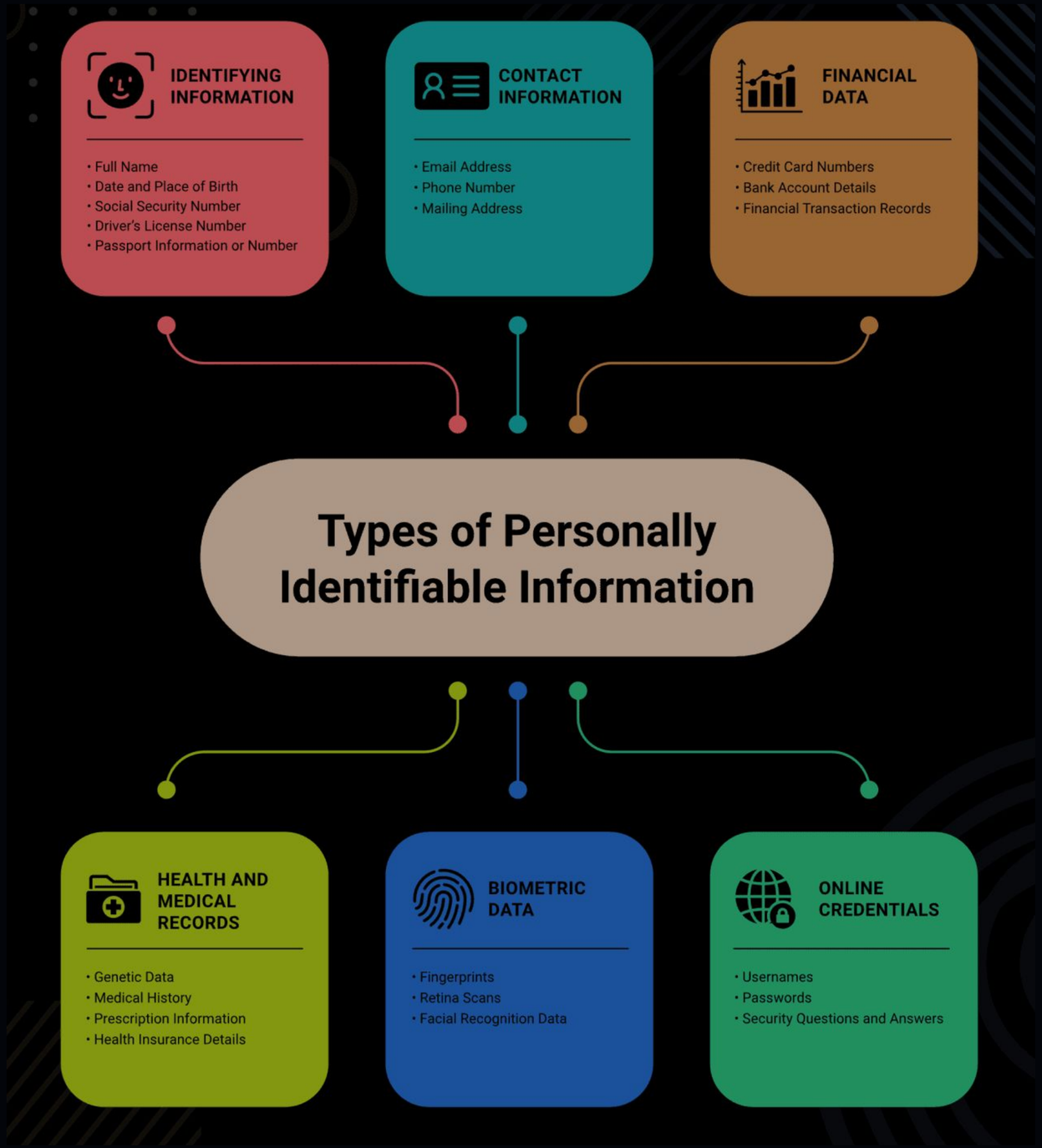
Motivations

Sensitive Information at Risk

- Financial records
- Medical data
- contact information
- location data
- biometrics
- online credentials/passwords

Important factors

- 82% of apps track personal data
- 4/5 companies store sensitive data in the cloud
- over 5000 data centers in US alone
- over 3500 data breaches per year



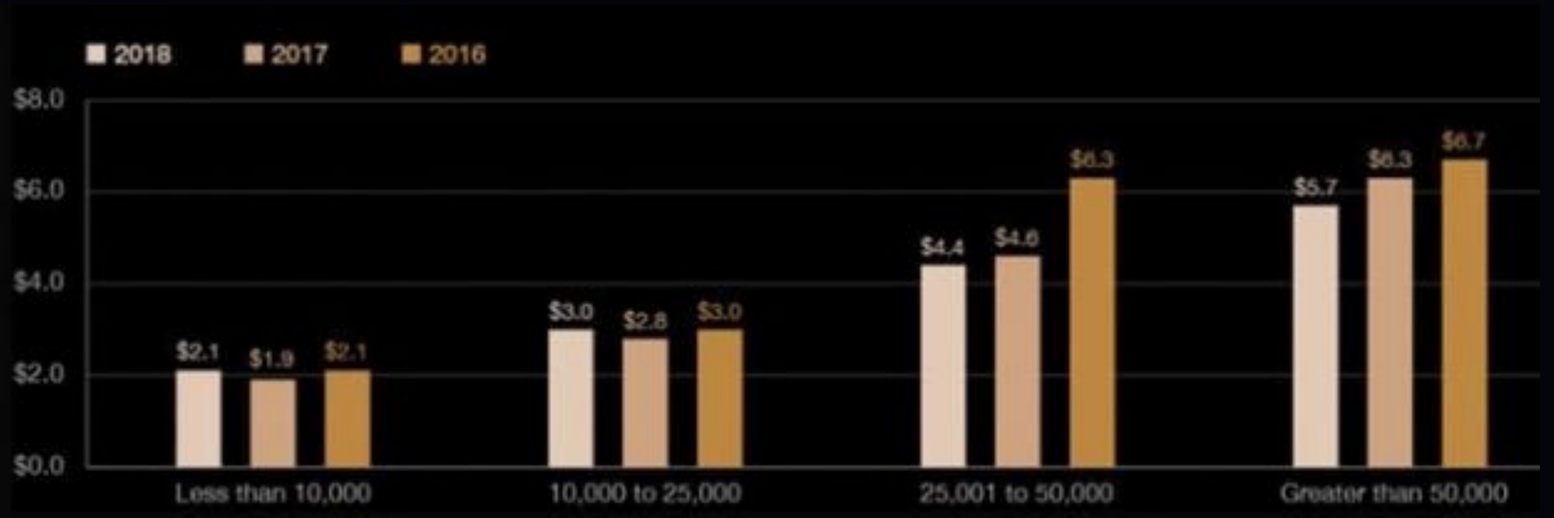
Motivations

Legal Responsibility

- HIPAA- protects medical data
- FERPA- protects student data
- GDPR- eu data protection law
- CCPA-californian data protection law

Consequences of Breaches

- Financial loss-avg of $3.86 million per breach
- Reputational damage
- Loss of customer trust



Average total cost by size of data breach (* million $)

Early History (1960s-1980s)

Database Security Origins

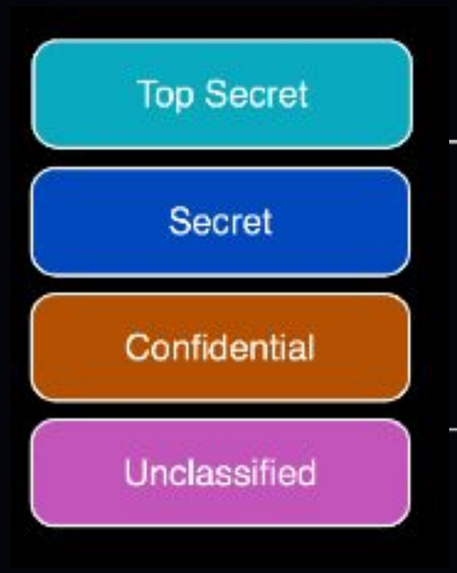
- Used mainly in government & military
- Security = basic authentication + physical access control
  - Locked server rooms
  - Physical keys

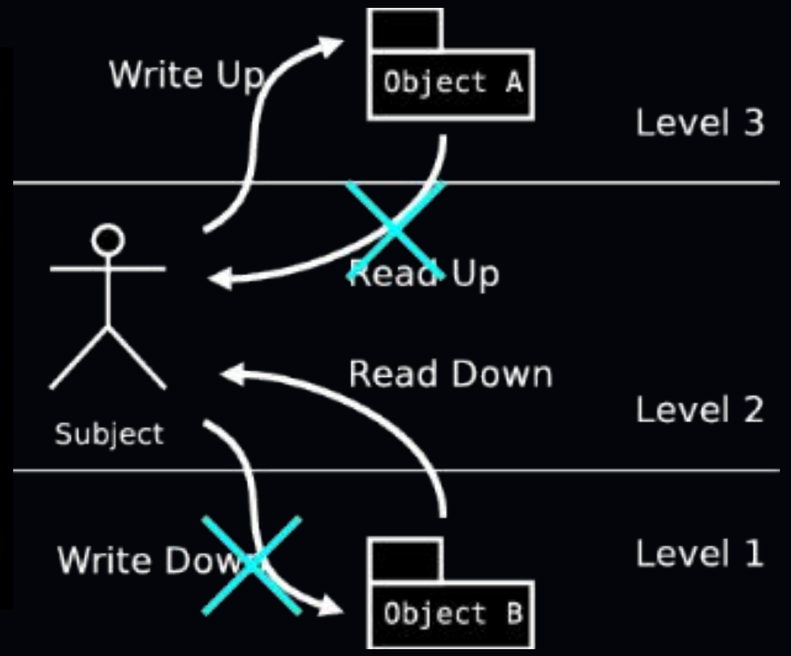
- Early Models of Confidentiality
  - Information security mindset: "classified vs unclassified"

Early History(1960s-1980s)

Bell–LaPadula & Access Control Models (1973)

- Divides entities into Discretionary access control
  - Subjects: users, processes
  - Objects: files, directories, ports
- Multi-level security (MLS)
- Rules
  - Mandatory access control(MAC)
    - Simple Security Property (no read up)
    - *-Property (no write down)
  - Discretionary access control(DAC)
    - Discretionary Security Property (access matrix)



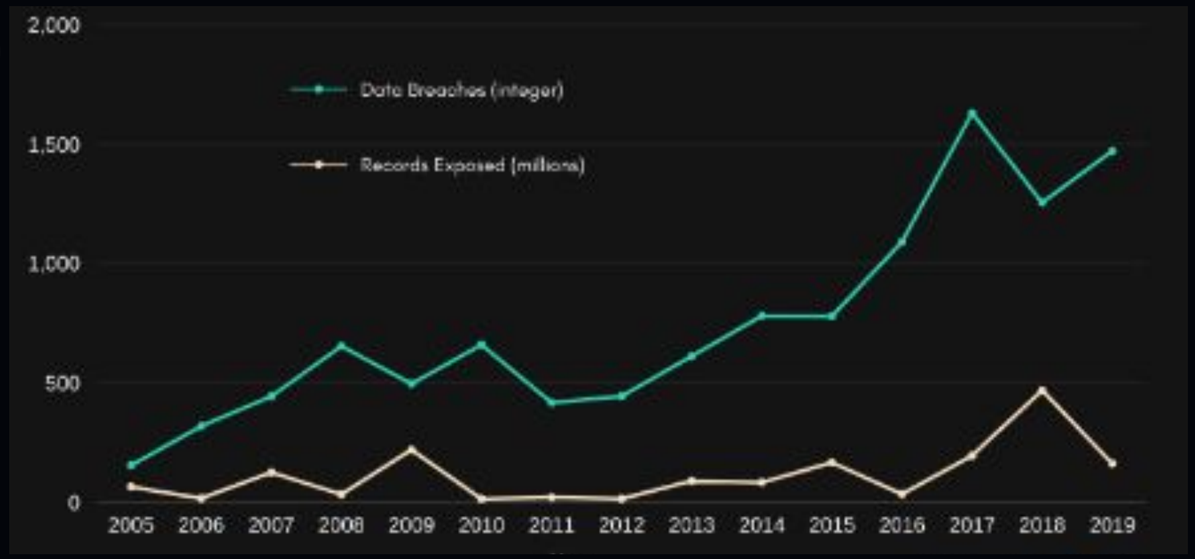


Commercial Databases & the 1990s Evolution

- Role-Based Access Control (RBAC) (1977)
  - introduction of private-public keys instead of just a single key
- Expansion of encryption
  - RSA(Rivest, Shamir, and Adleman)
  - DES (Data encryption standard) (1977)
    - later became AES (2001)
    - Government introduced encryption standard
- Early auditing/logging
  - Audit trails for recovery and misuse detection

The 2000s: The Breach Era

- Massive Growth in Breaches
  - 157 breaches (2005) → 662 (2010) → 3500+ (2023)
  - ChoicePoint (2005, 2008)
  - TJX/TJMaxx (2007): 45.7M credit cards stolen
- SQL Injection Becomes a Major Threat
  - Attackers inject malicious SQL
  - Dump entire DB contents
- Security Research Expands
  - Database firewalls
  - Intrusion Detection Systems (IDS)
    - Network, Host, Signature-based, Anomaly-based
- Stronger key management practices



Annual data breaches(integer) & records exposed(millions) in the us

2010s: Privacy-Preserving Computation

- Modern Privacy Threats
  - Cloud computing
  - Data reidentification risks
  - Adversarial AI & reconstruction attacks
- Key Technologies
  - Homomorphic Encryption (compute on encrypted data)
  - Differential Privacy (noise added to prevent re-ID)
  - Secure Multi-Party Computation (joint computation without sharing inputs)

2010s: Privacy-Preserving Computation

Trusted Execution & Federated Learning

- Trusted Execution Environments (TEEs)
  - Hardware isolations: Intel SGX, ARM TrustZone
  - Protect data during computation
  - Prevent OS, hypervisor, cloud admins from accessing sensitive data
- Cloud-Scale TEEs
  - Azure Confidential Computing
  - AWS Nitro Enclaves
- Federated Learning
  - Model sent to devices; data never leaves
  - Only model updates are shared

The Modern Day- Relevance to Modern Databases

- Threats Still Persist
  - SQL injection
  - DOS attacks
  - Mishandled data = re-identification
- Modern Breach Statistics
  - 8.2 billion+ records compromised in 2023
- Why Privacy Tech Is Needed
  - Distributed systems
  - Multi-organization collaboration
  - Cloud-computing/storage

Theorem of Access Control

RBAC – Role-Based Access Control

- **A role-centric rule**
- Permissions are assigned to roles, and roles are assigned to users.
- Benefit: Simplifies administration

ABAC – Attribute-Based Access Control

- **An attribute-centric rule**
- Access decisions are made based on attributes of user, resource, and environment.
- Benefit: Increases granularity and expressiveness

Theorem of Encryption

Definition:

-Encryption in Transit
Data transmitted between client and database is protected, preventing eavesdropping or Man-In-The-Middle attack.

-Encryption at Rest
Data stored in disks, backups, or cloud storage is encrypted.

Theorem-Confidentiality Preservation

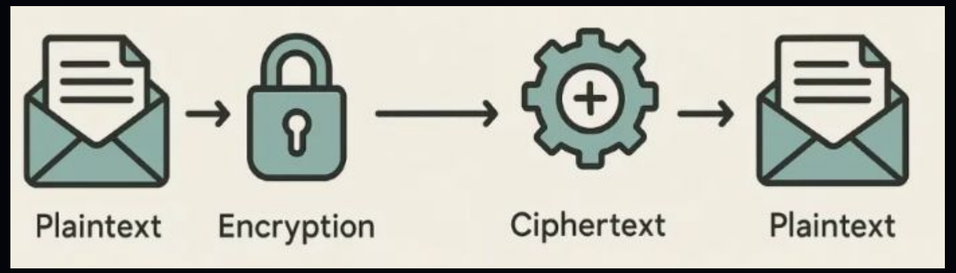
If encryption keys are properly isolated, then encryption in **transit + encryption** at rest ensures that the attacker must compromise key management to get any data from database.

Homomorphic Encryption Basics

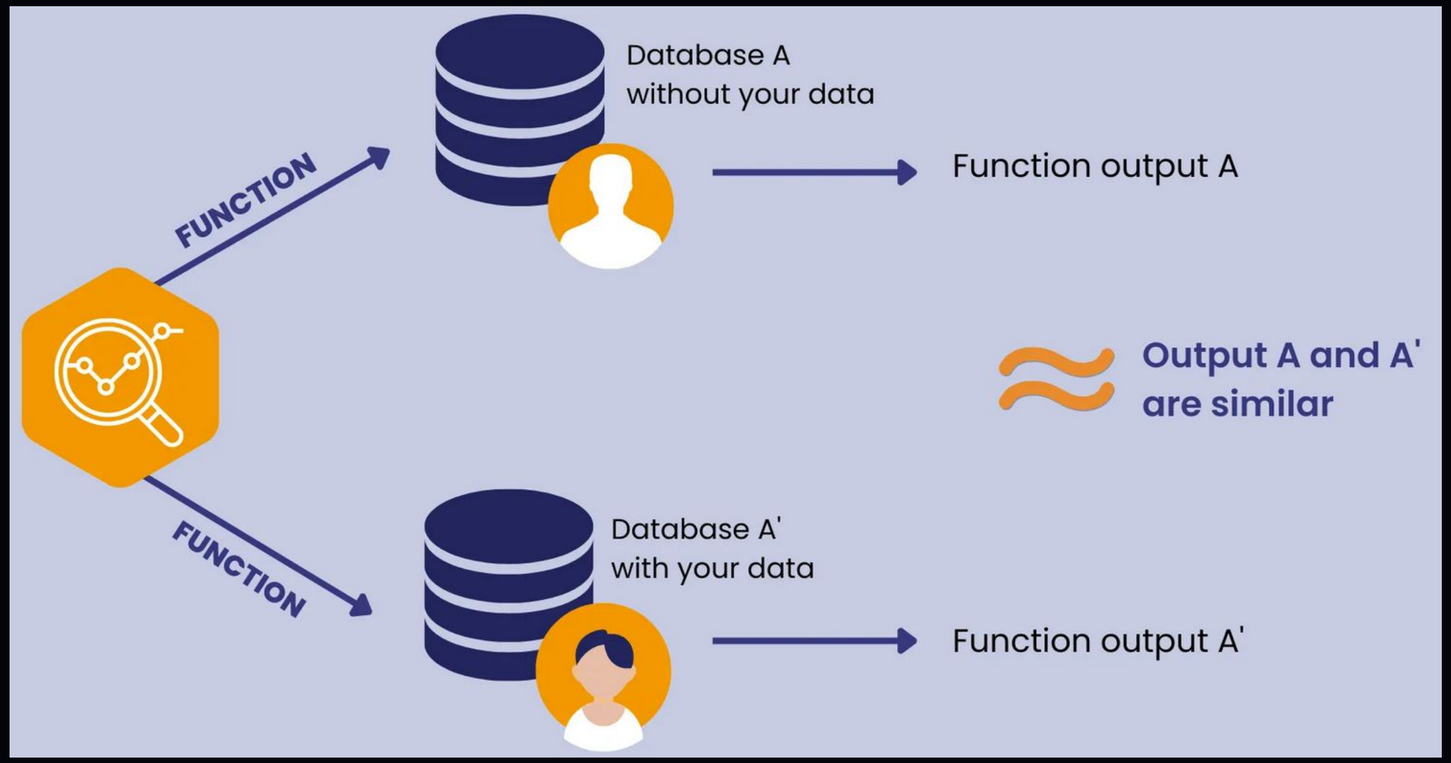
Definition:
<u>Computation on encrypted data</u>

Data can be stored and processed in encrypted form throughout the entire process.

The server will never see the plaintext data from beginning to end.



Differential Privacy Principles



- Adding controllable noise to data or results
- Maintain the overall statistical trend unchanged
- Similar result when without someone’s data

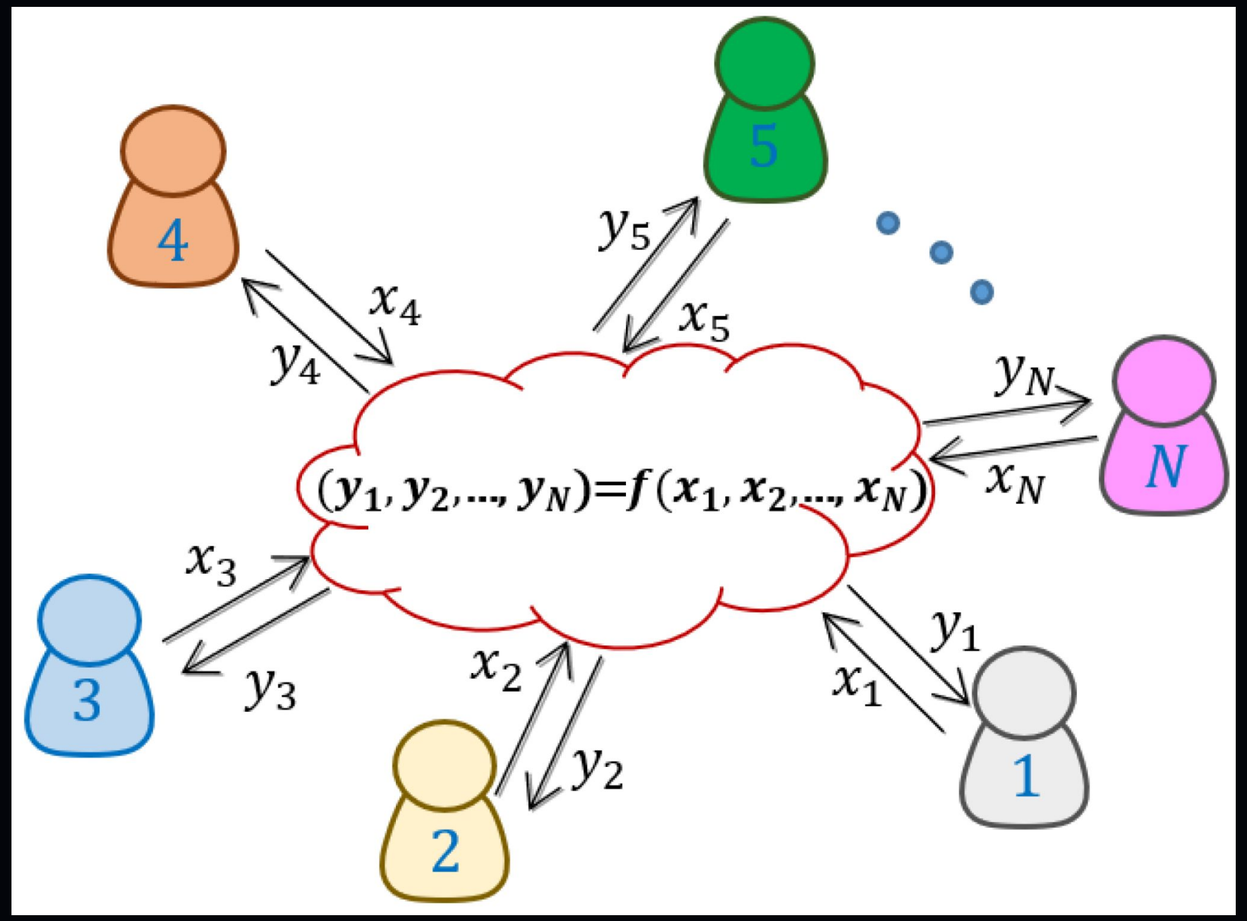
Secure Multi-party Computation

Secure Multi-Party Computation (MPC)

- Allows multiple parties to jointly compute a function without revealing their private inputs, beyond what the final output implies.

Guarantees (honest-but-curious model):

- Correctness of the joint output
- Privacy of each party's input
- No extra leakage beyond the function's result



CURRENT ADVANCES

1. Zero-Knowledge proofs
2. Synthetic data
3. Federated Learning

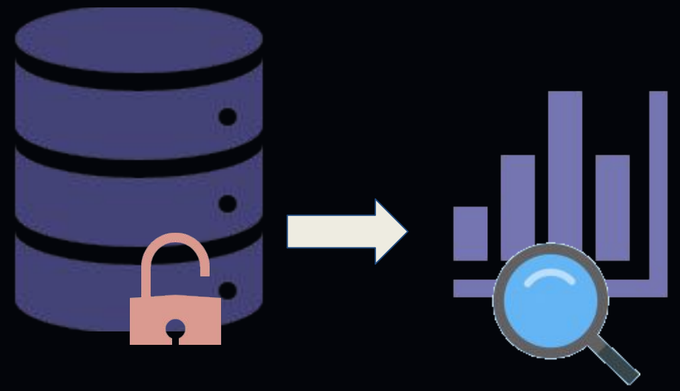
CLASSIC DATABASE SECURITY

- Protect Database servers and the connection to it
- Use access control:
  - RBAC(Role based Access Control): What you can do depends on your role
  - DAC(Discretionary Access Control): The owner grants or revoke permission
  - Or the ABAC

- On top of that, we use encryption:
  - Encryption at rest: data is encrypted while it is stored on disk or in backups
  - Encryption in transit: data is encrypted while it is moving over the network
- All of this works as long as we fully trust whoever runs the database

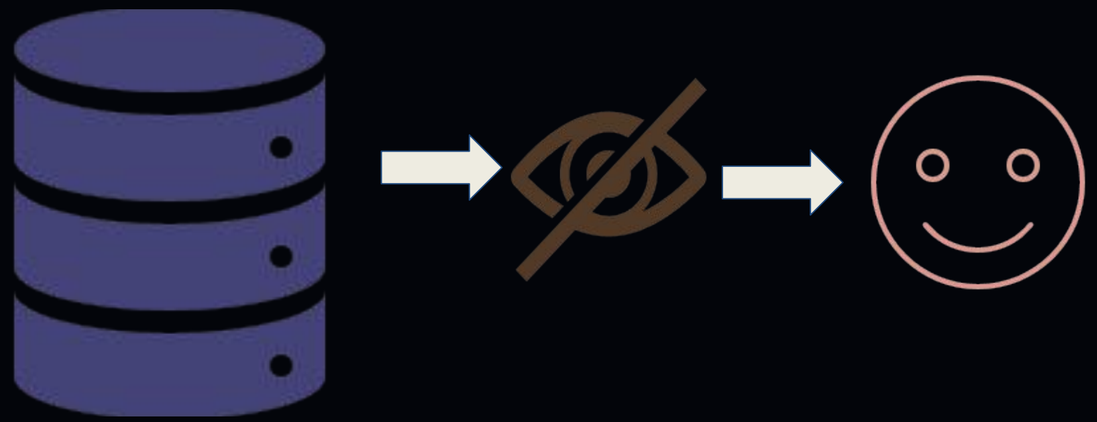
The problem with classic db security

- We decrypt the data to run queries, analytics, or train machine learning models
- This means, the data becomes fully visible to admins, insiders or even attackers who manage to get access at that point
- Modern applications do not fit this old trust model

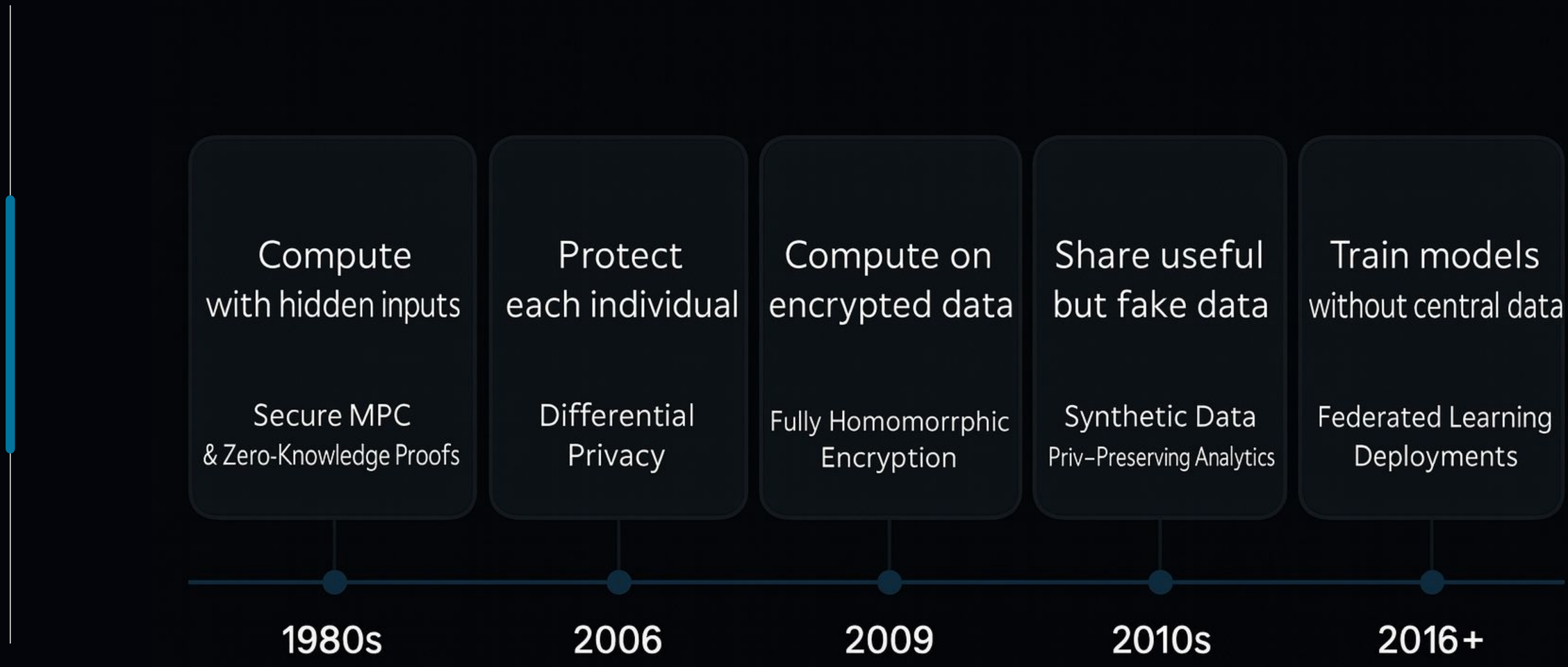


Solution

- We need techniques that let us use the data, but still hide as much as possible from the database itself and from other parties
- And that is what we mean by privacy-preserving



Evolution of privacy-preserving db tech



1. Zero-Knowledge proofs in database

- A zero-knowledge proof lets one party prove a statement about data is true, without revealing the data itself.

- In a database, a client can prove “my row satisfies this condition” (e.g., age >=18, balance >=$50) without showing the exact value stored in the table
- This allows the database or an external service to run checks or computations on data while keeping sensitive values (birthdate, balance, salary, etc.) hidden.

Zero-Knowledge proofs in practice

Normal table (plain balance)

```
Accounts(K
 user_id      INT PRIMARY KEY,
 balance      DECIMAL(10,2),
 ...}
```

ZK-friendly table (balance is hidden)

```
Accounts(
 user_id          INT PRIMARY KEY,
 balance_commitment VARBINARY(256),
-- this is c = Commit(balance, r) ..}.
```

balance_commitment is your:

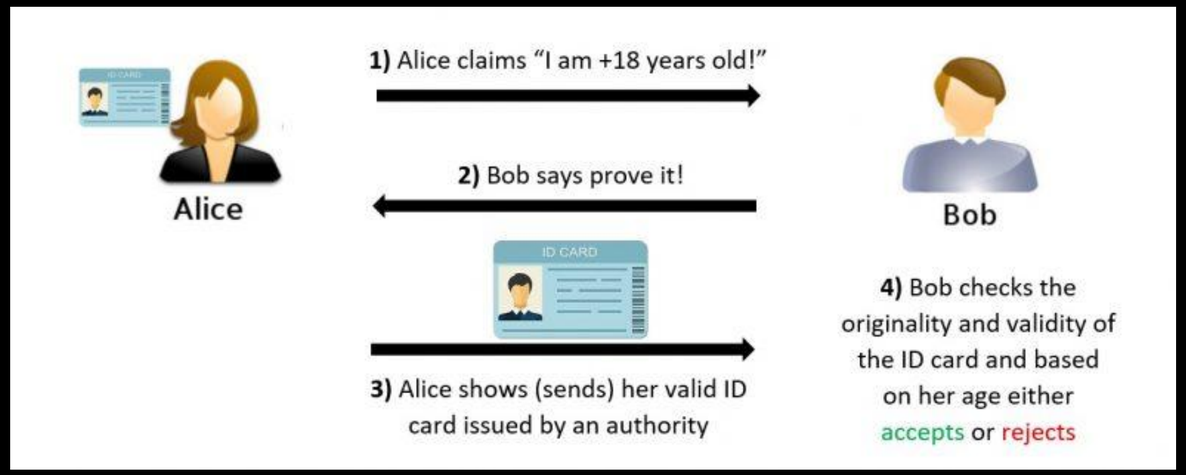
c = Commit(balance, r).

- The real balance and r stay on the client side (user's app, wallet, etc.)
- The DB only sees and stores the commitment value (looks like random bits).
- So yes: it is literally a column in a table, but the content is a commitment, not the plain value.

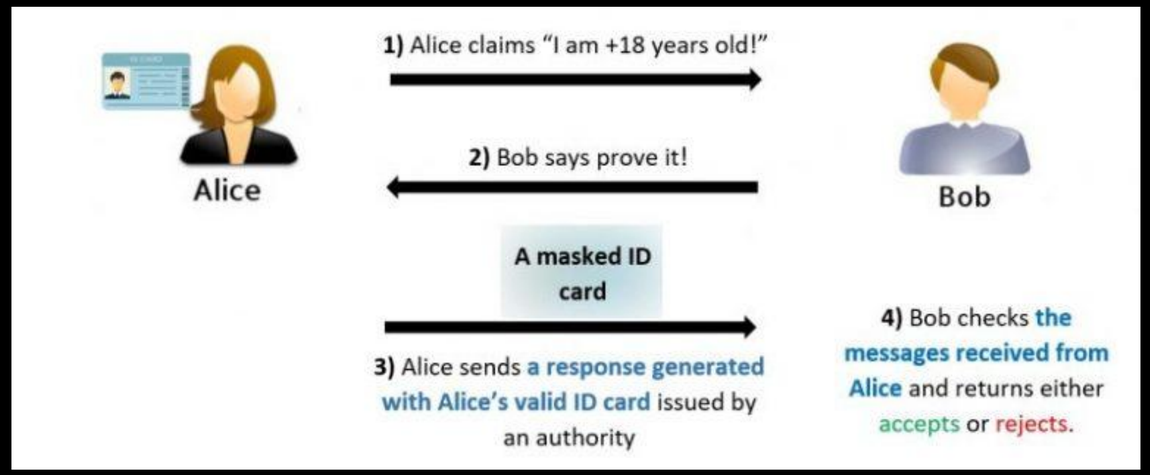
- Goal: replace a sensitive column with a commitment so the database cannot read the real value.
- Even a DB admin who sees balance_commitment cannot recover the balance, because they don't know r.

EXAMPLES

Classic check

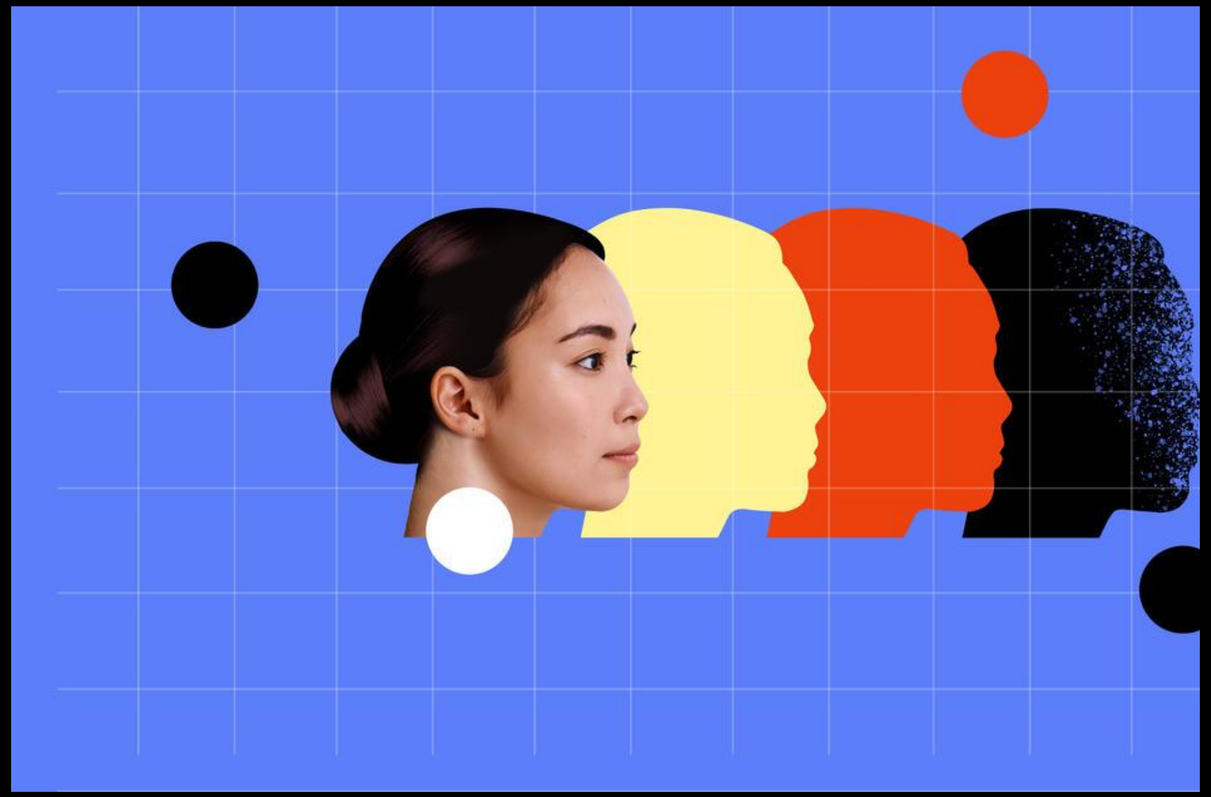


Zero-Knowledge proof



2. SYNTHETIC DATA: SHARING DATA WITHOUT SHARING PEOPLE

- **Synthetic data** = fake records generated by a model, built to follow the same patterns as the real database
- **Example use:** a hospital wants to share data with researchers or developers without exposing real patient records
- **Process:** Train a model on the real data, then use it to generate new, artificial rows (no row belong to a real person)
- **Benefit:** teams can explore, test, and build models on realistic data while reducing the risk of leaking someone's actual information



3. Federated Learning: Training models without collecting all the data together

Normally, we collect everyone's data into one big central database to train a machine-learning model.

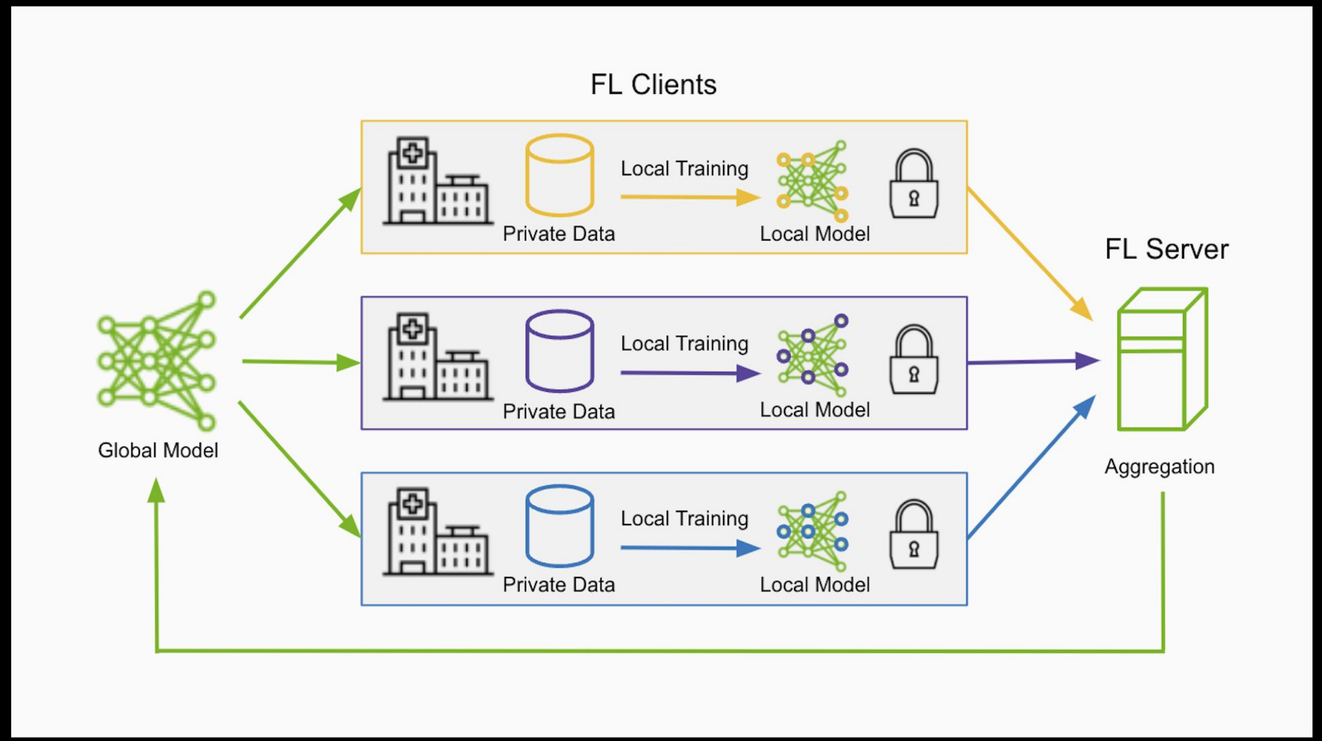
With federated learning, the data stays on each device or site (phone, hospital, school server).

The model is sent to the data, trained locally, and only small model updates are sent back – not raw records.

The server combines these updates to improve a global model, while never seeing the original data.

EXAMPLE: HOW YOUR KEYBOARD LEARNS WITHOUT SEEING YOUR MESSAGES

- **Goal:** improve next-word suggestions and autocorrect on your phone keyboard.
- **Traditional way:** Send copies of what people type to a central server and train the model there
- **Problem:** The server sees private messages, contacts, emails, and other sensitive text
- **Federated way:** your text stays on your phone; only model updates are sent back to improve a shared global model.



PRIVACY-PRESERVING ANALYTICS: USING DATA WITH LESS TRUST

Goal: Answer questions on data and train models without fully trusting any single database or server

Zero-knowledge proofs: let us prove that a condition is true (e.g., age ≥ 18) without showing the actual values used to check it.

Federated learning: keep data on users' devices or local sites, send only model updates to build a shared model

Synthetic data: share fake but realistic datasets so others can explore and test without touching real records

All these are often combined with other tools (encryption, access control, differential privacy) for stronger protections

*Take home message

CLASS ACTIVITY – CREDIT SCORE VERIFICATION (5 MINUTES)

A fine tech app wants to let users prove they are “creditworthy enough” to access a loan aggregator, without exposing their full credit report.

They consider three designs:

a. Design1:
   - The credit bureau stores encrypted credit reports.
   - When the user applies, the app sends the encrypted report to the aggregator, which decrypts it on its server, reads the full score and history, and decides.

b. Design2:
   - The bureau sends a report that only contains the credit score number (e.g., 742) and hides all individual loans, transactions, etc.
   - The aggregator sees the score and checks score ≥ 700.

c. Design 3:
   - The bureau stores a commitment to the user’s score, c = Commit(score, r).
   - When the user applies, their client generates a proof π that:
     - “The hidden score inside c is at least 700.”
   - The aggregator sees only π and a yes/no verification result; it never sees the exact score.

Questions:

1. For each design (1, 2, 3), answer Yes/No:
   - Does the verifier (aggregator) learn the exact score?
   - Does the design match the zero-knowledge spirit (prove “score ≥ 700” without revealing the score)?
2. Design 2 hides most of the report and only shows the credit score number. In 1–2 sentences, explain whether this design should be considered *zero-knowledge* or not, and justify your answer.

ANSWERS

Question 1:

a. Design 1:
   - Learns exact score? Yes.
   - Zero-knowledge? No. The server decrypts and sees the full score + history.

b. Design 2:
   - Learns exact score? Yes…742 (or whatever).
   - Zero-knowledge? No. The verifier learns more than just “score ≥ 700”; it learns the exact score. This is partial disclosure, not ZK.

c. Design 3:
   - Learns exact score? No.
   - Zero-knowledge? Yes. The verifier only learns that score ≥ 700 , not the score itself.

Question 2:

Design 2 still reveals the exact score value. The verifier can distinguish between 701 and 850, even though it only needs to know “score ≥ 700”. Zero-knowledge aims to reveal only that the condition is true, not the full underlying value.

Why Privacy in Data Matters Today

01 Rise of sensitive data collection across healthcare, finance and tech

02 Advancement in sophisticated threats: ransomware, insider attacks, AI-powered exploits

03 Regulations to meet threats: HIPAA, GDPR, CCPA

04 Consumer trust depends on data protection practices

Apple’s Use of Differential Privacy in User Analytics

What is differential privacy?

- Protects individual privacy in aggregate statistics
- Data transformed before leaving user’s device
- Enables learning about populations without exposing individuals

Apple’s Use of Differential Privacy

- Emoji usage patterns and keyboard predictions
- Health app trends and activity patterns
- Safari browsing data and search suggestions
- Apple Intelligence features (Genmoji, Writing Tools)

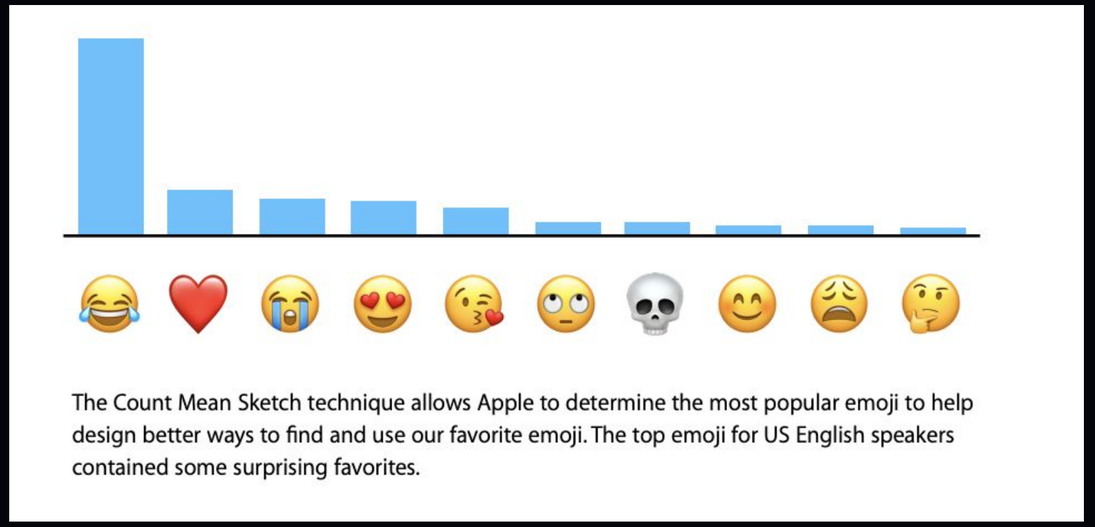


Image from Apple’s Differential Privacy Document

Data Protection in Modern Healthcare

Case Study: Mediguard Framework

The Challenge:

Getting accurate diagnoses from AI while completely hiding sensitive patient information.

The Solution:

Mediguard's system relies on "adaptive information obfuscation", which dynamically fuzzes out patient data while keeping enough context to solve complex problems.

Healthcare organizations must balance data accessibility for treatment with stringent privacy requirements under HIPAA and GDPR.

- Encryption at rest and in transit
- Role-based access control (RBAC)
- Data fingerprinting for PHI identification

Cloud-Based Healthcare Protection

- Digital Guardian DLP scans cloud files for PHI
- Automated remediation when sensitive data detected
- Separate folders for shareable vs. restricted data
- Real-time monitoring of cloud storage access

Detecting Financial Fraud While Preserving Data Privacy

FED-RD: Federated Learning for Finance

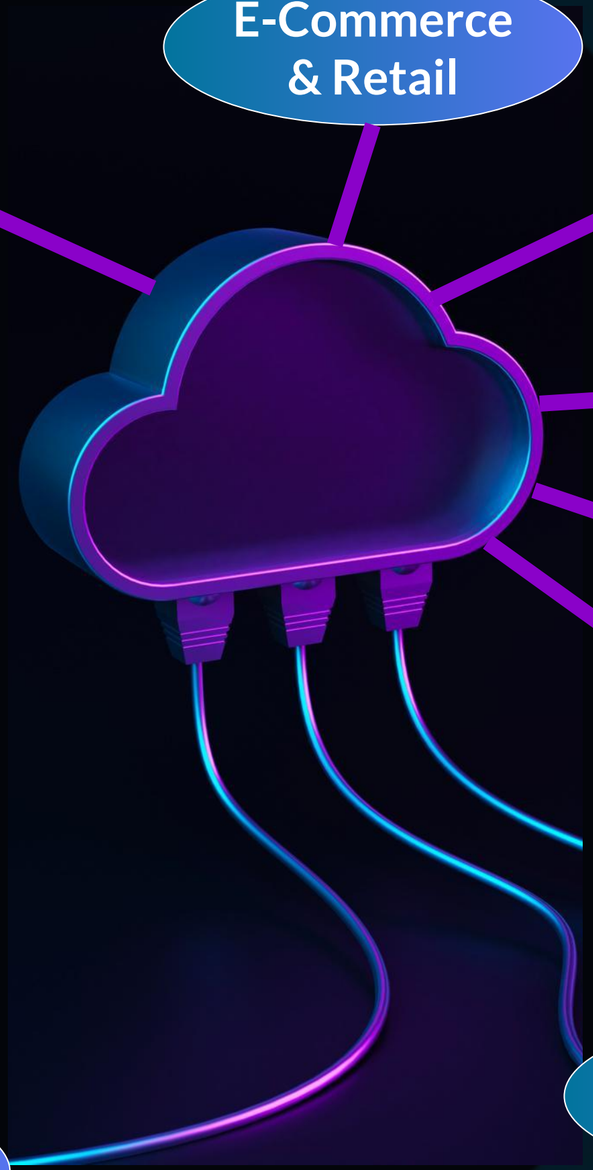
- Handles data partitioned across multiple banks
- Secure multiparty computation (MPC) for aggregation
- Minimal accuracy loss as privacy increases
- Detects anomalous transactions across institutions

How can we detect fraud while preserving privacy?

- Federated Learning allows banks to collaborate without sharing raw data
- Differential Privacy adds noise to protect individual transactions
- Synthetic data generation enables model training
- Maintains accuracy while preventing data exposure

Securing Cloud Database Systems

- Multi-factor authentication (MFA) for access control
- Encryption at rest and in transit (TLS/SSL)
- Database firewalls and intrusion detection
- Regular security suits and compliance monitoring



Manufacturing

E-Commerce & Retail

Healthcare

Finance

Education

Government

Tech & Software

Media & Entertainment

Privacy Practices in Relational Databases

01 Data masking and anonymization techniques

02 Query auditing and activity monitoring

03 Separation of sensitive data on different servers

04 Regular backups with encrypted storage

Business & Societal Impact

Consumer trust drives competitive advantage:

Studies show people will pay more for products from companies they trust with their data.

Fines for violating GDPR up to €20M, and HIPAA up to $1.5M:

Non-compliance penalties are severe—GDPR fines reach up to €20M or 4% of global revenue (whichever is higher), while HIPAA violations can cost up to $1.5M per violation per year.

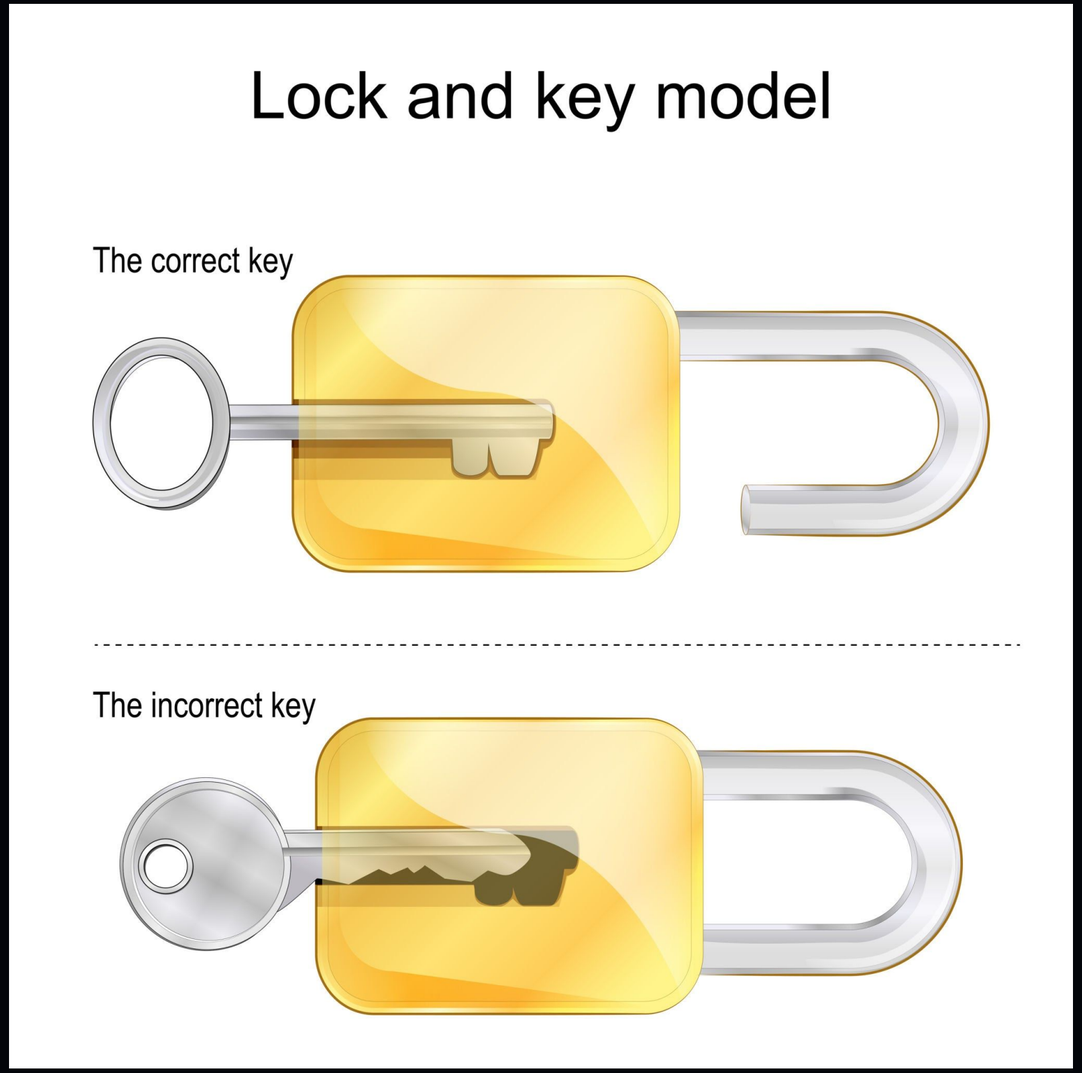
It’s important to enable data-driven innovation responsibly:

Privacy-preserving technologies unlock responsible innovation by allowing researchers to analyze healthcare trends and banks to collaborate on fraud detection without violating privacy laws.

Challenges in AI & Privacy

- AI Models as “black boxes” complicate transparency
- Massive data requirements increase exposure to risk
- Algorithm bias perpetuates privacy concerns
- Cross-jurisdiction compliance complexity
- Quantum computing threats to current encryption algorithms
- Balancing surveillance needs with ethical data use
- Rapidly evolving regulatory requirements
- Purpose specification challenged by secondary data use

SQL Injection



What is SQL Injection (SQLi)?

Definition: SQL Injection is a vulnerability that occurs when an attacker can interfere with the queries that an application makes to its database.

The Goal: To trick the application's database into executing commands that were not intended by the developer, typically to view, modify, or delete data.

Key Concept: The application mistakes **user input** for **SQL code**.

SQL Injections?

' OR '1'='1 –

%' UNION SELECT username FROM users --

1 ORDER BY 3

0 UNION SELECT username FROM users

admin' –

0 UNION SELECT email FROM users

Thank You

Please see our GitHub for slideshow content: