

# **Security and Data Privacy**

Instructor: Matei Zaharia

# Outline

Security requirements

Key concepts and tools

Differential privacy

Other security tools

# Outline

Security requirements

Key concepts and tools

Differential privacy

Other security tools

# Why Security & Privacy?

Data is valuable & can cause harm if released

- » Example: medical records, purchase history, internal company documents, etc

Data releases can't usually be “undone”

Security policies can be complex

- » Each user can only see data from their friends
- » Analyst can only query aggregate data
- » Users can ask to delete their derived data

# Why Security & Privacy?

It's the law! new regulations about user data:

**US HIPAA:** Health Insurance Portability & Accountability Act (1996)

» Mandatory encryption, access control, training

**EU GDPR:** General Data Protection Regulation (2018)

» Users can ask to see & delete their data

**PCI:** Payment Card Industry standard (2004)

» Required in contracts with MasterCard, etc

# Consequence

Security and privacy must be baked into the design of data-intensive systems

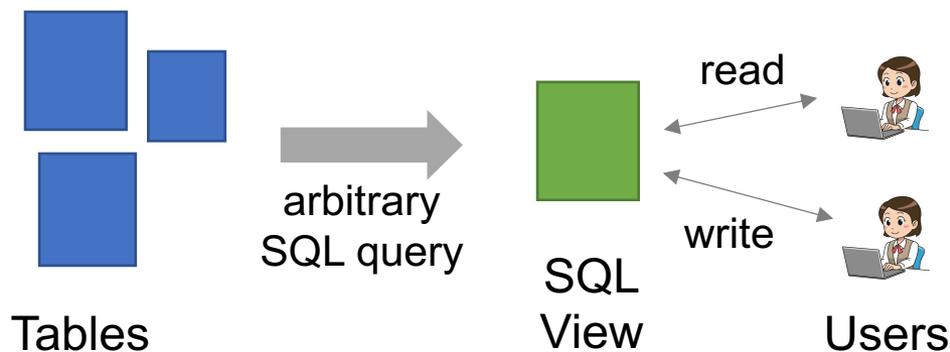
» Often a key differentiator for products!

# The Good News

**Declarative** interface to many data-intensive systems can enable powerful security features

» One of the “big ideas” in our class!

Example: System R’s access control on views



# Outline

Security requirements

Key concepts and tools

Differential privacy

Other security tools

# Some Security Goals

**Access Control:** only the “right” users can perform various operations; typically relies on:

- » **Authentication:** a way to verify user identity (e.g. password)
- » **Authorization:** a way to specify what users may take what actions (e.g. file permissions)

**Auditing:** system records an incorruptible audit trail of who did each action

# Some Security Goals

**Confidentiality:** data is inaccessible to external parties (often via cryptography)

**Integrity:** data can't be modified by external parties

**Privacy:** only a limited amount of information about “individual” users can be learned

# Clarifying These Goals

Say our goal was **access control**: only Matei can set CS 245 student grades on Axxess

What scenarios should Axxess protect against?

1. Bobby T. (an evil student) logging into Axxess as himself and being able to change grades
2. Bobby sending hand-crafted network packets to Axxess to change his grades
3. Bobby getting a job as a DB admin at Axxess
4. Bobby guessing Matei's password
5. Bobby blackmailing Matei to change his grade
6. Bobby discovering a flaw in AES to do #2

# Threat Models

To meaningfully reason about security, need a **threat model**: what adversaries may do

- » Same idea as failure models!

For example, in our Axxess scenario, assume:

- » Adversaries only interact with Axxess through its public API
- » No crypto algorithm or software bugs
- » No password theft

Implementing complex security policies can be hard even with these assumptions!

# Threat Models

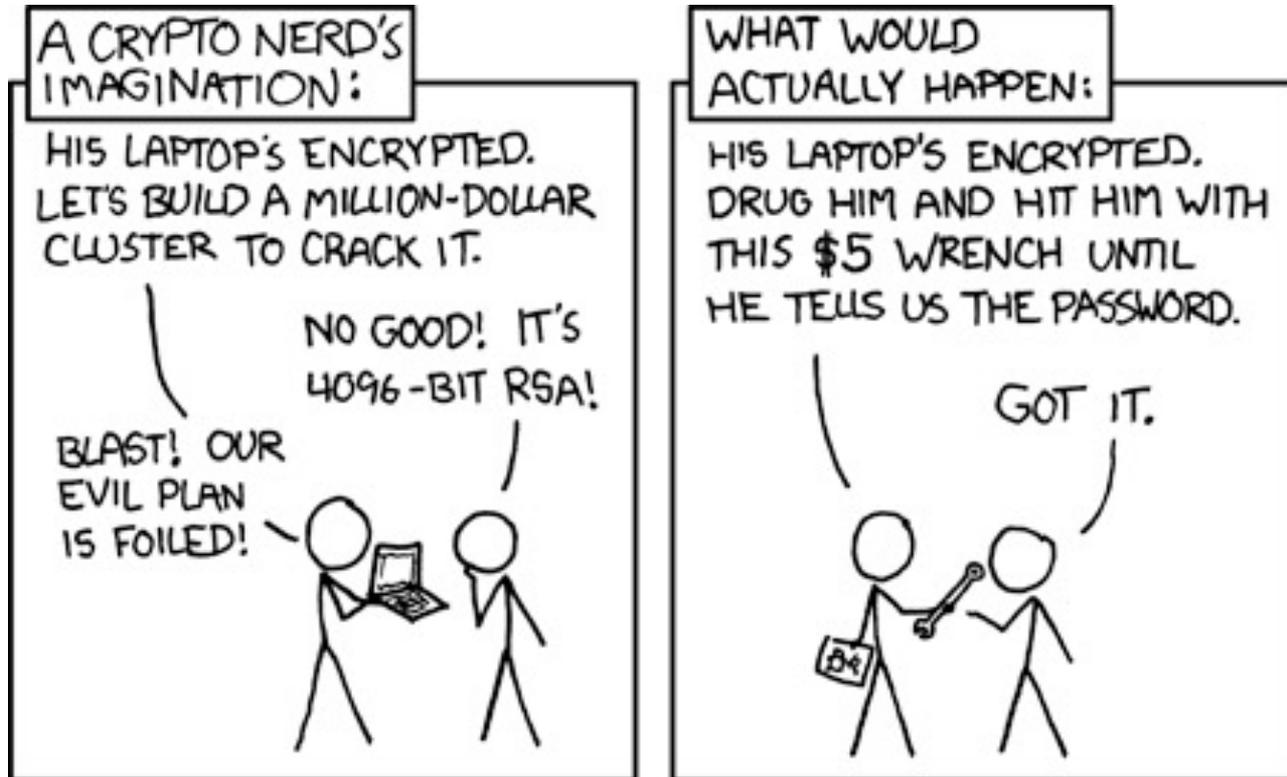
No useful threat model can cover everything

- » Goal: cover the most feasible scenarios for adversaries to increase the **cost** of attacks

Threat models also let us divide security tasks across different components

- » E.g. auth system handles passwords, 2FA

# Threat Models



# Useful Building Blocks

**Encryption:** encode data so that only parties with a key can efficiently decrypt

**Cryptographic hash functions:** hard to find items with a given hash (or collisions)

**Secure channels (e.g. TLS):** confidential, authenticated communication for 2 parties

# Security in a Typical DBMS

First-class concept of users + access control

» Views as in System R, tables, etc

Secure channels for network communication

Audit logs for analysis

Encrypt data on-disk (perhaps at OS level)

# Emerging Ideas for Security

Privacy metrics and enforcement thereof  
(e.g. differential privacy)

Computing on encrypted data (e.g. CryptDB)

Hardware-assisted security (e.g. enclaves)

Multi-party computation (e.g. secret sharing)

# Outline

Security requirements

Key concepts and tools

Differential privacy

Other security tools

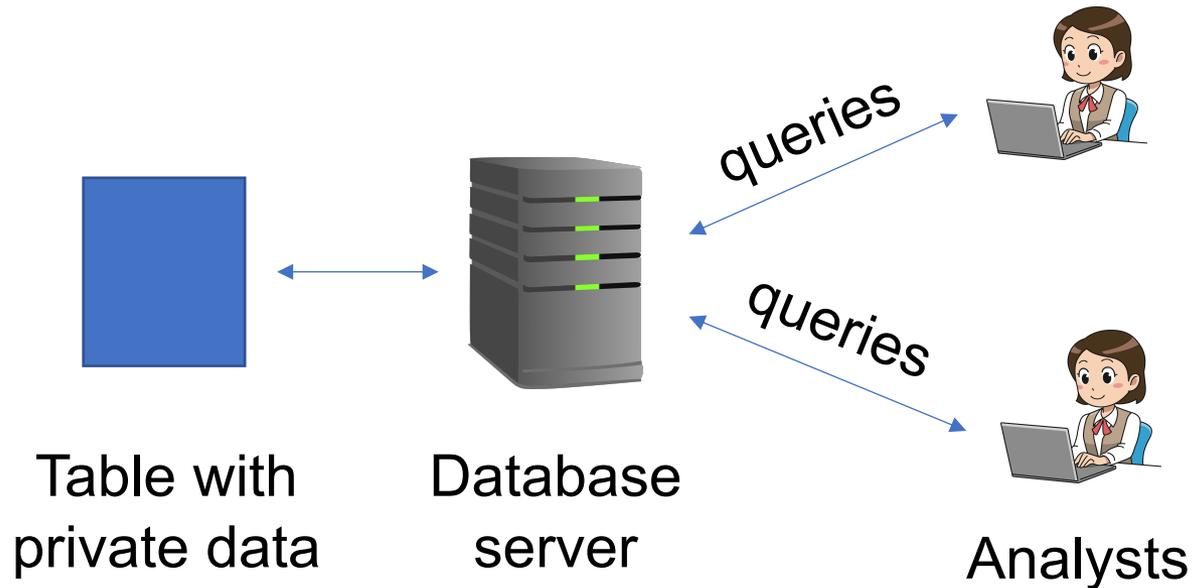
# Motivation

Many applications can be built on user data, but how to make sure that analysts with access to data don't see personal secrets?

**Example:** what word is most likely to be typed after “Want to grab” in a text message?  
» Need peoples' texts but they're very sensitive!

**Example:** what's the most common diagnosis for hospital patients aged <40 in Palo Alto?

# Threat Model



- » Database software is working correctly
- » Adversaries only access it through public API
- » Adversaries have limited # of user accounts

# How to Define Privacy?

This is conceptually very tricky! How to distinguish between

```
SELECT TOP(disease) FROM patients WHERE  
state="California"
```

and

```
SELECT TOP(disease) FROM patients WHERE  
name="Matei Zaharia"
```

# How to Define Privacy?

Also want to defend against adversaries who have some side-information; for instance:

```
SELECT TOP(disease) FROM patients WHERE  
birth_year="19XX" AND gender="M" AND  
born_in="Romania" AND ...
```

 Side information about Matei

Also consider adversaries who do multiple queries (e.g. subtract 2 results)

# Robust De-anonymization of Large Sparse Datasets

Arvind Narayanan and Vitaly Shmatikov

The University of Texas at Austin

## Abstract

*We present a new class of statistical de-anonymization attacks against high-dimensional micro-data, such as individual preferences, recommendations, transaction records and so on. Our techniques are robust to perturbation in the data and tolerate some mistakes in the adversary's background knowledge.*

*We apply our de-anonymization methodology to the Netflix Prize dataset, which contains anonymous movie ratings of 500,000 subscribers of Netflix, the world's largest online movie rental service. We demonstrate that an adversary who knows only a little bit about an individual subscriber can easily identify this subscriber's record in the dataset. Using the Internet Movie Database as the source of background knowledge, we successfully identified the Netflix records of known users, uncovering their apparent political preferences and other potentially sensitive information.*

## 1 Introduction

Datasets containing *micro-data*, that is, information about specific individuals, are increasingly becoming public in response to “open government” laws and to support data mining research. Some datasets include legally protected information such as health histories; others contain individual preferences and transactions, which many people may view as private or sensitive.

Privacy risks of publishing micro-data are well-known. Even if identifiers such as names and Social

and sparsity. Each record contains many attributes (*i.e.*, columns in a database schema), which can be viewed as dimensions. Sparsity means that for the average record, there are no “similar” records in the multi-dimensional space defined by the attributes. This sparsity is empirically well-established [7, 4, 19] and related to the “fat tail” phenomenon: individual transaction and preference records tend to include statistically rare attributes.

**Our contributions.** Our first contribution is a formal model for privacy breaches in anonymized micro-data (section 3). We present two definitions, one based on the probability of successful de-anonymization, the other on the amount of information recovered about the target. Unlike previous work [25], we do not assume *a priori* that the adversary's knowledge is limited to a fixed set of “quasi-identifier” attributes. Our model thus encompasses a much broader class of de-anonymization attacks than simple cross-database correlation.

Our second contribution is a very general class of de-anonymization algorithms, demonstrating the fundamental limits of privacy in public micro-data (section 4). Under very mild assumptions about the distribution from which the records are drawn, the adversary with a small amount of background knowledge about an individual can use it to identify, with high probability, this individual's record in the anonymized dataset and to learn all anonymously released information about him or her, including sensitive attributes. For *sparse* datasets, such as most real-world datasets of individual transactions, preferences, and recommendations, very little background knowledge is needed (as few as 5-10 attributes in our case study). Our de-anonymization algorithm is *robust*

# Differential Privacy

Privacy definition that tackles these concerns and others by looking at **possible databases**

» Idea: results that an adversary saw should be “nearly as likely” for a database without Matei

Definition: a randomized algorithm  $M$  is  $\epsilon$ -differentially private if for all  $S \subseteq \text{Range}(M)$ ,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] e^{\epsilon \cdot |A \oplus B|}$$

Number of records that differ in sets  $A$  and  $B$

# Equivalent Definition

A randomized algorithm  $M$  is  $\epsilon$ -differentially private if for all  $S \subseteq \text{Range}(M)$  and all sets  $A, B$  that differ in 1 element,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] e^\epsilon$$

# What Does It Mean?

Say an adversary runs some query and observes a result  $X$

Adversary had some set of results,  $S$ , that lets them infer something about  $\text{Mate}_i$  if  $X \in S$

Then:

$$\Pr[X \in S \mid \text{Mate}_i \in \text{DB}] \leq e^\epsilon \Pr[X \in S \mid \text{Mate}_i \notin \text{DB}]$$

$\approx 1 + \epsilon$

and

$$\Pr[X \notin S \mid \text{Mate}_i \in \text{DB}] \leq e^\epsilon \Pr[X \notin S \mid \text{Mate}_i \notin \text{DB}]$$

Similar outcomes whether or not  $\text{Mate}_i$  in DB

# What Does It Mean?

Example (assume  $\epsilon=0.1$ ):

```
SELECT TOP(diagnosis) FROM patients WHERE age=XX  
AND city="Palo Alto" → flu
```

```
SELECT TOP(diagnosis) FROM patients WHERE age=XX  
AND city="Palo Alto" AND born="Romania" → drug overdose
```

Does this mean Matei specifically took drugs?

- » Result would have been nearly as likely (within 10%) even if Matei were not in the database
- » Could be we just got a low-probability result
- » Could be *most* Romanians do drugs (no info on Matei)

# Some Nice Properties of Differential Privacy

**Composition:** can reason about the privacy effect of multiple (even dependent) queries

Let queries  $M_i$  each provide  $\epsilon_i$ -differential privacy; then the sequence of queries  $\{M_i\}$  provides  $(\sum_i \epsilon_i)$ -differential privacy

Proof:

$$\Pr[\forall i M_i(A)=r_i] \leq e^{(\epsilon_1+\dots+\epsilon_n)|A\oplus B|} \Pr[\forall i M_i(B)=r_i]$$

Adversary's ability to distinguish DBs A & B grows in a bounded way with each query

# Some Nice Properties of Differential Privacy

**Parallel composition:** even better bounds if queries are on disjoint subsets

Let  $M_i$  each provide  $\epsilon$ -differential privacy and read disjoint subsets of the data; then the set of queries  $\{M_i\}$  provides  $\epsilon$ -differential privacy

Example: query both average patient age in CA and average patient age in NY

# Some Nice Properties of Differential Privacy

**Easy to compute:** can use known results for various operators, then compose for a query

- » Enables systems to **automatically** compute privacy bounds given declarative queries!

# Disadvantages of Differential Privacy

# Disadvantages of Differential Privacy

Each user can only make a limited number of queries (more precisely, limited total  $\epsilon$ )

- » Their  $\epsilon$  grows with each query and can't shrink
- » Some methods bound total  $\epsilon$  but limit query types

How to set  $\epsilon$  in practice?

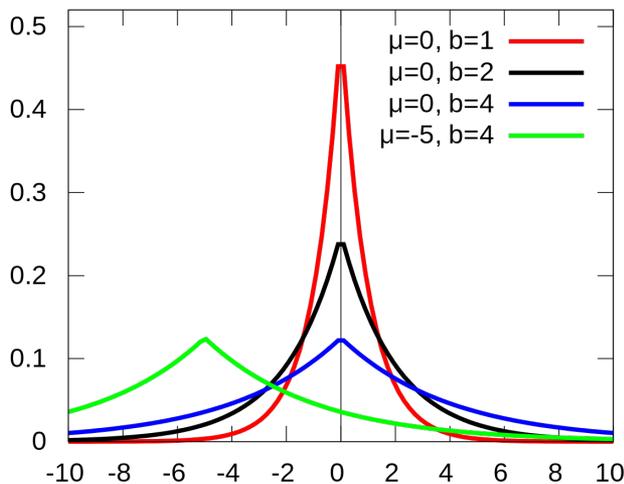
- » Hard to tell what various values mean, though there is a nice Bayesian interpretation
- » Apple set  $\epsilon=6$  and researchers said it's too high

Can't query using arbitrary code (must know  $\epsilon$ )

# Computing Differential Privacy Bounds

Let's start with COUNT aggregates:  
`SELECT COUNT(*) FROM A`

The randomized algorithm  $M(A)$  that returns  $|A| + \text{Laplace}(1/\epsilon)$  is  $\epsilon$ -differentially private



Laplace( $b$ ) distribution:  
$$p(x) = 1/(2b) e^{-|x|/b}$$

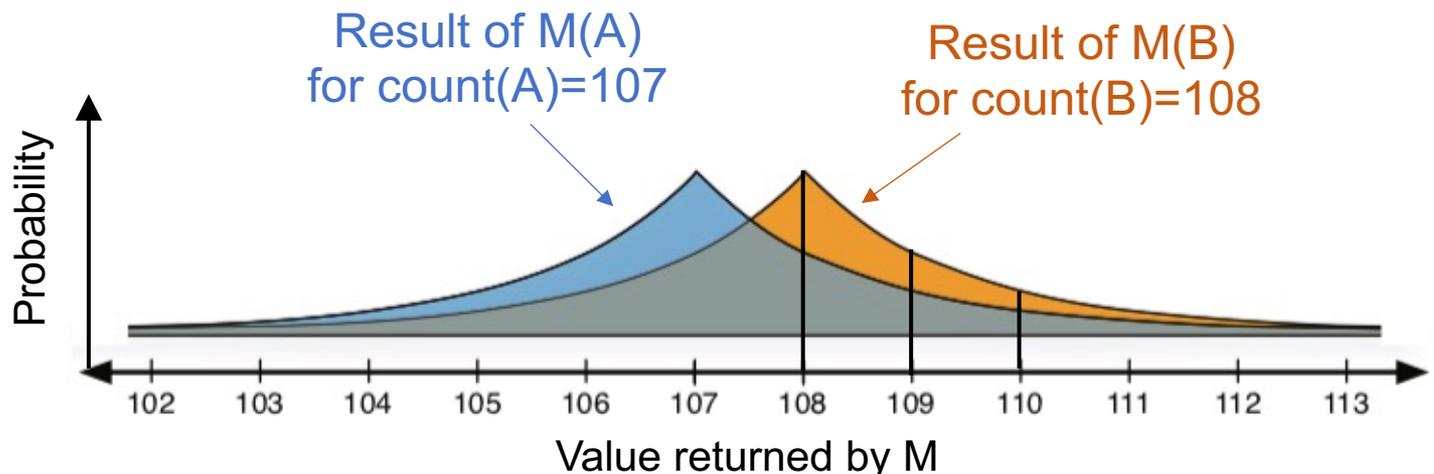
Mean: 0

Variance:  $2b^2$

# Computing Differential Privacy Bounds

Let's start with COUNT aggregates:  
`SELECT COUNT(*) FROM A`

The randomized algorithm  $M(A)$  that returns  $|A| + \text{Laplace}(1/\epsilon)$  is  $\epsilon$ -differentially private



# Computing Differential Privacy Bounds

What about AVERAGE aggregates:  
`SELECT AVERAGE(x) FROM A`

# Computing Differential Privacy Bounds

What about AVERAGE aggregates:  
`SELECT AVERAGE(x) FROM A`

How much can one element of A affect result?

- » In general case, unboundedly! No privacy
  - `SELECT AVG(wealth) WHERE city="Omaha, NB"`
- » If  $x \in [0, m]$  for all  $x$  in A, then by at most  $m$ 
  - Adding Laplace( $m/\epsilon$ ) noise is  $\epsilon$ -differentially private

Paper bounds AVG, SUM for values  $x \in [-1, 1]$

# Computing Differential Privacy Bounds

General notion to capture the impact of one element: **sensitivity**

Sensitivity of a function  $f: U \rightarrow \mathbb{R}$  on sets is

$$\Delta f = \max_{A, B \in U \text{ differ in 1 element}} |f(A) - f(B)|$$

# Sensitivity Examples

	Sensitivity
$f(A) =  A $	1
$f(A) = \text{sum}(A), x \in [0, m] \forall x \in A$	m
$f(A) = \text{avg}(A), x \in [0, m] \forall x \in A$	m
$f(A) =  \{x \in A \mid x \text{ is male}\} $	1
$f(A) =  A \bowtie B $	unbounded
$f(A) =  A \bowtie B $ , each key has $\leq k$ matches	k

# Multi-dimensional Sensitivity

Can also define sensitivity for functions that return multiple numerical results:

Sensitivity of a function  $f: U \rightarrow \mathbb{R}^d$  on sets is

$$\Delta f = \max_{A, B \in U \text{ differ in 1 element}} \|f(A) - f(B)\|_1$$

Example:  $f$  fits a linear model to the data...

# Computing Differential Privacy Bounds

Another concept, used to reason about set transformations in PINQ: **stability**

A function  $T$  on sets is  $c$ -stable if for any two input sets  $A$  and  $B$ ,

$$|T(A) \oplus T(B)| \leq c |A \oplus B|$$

Number of records that differ in  $A$  and  $B$



**PINQ's approach:** let user do any # of set ops; compute their stability; then let them do one aggregate op and compute its sensitivity

# Stability Examples

	Stability
$T(A) = \sigma_{\text{predicate}}(A)$ (“Where”)	1
$T(A) = \pi_{\text{exprs}}(A)$ (“Select”)	1
$T(A, B) = A \cup B$	1
$T(A) = \text{GroupBy}(A, \text{expr})$ (retruns 1 record/group)	2
$T(A) = A \bowtie B$ limited to at most 1 match per key	1

# Partition Operator

Partition(dataset, key\_list) returns a set of IQueryables: one for each key in your list

- » User give the desired keys in advance (e.g. “CA” or “NY”); can’t use to discover keys
- » Lets PING use **parallel composition** rule since the sets returned are all disjoint

Stability = 1

# Analyzing Queries in PINQ

User calls multiple set transformation ops and finally one aggregation/result op

» Transformations are lazy; can't see result

PINQ computes stability of set ops and multiplies by sensitivity of each aggregate to get total sensitivity

User provides an  $\epsilon$  to aggregate; PINQ adds noise proportional to sensitivity/ $\epsilon$

# Putting It All Together

---

**Example 5** Measuring query frequencies in PINQ.

---

```
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data  = new PINQQueryable<string>(rawdata, agent);

// break out fields, filter by query, group by IP
var users = data.Select(line => line.Split(','))
                .Where(fields => fields[20] == args[0])
                .GroupBy(fields => fields[0]);

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + ": " + users.NoisyCount(0.1));
```

---

cricket: 127123.313

# Putting It All Together

---

## Example 7 Transforming IP addresses to coordinates.

---

```
// ... within the per-query loop, from before ...  
  
// use the searches for query, group by IP address  
var users = parts[query].GroupBy(fields => fields[0]);  
  
// extract IP address from each group, and match  
var coords = users.Join(iplatlon,  
                        group => group.Key,  
                        entry => entry[0],  
                        (glist,elist) => elist.First());
```

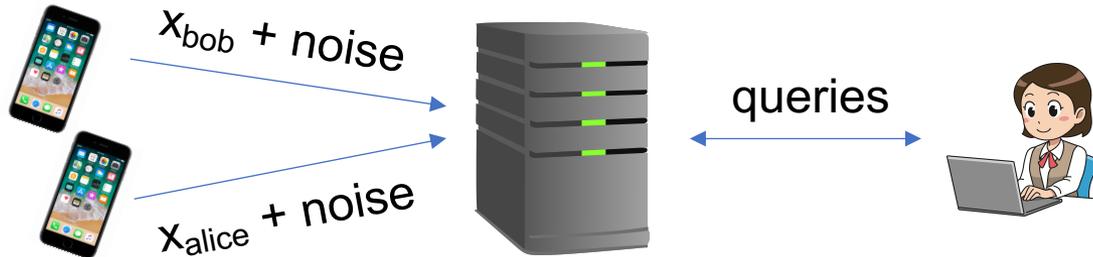
---



# Uses of Differential Privacy

Statistics collection about iOS features

“Randomized response”: clients add noise to data they send instead of relying on provider



Research systems that use DP to measure security (e.g. Vuvuzela messaging)

# Outline

Security requirements

Key concepts and tools

Differential privacy

Other security tools

# Computing on Encrypted Data

**Threat model:** adversary has access to the database server we run on (e.g. in cloud)

**Idea:** some encryption schemes allow computing on data without decrypting it:

$$f_{\text{enc}}(\text{Enc}(X)) = \text{Enc}(f(X))$$

Usually very expensive, but can be done efficiently for some functions  $f$ !

# Example Systems

CryptDB, Mylar (MIT research projects)

Encrypted BigQuery (CryptDB on BigQuery)

Leverage properties of SQL to come up with efficient encryption schemes & query plans

# Example Schemes

Equality checks with deterministic encryption

```
SELECT * FROM table WHERE state="CA"
```



Encrypt "state" column

```
SELECT * FROM table WHERE state="XAYDS9"
```

# Example Schemes

Equality checks with deterministic encryption

```
SELECT * FROM table WHERE state="CA"
```



Encrypt "state" column

```
SELECT * FROM table WHERE state="XAYDS9"
```

Potential challenges with this scheme:

- » Adversary can see relative frequency of keys
- » Adversary sees which keys are accessed on each query (e.g. Matei logs in → CA key read)

# Other Encryption Schemes

Additive homomorphic encryption:

$$\text{Enc}(A + B) = \text{Enc}(A) \otimes \text{Enc}(B)$$

Fully homomorphic encryption:

$$\text{Enc}(f(A)) = f_{\text{enc}}(\text{Enc}(A))$$

Possible but very expensive  
( $10^8$  or more overhead)

Order-preserving encryption:

$$\text{if } A < B \text{ then } \text{Enc}(A) < \text{Enc}(B)$$

# Hardware Enclaves

**Threat model:** adversary has access to the database server we run on (e.g. in cloud) but can't tamper with hardware

**Idea:** CPU provides an “enclave” that can provably run some code isolated from the OS

- » Enclaves returns a certificate signed by CPU maker that it ran code C on argument A

# Hardware Enclaves in Practice

Already present in all Intel CPUs (Intel SGX),  
and many Apple custom chips (T2, etc)

Initial applications were digital rights mgmt.,  
secure boot, secure login

» Protect even against a compromised OS

Some research systems explore using these  
for data analytics: Opaque, ObliDB, others

# Databases + Enclaves

1. Store data encrypted with an encryption scheme that leaks nothing (randomized)
2. With each query, user includes a public key  $k_q$  to encrypt the result with
3. Database runs a function  $f$  in the enclave that does query and encrypts result with  $k_q$
4. User can verify  $f$  ran, DB can't see result!

Performance is fast too (normal CPU speed)!

# Are Enclaves Enough to Secure Against Non-HW Adversaries?

# Are Enclaves Enough to Secure Against Non-HW Adversaries?

No! adversary can still get info by observing **access patterns** to RAM or **timing**

» Similar to some attacks on encrypted DBs

**Oblivious algorithms** can help prevent this but add more computational cost

» Oblivious = same access pattern regardless of underlying data, query result, etc

# Multi-Party Computation (MPC)

**Threat model:** participants  $p_1, \dots, p_n$  want to compute some joint function  $f$  of their data but don't trust each other

» E.g. patient stats across 2 hospitals

**Idea:** protocols that compute  $f$  without revealing anything else to participants

» Like with encryption, general computations are possible but expensive

# Example: Secret Sharing

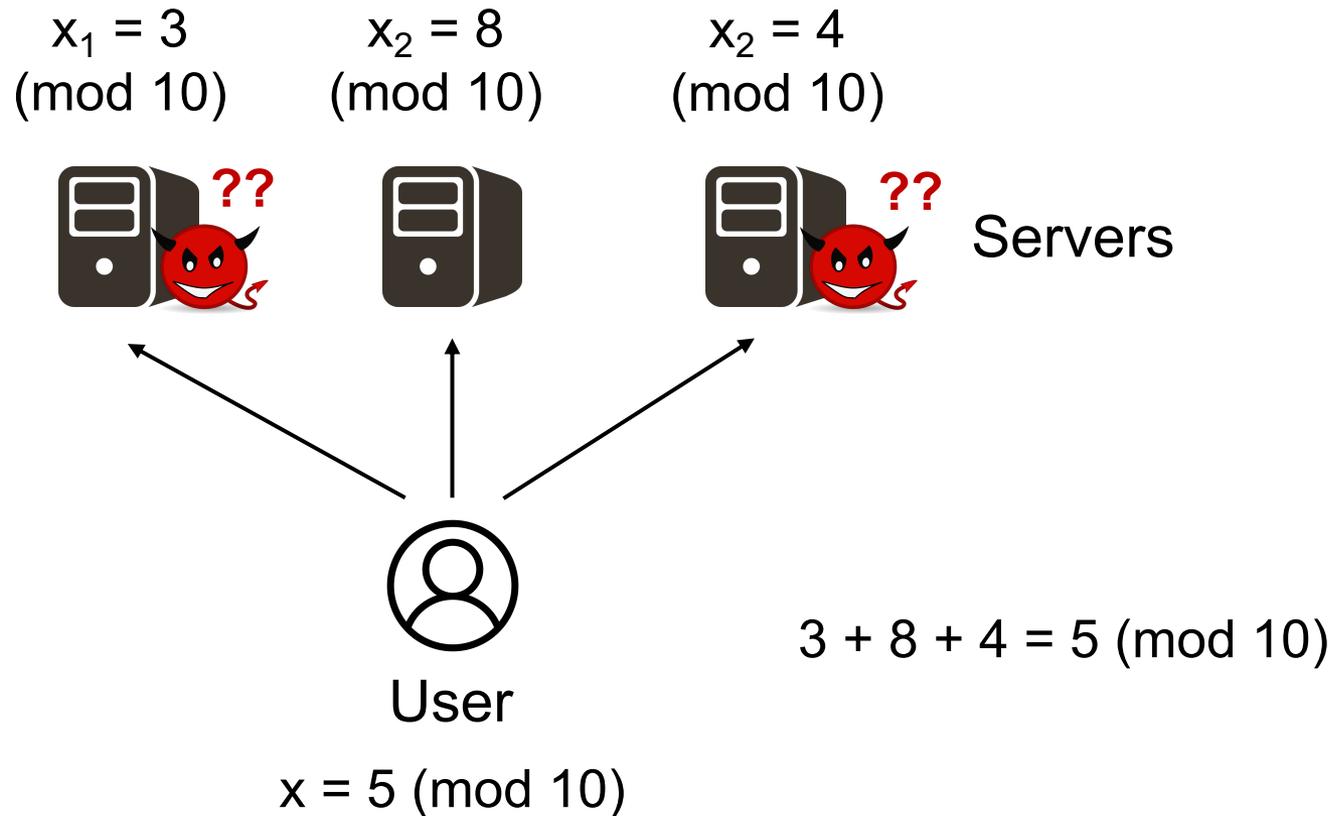
Users wants to store a secret value  $x$  among  $n$  servers, but doesn't fully trust them

» E.g. the servers are public clouds... what if one gets hacked?

**Idea:** split  $x$  into “shares”  $x_i$  so that all shares are needed to recover  $x$

Additive secret sharing:  $x = \text{integer mod } P$ ,  
 $x_i$  are random integers so  $\sum x_i = x$

# Secret Sharing Example

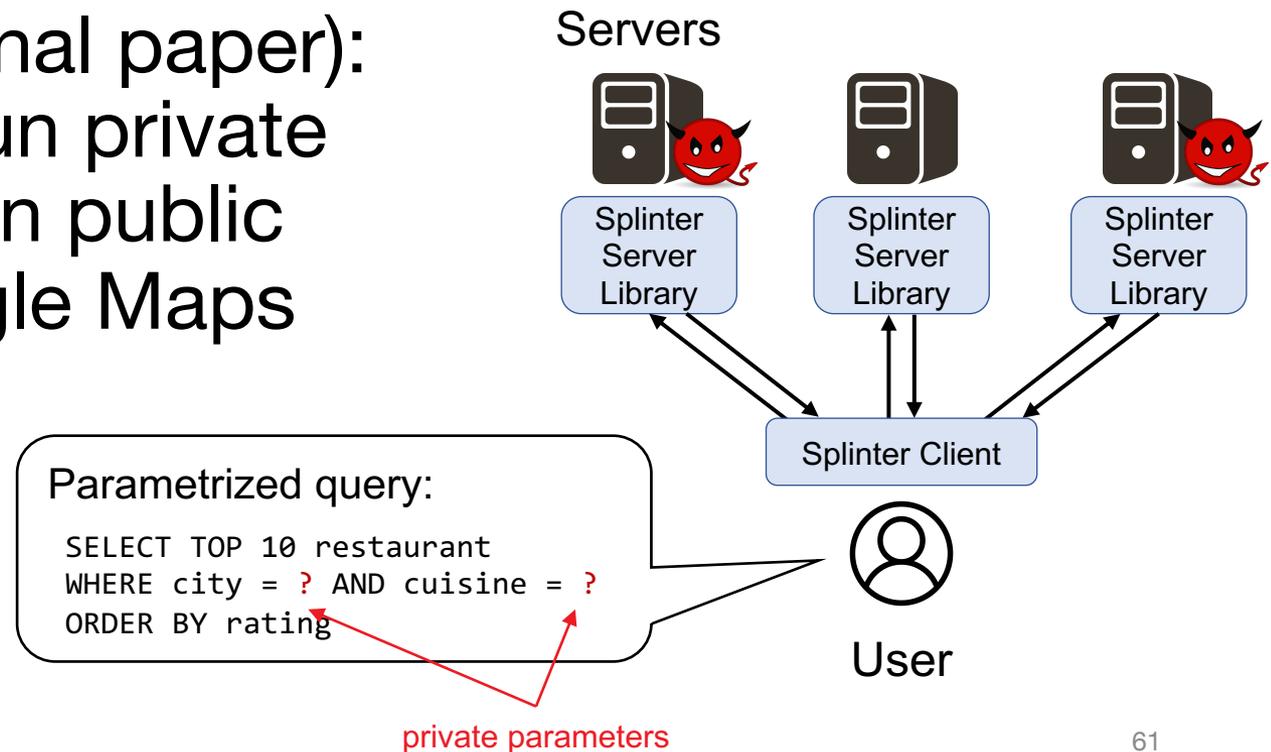


**Note:** performance is quite fast (just additions)

# Function Secret Sharing

Recent result that allows sharing some functions too (keeping queries private)

Splinter (optional paper):  
uses FSS to run private  
SQL queries on public  
data like Google Maps



# Lineage Tracking and Retraction

Goal: keep track of which data records were derived from an individual input record

- » Facilitate removing a user's data in GDPR, verifying compliance, etc

Some real systems provide this already at low granularity, but could be baked into DB

# Summary

Security and data privacy are essential concerns for data-intensive systems

**Threat models** are a systematic way to measure security and reason about designs

Many nice theoretical tools exist to reason about security needs of relational & math ops  
» Build on declarative and relational APIs!