Module 5

Data Security and Privacy

Data Security and Privacy

- Data Security
  - Who has access to the data?
  - Who can change the data?
  - What are the threats to the data?
  - How do we mitigate the threats?
- Data Privacy
  - Who is the data about?
  - How can we share data without threatening people's privacy?

Threats to Data Security

- Random corruption
  - Especially threatening to server RAM, critical infrastructure
- Software flaws
- Human errors
- Malicious corruption
- Malicious injection

Protecting Data Security

- Access Control
- Error Checking and Correction
- Backup and Replication

Access Control

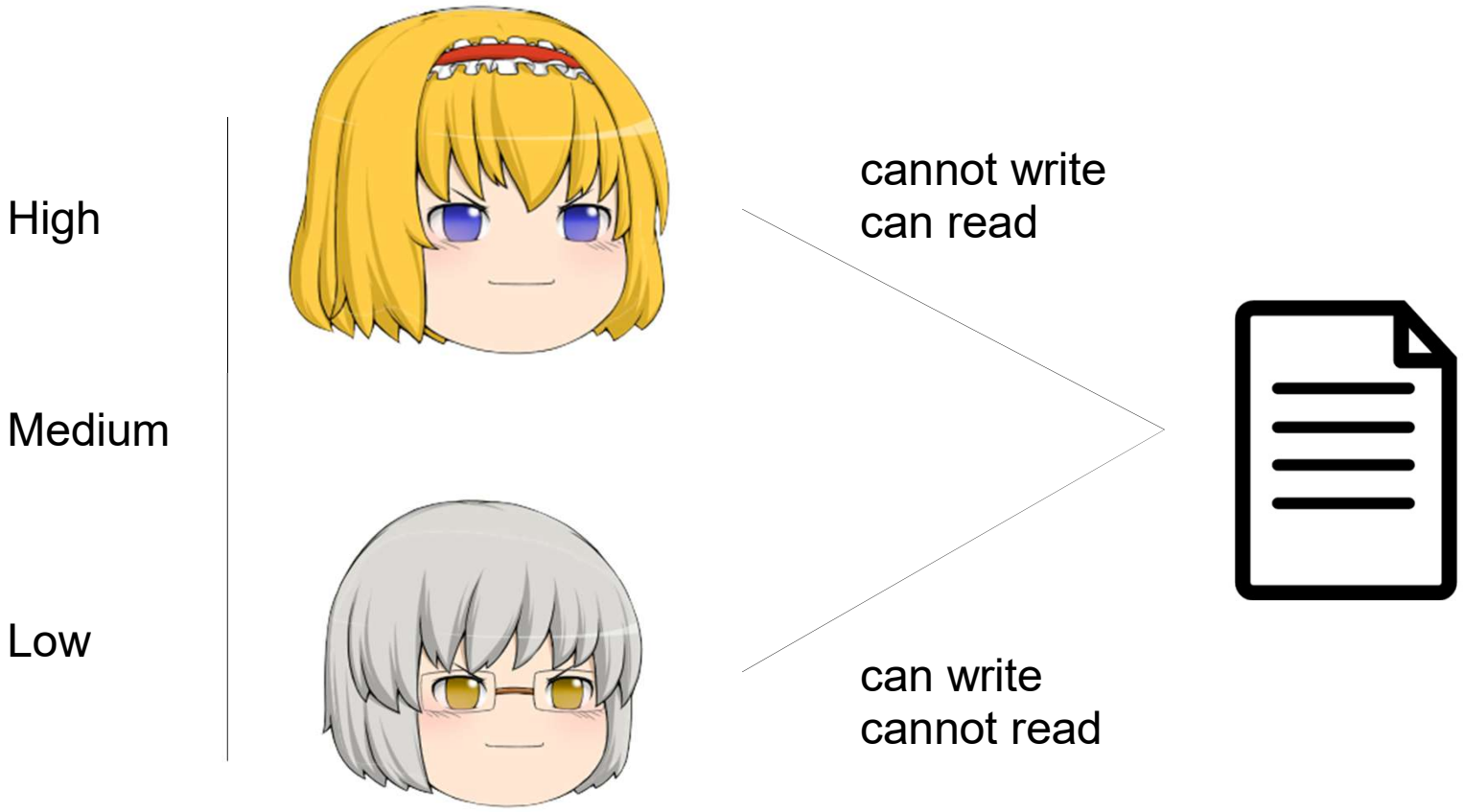
- Define access control for (legitimate) users
- Mandatory vs Discretionary models
  - Mandatory: Admin controls all r/w/x permissions
    - Includes: Multi-level security
  - Discretionary: Each user decides
    - Includes: Unix file access control

Bell-LaPadula Model

- Example of Multi-Level Security
- Subjects and Objects both have security levels (e.g. High, Low)
- All read/write must follow two rules (next slide)
- Prevents leakage of information (i.e. confidentiality)
- Originally the “most widely accepted basis for verifying the security of systems”

Bell-LaPadula Model

1. A lower security subject cannot read a higher security object
2. A higher security subject cannot write to a lower security object



Biba Integrity Model

- Like Bell-LaPadula, but reversed.
- Two rules:

1. A higher security subject cannot read from a lower security object
2. A lower security subject cannot write to a higher security object

- Prevents flow of incorrect information (i.e. integrity)

High-water and Low-water mark

- Replaces rule 2) of each model
- High-water Bell-LaPadula: After higher security subject writes to lower security object, increase security level of object to level of subject
- Low-water Biba Integrity: After high security subject reads from lower security object, decrease security level of subject to level of object

File Access Control

- Access control matrix
- Access control list
- Capabilities
- Role-based

		Objects		
		Data1	Data2	Data3
Subjects	Alice	rw	r	-
	Bob	-	-	r
	Carol	r	r	r

Access Control List

- "Which subjects can read/write/execute this object?"
- e.g. chmod 744 on Unix (what does it mean?)

		Objects		
		Data1	Data2	Data3
Subjects	Alice	rw	r	-
	Bob	-	-	r
	Carol	r	r	r

ACL for Data1

Capabilities

- A transferable “reference” that gives a subject permissions to an object
- “Which objects can this subject read/write/execute?”
- f = open("filename", r);

		Objects		
		Data1	Data2	Data3
Subjects	Alice	rw	r	-
	Bob	-	-	r
	Carol	r	r	r

Alice's capabilities

Biometrics

- Visual, sound, fingerprint, gait, handwriting, etc.
- Can be a factor in multi-factor authentication
- Also suffers from base rate fallacy
  - FIDO Biometrics requirement: >95% TPR @ 0.01% FPR
- Some attacks:
  - Steal biometrics: fingerprint residue, photos/recordings
  - Puppet attack: force live victim to authenticate for you
    - “$5 wrench attack”
  - Generative attacks

Token devices

- For key K and time T, output is:

$$h(K \oplus T)$$

- Only device owner and authorization checker have the key K

Physical Security

- Preventing damage: Rain, Bug, Storm, Electricity, Earthquake, Tornado, etc.
- Access: Fencing, Walls, Windows
- Monitoring: Guards, Cameras

Error detection

- Small number of bit errors should be detectable
- Append a tag to a file:
  - Parity
  - Checksums, e.g. CRC32
  - Hashes; a weak cryptographic hash may be a good error detection hash (e.g. MD5)
- Input can be any size, output is fixed (32-bit for CRC32, 128-bit for MD5)
- Cannot correct an error

Error correction

- Error Correcting Codes (ECC)
- Used in memory, storage, etc.
- Hamming code example:
  - For 2^k-1 bits of transmission, use k bits of parity
  - parity k is in location 2^k
  - There are 2^k-k-1 bits of data in all other locations
  - parity k covers all locations with 1 in the *k*th bit except itself
  - Can correct any 1 bit error
  - Can detect any 2 bit error if we add a parity bit covering all other bits

Error correction

Data is:

$$d_1 d_2 d_3 ... d_{11}$$

Add parity bits in the right places:

$$\mathbf{p_1 p_2} d_1 \mathbf{p_3} d_2 d_3 d_4 \mathbf{p_4} d_5 d_6 d_7 d_8 d_9 d_{10} d_{11}$$

Compute parity bits:

$$\mathbf{p_1 = H_3 \oplus H_5 \oplus H_7 \oplus H_9 \oplus H_{11} \oplus H_{13} \oplus H_{15} = d_1 \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_9 \oplus d_{11}}$$

$$\mathbf{p_2 = H_3 \oplus H_6 \oplus H_7 \oplus H_{10} \oplus H_{11} \oplus H_{14} \oplus H_{15} = d_1 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \oplus d_{10} \oplus d_{11}}$$

...

Let's say d_6 was flipped. Which parity bits will seem “wrong”?

d_4 is H_{10}, and 10 = $(1010)_2$, so p_2 and p_4 will seem “wrong”

Let's say the receiver notices parity bits 2 and 3 seem wrong.
Which bit should they correct?

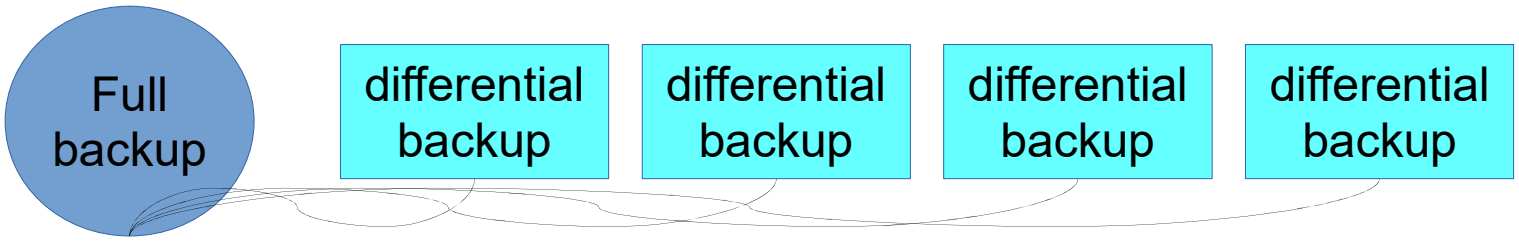
$(0110)_2$ is 6, and H_6 is d_3, so they should correct d_3

Backup

- Used for disaster recovery – we want to recover our data after corruption
- Full backups store all data, but we cannot store too many
- We need to use differential and incremental backups

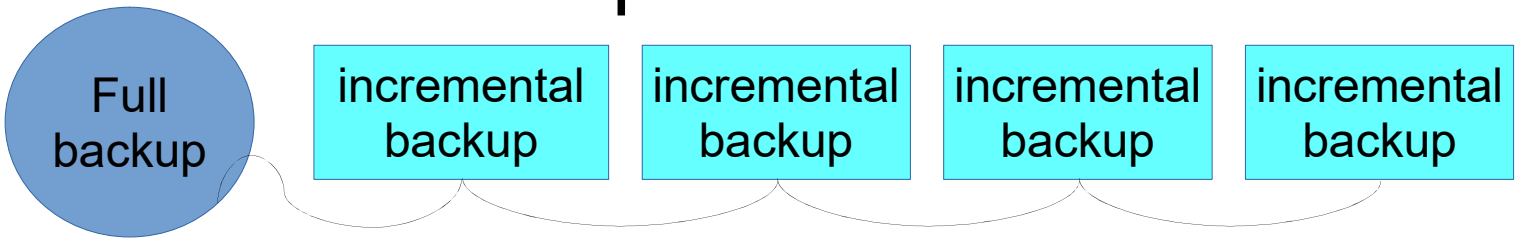
Differential backup

- Stores all changes between current time and last full backup
- How can we find changes?
  - e.g. rsync in Unix: Divide file into chunks, then hash each chunk, and compare the hash for each chunk with stored MD5 hashes
  - Only updates chunks with changed hashes



Incremental backup

- Stores all changes between current time and last backup (not necessarily full backup)
- Smallest storage space
- Hard to recover (if full backup was a long time ago)
- What happens if we combine differential and incremental backups?



Replication

- Synchronous replication: All file updates should happen (almost) immediately
  - All data changes go to all replicas simultaneously
- Asynchronous replication: Small delay when pushing to replicas is acceptable
  - Any data changes go to a central server, then copied to replicas in some order
- Different from backups: replication keeps no historical state
- Reduces data access latency and ensures availability if one replica is taken down
  - But does not defend against threats that would require a backup

Data Privacy

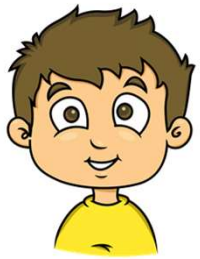
Data has sensitive attributes and personally identifiable information

How can the data owner allow a data user to utilize the data without compromising privacy?

Idea: Restrict queries by data user
But this leads to *inference attacks!*

Inference Attack

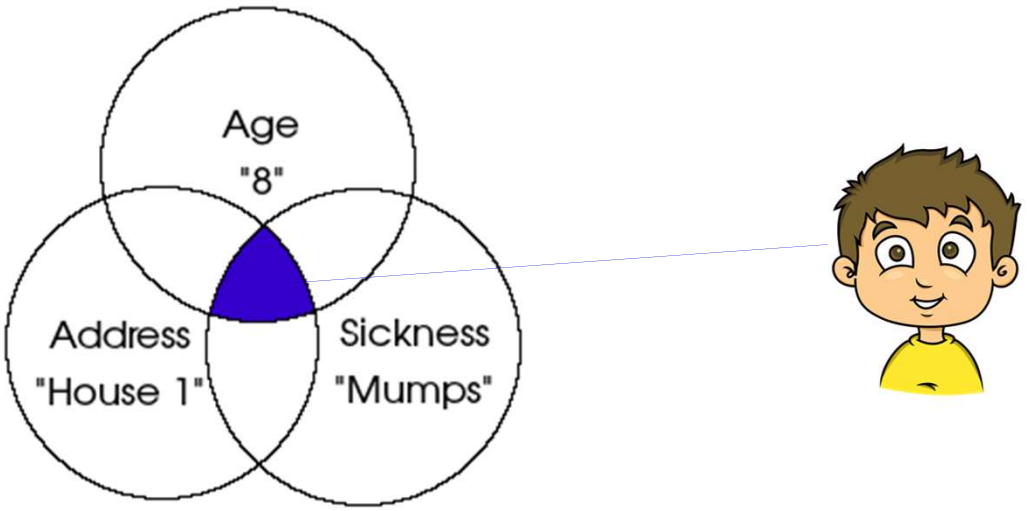
- Use restricted queries to infer sensitive attributes
  - Example: A hospital has a database of patients and their sicknesses, and wants to allow queries on it for research
  - For simplicity: Database includes Age, Address, Sickness
- The hospital restricts all queries to COUNT queries
- Bob is the only boy who is 8 and lives in House 1
- Can the data user (who knows Bob's Age and Address) figure out if Bob has mumps?



Inference Attack

Queries only including Bob

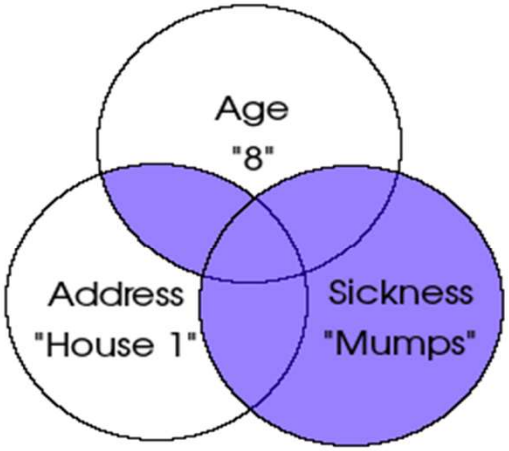
- Data user makes a query returning 0 or 1 result:
  - COUNT(Age=”8” and Address=”House 1” and Sickness=”mumps”)
- Such queries should also be restricted
- But this does not solve difference and intersection inference attacks (next)



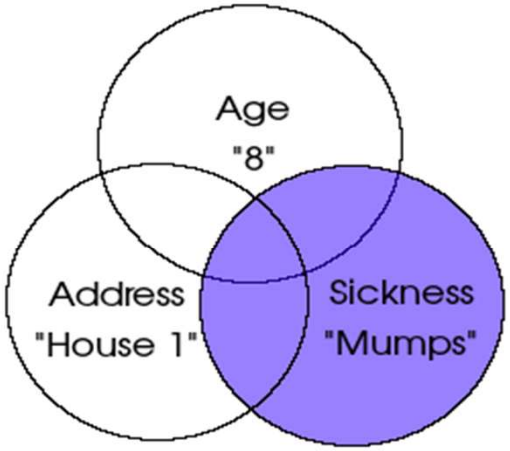
Inference Attack

Difference of queries

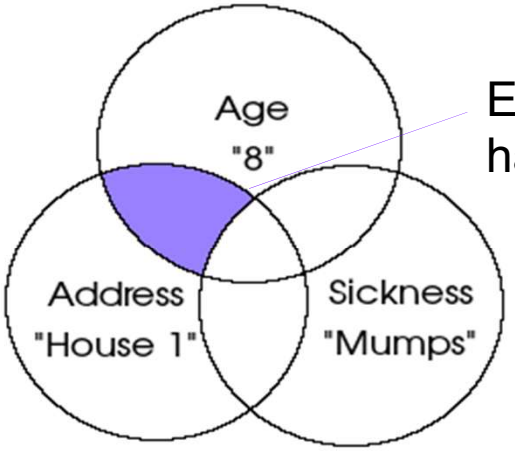
- Data user makes two queries, and takes their difference:
  - Q1 = COUNT(Sickness=”mumps”)
  - Q2 = COUNT((Age=”8” and Address=”House 1”) or Sickness=”mumps”)
- Q2-Q1 = 0 if Bob has mumps, and 1 if not



Q2



Q1



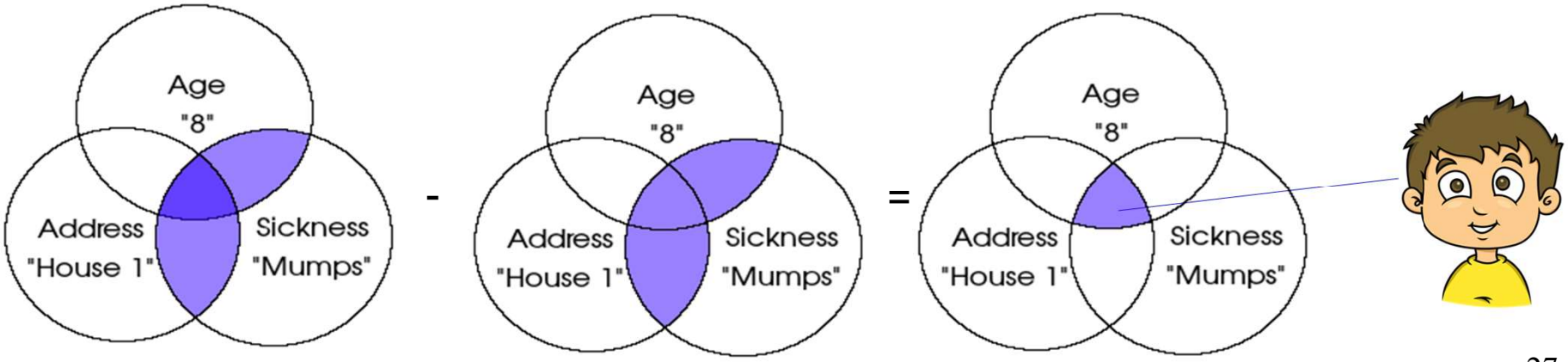
Empty if Bob has mumps

Q2-Q1

Inference Attack

Intersection of queries

- Data user makes three queries:
  - Q1 = COUNT(Age=”8” and Sickness=”mumps”)
  - Q2 = COUNT(Address=”House 1” and Sickness=”mumps”)
  - Q3 = COUNT((Age=”8” or Address=”House 1”) and Sickness=”mumps”)
- Q1+Q2-Q3 is 1 if Bob has mumps, and 0 if not



Q1+Q2 Q3 Q1+Q2-Q3

Inference Attack

- Takeaway: It is practically impossible to prevent or even detect inference attacks from query history
- We need better ways to protect data privacy

Data Privacy

How can the data user compute Q on data D without compromising the data owner's privacy?

k-Anonymity: (D sensitive, Q possibly sensitive) Publish distorted D
Differential Privacy: (D sensitive) Allow only special queries with mathematical error guarantees
Secure Multiparty Computation: (D1, D2 sensitive) jointly compute Q without revealing D1, D2 to each other
Private Information Retrieval: (Q sensitive) Retrieve some information from D without revealing Q

k-Anonymity

- Motivation: many data elements are "quasi-identifiers"; hiding direct identifiers (names, IDs) is not enough to preserve privacy
  - QIDs: Address, height, age, etc.
  - Sweeney '02: 87% of US population can be uniquely identified with {5-digit ZIP, gender, date of birth}
- However, removing quasi-identifiers completely could damage the value of data
  - Originally, the link between asbestos and diseases was found by correlating addresses with medical conditions and a construction database
- Is there a principled way to hide quasi-identifiers without compromising data value?

k-Anonymity

- First, identify QIDs and sensitive attributes
  - Sensitive attributes are not QIDs and cannot be hidden: e.g. medical conditions, salary, political views, etc.
- After anonymization, each **set** of identifiers in the table must appear at least *k* times (= anonymity sets have at least k elements)

(data points)

k-Anonymity

	Quasi-identifiers		
	Age	Weight (kg)	Heart disease?
Hospital Subjects	23	86	N
	15	65	Y
	34	123	Y
	55	95	N
	32	63	Y
	45	89	Y
	59	112	N
	61	81	Y
	15	73	Y

—

	Quasi-identifiers		
	Age	Weight (kg)	Heart disease?
Hospital Subjects	25	100	N
	25	50	Y
	25	100	Y
	50	100	N
	25	50	Y
	50	100	Y
	50	100	N
	50	100	Y
	25	50	Y

"Round Age to nearest 25, Weight to nearest 50" → k = 2
(There are three anonymity sets: Size 2, Size 3, Size 4.
We take the minimum to be k.)

k-Anonymity

	Quasi-identifiers		
	Age	Weight (kg)	Heart disease?
Hospital Subjects	25	100	N
	25	50	Y
	25	100	Y
	50	100	N
	25	50	Y
	50	100	Y
	50	100	N
	50	100	Y
	25	50	Y

- A flaw in k-anonymity: All members of an anonymity set may have the same sensitive attribute
  - e.g. If someone you know is around age 25 and weight 50kg, and you know they're in the table, you know they have heart disease
- To fix this, we can also enforce *l*-diversity: Every anonymity set must have at least *l* different sensitive attributes

k-Anonymity

Another weakness is that complementary releases can compromise k-anonymity:

	Quasi-identifiers	
	Weight (kg)	Condition
Hospital Subjects	30-60	A
	30-60	B
	30-60	C
	30-60	D
	30-60	E
	60-150	F
	60-150	G
	60-150	H
	60-150	I

$k = 4$

	Quasi-identifiers	
	Weight (kg)	Condition
Hospital Subjects	25-55	A
	25-55	B
	25-55	C
	25-55	D
	55-150	E
	55-150	F
	55-150	G
	55-150	H
	55-150	I

$k = 4$

If you know someone whose weight is 56 kg, you know they have E

k-Anonymity

- **Knowing the anonymization scheme** can also compromise the scheme. See assignment!
- Quality of data can still be significantly damaged
  - Curse of dimensionality: information loss grows quickly with more QID columns
- Cannot be naively applied to many kinds of data
  - Time series data, packet data, location sequences, etc.

Differential privacy

- Ensures privacy of data items using a *differential* mathematical formulation
- Hard to understand, easy to implement
- Anonymization function is random
- Used in iOS (2016~), Windows (2017~), COVID-19 research, etc.

Motivation: Differencing attack

- Suppose you query the mean salary of a company M_1 with 500 people, then someone joins, and you query it again to obtain M_2
- What is the salary of the person who joined?
- Similar: Query for total count of people with a certain condition, ethnicity, etc.

Differential privacy

Definition: Two databases are *neighboring* if they are the same except for one element (one person's data).

A query Q is *ε-differentially private if for all neighbouring databases D_1 and D_2 and for all q:*

$$\frac{Pr(Q(D_1) = q)}{Pr(Q(D_2) = q)} \leq e^{\varepsilon}$$

Intuitively, changing one person's data is unlikely to change the result (distribution) of a differentially private query => the query result ***does not reveal that person's existence****.*

Hypothesis Testing

Differential privacy can be explained with hypothesis testing. Suppose that two hypotheses are:

H_0: The underlying dataset is D_1 which does contain Alice
H_1: The underlying dataset is $D_2 = D_1 \backslash \{Alice\}$

We use the definition of differential privacy:

$$\frac{Pr(Q(D_1) = q)}{Pr(Q(D_2) = q)} \leq e^{\varepsilon}$$

The probability of rejecting the null hypothesis H_0 at significance level *a* is no higher than $e^{\varepsilon}a$ ~= *a*, i.e. any test cannot be powerful

Achieving differential privacy

In many cases, differential privacy is easily achieved with *Laplacian* noise, with pdf defined as follows:

$$f(x, b) = \frac{1}{2b} \exp\left(\frac{-|x|}{b}\right)$$

Consider two neighboring databases' true query results $\overline{Q}(D_1)$, $\overline{Q}(D_2)$, then for any query output *q we can write* $q = \overline{Q}(D_1) + x_1$ *and* $q = \overline{Q}(D_1) + x_2$

$$\frac{Pr(Q(D_1) = q)}{Pr(Q(D_2) = q)} = \frac{\frac{1}{2b} \exp\left(\frac{-|x_1|}{b}\right)}{\frac{1}{2b} \exp\left(\frac{-|x_2|}{b}\right)} \leq \exp\left(\frac{\left|\overline{Q}(D_1) - \overline{Q}(D_2)\right|}{b}\right)$$

Sensitivity

The sensitivity of a query is the maximum amount it can change between neighboring datasets:

$$S(Q) = max_{neighbours} \left| \overline{Q}(D_1) - \overline{Q}(D_2) \right|$$

Therefore, Laplacian noise with mean 0 and sensitivity b achieves *ε-DP* where *ε = S(Q)/b*

Sensitivity of common queries:

- Count (including conditional count): 1
- Summation: *m* where *m* is the largest possible element
- Mean: *m/n* where *m* is as above and *n* is the smallest possible dataset

Some queries have excessively large sensitivity!

Intuition of differential privacy

- A small amount of noise on a query output can be equivalent to a large amount of noise on each individual element
- We are anonymizing a *query*, not a database
  - A differentially private query protects **any** database it is applied to
- If a query output is possible for a database but not for a neighboring database, DP is not achieved

Usefulness of differential privacy

Composition theorem

If Q_1 and Q_2 are differentially private at levels ε_1 and ε_2, (Q_1, Q_2) is $(\varepsilon_1 + \varepsilon_2)$-differentially private

- This protects against sequential release

Post-processing theorem

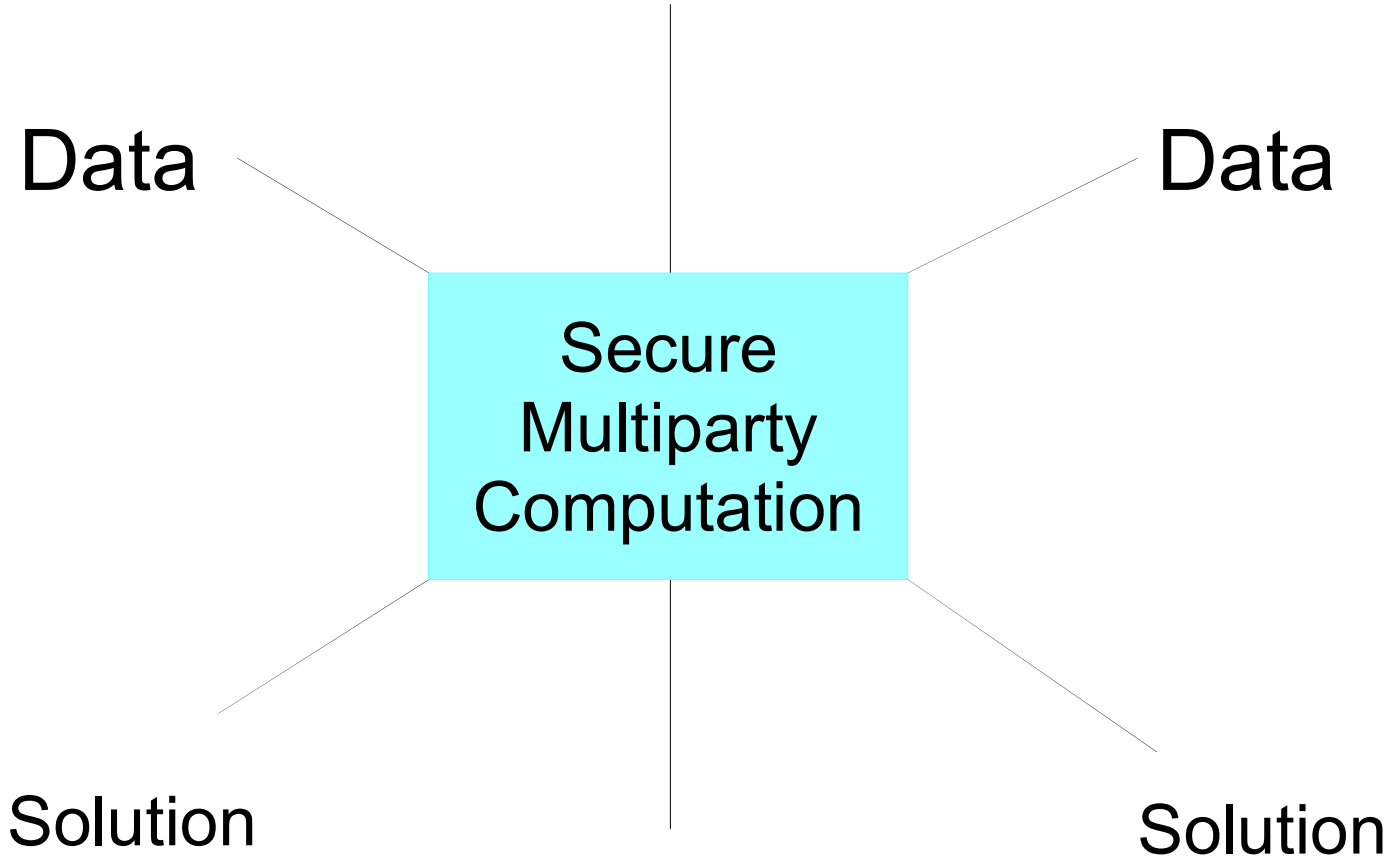
If Q is ε-differentially private, $g(Q)$ is ε-differentially private for any function g

- Any operation on the output is safe

Differential privacy for data aggregation

- Scenario: Each individual has (private) data, we want to compute aggregate without compromising privacy
- DP intuition: Large noise for each person = small noise on result
- Example: Private Count Mean Sketch for identifying energy-hungry websites
  - Energy-hungry website is first mapped to a one-hot vector using a hash function
  - Each bit of the vector is then randomly flipped with some probability to introduce noise
  - Summing all users' vectors still produces a number close to the true value
- Also possible to jointly train a ML model

Secure Multiparty Computation



Useful for research, data analytics, collaboration, etc.

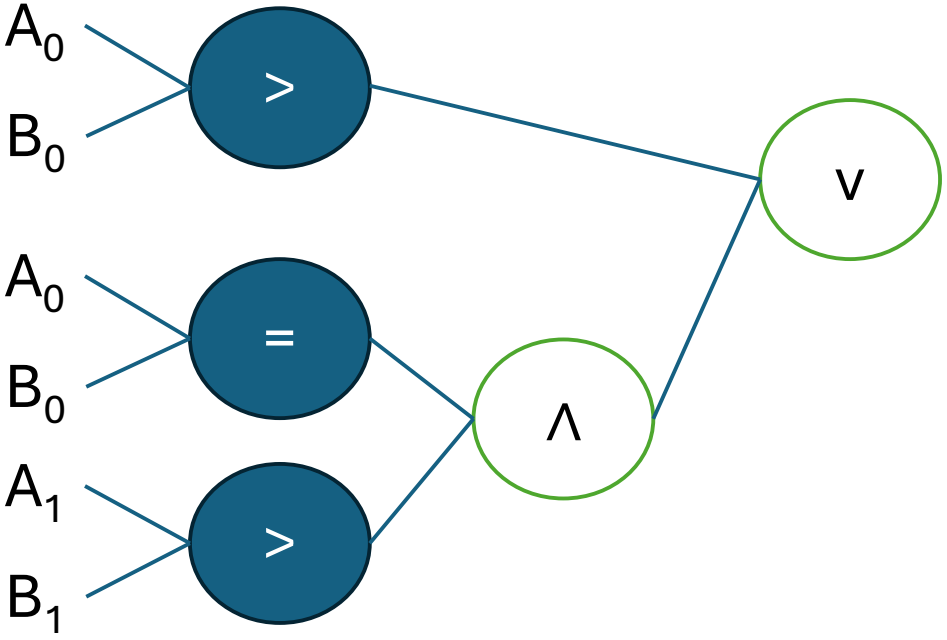
Secure Multiparty Computation

- Two parties with different data can jointly compute a known function on the union of their data while sharing no data at all
- e.g. “Who has more customers on this day?”
- Generally (much) slower than directly running the algorithm
- e.g. 20 minutes on 2 cores to complete one AES encryption of 128 bits under SMPC
- Guaranteed correctness without noise

Secure Multiparty Computation

Boolean circuits

- Computable functions can be represented as Boolean circuits
- Example Boolean circuit for "$\overline{A_0A_1} > \overline{B_0B_1}$?"
  - $\overline{A_0A_1}$ means a two-bit number, A_0 then A_1



>: ¬ A_0 ∧ A_1

=: (A_0 ∧ A_1) ∨ (¬ A_0 ∧ ¬ A_1)

Secure Multiparty Computation

Yao's garbled circuit

- Construct a boolean circuit representing the problem
- Suppose A is the "garbler", and B is the "evaluator"
- For each boolean gate, A will compute a *garbled* table and share all garbled tables with Bob

Secure Multiparty Computation

Yao's garbled circuit

- For each boolean gate, A computes a garbled table:
- Generate random garbled strings I_0, J_0, I_1, J_1 representing input bits being 0 or 1
- Generate random garbled strings O_0, O_1 representing output bits being 0 or 1
- Encrypt the four possible outputs (random strings) of the table using its inputs as keys, with some long public string A (e.g. 128 bits of 0)

Input 1 (I)	Input 2 (J)	Output (O)
0	0	0
0	1	1
1	0	1
1	1	0

Encrypted Output (O)
$Enc_{I1, J0}(O_1 \| A)$
$Enc_{I0, J0}(O_0 \| A)$
$Enc_{I1, J1}(O_0 \| A)$
$Enc_{I0, J1}(O_1 \| A)$

This table is shared with Bob

Secure Multiparty Computation

Yao's garbled circuit

- Alice randomize the order of the table and sends it to Bob
  - Also sends either I_0 or I_1, whichever is the correct bitstring corresponding to her data
  - Bob can request J_0 or J_1 without revealing which bitstring has been requested using a protocol called oblivious transfer
  - For gates that only have inputs from other parts of the circuit, Alice does not need to send any input bitstrings
    - For example, if a gate takes this table's output as input, then one of its input columns would be coded with O_0 and O_1

Input 1 (I)	Input 2 (J)	Output (O)
0	0	0
0	1	1
1	0	1
1	1	0

Encrypted Output (O)
$Enc_{I1, J0}(O_1 \| A)$
$Enc_{I0, J0}(O_0 \| A)$
$Enc_{I1, J1}(O_0 \| A)$
$Enc_{I0, J1}(O_1 \| A)$

This table is shared with Bob

Secure Multiparty Computation

Yao's garbled circuit

What does Bob do with the table?

Encrypted Output (O)
$Enc_{I1, J0}(O_1 \| A)$
$Enc_{I0, J0}(O_0 \| A)$
$Enc_{I1, J1}(O_0 \| A)$
$Enc_{I0, J1}(O_1 \| A)$

- Suppose Alice's input is I (and it is 0) and Bob's input is J (and it is 1)
- Alice sends Bob I_0, Bob requests J_1 from Alice
- Using those two strings as a key, Bob tries to decrypt all 4 rows of this table
- Only one row will succeed (Bob knows it succeeds if ***A*** shows up)
- Bob will then obtain O_1 (without knowing the meaning of O_1)
- This can be used in the next gate, or if it is the final output, its meaning can be determined by asking Alice

Secure Multiparty Computation

Yao's garbled circuit

- We achieve these desired properties:

1. Bob can compute the gate without knowing the real inputs
2. The output is unknown to Bob until Alice reveals what the output means
3. Alice does not see the output garbled string until Bob reveals it

- There are further enhancements to the protocol that can prevent cheating/save gates

Secure Multiparty Computation

Oblivious Transfer

A simple protocol based on Diffie-Hellman:

Suppose Alice is the sender (messages are M_0, M_1) and Bob is the receiver. Public parameters p, g. In the following all integers and computations are done as integers mod p

1. Alice chooses a secret integer A, Bob chooses another secret integer B
2. Alice sends g^A
3. Bob sends
    1. If the message he wants is M_0, $C = g^B$
    2. If the message he wants is M_1, $C = g^{A+B}$
4. Alice sends both:
    1. $C^A M_0$
    2. $C^A g^{-A} M_1$
5. Bob can only recover M_0 or M_1

A different scenario

- What if the data holder's data is not sensitive, but the data user's query is sensitive?
- For example:
  - Searching for a patent
  - Searching for attributes of a sickness
  - Searching for darknet sites
- We want to use Private Information Retrieval in these cases

Private Information Retrieval

- Goals:
  - Data returned must be accurate
  - Data owner must not know anything about the query
- Trivial solution: Let data user download entire database
  - This achieves goals of PIR but it is inefficient and sometimes undesirable
- Multi-database PIR can be information-theoretically secure (and efficient)
- Single-database PIR is possible

4-Database Information-Theoretic PIR

First, represent the database as a 2-dimensional table; Alice wants to obtain one of these elements

Query: row A, column B

- Working example: A = 2, B = 2

$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	$X_{4,1}$
$X_{1,2}$	$X_{2,2}$	$X_{3,2}$	$X_{4,2}$
$X_{1,3}$	$X_{2,3}$	$X_{3,3}$	$X_{4,3}$
$X_{1,4}$	$X_{2,4}$	$X_{3,4}$	$X_{4,4}$

4-Database Information-Theoretic PIR

- Alice randomly picks each row and column with ½ chance (not related to the element she wants)
- R = {Rows 1, 3}
- C = {Column 3}

$X_{1,1}$	$X_{2,1}$	$X_{3,1}$	$X_{4,1}$
$X_{1,2}$	$X_{2,2}$	$X_{3,2}$	$X_{4,2}$
$X_{1,3}$	$X_{2,3}$	$X_{3,3}$	$X_{4,3}$
$X_{1,4}$	$X_{2,4}$	$X_{3,4}$	$X_{4,4}$

4-Database Information-Theoretic PIR

- Alice creates R' and C' by flipping the rows in R and columns in C corresponding to A, B.
- Suppose she wants $x_{2,2}$:
  - Flip row 2, R' = {Rows 1, 2, 3}
  - Flip column 2, C' = {Columns 2, 3}
- Then she creates 4 requests to 4 servers. Each request is an XOR of all elements in certain rows and columns:
  - DB1 = XOR all elements in the intersection of R and C
  - DB2 = XOR all elements in the intersection of R' and C
  - DB3 = XOR all elements in the intersection of R and C'
  - DB4 = XOR all elements in the intersection of R' and C'

4-Database Information-Theoretic PIR

Yellow = activated Rows, Green = activated Columns, Orange = intersection;
DB replies with XOR of all elements in orange

4-Database Information-Theoretic PIR

- Alice can obtain $x_{2,2}$ just by XORing all 4 responses (XOR of all orange boxes in the previous slide)
  - You can prove this!
- Information-theoretic privacy follows from each row/column being randomly selected at ½ chance from any database's perspective
  - A database cannot tell which row/column was perturbed, if any
  - This is true even if probability of any row/column being perturbed was uneven
- Query length is O(sqrt(n))
- Communication cost is minimum possible – only one bit
- Several other protocols exist with fewer databases/better query length

Summary on Data Privacy

To compute Q(D):

- k-Anonymity:
  - (D sensitive, Q possibly sensitive/unknown)
  - Publish distorted D
- Differential Privacy:
  - (D sensitive)
  - Allow only specific queries with error for mathematical guarantees
- Secure Multiparty Computation:
  - (D1, D2 sensitive)
  - Jointly compute Q without revealing D1, D2 to each other
- Private Information Retrieval:
  - (Q sensitive)
  - Retrieve some information from D without revealing Q