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
- Software flaws
- Human errors
- Malicious corruption
- Malicious injection

Protecting Data Security

- Access Control
- Error Checking/Correction
- Backup

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

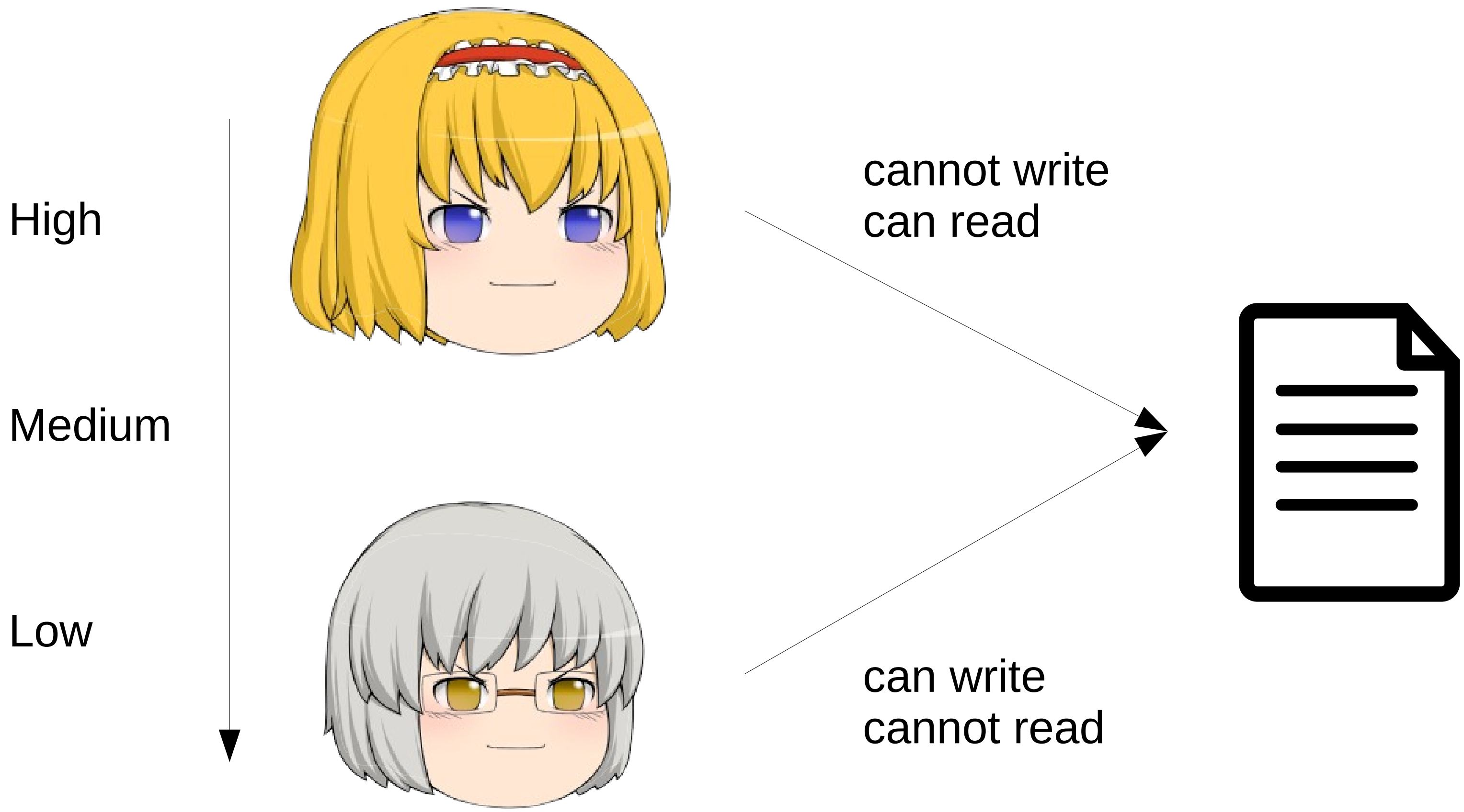
Bell-LaPadula Model

- 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)

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
- Can be mimicked – photos, recordings, etc.
- Also suffers from base rate fallacy

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 fix 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:

$$d_1d_2d_3...d_{11}$$

Add parity bits in the right places:

$$\mathbf{p_1p_2}d_1\mathbf{p_3}d_2d_3d_4\mathbf{p_4}d_5d_6d_7d_8d_9d_{10}d_{11}$$

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?

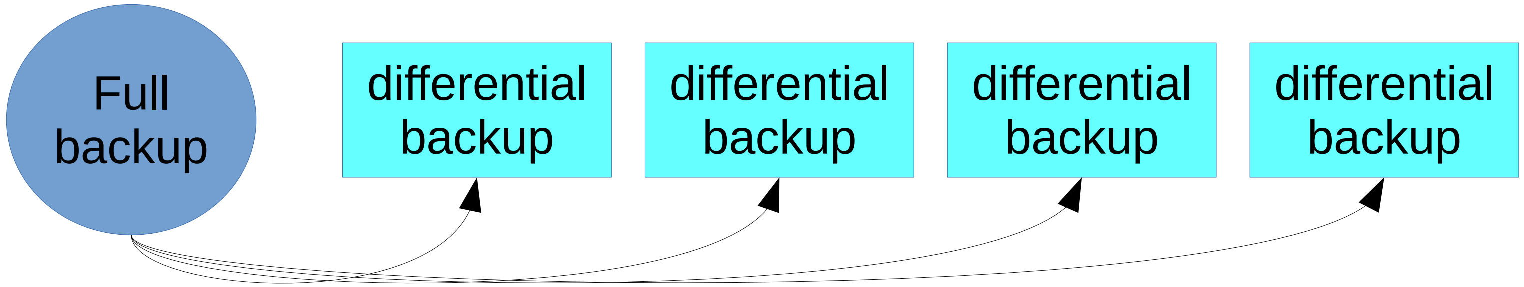
$(0110)_2$ is 6, and H_6 is d_3, so they should correct d_3

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

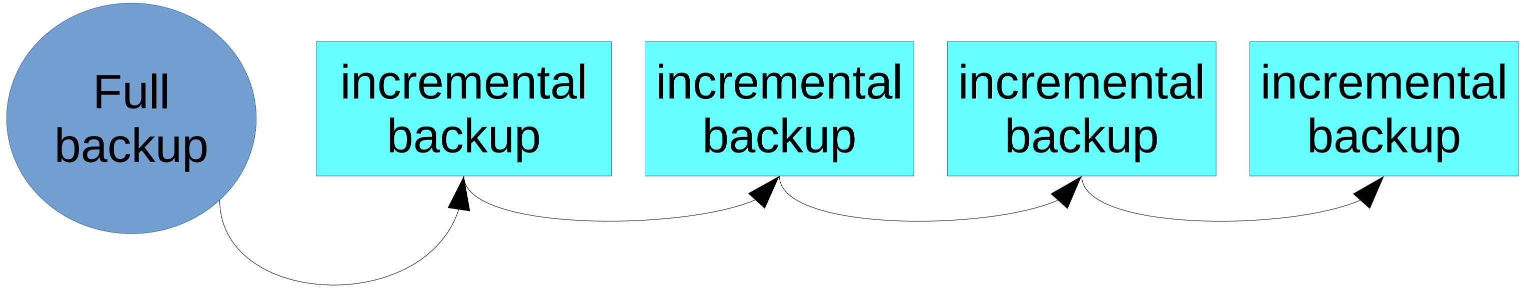
Differential backup

- 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

- Different from backups: replication keeps no historical state
- Synchronous replication: All file updates should happen (almost) immediately
- Asynchronous replication: Small delay when pushing to replicas is acceptable
- Shadowing for databases

Data Privacy

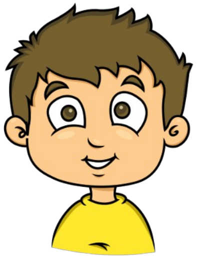
Data has sensitive attributes and personally identifiable information

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

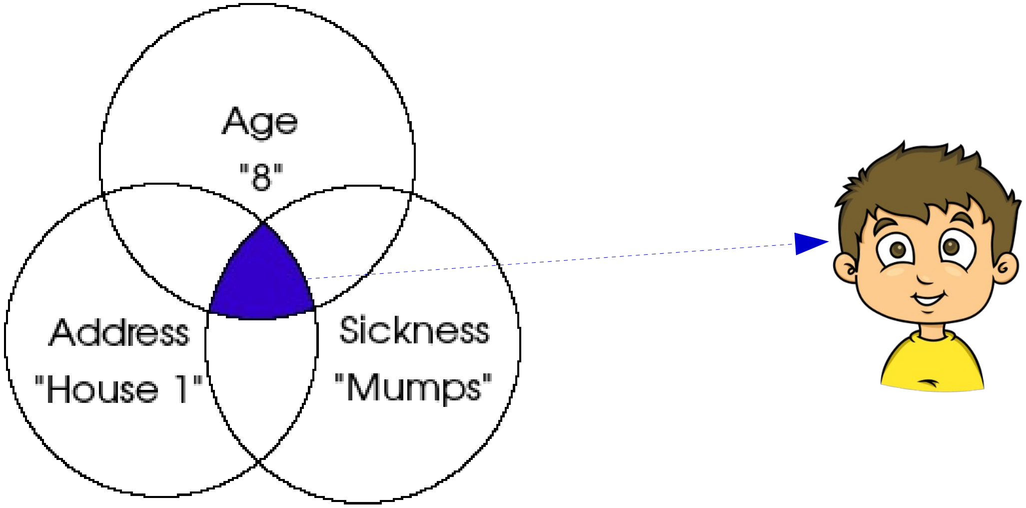
- Use restricted queries to infer sensitive attributes
- Example: A hospital has a database of patients and their sicknesses, and wants to allow queries on it for research
- For simplicity: Database includes Age, Address, Sickness
- The hospital restricts all queries to COUNT queries
- Bob is the only 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

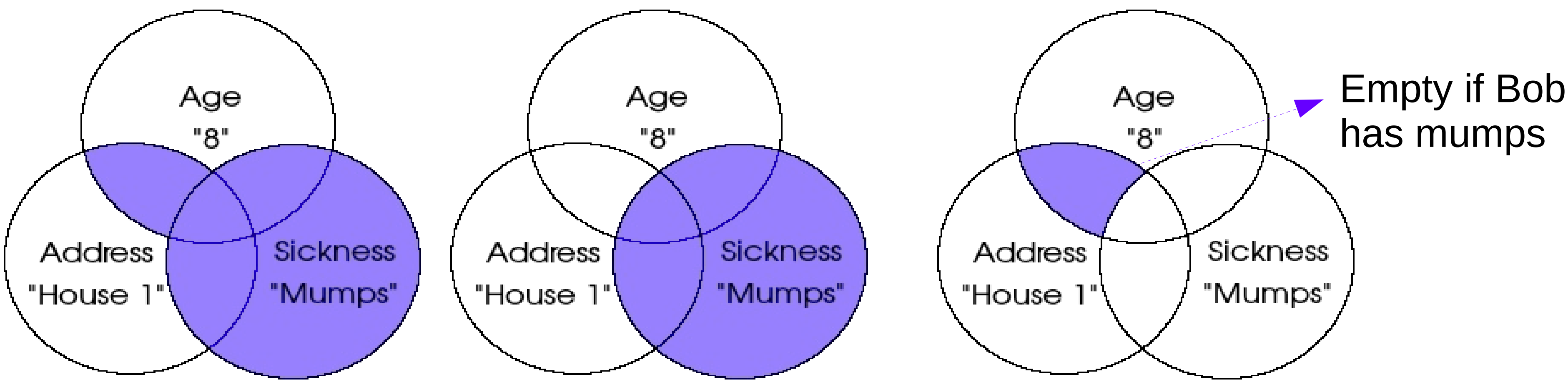
- Data user makes a query returning 0 or 1 result:
  - COUNT(Age=”8” and Address=”House 1” and Sickness=”mumps”)
- Such queries should also be restricted
- But this does not solve difference and intersection inference attacks (next)



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

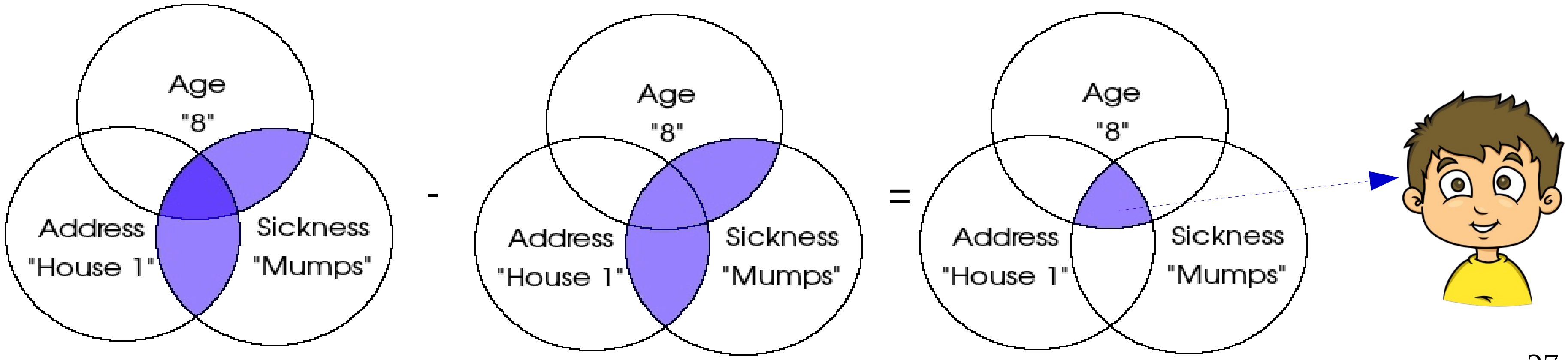


Q2 Q1 Q2-Q1

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

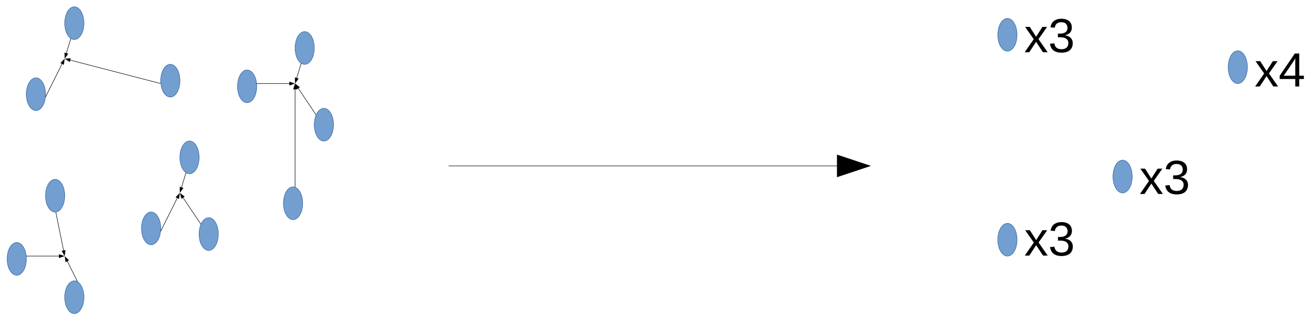
Data Privacy

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

- Remove link between identifiers (PII) and sensitive attribute
- Anonymization function is usually deterministic
- 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 your friend 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 completentary releases can compromise k-anonymity:

	Quasi-identifiers	
	Weight (kg)	Sickness
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)	Sickness
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

$k = 4$

Different rounding schemes will weaken k - if your friend has weight 56 kg, you know they have E

k-Anonymity

- Knowing the anonymization scheme can also compromise the scheme. Suppose Age is the only QID. If you know the anonymization scheme is the following:
  - Sort patients by age, start with an anonymity set containing only the smallest age.
  - Add patients in order to the current anonymity set until desired *k* and *l* have been achieved. Then start a new anonymity set with the next person that has not been added.
  - Repeat until all patients added; if final anonymity set is too small, merge it with the previous completed anonymity set.
- Suppose the hospital want to achieve *k = 3, l* = 2. It releases two anonymity sets {Age}:{Heart Disease} as follows:

  \- {0-40}: {N, Y, Y, Y, Y}   {40-80}: {Y, N N}
- If you know that your friend is the youngest person in the database, then they definitely have heart disease, otherwise the first set would not be so large!

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 10 (2016)

Differential privacy

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:*

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

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!

Differential privacy

Suppose the salaries of 5 employees are:

Employee	Salary
A	$200
B	$210
C	$240
D	$150
E	$400

For legal compliance, the company is obligated to reveal the average salary of its employees.

Now E has left the company. Suppose someone queries the average salary twice: before and after E left. The results are: $200, $240. This reveals E's salary: $240 * 5 - $200 * 4 = $400.

Differential privacy

Differential privacy: Implicitly add noise before returning the average.

Average = \$200 + Noise ⟷ Same result

Employee	Salary
A	\$200+Noise
B	\$210+Noise
C	\$240+Noise
D	\$150+Noise
E	\$400+Noise

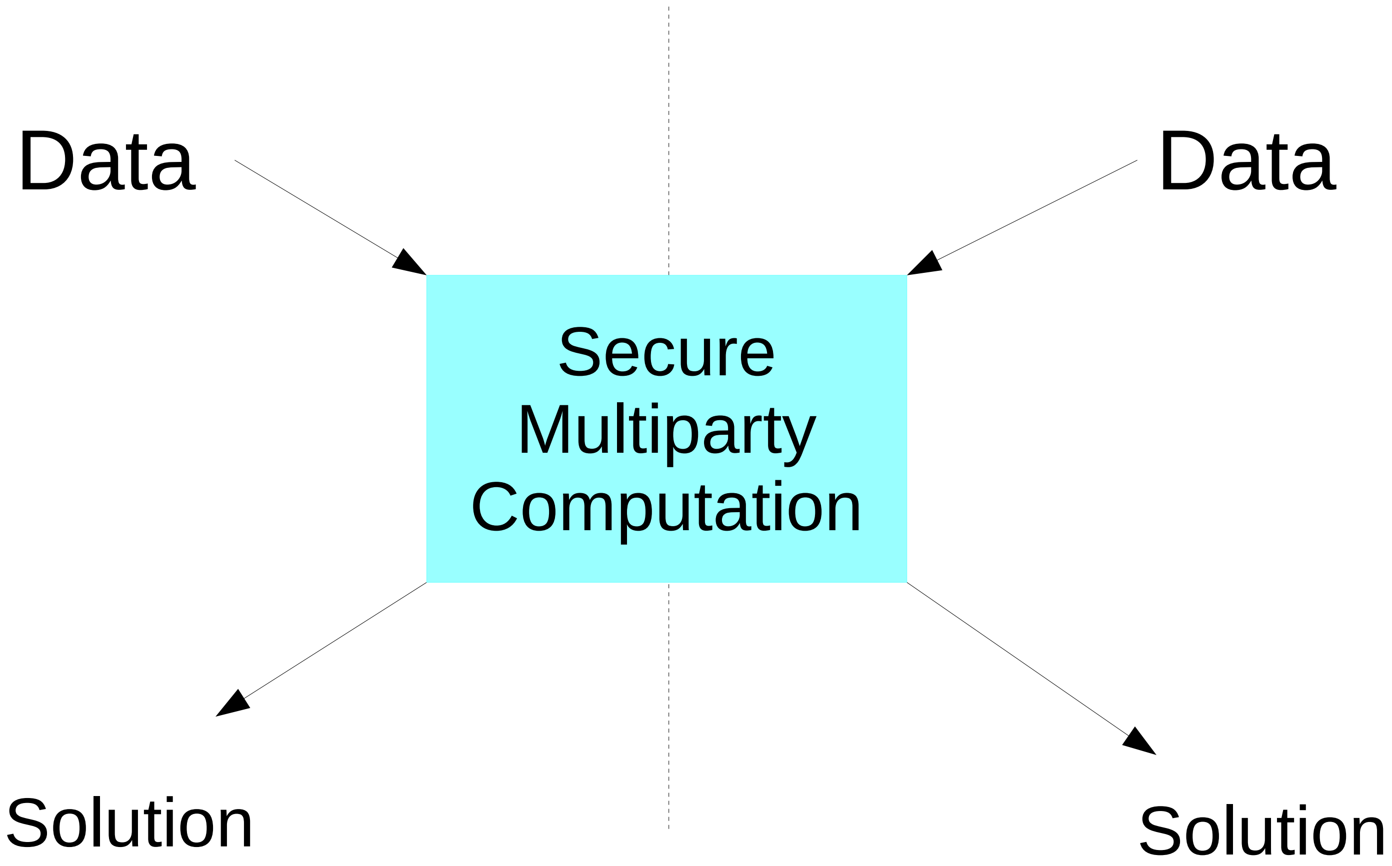
This satisfies differential privacy:

- Difference between two neighboring data sets is dominated by noise
- In other words, one person joining/leaving doesn't change the result as much as the noise itself

Differential privacy

- See notes for two examples with full workings
- Utility can be high with large data sets
- To avoid multiple queries reducing the noise level of the data, we can sample the noise only once until the data changes
- Can be applied to many types of data, but care must be taken regarding definition of “neighboring databases”

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

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)
  - Randomize the order of the table, share this table

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 bit 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:
  - Bob can compute the gate without knowing the real inputs
  - The output is unknown to Bob until Alice reveals what the output means
  - Alice does not see the output garbled string until Bob reveals it
- Bob's request from Alice needs to be protected: Bob can't take both J_0 and J_1 (otherwise he can cheat), but Bob can't say “I want J_1” (otherwise Alice knows Bob's input is 1)
- This can be done with oblivious transfer

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

- Mechanisms to hide query from data owner
- Data returned is accurate
- Trivial solution: Let data user download entire database, but this is not efficient
- Multi-database PIR can be information-theoretically secure (and efficient)
- Single-database PIR is possible (see notes)

4-Database Information-Theoretic PIR

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

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$

4-Database Information-Theoretic PIR

- Alice randomly picks each row and column with ½ chance (not related to the element she truly wants)
  - R = {Rows 2, 3, 5}
  - C = {Column 3}

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$

4-Database Information-Theoretic PIR

- Alice creates R' and C' by flipping the rows in R and columns in C corresponding to the element she truly wants. Suppose she wants $x_{2,2}$:
  - Flip row 2, R' = {Rows 3, 5}
  - 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

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$

DB1

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$

DB2

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$

DB3

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$

DB4

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 sequence: XOR of all orange boxes in the previous slide)
- The desired element is in the intersection of exactly one of R and R', and exactly one of C and C', so only once in the XOR sequence
- No other element has the above property:
  - Every other element is in either both R and R', or both C and C', or neither R and R', or neither C and C'
  - In the last two cases it is not in the XOR sequence
  - If it is in R and R', it appears an even number of times in the XOR sequence depending on if it's in both C and C' (4), one of them (2), or neither of them (0) – so it will be cancelled out with XOR
  - Vice-versa for C and C'

4-Database Information-Theoretic PIR

- 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

Which algorithm to use?

- If downloading the entire database solves the problem, PIR is a good solution
  - i.e. data is not private
- If no noise is tolerable, k-anonymity and differential privacy are not acceptable
- k-anonymity is used to hide QIDs
- Differential privacy can also collect data