CMPT 980 – Information Privacy

Module 5: Data Privacy

Privacy concerns

Data about people naturally bears privacy concerns:

- Personally identifiable information (PII) – Information that identifies a specific individual
  - Name, SIN, etc.
- Quasi-identifiers – Information that may help identify an individual
  - First 4 digits of phone number, zip code, gender
- Sensitive attributes – Information about a person that one does not wish to share
  - Health information, address, salary, etc.

Usability concerns

We want to support use of privacy-sensitive data:

- Data query – allow queries on data
  - Data owner does not release full data, but allows people to query the database
- Data sharing – sharing data between two entities
  - A data user may have analysis and research capabilities, but they need to acquire data from a data owner/collector
- Data release – publish data to broader audience
  - Allows anyone to analyze the data
  - May be necessary to satisfy open information laws

Query controls

- A database owner wants to safeguard data privacy by limiting what queries can be made on the database
  - For example, a range query on salaries will be rejected unless the minimum and maximum are at least $5,000 apart
- This is difficult, because there are many ways to obtain privacy-sensitive information
  - For example, there are 33 people between $50,000 and $55,000, and (after a certain date) 34 people between $50,000 and $55,001
- These are known as inference attacks
  - Specifically, tracker attacks are inference attacks that identify 1 person's attribute

Tracker Attacks

- 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
  - Also, the hospital rejects any query with only 1 result
- 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?

Tracker Attacks

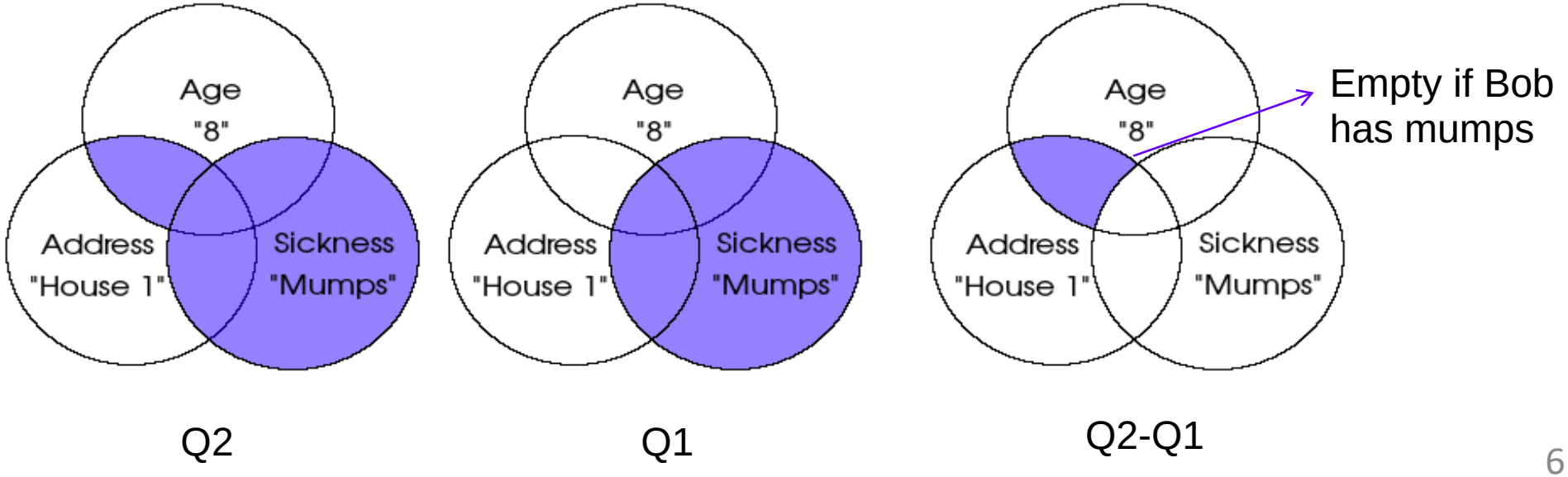
Difference of queries:

- Data user makes two queries, and takes their difference:

  Q1 = COUNT(Sickness=”mumps”)

  Q2 = COUNT((Age=”8” and Address=”House 1”) or Sickness=”mumps”)

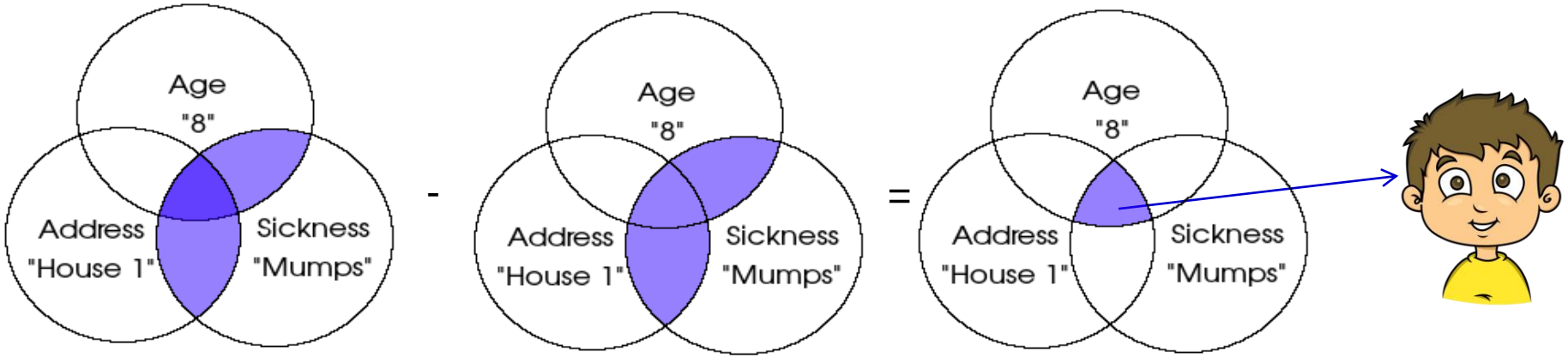
- Q2-Q1 = 0 if Bob has mumps, and 1 if not



Tracker Attacks

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

Tracker Attacks

- Given a set of queries, determining if there is a tracker attack is generally hard
  - SUM+MAX queries – NP-complete (Chin 1986)
  - This set may be very large if we consider the possibility that all data users could be colluding
- Other forms of query control can often also be attacked
  - Adding noise – can be averaged out
  - Data perturbation/suppression
  - Usability is always a concern

k-Anonymity

- Scenario: Data owners wants to suppress privacy-sensitive information for release
- Data columns are divided into two types:
  - Quasi-identifiers (QIDs)
  - Sensitive attributes
- 87% of people in US can be identified based on 3 QIDs: 5-digit ZIP, gender, date of birth (Sweeney 2000)
- After anonymization, each **set** of QIDs in the table must appear at least *k* times (= anonymity sets have at least k elements)

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. Take the minimum)

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
- This is because k-anonymity doesn't consider sensitive attributes

k-Anonymity

- *l*-diversity: Each anonymity set also has representatives from at least *l* different attributes
  - This can be bad in heavily skewed data sets (e.g. sickness):
    - May require very large sets to satisfy
    - May still leak information about people
- *t*-closeness: Each anonymity set should have a similar distribution of sensitive attributes as the original data set
  - *t* measures the difference in distribution

k-Anonymity

Complementary release attack: If the same person's data is released twice under k-anonymity, the combination of the data sets can have no 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

	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-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 youngest.
  - 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 wants 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!

k-Anonymity

- Other problems:
  - What is the usefulness of data under k-Anonymity?
    - Can we achieve optimality in minimizing data change?
  - Curse of dimensionality
  - What about other types of data (e.g. trajectories)?
  - What if the distinction between QID and sensitive attribute is not clear?

Differential Privacy

- Scenario: Data owner wants to answer queries in a way that does not compromise privacy
  - We saw that some naïve methods of doing so don't work
- Hard to explain to laymen, but easy to implement
- Answers all queries with differential *noise*
- Used by Apple, Microsoft, Google, etc. for collecting data
  - Word prediction, website visits, identifying heavy websites, etc.

Differential Privacy

- We say two databases are neighboring if at most 1 row is different between them
- A query Q is ϵ-differentially private if for all neighbouring databases D_1 and D_2 and for all q:

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

- Intuition: changing one person's data does not change the response to the query Q much

Differential Privacy

- Suppose the salaries of 5 people are:

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

- For legal compliance, the company is required to disclose the mean salary.
- Now E has left the company. If someone queried the mean salary before, it was $240. Now it is $200. $240*5 - $200 * 4 = $400.

Differential Privacy

- Instead, let us add noise before returning the result:

Employee	Salary
A	$\$200+N(0, \sigma^2)$
B	$\$210+N(0, \sigma^2)$
C	$\$240+N(0, \sigma^2)$
D	$\$150+N(0, \sigma^2)$
E	$\$400+N(0, \sigma^2)$

- The real data is never revealed – noise is only added once, internally
- The difference between two neighboring data sets is dominated by Gaussian noise
- However, the mean has less noise than any element: $N(0, \sigma^2/5)$

Satisfying Differential Privacy

Does normal distribution noise satisfy differential privacy for mean query?

Suppose two neighboring datasets D_1 and D_2 have means M_1 and M_2
We write $k = M_1 + x_1 = M_2 + x_2$

$$\frac{\Pr(Q(D_2) = k)}{\Pr(Q(D_1) = k)} = \frac{\left(\frac{e^{-{x_2}^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}}\right)}{\left(\frac{e^{-{x_1}^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}}\right)} = e^{({x_1}^2 - {x_2}^2)/2\sigma^2}$$

Unfortunately, this cannot be bounded as x_1 and x_2 can be infinitely large

Satisfying Differential Privacy

Does Laplace distribution noise satisfy differential privacy for mean query?

Suppose two neighboring datasets D_1 and D_2 have means M_1 and M_2
We write $k = M_1 + x_1 = M_2 + x_2$

$$\frac{\Pr(Q(D_2) = k)}{\Pr(Q(D_1) = k)} = \frac{\left(\frac{e^{-|x_2|/b}}{2b}\right)}{\left(\frac{e^{-|x_1|/b}}{2b}\right)} = e^{(-|x_2|+|x_1|)/b} \leq e^{|x_1-x_2|/b} = e^{|M_1-M_2|/b}$$

If we can bound the $|M_1\text{-}M_2|$, which is the maximum difference between two query results, we achieve (1/b)-differential privacy

Satisfying Differential Privacy

- The previous result does not at all use the fact that we are computing the mean – it is generalizable to any function for which we can bound the *sensitivity* defined as:

$$\max_{neighbors\ D_1, D_2} ||Q(D_1) - Q(D_2)||$$

- Some examples:
  - COUNT: Sensitivity = 1
  - SUM: Sensitivity = 1
  - Histogram results (e.g. 23 students in A range, 35 students in B range…): Sensitivity = 1
  - MIN, MAX, Median: Bad

Smooth Sensitivity

- Some functions may have poor sensitivity in general, but good sensitivity on real data (e.g. median salary)
- However, if we consider only *local* sensitivity (i.e. fix D_1, take neighbors D_2), then this does not satisfy differential privacy (why?)
- A middle ground between local sensitivity and global sensitivity is smooth sensitivity, and it does satisfy differential privacy [Nissim, Raskhodnikova, Smith 2007]:

$$\max_k e^{-k\varepsilon}\left(\max_{y:d(x,y)=k} LS(y)\right)$$

  - x is the original data, y is any other data set
  - d(x,y) is the distance between x and y
  - LS(y) is the local sensitivity computed at y

Benefits of differential privacy

- Effect of attacker's side knowledge is well-modeled
- Composition is easy
  - Any function applied after differential privacy does not affect its guarantee
  - Combination of several differentially private databases still has privacy
- Can be used for data collection
- Can be used for machine learning

Another Differential Privacy Example

Count-mean sketch (e.g. web domain visit count)

- When event happens, from a number of predefined hash functions, randomly select one to hash the event (e.g. h_3(www.google.com)=5)
- Write the vector as (0, 0, 0, 0, 1, 0, 0…) but randomly flip each of those bits at probability $1/(\epsilon^{\epsilon/2}+1)$
- Submit the vector to the aggregator; the aggregator adds it to row 3 (one row for each hash function)
- After normalization, mean of all i, h_i(www.google.com) is an estimate for # of page visits

Differential privacy for bidding mechanisms

Unlimited supply auctions

- Objective: Not privacy; an auction system where people give honest bids rather than try to game the system
- Adding noise directly to bids is not a good idea because it will often output a price that no one is able or willing to pay
- Instead, given a database *X* of all bids, we output a given price *p* with probability proportional to

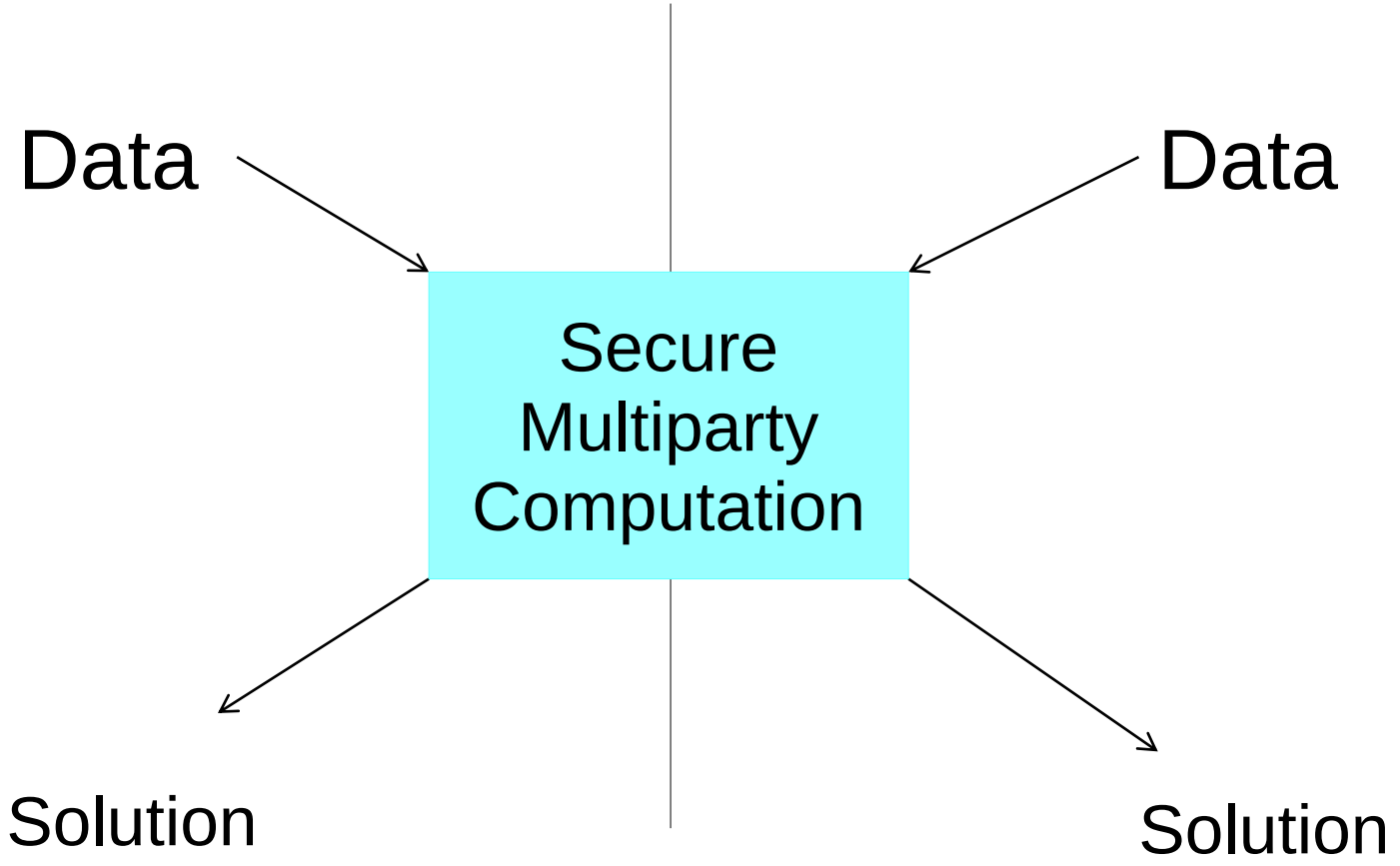
$$e^{-\varepsilon u(X,p)/2}$$

- *u(X,p)* is the total revenue if sold at price *p*
- Intuition: Changing one bid almost does not change the result
- Good properties: truthfulness is nearly dominant, resistance to collusion, composition is robust (e.g. rerun auctions)

Secure Multiparty Computation

- Suppose that two entities want to compute a function that requires input data from both entities, without sharing the data
  - The function is publicly known and relatively simple
  - Noise is not desirable
  - Both parties are honest (but curious)
- e.g. Yao's millionaires problem: Who is richer?
- This is possible through SMPC
  - Implemented by Yao's garbled circuits
- Any Boolean function with fixed-size input can be expressed as a circuit

Secure Multiparty Computation



Secure Multiparty Computation

- One person (Alice) will take up the role of garbler, and the other person (Bob) will take up the role of evaluator
  - Alice will create the garbled circuit, and will know details about the encoding of each gate in the circuit
  - Bob will evaluate on the circuit without knowing the meaning of his evaluation
  - It does not matter who is who
- Each gate is encoded separately
- But first, we need Oblivious Transfer

Oblivious Transfer

- Bob asks Alice to send one of m_0, m_1 *without telling Alice which one to send*
  - Only Bob knows which one was requested; Alice doesn't know
  - Also, Bob is only allowed to receive one of them
- Consider RSA with key pair (e, d) in modulo N

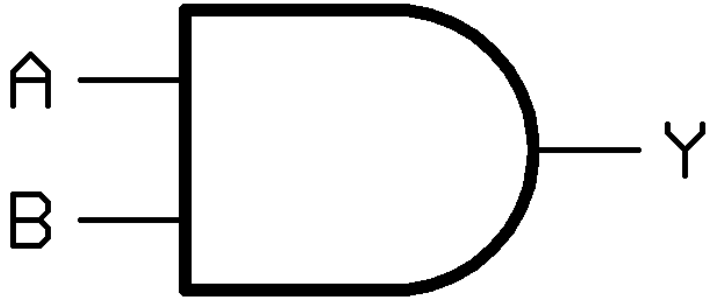
Protocol

1. Alice generates and sends random x_0, x_1 to Bob
2. Suppose Bob wants Alice to send m_b. Bob sends $v = (x_b + k^e) \bmod N$

   k is random and secret
3. Alice returns both:
   - $y_0 = m_0 + (v - x_0)^d \bmod N$
   - $y_1 = m_1 + (v - x_1)^d \bmod N$
4. Bob can only decrypt the one he requested

Yao's Garbled Circuits

- For each gate, the garbler creates a *garbled table*



Input A	Input B	Output	Garbled Input A	Garbled Input B	Garbled Output
0	0	0	A0	B0	$Enc_{A0, B0}(Y0)$
0	1	0	A0	B1	$Enc_{A0, B1}(Y0)$
1	0	0	A1	B0	$Enc_{A1, B0}(Y0)$
1	1	1	A1	B1	$Enc_{A1, B1}(Y1)$

Yao's Garbled Circuits

Then, the garbler shares only the garbled output table with the evaluator, after random shuffling

Garbled Output
$Enc_{A1, B0}(Y0)$
$Enc_{A1, B1}(Y1)$
$Enc_{A0, B0}(Y0)$
$Enc_{A0, B1}(Y0)$

- The evaluator can unlock one and only one of the outputs if given both input keys
  - The evaluator tries to decrypt all four, and only one will succeed
  - To know which one succeeded, we can specially format Y0 and Y1 (e.g. ends with 32 zeros)
- Only Alice will know the real meanings of the garbled inputs A0, A1, B0, B1; Bob can receive those strings but won't know if they correspond to a real 0 or 1

Yao's Garbled Circuits

- Now, to evaluate the gate, Bob needs to know one of A0/A1, and one of B0/B1
  - A0/A1 is Alice's input; Alice can just send him the right string (since Alice knows which one is correct)
  - B0/B1 is Bob's input, but only Alice knows their meaning. However Alice cannot know Bob's input.
    - Using *oblivious transfer*, Bob can request the right string without Alice knowing if he requested B0 or B1
- Alice makes a garbled table for each gate in the circuit
  - If a future gate takes Y as input, then the corresponding Y0 and Y1 are the same in that table

Yao's Garbled Circuits

- Largely considered theoretical for 15 years until first practical implementation in 2004 (Fairplay)
  - Still slow: e.g. 20 minutes for AES encryption in 2009 (Pinkas et al.), reduced to several seconds recently with parallelism
- Some efficiency optimizations:
  - Point and permute: Randomly select A0 (B0) to start with a bit of 0/1 and A1 (B1) to start with a bit of 1/0. Indicate in the garbled circuit table which one to decrypt depending on the starting bit
  - Choose the output labels such that one of the encryptions becomes 0
  - Circuit tricks (e.g. remove gates where both inputs are from the same party, specific designs, free XOR)
  - Many parallelism tricks (e.g. garble-send-evaluate pipeline)

Free XOR

We can implement XOR gates without any garbling:

Input A	Input B	Output	Garbled Input A	Garbled Input B	Garbled Output
0	0	0	A0	B0	$A0 \oplus B0$
0	1	1	A0	B1	$A0 \oplus B1$
1	0	1	A1	B0	$A1 \oplus B0$
1	1	0	A1	B1	$A1 \oplus B1$

To maintain consistency (ensure $A0 \oplus B0 = A1 \oplus B1$, $A0 \oplus B1 = A1 \oplus B0$), we set:

$$A1 = A0 \oplus R$$

$$B1 = B0 \oplus R$$

This means XOR gates do not contribute to the practical cost of garbled circuits

Yao's Garbled Circuits

- Gate garbling is also limited by the cost of oblivious transfer
  - This is possible by having most instances of oblivious transfer be "based" on a small number of instances (Asharov et al. 2013)
- A malicious garbler can be forced to follow rules with zero-knowledge proofs
  - Alternatively, the "cut-and-choose" algorithm: have them garble many circuits and open a few of them
- Multi-party computation is achieved through secret sharing of each garbled gate table in an arithmetic circuit

Private Information Retrieval

- Sometimes we want to query a database without revealing the query to the database
  - e.g. Asking a medical database about certain symptoms
  - e.g. Asking a patent database about a new idea
- Trivial but impractical solution: Download the entire database
- Practical solutions reduce communication overhead
- Two types:
  - Information-theoretic PIR: Databases do not obtain the query
    - Proven impossible for single database (except full download)
  - Computational PIR: Databases need to compute difficult problem to obtain query

4-Database Information-Theoretic PIR

- First, represent the database as a 2-dimensional table, and Alice wants to obtain one of those 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 column and row 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

- She creates R’ and C’ which flip the rows and columns in R and C based on which element she really wants. Suppose she wants $x_{2,2}$ (row 2, column 2):
  - R’ = {Rows 3, 5}
  - C’ = {Columns 2, 3}
- Then she creates four requests to the four 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’

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

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

4-Database Information-Theoretic PIR

- Alice can obtain $x_{2,2}$ just by XORing all 4 responses
  - The desired element is in the intersection of only one of R and R', and only one of C and C', so only one of DB1, DB2, DB3, and DB4
  - No other element has this property: it 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 any of the returned answers
    - If it is in R and R', it is in an even number of returned answers 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 element
- Computation cost is O(n) – XOR of about ¼ of all elements
- Several other protocols exist with fewer databases/better query length

Single-Database Computational PIR

- Decisional Diffie-Hellman: Decide if (x, y, z) is a Diffie-Hellman triplet
  - They are if $x \equiv g^a \pmod p$, $y \equiv g^b \pmod p$, $z \equiv g^{ab} \pmod p$
  - g and p are given
  - This is (presumably) computationally difficult
- Two triplets can be multiplied in a specific manner to create another triplet:
  - $(g^a, g^{b1}, g^{ab1}) * (g^a, g^{b2}, g^{ab2}) = (g^a, g^{b1+b2}, g^{ab1+ab2})$ (all in mod p space)
- We will use this to create a single-database computational PIR

Single-Database Computational PIR

- Suppose the database consists of n bits
- Suppose Alice wants x_i, the i-th bit
- Alice creates DDH triplets for all elements except i: (g^a, g^{bj}, $g^{cj} = g^{abj}$)
  - For i, she creates a random (g^a, g^{bi}, g^{ci}) that is not a triplet
- She sends all DDH triplets to the server
- The server returns a multiple of all triplets for which the bit is 1
- He returns (g^a, B, C) such that

$$B = \prod_{x_j=1} g^{bj} \qquad C = \prod_{x_j=1} g^{cj}$$

Single-Database Computational PIR

$$B = \prod_{x_j=1} g^{bj} \qquad C = \prod_{x_j=1} g^{cj}$$

- Alice determines if the response (g^a, B, C) is a DDH triplet
  - If $x_i = 1$, then the randomly created non-triplet (g^a, g^{bi}, g^{ci}) would be in the multiple. The result would not be a DDH triplet
  - If $x_i = 0$, then it is not in the multiple; the result is a multiple of triplets, so it is a triplet
- Query length is O(n), response length is O(1)
- Can balance out query/response length with 2 dimensions and further optimized with recursion

Location Privacy

- Many sources of leakage, including:
  - **Location-based services**: User interaction with e.g. social networks, which often allow proximity-based queries
  - **Data leakage:** Data loss from data collectors (may be untrustworthy)
  - **Collection leakage:** Hardware, IoTs collecting data unexpectedly
- Threatens user privacy:
  - Exact location is highly private – reveals activities, relationships, etc.
    - Much easier to attack someone you can find
  - Four spatiotemporal points is enough to identify an individual

Attacks on Location-Based Services

- While LBSs usually only reveal relative distance (e.g. proximity), this can be used to reveal exact locations
- Fundamental issue: Location spoofing is easy
  - The device API is trusted to give accurate results to the app; the API can be changed
  - Some apps consult an external server, which can always be intercepted
- By faking our location, we can measure our relative distance to the target from multiple angles; this allows us to deduce exact location
  - Trilateration
  - Space partitioning

Attacks on Location-Based Services

Trilateration:

- Given multiple distance measurements, we can pinpoint the location of an object
  - However, distance measurements are usually imprecise
- A least-squares formulation:

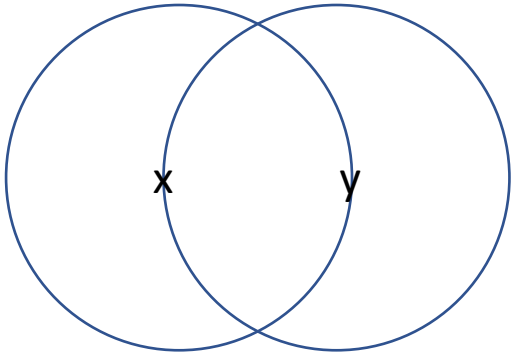
Given measurement points p_i and measures distances r_i,
Find p = $\underset{p}{\operatorname{argmin}} \sum_{i=1}^{n} (||p_i - p|| - ri^2)^2$

- This formulation can be solved and is more robust to noisy measurements

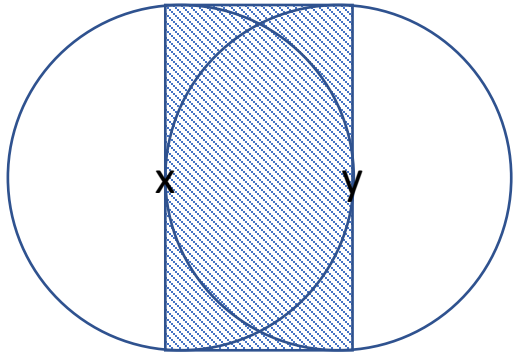
Attacks on Location-Based Services

Space partitioning:

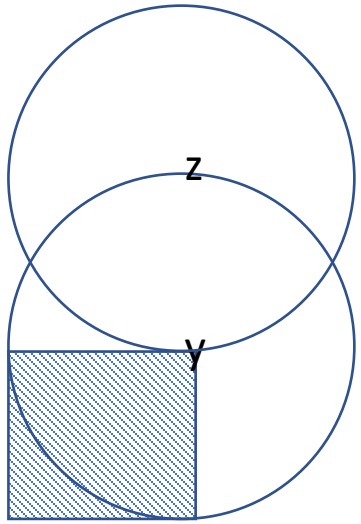
- We can use two points to decide if the target is in the left or right half of a circle
- Iterate repeatedly to reduce the circle size
- Example:



If the target point is in x's circle, and it is also in y's circle…



Then we know it is in the right half of the circle (simplified as rectangle)



We can then repeat it with z to decide which half of the rectangle it is in (here we find it is in y but not in z)

Defending Location-Based Services

Some solutions:

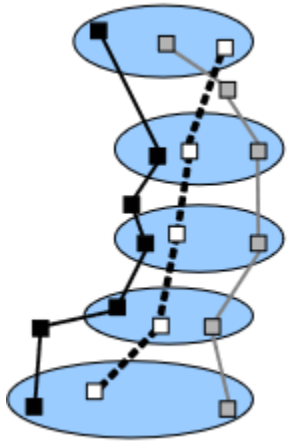
- Add noise – but this may perform worse than designed because many locations are “unconvincing”
- Each user is placed in a grid, not a point; increase granularity of grid
- k-anonymity:
  - Query with k-1 fake points and 1 real point (“dummy locations”)
  - Query for a large region that can contain k points (“cloaking”)
  - Weaknesses: Most points are not convincing, side knowledge is not considered (also a weakness in original k-anonymity)
- Differential location privacy (geo-indistinguishability): Query point includes noise such that the query distribution between nearby points is similar, and faraway points is less similar

Trajectory anonymity

- Trajectories are points with order and/or time information connected to one individual
- Trajectory analysis is valuable, but highly privacy-sensitive
- (k, d)-anonymity:
  - Two trajectories are called d-colocalized if for any time t, (x1, y1, t) in trajectory 1 and (x2, y2, t) in trajectory 2 have at most distance d
  - A (k, d)-anonymity set is a set with k d-colocalized trajectories
  - Anonymity sets can be created with a greedy algorithm
  - Publishing these anonymity sets still allows analysis of crowd sizes and movement

Trajectory anonymity

- Trajectory generalization:
  - Create an anonymity set iteratively
    - In each step, a nearby trajectory is added to the anonymity set
    - Points of two trajectories are merged by finding “point links” between points of similar time and location
    - Further trajectories are added by adding them to the closest previous point links
    - Unmatched points are suppressed (not published)
  - After creating these anonymity sets, we can publish the anonymity sets or generate k trajectories from these sets to use to answer queries



[Nergiz et al. 2008]

Hardware risks

- Mobile devices are built with hardware from many different vendors: universal integrated circuit cards, processors, sensors, etc.
- It is nearly impossible to detect malicious hardware
- Other unexpected risks: accelerometers, gyroscopes, magnetometers
  - How a person walks is identifiable
- Some solutions:
  - Trusted execution environments
  - Sandboxing
  - Synthetic trajectories
  - Better user interfaces