Lecture notes: Data Privacy

Data owners (such as businesses and governmental organizations) may want to share their data to allow data analysis. As a motivating example, hospitals may want to enable medical research by sharing clinical data. Sharing such data may lead to a serious compromise of privacy when the data contains private information. We consider four ways to allow a data user to perform some query Q on the data owner's database D while protecting privacy, suitable for four somewhat different scenarios:

- **k-anonymity:** The data owner publishes a distorted version of D, denoted as D_k. Then, the data user can perform any query on $Q(D_k)$.
- **Differential privacy:** The data owner does not publish any data. Instead, the data owner only accepts queries Q which satisfy differential privacy from the data user.
- **Secure multi-party computation:** There are two data owners owning D_1 and D_2, and they want to jointly compute $Q(D_1, D_2)$ without revealing either.
- **Private information retrieval:** The query Q is private, and the data owner needs to support $Q(D)$ without knowing what Q is.

Any material in here that was not covered by the lectures/slides is not in the exam and should be considered bonus material for your knowledge.

k-anonymity

k-anonymity allows the data publisher to distort their data D into D_k and publish it. Let us visualize the data D as a table, where each row represents one person. There are three types of columns in the table:

- **Identifiers:** These columns strictly identify the person and provide no value during analysis. For example, they may be the name or ID of the person. The identifiers are never published.
- **Quasi-identifiers (QIDs):** These columns do not directly identify the person and they are valuable during analysis. For example, they may include age, sex, height, and weight. They need to be published for analysis, but they may still reveal the person's identity indirectly. Therefore, they need to be distorted.
- **Sensitive attributes:** These columns are sensitive and we do not want the data user to figure out the real identity of people with these sensitive attributes. They are also published, but they are not distorted.

We say that two people are in the same **anonymity set** if they have the same QIDs. It is not necessary for them to have the same sensitive attributes or identifiers. It is not likely for two people to have the same QIDs in the original data set D. Therefore, we distort D into D_k so that people would be put into the same anonymity set.

We say that D_k satisfies k-anonymity if every anonymity set consists of at least k elements.

Table 1 shows an example of a hospital's data set D. The Name is an identifier (so it is never published), Age and Height are QIDs, and Sickness is the sensitive attribute. Table 2 shows a distorted version D_k that is ready to be published. The rows in the same anonymity set in Table 2 are marked with the same color (red, green, or blue). We can see that the three anonymity sets have size 3, 4, and 4 respectively. Therefore, D_k satisfies 3-anonymity.

Table 1: Table of patient sicknesses held by hospital; will not be published.

Name	Age	Height (cm)	Sickness
Alice	13	145	Hepatitis A
Bob	15	161	Hepatitis A
Carol	21	165	No sickness
Dave	33	177	Chronic coughing
Eve	33	160	Hepatitis A
Frank	35	172	Hepatitis B
Grace	41	180	Flu
Henry	43	156	Hepatitis A
Ivy	45	163	Flu
James	45	178	Flu

Table 2: Published table of patient sicknesses held by hospital satisfying 3-anonymity.

Age	Height (cm)	Sickness
20	150	Hepatitis A
20	150	Hepatitis A
20	150	No sickness
40	200	Chronic coughing
40	150	Hepatitis A
40	150	Hepatitis B
40	200	Flu
40	150	Hepatitis A
40	150	Flu
40	200	Flu

Table 3: Data error of the above table (same as "change" in Assignment 3).

Data error	
$\lvert 20-13\rvert + \lvert 150-145\rvert$	$= 12$
$\lvert 20-15\rvert + \lvert 150-161\rvert$	$= 16$
$\lvert 20-21\rvert + \lvert 150-165\rvert$	$= 16$
$\lvert 40-33\rvert + \lvert 200-177\rvert$	$= 30$
$\lvert 40-33\rvert + \lvert 150-160\rvert$	$= 17$
$\lvert 40-35\rvert + \lvert 150-172\rvert$	$= 27$
$\lvert 40-41\rvert + \lvert 200-180\rvert$	$= 21$
$\lvert 40-43\rvert + \lvert 150-156\rvert$	$= 9$
$\lvert 40-45\rvert + \lvert 150-163\rvert$	$= 18$
$\lvert 40-45\rvert + \lvert 200-178\rvert$	$= 27$
Sum:	$= 193$

How effective is k-anonymity?

We can measure the effectiveness of the data distortion procedure in k-anonymity by the **data error**. Various definitions of data error exist; we use the following in Assignment 3. If a certain QID (say, Age) of person A is I in the original data D, and it is distorted to become I' in D_k, we say that the data error of this element is $|I' - I|$. Then, the data error of the data distortion procedure is simply the sum of the data error for all QIDs and all people in the table. Table 3 shows the calculation of data error for Table 2.

Problems with k-anonymity

One problem with k-anonymity is that it may still be possible to deduce the sensitive attribute of someone if every person in their anonymity set has the same sensitive attribute. This is resolved by another property known as ℓ-diversity. A published data set is said to satisfy ℓ-diversity if every anonymity set has at least ℓ different sensitive attributes in it. Table 2 satisfies 2-diversity.

Another issue is that k-anonymity can be compromised if two related distorted data sets are published. The data user can correlate the entries in the data sets to deduce private information. Even if the two distorted data sets each individually achieve k-anonymity for any value of k, if we consider the two distorted data sets as a whole set, it is possible it cannot achieve any k-anonymity with $k > 1$.

Notably, achieving minimum data error for k-anonymity is known to be an NP-complete problem in general, when either the number of dimensions (QID columns) or the number of clusters required is not fixed. This is true for any L_p-norm (the above definition uses the L_1-norm). This is known as the optimal k-means clustering problem.

This is not true if k and the number of dimensions is fixed. For example, if there is only one QID, a $O(n^2)$ time algorithm, where n is the number of rows, exists for any k using dynamic programming as follows. Suppose the QIDs are arranged in ascending order in D, and denote $D_{i...j}$ as all the elements between D_i and D_j, inclusive. (The data set is $D_{1...n}$. Let the final anonymity set be $D_{m...n}$, where $n-m \geq k$. The optimal k-anonymity solution for $D_{1...n}$ is therefore the optimal k-anonymity solution for $D_{1...m}$ plus the cost of $D_{m...n}$. Therefore, by varying m, we can reduce the larger n-sized problem to the smaller m-sized problem.

Differential privacy

A different way to allow privacy-preserving queries of a data set is to restrict the types of queries that can be made to the data set. Two data sets D_1 and D_2 are said to be neighbouring if at most one row (one person's data) is different between D_1 and D_2. For example, if D_1 includes all the patients that have visited the hospital today by 1 PM, one patient came in between 1 PM and 1:05 PM, and D_2 includes all the patients by 1:05 PM, then D_1 and D_2 are neighboring.

Observe that a query Q can reveal private details of that last patient if applied to the above D_1 and D_2. Suppose Q returns the number of people who have HIV at the hospital. If $Q(D_1) + 1 = Q(D_2)$, then we know that the last patient who arrived has HIV. This motivating example serves to show that we need $Q(D_1)$ and $Q(D_2)$ to return values with similar probabilities.

For a query Q to be differentially private, $Q(D)$ needs to be understood as a random variable with a probability distribution of possible outputs. Then, a query Q is said to be a ε-differentially private query if for any neighbouring data sets D_1 and D_2, and for any possible output value k,

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

Note the following about the above definition:

- e is the natural number. ε must be a constant, independent of D.
- The above must be true for all values k that can be the output of a query on a data set.
- ε cannot be negative as we cannot achieve $e^{\varepsilon} < 1$: If $\frac{Pr(Q(D_1)=k)}{Pr(Q(D_2)=k)} < 1$, then $\frac{Pr(Q(D_2)=k)}{Pr(Q(D_1)=k)} > 1$.
- Generally, the way to achieve differential privacy is to take the regular, non-private query $\bar{Q}(D)$ (which returns the true value) and add random noise, so $Q(D) = \bar{Q}(D) + R$ where R is some random noise taken from a random distribution.
- As a consequence of the definition, if any valid data set can return some given value k with non-zero probability, then all valid data sets must be able to return k with some probability. A further corollary is that deterministic queries cannot be differentially private, except the trivial query that returns the same value for all input.

We give two examples of mechanisms that can satisfy the above definition.

Example 1. Consider the above hospital database. Suppose there are $\bar{Q}(D) = N$ patients with HIV. The hospital will output $Q(D) = \bar{Q}(D) + R$, where R is generated as follows. First, decide the sign of R randomly by flipping a coin (heads will be positive, tails will be negative). Then, flip coins until we see tails. With each heads flip, the size of the counter will be increased by 1. We can see that the probability of returning a $N + \delta$ as a result of this procedure is the two-sided geometric distribution, $Pr(Q(D) = N + \delta) = (1/2)^{|\delta|+2}$ for $\delta \neq 0$, and $Pr(Q(D) = N) = (1/2)$. For any two neighbouring data sets, the total number of patients with HIV differs by at most 1; consider D_1 with N HIV patients with D_2 with $N + 1$ HIV patients. Then for any $k = N + \Delta k$ where $\Delta k \geq 1$,

$$\frac{Pr(Q(D_2) = k)}{Pr(Q(D_1) = k)} = \frac{(1/2)^{|\Delta k|}}{(1/2)^{|\Delta k|-1}} = 2$$

The inverse can be shown to be equal to 2 as well if $\Delta k \leq 0$. If $\Delta k = 0$, the above becomes 4. Therefore, Q satisfies ln 4-differential privacy.

Example 2. Consider a database of salaries of professors. Suppose the university allows you to make a query on the mean salary paid to professors, with noise from $Laplace(0, b)$ (a Laplace distribution with mean 0 and diversity b) added to the output. Suppose the mean salary of D_1 is M_1 and the mean salary of D_2 is M_2 and without loss of generality $M_1 > M_2$. Suppose any professor's maximum salary is known to be G. Then for any $k > M_1$ where $k = M_1 + x_1 = M_2 + x_2$,

$$\begin{aligned}
\frac{Pr(Q(D_2) = k)}{Pr(Q(D_1) = k)} &= \frac{(1/2b)e^{-\frac{x_2}{b}}}{(1/2b)e^{-\frac{x_1}{b}}} \\
&= e^{\frac{x_1 - x_2}{b}} \\
&\leq e^{\frac{|x_1 - x_2|}{b}} \\
&= e^{\frac{|M_1 - M_2|}{b}}
\end{aligned}$$

For $M_2 < M_1$, the inverse of the above equation produces the same result. Therefore, Q satisfies $\frac{|M_1 - M_2|}{b}$-differential privacy.

Secure Multiparty Computation

When two parties want to jointly compute a function Q on their separate data sets D_1 and D_2 without leaking their data sets to each other, they may do so using secure multiparty computation (SMPC). We will assume that both parties will honestly execute the protocol because they both want to know $Q(D_1, D_2)$, but they should be prevented from knowing the other party's data in the process. For example, two millionaires who want to find out who is richer can use SMPC to find the answer without revealing their wealth.

SMPC is generally computationally costly: it takes around 20 minutes to compute a single AES encryption with two keys held by two parties. (Consider that any internet-capable device encrypts thousands of packets with AES every second.) One advantage of SMPC is that there is no distortion or noise; the true output is given to both parties.

Achieving SMPC

We can achieve SMPC by using Yao's garbled circuits. Represent Q as a circuit, where the circuit inputs are D_1 and D_2, and both parties can arrive at the output.

We describe how garbling works as follows. One person (say, Alice) will garble the circuit and one person (say, Bob) will evaluate it. Alice will garble each gate of the circuit as follows. Suppose the truth table of the gate with inputs I, J and output O is written as follows (in this example, it is an XOR gate):

I	J	O
0	0	0
0	1	1
1	0	1
1	1	0

Alice will randomly generate six random strings known as garbled strings, I_0, I_1, J_0, J_1, O_0, O_1. The garbled strings look like long, random bit strings. Alice will remember, for example, that I_0 corresponds to a 0 bit in the first input of this gate, and O_1 corresponds to the output bit of this gate being 1. Bob will not know this information, so if Bob sees I_1, for example, he does not know what it means. Alice will then encrypt each output garbled string by using the corresponding input garbled strings as keys. Alice also chooses some publicly known long string A (say, 64 bits of zero). For the XOR gate above, the encrypted output garbled strings, after random reordering, is as follows:

Encrypted O
$Enc_{I_1,J_0}(O_1 \| A)$
$Enc_{I_0,J_0}(O_0 \| A)$
$Enc_{I_0,J_1}(O_1 \| A)$
$Enc_{I_1,J_1}(O_0 \| A)$

The above table, the garbled gate table, is shared with Bob. Given some garbled input strings, Bob will try to decrypt all 4 values above. Only one of them will return an output with proper format (Bob should see the string A), and Bob can then obtain either O_0 or O_1 from it. The other four will return bad output that cannot be used by Bob in any way.

Note that Bob cannot know, from decrypting the above table, whether O corresponds to 0 or 1 either. This is because it is randomly reordered. For example, the second element above actually corresponds to $I_0 \oplus J_0 = O_0$, but Bob does not know that.

We re-iterate the above construction for all gates in the circuit. For example, if the output bit of this gate is connected to some other gate, then it must be constructed with the same garbled strings O_0 or O_1 at the input.

Alice will give Bob the garbled gate tables for all gates, and Alice will give him her garbled input strings at the start of the circuit. Her privacy is preserved since Bob does not know what her garbled inputs mean. However, Bob also needs to know the garbled input strings corresponding to his inputs as well, without telling Alice his actual inputs. The mechanism to do so is known as oblivious transfer.

Oblivious transfer can be done as follows. Suppose Bob's true input bit is 0, and Bob wants J_0 from Alice. Alice generates the RSA public parameters, N as a product of two large primes, public key e, and private key d, where $m^{de} \equiv m \pmod{N}$ for any message m. (All of the following operations are in modulo N space.) Then Alice generates two random strings x_0 and x_1 and sends them to Bob. Bob chooses x_0 (because his input bit is 0), and generates a random k, and sends $v = x_0 + k^e$ to Alice. Alice will reply with $m_0 = J_0 + (v - x_0)^d$ and $m_1 = J_1 + (v - x_1)^d$ to Bob. Note that $m_0 = J_0 + k$ but $m_1 \neq J_1 + k$; m_1 is effectively a failed attempt to decrypt v but Alice does not know that. After sending m_0 and m_1 to Bob, Bob will take $m_0 - k$ (discarding m_1) and thus obtain J_0.

We see that Alice and Bob now both know the garbled circuit and the correct way to obtain the output for each gate: attempt to decrypt all four values in the corresponding garbled gate table. Therefore both Alice and Bob can execute the entire circuit and obtain the garbled output string of the entire circuit. Only Alice will know the meaning of the garbled output string, so Alice will share that with Bob. (It is possible to ensure Alice cannot cheat at this stage with cryptographic commitments.)

Private Information Retrieval

Private Information Retrieval (PIR) is useful if a data user wants the data owner to answer $Q(D)$ without knowing what Q was. A scheme can be said to be information theoretic or computational:

- **Information theoretic** PIR: We can prove that the data owner cannot know Q without any restrictions on the data owner.
- **Computational** PIR: We can prove that the data owner cannot know Q if we assume that the data owner cannot compute a certain hard problem.

This is analogous to cryptography: one-time pads are information theoretically secure, whereas RSA relies on a computational assumption (the difficulty of the integer factorization problem). PIR schemes can also be said to be single-database or multi-database:

- **Single-database** PIR: There is only one data owner.
- **Multi-database** PIR: There are multiple non-colluding data owners who are all willing to serve the data user.

A trivial single-database information theoretic PIR scheme is to have the user download the entire database. This costs a lot of communication overhead, so it is not desirable. However, it can be proven that there is no better single-database information theoretic PIR scheme.

We construct a multi-database information theoretic PIR scheme as follows. Suppose the database is written as a block of bits:

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

Each $x_{i,j}$ represents one bit. Suppose there are four database owners. Alice wants to retrieve some specific $x_{I,J}$ without letting any of the four database owners know what she wants to retrieve. She will randomly choose each column with one half chance and each row with one half chance. Suppose she chooses the following (coloured):

R

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

C

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

Let us call the set of rows chosen R and the set of columns chosen C respectively. Suppose the real data she wants is $x_{I,J}$, i.e. the data in row I column J. She will generate $R' = R \oplus I$, where $\oplus$ means putting row I in R if it was not in R, and removing row I from R if it was in R. She will also generate $C' = C \oplus j$. Then she will send the following four requests to the four databases:

1. DB1: Return XOR of all data in rows R and columns C.
2. DB2: Return XOR of all data in rows R' and columns C.

3. DB3: Return XOR of all data in rows R and columns C'.

4. DB4: Return XOR of all data in rows R' and columns C'.

Suppose she wants $x_{4,3}$, in our example the requests are drawn in the following four tables:

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

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

The replies are the XORs of all the darker green cells, which are:

- DB1: $x_{3,2} \oplus x_{3,3} \oplus x_{5,2} \oplus x_{5,3}$.
- DB2: $x_{3,2} \oplus x_{3,3} \oplus x_{4,2} \oplus x_{4,3} \oplus x_{5,2} \oplus x_{5,3}$.
- DB3: $x_{3,2} \oplus x_{5,2}$.
- DB4: $x_{3,2} \oplus x_{4,2} \oplus x_{5,2}$.

Alice simply XORs the replies of all four databases to get her $x_{4,3}$. It is easy to prove, in general, that following this scheme, only $x_{I,J}$ will appear an odd number of times in the output; any other bit will appear an even number of times. This means that XORing all the replies will produce $x_{I,J}$. Note that each database only sees, from its own perspective, a series of random row and column requests. Also note that if any two databases collude, they can deduce either the row or the column of Alice's real request, or both. The communication required to satisfy such a request is $O(\sqrt{n})$, where there are n bits in total in the database.

We construct a single-database computational PIR scheme as follows. This requires the use of a homomorphic encryption scheme, which satisfies the special property that encryption $\mathcal{E}$ satisfies:

$$\mathcal{E}(M_1 \cdot M_2) = \mathcal{E}(M_1) \cdot \mathcal{E}(M_2)$$

The key is abbreviated away from the notation to avoid confusion. Note that the above property is not true for any of the encryption schemes we have learned. In particular, homomorphic encryption is computationally expensive, sometimes prohibitively so.

Suppose that we are using such a scheme, where the data user Alice has the ability to encrypt with $\mathcal{E}$ and decrypt with $\mathcal{F}$, and the data owner Bob has neither. Suppose the data is written as $x_1, x_2, \cdots, x_n$, each x_i representing one bit, and Alice wants x_I. For each bit x_i she constructs q_i for all i from 1 to n where:

$$q_i = \begin{cases} \mathcal{E}(1), & \text{if } i = I \\ \mathcal{E}(0), & \text{otherwise.} \end{cases}$$

She would of course ensure that $\mathcal{E}(0)$ is different every time it is encrypted (by using an IV). The data owner, upon receiving $q_1, q_2, \cdots, q_n$, replies with:

$$\mathcal{R} = \Sigma_{i=1}^{n} x_i \cdot q_i$$

When Alice decrypts R, homomorphic encryption ensures that:

$$\mathcal{D}(R) = \mathcal{D}\big(\Sigma_{i=1}^{n} x_i \cdot q_i\big) = \Sigma_{i=1}^{n} x_i \cdot D(q_i) = x_I$$

In particular, the last equality is because $D(q_i) = 0$ for all i except $i = I$, where instead $D(q_i) = 1$. Thus she has retrieved x_I revealing no information about what she requested, assuming the attacker cannot break the homomorphic encryption, which is the computational assumption.