final

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2017-2018

MEng Honours Degree in Mathematics and Computer Science Part IV
MEng Honours Degrees in Computing Part IV
MSc in Advanced Computing
MSc in Computing Science (Specialist)
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C408H

PRIVACY ENHANCING TECHNIQUES

Wednesday 13 December 2017, 10:00
Duration: 70 minutes

Answer TWO questions

Paper contains 3 questions
Calculators not required

1 This is a question about differential privacy.

a Consider the following mechanism. The input to the mechanism is a bit ***B*** (0 or 1). The mechanism chooses two new random bits ***R*** and ***S***. If ***R***=0 then the mechanism returns ***B***. If ***R***=1 then the mechanism returns ***S***. Show that this mechanism is differentially private.

b Consider the following mechanism. The input to the mechanism is a bit ***B*** (0 or 1). The mechanism returns a binary value ***B xor R*** where ***R*** is a new random bit value. Show that this mechanism is differentially private and comment on its information loss/gain.

c Write an ε-differentially private function **`laplace_sum(list, epsilon)`** that sums a list a numbers with values that lie in the range -1.0 to +1.0. Explain how this function could be adapted for numbers outside this range. You can assume that there is a function **`laplace(scale)`** that returns a random number from the Laplace distribution located at 0 with scale **`scale`**. Use Python or a similar language (or pseudo-code) to write your function.

d You've performed a sequence of differentially private queries on a national tax database and have one query left that you can perform. Would you ask for the mean or median to understand the income of the average person in the country? Explain your choice.

e Using **`laplace_sum`** from part c, write an ε-differentially private function **`laplace_pearson(X, Y, epsilon)`** that computes the following statistic:

$$\frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$

where **X** is the list $[x_1, .., x_n]$, **Y** is the list $[y_1, .., y_n]$, and n>1.

The five parts carry, respectively, 15%, 15%, 15%, 15%, and 40% of the marks.

2 This is a question about constructing and evaluating a garbled circuit for a function **`f(x,y)`** where **`x`** is Alice's input and **`y`** is Bob's input. Assume inputs **`x`** and **`y`** are **`n`**-bit binary values and the output of **`f`** is an **`m`**-bit binary value. You can also assume **`f`** is a simple boolean function like **`max`**. Use Python or a similar language (or pseudo-code) for your program code.

a Outline an approach to construct a non-garbled circuit for **`f`**. You do not need to carry out your method for **max**.

b Using a very small example outline how you would specify a non-garbled circuit that could be read and parsed by a garbled circuit interpreter. Highlight the main features of your specification. You do not need to provide a grammar for your garbled circuit specifications.

c Describe a program data structure to hold a garbled circuit.

d Give the program code Alice could use to initialise the data structure in part c from the specification in part b. You can assume that the specification is correct and do not need to explain how it is read or parsed. You can also assume suitable encrypt/decrypt functions.

e Outline what message exchanges would Alice and Bob perform in order for Bob to carry out the evaluation of the garbled circuit? Describe clearly what messages would be send between Alice and Bob. You do not need to show program code for this.

f Give the program code Bob could use to evaluate the garbled circuit.

The six parts carry, respectively, 10%, 10%, 15%, 30%, 15%, 20% of the marks.

3 Consider the following table anonymised by a team of "privacy experts":

pseudo_id	gender	age	city	disease
737..d40	male	62	Hamburg	gastritis
8af..b26	male	62	Hamburg	gastritis
ed3..9a3	male	34	Hamburg	skin cancer
52c..df4	male	34	Hamburg	hypertension
271..1ad	female	55	Berlin	gastritis
58e..d10	female	55	Berlin	skin cancer
14c..d09	female	55	Berlin	gastritis
0a4..a62	female	55	Berlin	hypertension

Gender, age and city are quasi-identifiers, while disease is the sensitive attribute. In the following you're asked to assess what they did.

a What is the maximum *k* for which one can claim this table to be *k*-anonymous? What is the maximum *l* for which one can claim this table to be *l*-diverse?

b Suppose that you are allowed to delete one entire column of quasi-identifiers from the table, and this is the only thing you can do. Which column would you delete to obtain a table which is *k*-anonymous, for *k* as high as possible? Does your choice affect *l*-diversity as well and why? Note: you have freedom on the choice of the column, you are not concerned with how this affects utility.

c Suppose you know that the original IDs are National Insurance numbers, which are strings of 6 uppercase letters (no number or symbol). To obtain every **pseudo_id**, they have been hashed using MD5 with a salt of 2 **distinct** digits (from 0 to 9) added **at the end**. The security experts didn't tell you what salt they used. Assume you know that your National Insurance number is LKNMSD and your **pseudo_id** in the table is 52c..df4. Since you are a good computer scientist, you realise that you can easily find what the salt is, using bruteforce

How will you do it, precisely? How much computational time will it take? And, once you found out what the salt is, how would you then de-anonymise the whole table?

(you can assume that computing the md5 hash of any string of length <100 takes 10-6 seconds, and that every other computational task takes a negligible amount of time; for simplicity you can also assume that md5 is collision-free)

Continued on next page

d Now that you have found the salt, the security experts realised that their method was not secure enough. So they have come up with a new “great” idea: every salted ID is hashed **twice** with the MD5 function, that is:

pseudo_id = MD5(MD5(salted_id))

Furthermore, the salt still consists of 2 **distinct digits**, but you don’t know **where** they are added nor if the two digits are **together**. So, for example, your **salted_id** could be LK**1**NMS**8**D, or **2**LKNMSD**5**, or LK**49**NMSD,...

How long will it take to find both the salt and its position under the same computational assumptions as in part c? You know that your ID is still LKNMSD and that now your hashed ID is 9db...596 (not shown in the table).

The four parts carry, respectively, 15%, 15%, 20%, and 50% of the marks.