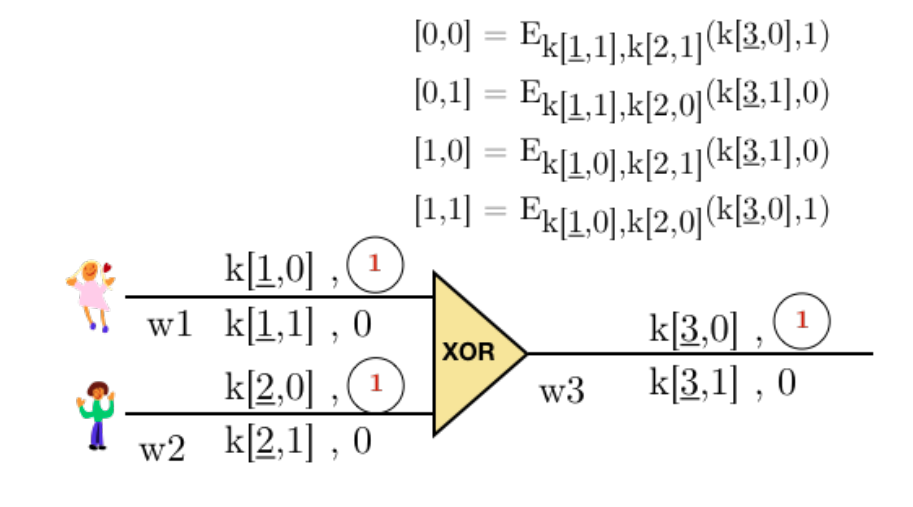
Privacy Engineering 2019-2020

# Question 1

## Part (a)

From Tutorial 2.



## Part (b)

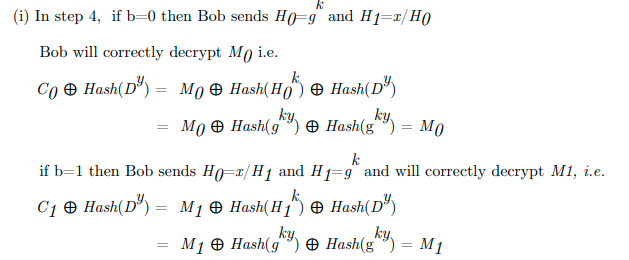
* A will send B the decryption mapping table, the permuted truth table and k[1, 1]
* Bob must do an oblivious transfer with A to get k[2, 0]
* By comparing the keys with the parity bits bob knows to look under [0, 1] in the truth table, bob decodes the value saved there and get (k[3, 1], 0)
* Using the decryption mapping table Bob gets the final output 1

## Part (c)

From Tutorial 2, question 9.

### Part (i)

Follow the definitions remembering that (x xor y) xor y = x



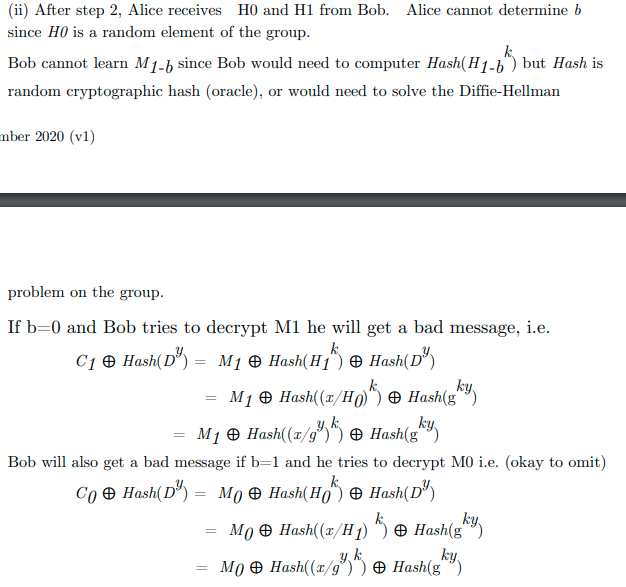
### Part (ii)

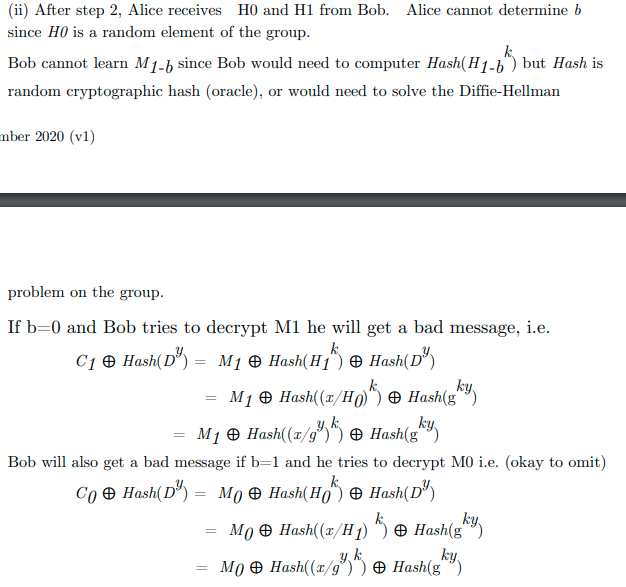
Alice Learns nothing about b:

Bob sends Alice g^y and x / g^y, without knowing y A can’t distinguish which is which and therefore doesn’t know the value of b

Bob doesn’t learn the other message:

For the other message is xor-ed with two different values (assuming the hash function is well chosen). The expression can’t be simplified, and B doesn’t know k so he can’t discover the other message.



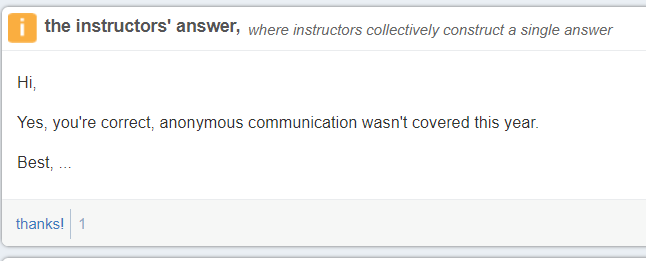


## Part d

* When noise gets too big it can be reset by decryption and re-encryption
* FHE can execute any function, and so it executes the decrypt function by encrypting the key itself, passing it to the FHE, and doing the noise reset by decrypting with the encrypted key
* We don’t want to keep sending keys back and forth
* The H.E method is Turing complete so the untrusted server can perform encryption with an encrypted version of the key

# Question 2

Not assessed anymore.



Pretty sure you could still ask 2c) i), so just in case:

Round 1:  
P1 X1: 1 / ( g^X2 g^X3 )  
P2 X2: g^X1 / g^X3  
P3 X3: (g^X1 g^X2) / 1

Round 2:  
P1: g^X1(-X2-X3)  
P2: g^X2(X1-X3)  
P3: g^X3(X1+X2)

Completion:  
g^X1(-X2-X3) \* g^X2(X1-X3) \* g^X3(X1+X2) = g^A = 1  
A = -X1X2 - X1X3 + X1X2 - X2X3 + X1X3 + X2X3 = 0

# Question 3

## Part (a)

### Part (i)

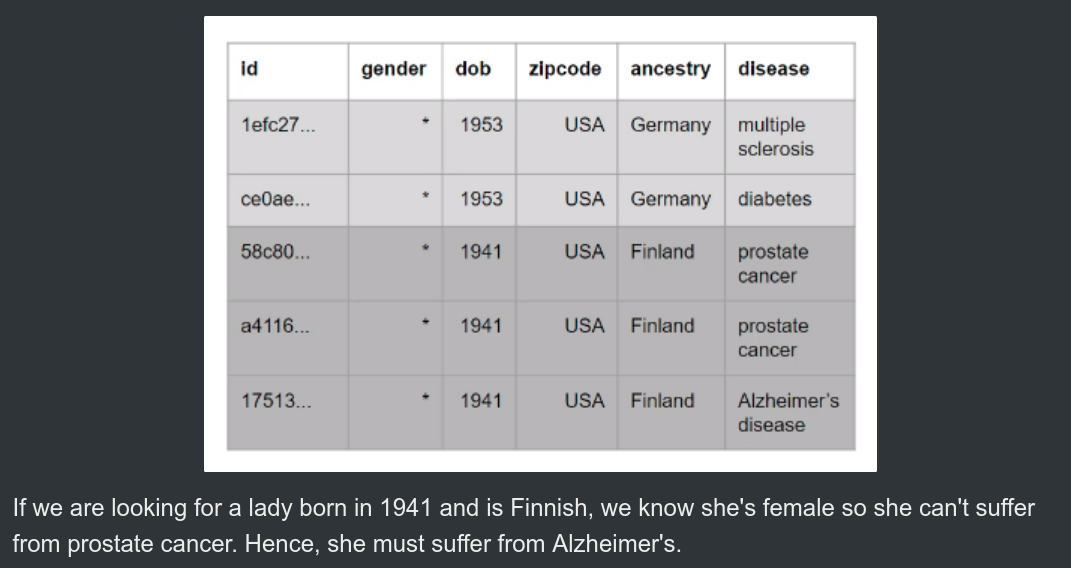
Over the last couple of decades technology has popularized internet shopping. Many storefronts track their users viewed and purchased items; technology has made it possible to mine this data to deliver personalized advertising with the goal of encouraging the customer of making more purchases. This isn’t feasible with conventional in-store shopping where customers can browse unobserved.

### Part (ii)

From a dataset we can form equivalence classes by collecting records together that match on a set of quasi-identifiers. The distribution of values for any sensitive attributes may differ from the global distribution within any equivalence class. T-closeness is a restriction that the distribution is not more different than some threshold t.

L-diversity guarantees that there will be at least l different values for any sensitive attribute in an e-class. However, the distribution of these values could still be significantly different from the global population and give the attacker the opportunity to learn something. For example, an attacker might learn that people living in South Kensington still suffer from a variety of illnesses but at an increased risk of having a particular ailment.

### Part (iii)



B

I)

Yes, a hash table is better because the original id value can be hashed at the point of data capture and then doesn’t need to be stored anymore. Hash functions are also peer reviewed for their safety.

**Alternative:**

No. A hash table is still susceptible to attack since the hashing function is deterministic--based on the original ID. Assuming that the correspondence table is *secure*, it leaks no information about the original IDs since the pseudonymized IDs are randomly generated.

Hash functions may also produce collisions.

Obviously if your correspondence table gets leaked, you’re in a bit of a pickle 🥒.

II)

Maintaining correspondence tables can be cumbersome, particularly to update when new data comes in. There are many well supported libraries available for hashing using well documented hash functions. Lookup on correspondence tables is O(n)--hashing O(1). Hashing: Supports salting & peppering.

III)

There are only a limited number of well documented and supported hash functions, so it is reasonable to disclose it. What should be kept secret is the precise **value of the salt** as well as **the length** and it’s **placement** in the input value. This should make the input space to the hash-function too large to generate a lookup-table

**Alternative:**

Given a sufficiently long salt, you can freely disclose the hash function, length and placement, keeping only the value of the salt secret. This is because disclosing the length and placement only have polynomial increases in the amount of time taken, whereas increasing the length of the salt has an exponential effect on the runtime to produce the lookup table. If the salt is 10,000 characters long, knowing where it’s placed won’t help you.

IV)

An attacker can just attempt hashing MARKLAMBETH0000, MARKLAMBETH0001 etc. until he finds a match in the dataset.

There are no collisions so the attacker will have found Mark

There are 10^4 possible salts and only 1 salt location so they needs to call the hash function 10^4 times

This is 10 seconds

V)

The attacker must now try \*salt\*MARKLAMBETH M\*salt\*ARKLAMBETH etc. for each possible salt. There are 12 positions in Mark’s name, so the attack now takes 2 minutes

VI)

We want to find n where 10^n > 120,000. The smallest such n is 6 so we would only need to add 2 digits

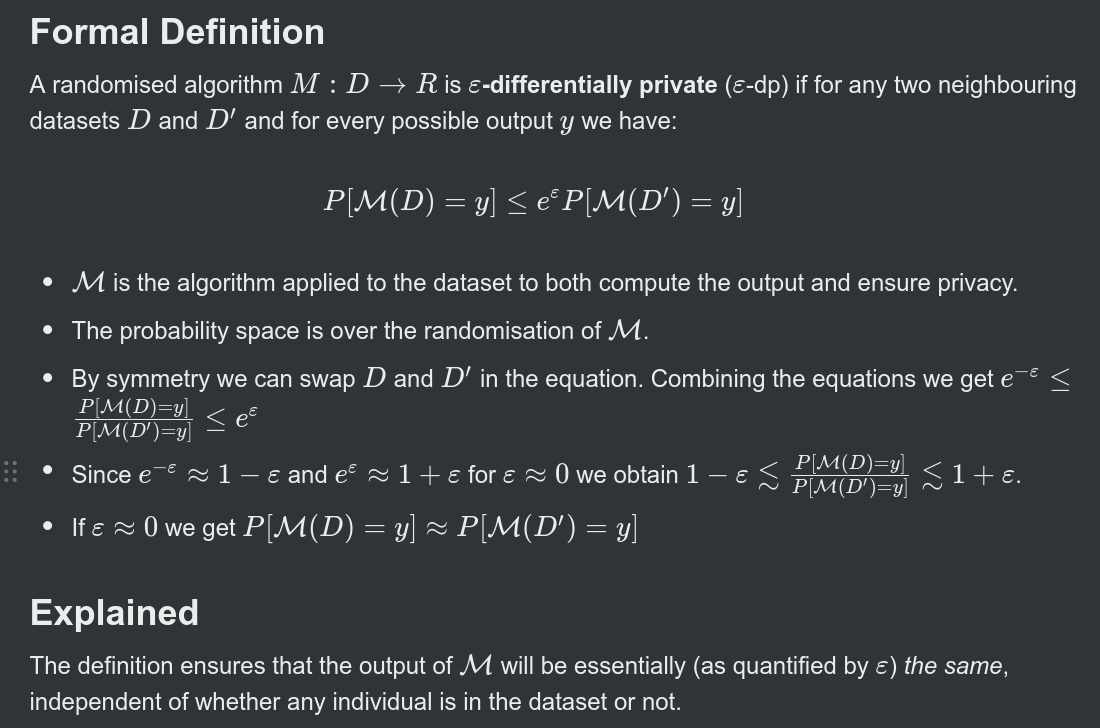
VII)

Given access to (or discovery of) some correct mappings, an attacker could easily find the polynomial using Lagrange interpolation. And degree 2 only needs 3 different points.

Note also that polynomials of degree T > 1 may produce collisions.  
  
(Some other auxiliary stuff: f(.) may produce large values (it is after all unbounded) impractical to be used as pseudonyms)

# Question 4

## Part (a)

z

## Part (b)

### Part (i)

A uniqueness attack is an attack that can be performed when an attacker finds a set information about their target which doesn’t identify them directly but makes them unique in the dataset.

Knowing John is the only person with his birthday the attacker can query count(dob=1-1-1994 AND salary = 1K), count(dob=1-1-1994 AND salary = 2K) etc. Only one of these queries will return with the value 1, all the others will be 0.

### Part(ii)

Averaging attack.

To obtain the true value for a particular query the attacker needs only to ask the query multiple times and average out the value returned. By the CLT the mean value of the noise added will be 0 so the mean average of a set of results will be the true value.

The attacker can ask more queries to improve the likelihood of their final answer being correct.

III)

The attack from (I) won’t work as the results will still be noisy. The attack in (II) no longer defeats the noise addition as asking the same query multiple times will see the same noise added so the average value of the result will be the true value + some random noise.

IV)

Q1 = true\_count[Q1] + noise[salary=20k]

Q2 = true\_count[Q2] + noise[dob!=1-1-1994] + noise[salary=20k]

Q1 – Q2 = true\_count[Q1] – true\_count[Q2] – noise[dob!=1-1-1994]

V)

Q3 = true\_count[Q3] + noise[salary=21k]

Q4 = true\_count[Q4] + noise[dob!=1-1-1994] + noise[salary=21k]

Q3 – Q4 = true\_count[Q3] – true\_count[Q4] - noise[dob!=1-1-1994]

VI)

For each pair of queries, if Bobs salary is the value used the difference between the true counts will be 1, otherwise it will be 0.

If we output the result of every pair of value, the one corresponding to bobs actual salary will be 1 larger than the rest.