Author: Student 217490462

Experiment 02 – Discovery Task – Single Suspect Multiple Parties

Objective

Assess matching and encryption methods on a suspect discovery task where one suspect may be present in multiple parties' data. Report back positive matches in a useful data format which enables automated follow up queries to the parties holding the data.

Method

Using python, create one or more methods to match one suspect against ten lists.

- **Suspect** will be a *single* field containing integers of a set length known as the 'ID' field. This replicates the type of ID number which would be used in an ideal financial intelligence scenario to uniquely identify a given customer.
- **Multiple parties** will be various holders of 'lists' containing records including names, addresses, and an 'ID field'. The 'ID' field will be matched against the suspect ID, the suspect ID should be present in some, though not all lists.

First iteration

The suspect will be matched against the multiple parties lists with no encryption and the results will be returned along with the time required to process the matches. Results will be stored in a useful format which could include a numpy array, dataframe or other format.

Performance is expected to be high, and this first iteration will be the baseline against which encrypted iterations are compared.

Second iteration

The suspect will be matched against the multiple parties lists using the SHA-256 hash algorithm and the results will be returned along with the time required to process the matches. Results will be stored in a useful format which could include a numpy array, dataframe or other format. Hashing provides a low level of protection which is easily defeated using a rainbow table for short integer strings like those proposed in the ID fields.

Performance is expected to be similar to or better than the first iteration, based on the performance measured during SIT378 T1 Experiment 01.

Third iteration

The suspect will be matched against the multiple parties lists using homomorphic encryption (HE). HE should always provide high protection to the data, even during the match operation. The results will be returned along with the time required to process the matches.

Performance is expected to be significantly lower than both the first and second iterations due to the increased length of time required to complete match operations using homomorphic encryption. The total number of operations is expected to be the main predictor of performance. i.e. 10,000 total operations (10 suspects, 1000 customers in match experiment 01) is expected to have similar performance to experiment 02 (1 suspect, 10 lists of 1000 customers).

Next steps

Additional iterations of the experiment can be considered depending on the results of the experiment. Permutating the number of suspects, lists and customers may be useful.

Author: Student 217490462

Experiment 02

Parameters

1 suspect ID matched against 10 lists of 1000 customer IDs.

Outcome

?

Results

First iteration – cleartext matching

Expected outcome: Performance expected to be high.

Actual outcome: ?

Second iteration – Hashed matching

Expected outcome: Performance was expected to be similar to cleartext matching.

Actual outcome: ?

Third iteration – HE matching

Expected outcome: Performance expected to be significantly lower than cleartext or hashed matching. A linear decrease in performance as number of operations increases.

Actual outcome: ?

Evaluation of results

?

Attachment A: experiment 02 results from python in a Linux environment.

Attachment B: the python code to create synthetic match data containing ID numbers

```
#import necessary libraries

from faker import Faker

import pandas as pd

import numpy as np

import time

fake = Faker()
```

Author: Student 217490462

```
#create a function for creating realistic customer data, including name, DOB, ID etc.

def create_data(x):

        person ={}

        for i in range(0,x):

                person[i] = {}

                person[i]['Name'] = fake.name()

                person[i]['DOB'] = fake.date_of_birth(minimum_age=16,maximum_age=120)

                person[i]['ID'] = np.random.choice(np.arange(1000000,9999999),replace=False)
#this will avoid duplicate ID numbers but runs slowly, test on your system with a small value before
generating large datasets.

                #person[i]['ID'] = np.random.randint(1000000,9999999) #this will include duplicate
ID numbers, but runs very quickly. It should be used unless duplicates are undesireable.

                person[i]['Address'] = fake.address()

                person[i]['Country'] = fake.country()

        return  person

tic = time.perf_counter() #starts a timer

df = pd.DataFrame(create_data(1000)).transpose() #runs def create_data(number of rows required
in output goes here)

toc = time.perf_counter() #ends the timer

print(f"Synthetic data creation time was: {toc - tic:0.6f} seconds") #returns text and the amount of
time the process required

print(df.head()) #prints first and last entries of the dataframe


df.to_csv("suspects.csv") #outputs the dataframe to a csv file

#df.to_csv("customers.csv") #outputs the dataframe to a csv file
```

Attachment C: python code to run a match protocol using cleartext

Attachment D: python code to run a match protocol using hashing

Attachment E: python code to run a match protocol using homomorphic encryption