Author: Student 217490462

Experiment 01 – Matching Task – Suspects in Customer List

Objective

Assess the performance impact of different encryption methods on a standard list matching task in a financial intelligence use case.

Method

Using python, create a method to match two lists (A and B) to identify and return matches based on a single field.

- **List A** will be a 'suspect list' containing a *small number* of records including one field of unique integers of a set length known as the 'ID field'. This field is intended to replicate the type of ID number which would be used in an ideal financial intelligence scenario to uniquely identify a given customer.
- **List B** will be a 'customer list' containing a *large number* of records including transaction values, names and an 'ID field'. The List A ID field will be matched against the List B ID field.

First iteration

List A (ID field) will be matched against List B (ID field) with no encryption and the results will be returned along with the time required to process the matches.

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

Second iteration

The ID fields in both lists A and B will be hashed using a common one-way SHA-256 hash algorithm. 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.

List A (hashed ID field) will be matched against List B (hashed ID field) and the results will be returned along with the time required to process the matches.

Performance is expected to be lower than the first iteration due to the increased length of the hashed values versus the unencrypted values in the first iteration.

Third iteration

The ID fields in both lists A and B will be encrypted using homomorphic encryption (HE). HE should always provide high protection to the data, even during the match operation.

List A (homomorphically encrypted ID field) will be matched against List B (homomorphically encrypted ID field) and the results will be returned along with the time required to process the matches.

Performance is expected to be much lower than both the first and second iterations due to the increased length of time required to complete match operations while using homomorphic encryption.

Next steps

Additional iterations of the experiment can be considered depending on the results of the first three iterations. The performance impact of HE on matching will inform our privacy product design.

Author: Student 217490462

Experiment 01

Parameters

Two scales of matching operations were used (11000 and 100,000). Respectively; 11 suspect IDs matched against 1000 customer IDs, and 100 suspect IDs matched against 1,000 customer IDs.

Outcome

The experiment was successful.

The team now has quantitative data on the performance impact of different encryption methods on a list matching task in a financial intelligence use case.

Results

First iteration – cleartext matching

Expected outcome: Performance expected to be high.

Actual outcome: Performance was high. Average seconds per operation was 0.000002199.

Second iteration – Hashed matching

Expected outcome: Performance was expected to be relatively high, though lower than the first iteration due to the increased length of the hashed values versus the unencrypted values in the first iteration.

Actual outcome: Performance was higher than expected and slightly better than the first iteration using cleartext. Average seconds per operation was 0.000002005.

Third iteration – HE matching

Expected outcome: Performance was expected to be much lower than both the first and second iterations due to the increased length of time required to complete match operations while using homomorphic encryption.

Actual outcome: Performance was orders of magnitude slower than both the first and second iterations. Average seconds per operation was 0.040338516. More than 20,000 times slower.

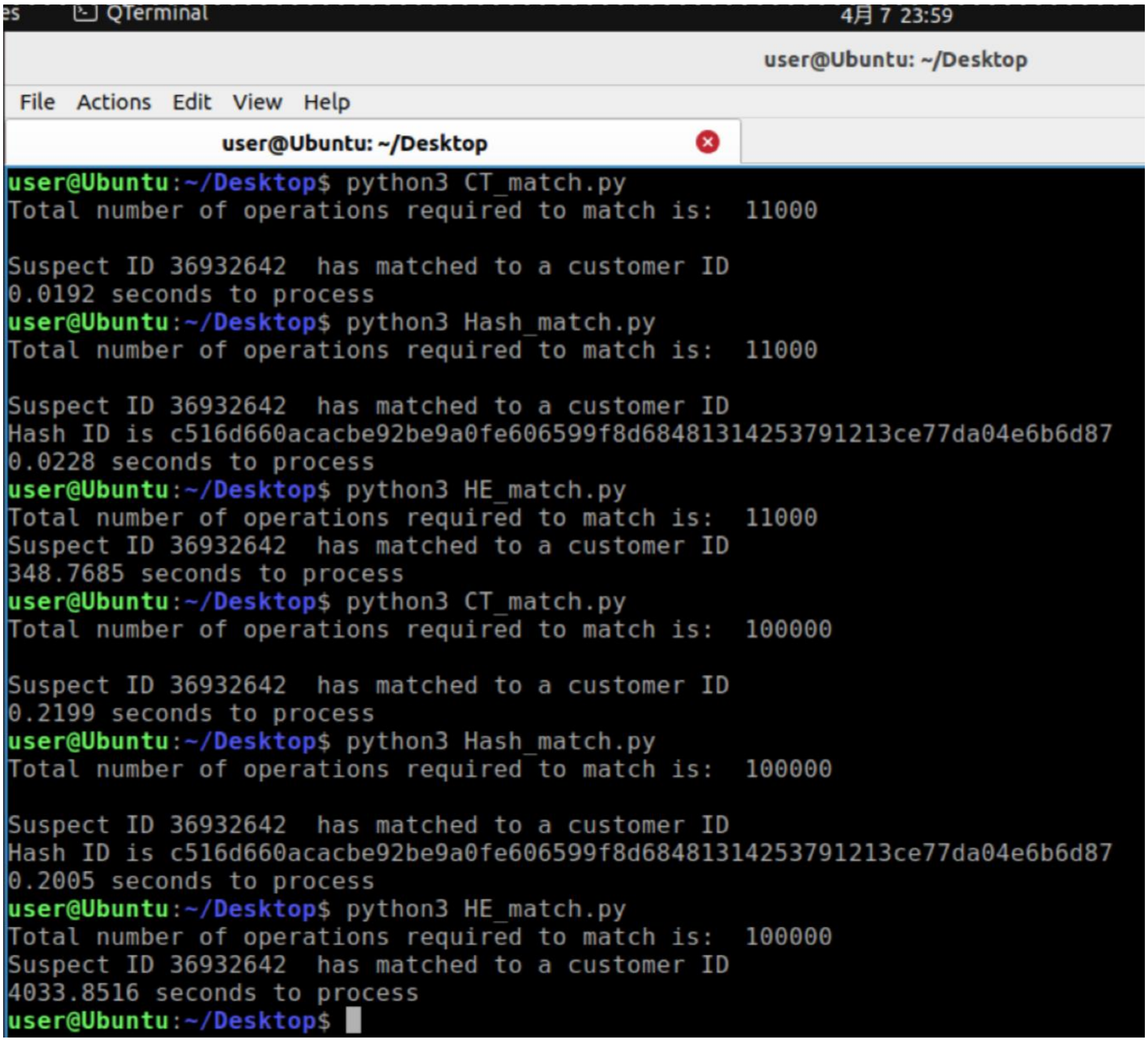
Evaluation of results

Cleartext and hashed matching are much more performant than homomorphically encrypted (HE) matching. Increasing the number of operations is trivial and both methods could be used for millions of match operations without specific investments in time or resources.

HE matching is expensive and attempting to scale up to millions of match operations would require investment in time (waiting for results) or resources (running the operations on better hardware). For a financial intelligence use case, additional investment in hardware is feasible, additional time costs are likely to be undesirable.

Author: Student 217490462

Attachment A: experiment 01 results from python in a Linux environment.



```
user@Ubuntu:~/Desktop$ python3 CT_match.py
Total number of operations required to match is:  11000

Suspect ID 36932642  has matched to a customer ID
0.0192 seconds to process
user@Ubuntu:~/Desktop$ python3 Hash_match.py
Total number of operations required to match is:  11000

Suspect ID 36932642  has matched to a customer ID
Hash ID is c516d660acacbe92be9a0fe606599f8d68481314253791213ce77da04e6b6d87
0.0228 seconds to process
user@Ubuntu:~/Desktop$ python3 HE_match.py
Total number of operations required to match is:  11000
Suspect ID 36932642  has matched to a customer ID
348.7685 seconds to process
user@Ubuntu:~/Desktop$ python3 CT_match.py
Total number of operations required to match is:  100000

Suspect ID 36932642  has matched to a customer ID
0.2199 seconds to process
user@Ubuntu:~/Desktop$ python3 Hash_match.py
Total number of operations required to match is:  100000

Suspect ID 36932642  has matched to a customer ID
Hash ID is c516d660acacbe92be9a0fe606599f8d68481314253791213ce77da04e6b6d87
0.2005 seconds to process
user@Ubuntu:~/Desktop$ python3 HE_match.py
Total number of operations required to match is:  100000
Suspect ID 36932642  has matched to a customer ID
4033.8516 seconds to process
user@Ubuntu:~/Desktop$ 
```

Screenshot showing the three experiment 01 iterations (cleartext, hashing and HE) across two scales of matching operations (11000 and 100,000). Average seconds per operation was calculated from the 100,000 match operation i.e. 100 suspect IDs matched against 1,000 customer IDs.

Author: Student 217490462

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()
#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(100)).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
```

Author: Student 217490462

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

```
import time

import numpy as np

import pandas as pd


df1 = pd.read_csv('suspects1.csv')

df2 = pd.read_csv('customers1M.csv')


suspects = df1['ID'].to_list()

customers = df2['ID'].to_list()


print("Total number of operations required to match is: ",(len(suspects))* len(customers))


def match(sus,cus):

        int1 = np.array([sus], dtype=np.int64)

        int2 = np.array([cus], dtype=np.int64)


        resSub = int1 - int2


        if (resSub[0]) == 0:

                print("\nSuspect ID", sus ," has matched to a customer ID")


tic = time.perf_counter()


for s in range(0, (len(suspects))):

        for c in range(0, (len(customers))):

                match(suspects[s],customers[c])


toc = time.perf_counter()

print(f"{toc - tic:0.4f} seconds to process")
```

Author: Student 217490462

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

```
import time

import numpy as np

import pandas as pd

import hashlib

df1 = pd.read_csv('suspects1.csv')

df2 = pd.read_csv('customers1M.csv')

suspects = df1['ID'].to_list()

customers = df2['ID'].to_list()

print("Total number of operations required to match is: ",(len(suspects))* len(customers))

def match(sus,cus):

        sus = str(sus)

        cus = str(cus)

        ec1 = sus.encode()

        ec2 = cus.encode()

        hash1 = hashlib.sha256(ec1).hexdigest()

        hash2 = hashlib.sha256(ec2).hexdigest()

        if hash1 == hash2:

                print("\nSuspect ID", sus ," has matched to a customer ID")

                print("Hash ID is", hash1)

tic = time.perf_counter()

for s in range(0, (len(suspects))):

        for c in range(0, (len(customers))):

                match(suspects[s],customers[c])

toc = time.perf_counter()

print(f"{toc - tic:0.4f} seconds to process")
```

Author: Student 217490462

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

```
import time

import numpy as np

import pandas as pd

from Pyfhel import Pyfhel

HE = Pyfhel()

HE.contextGen(scheme='bfv', n=2**14, t_bits=20)

HE.keyGen()

df1 = pd.read_csv('suspects1.csv')

df2 = pd.read_csv('customers1M.csv')

suspects = df1['ID'].to_list()

customers = df2['ID'].to_list()

print("Total number of operations required to match is: ",(len(suspects))* len(customers))

def match(sus,cus):

	int1 = np.array([sus], dtype=np.int64)

	int2 = np.array([cus], dtype=np.int64)

	ctxt1 = HE.encryptInt(int1)

	ctxt2 = HE.encryptInt(int2)

	ctxtSub = ctxt1 - ctxt2

	resSub = HE.decryptInt(ctxtSub)

	if (resSub[0]) == 0:

		print("Suspect ID", sus ," has matched to a customer ID")

tic = time.perf_counter()

for s in range(0, (len(suspects))):

	for c in range(0, (len(customers))):

		match(suspects[s],customers[c])

toc = time.perf_counter()

print(f"{toc - tic:0.4f} seconds to process")
```