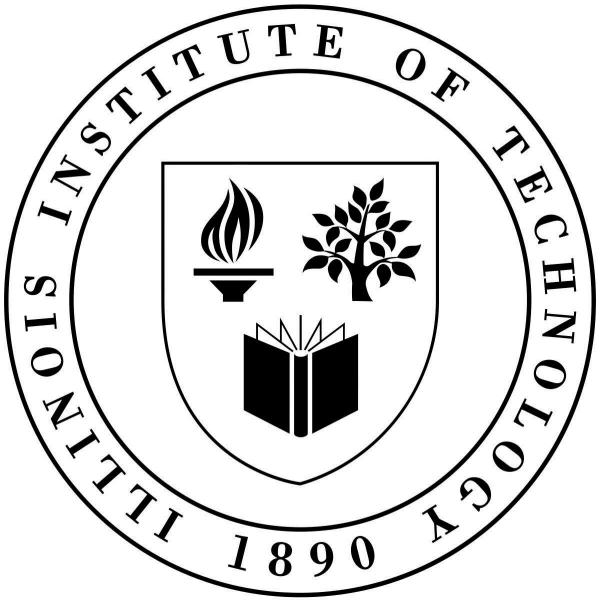

DEPARTMENT OF COMPUTER SCIENCE

Data Privacy and Security - CS 528

Project Report:

Secure Multiparty Computation On Market Basket Analysis

By:

Rakesh Abbireddy

Rishith Reddy Odeti

Yagna Veeresh Pamulapati

Contents

1 Team Description

Our project team consists of three people: Rakesh, Rishith, and Yagna Veeresh. Rakesh is an aspiring and enthusiastic individual with industry experience. In the future, he will work as a Network Architecture research intern at Bell Labs. He contributed the majority of the work to this project. Rishi is interested in security and is willing to put in the effort to gain knowledge and skills for his future career in cybersecurity. Yagna is an intermediate learner in the field of security and believes that cybersecurity can enhance our lives. Rakesh provided technical assistance and implemented the project, while Yagna and Rishi contributed by finding better techniques and formatting the report

2 Application

In this project, we implemented a secure multiparty computation model where parties securely compute on dictionary items to determine the most frequently bought item by customers across different party datasets. We developed an application that implements a protocol for communicating with different parties in a pool. The parties are convenience, department, and pharmacy stores. The model assumes semi-honest parties, meaning that each party may try to learn more information from the outputs than intended. We utilized argument parsers to input the number of parties and assign party numbers for running the protocol. Through this implementation, we successfully performed secure computations using homomorphic encryption and obtained the output secure.

3 Datasets

For the dataset, we used three different datasets from retail transaction data. Each dataset contains data from different store types: convenience stores, department stores, and pharmacies. Each dataset has over 4,900 entries. We also used similar datasets for testing purposes. This is the GitHub link for datasets Stores dataset[1].

4 Privacy and Security Technique

In this project, we developed a secure and privacy-preserving solution for multiparty computation using advanced cryptographic techniques. We leveraged the Paillier homomorphic encryption scheme to enable computations on encrypted data without decryption and employed Shamir's secret sharing for the secure distribution of cryptographic keys among parties. The TNO-MPC library[2] was utilized to facilitate the implementation of the secure multiparty computation protocol.

To enhance the system's resilience and security, we incorporated Threshold Cryptography techniques, ensuring that the system remains secure even if a subset of parties is compromised. The combination of Paillier homomorphic encryption, Shamir's secret sharing, and Threshold Cryptography creates a powerful framework for multiparty computation.

Our project demonstrates the feasibility and effectiveness of these techniques in enabling collaborative computations of sensitive data while preserving privacy. The implemented solution has potential applications in various domains requiring joint computation without revealing private inputs, contributing to the field of secure computation and paving the way for further research and practical applications in privacy-preserving collaborative anal

5 Code Implementation

In this project, we used libraries that helped us to achieve the goal of doing multiparty computation in the semi-host model. TNO-MPC libraries helped us in implementing the distributed key generation and secure communications and computations between parties. As shown in the figure 1.

```
import argparse
import asyncio
from typing import List, Tuple
from tno.mpc.communication import Pool
from tno.mpc.protocols.distributed_keygen import DistributedPaillier
import pandas as pd
import numpy as np
from collections import Counter
```

Figure 1: Libraries

We can configure different types of security parameters like corruption threshold, key length, prime number threshold, and Shamir secret sharing statistical parameters.

```
corruption_threshold = 1
key_length = 128
prime_thresh = 2000
correct_param_biprime = 40
stat_sec_shamir = (
    40
)
```

Figure 2: security parameter

The setup_local_pool function will create a pool for all participating members for secure multiparty computations.

```
def setup_local_pool(server_port: int, others: List[Tuple[str, int]]) -> Pool:
    pool = Pool()
    pool.add_http_server(server_port)
    for client_ip, client_port in others:
        pool.add_http_client(
            name: f"client_{client_port}", client_ip, client_port
        )
    return pool
```

Figure 3: pooling members

Each party will have its dataset we transformed the data for our use case where all items have several counts the customer purchased frequently. We encrypted each item count using the Pallier encryption scheme.

```
store_data = pd.read_csv(r'C:\Users\rakes\PycharmProjects\DPS-project\convenience_store.csv')

store_data['Product'] = store_data['Product'].apply(eval)
store_data = store_data['Product']
consolidated_list = []
for product_list in store_data:

    for product in product_list:
        consolidated_list.append(product.strip())

consolidated_list = np.array(consolidated_list)
product_counts = Counter(consolidated_list)
unique_products = dict(product_counts)
for key, value in unique_products.items():
    unique_products[key] = distributed_paillier.encrypt(value)
```

Figure 4: data transformation

The convenience store party will send the homomorphically encrypted dictionary to the department store party as it is shown in Figure 5. Then the department store party will await the dictionary and compute its encrypted items with the convenience store encrypted items as shown in Figure 6. The total encrypted sum new dictionary will be sent to the pharmacy store party and it awaits for it. Now the pharmacy store will do the same process as the department store party Figure 7. In total, it is a secure multiparty computation protocol.

```
await distributed_paillier.pool.send( handler_name: "client_8889", unique_products,
                                     msg_id="step1")
```

Figure 5: Communication of encrypted items

```
unique_products = await distributed_paillier.pool.recv( handler_name: "client_8888",
                                                        msg_id="step1")
for key, value in current_unique_products.items():
    if key in unique_products:
        unique_products[key] += value
    else:
        unique_products[key] = value
```

Figure 6: computation Party 2 on Party1 data

```
unique_products = await distributed_paillier.pool.recv( handler_name: "client_8889",
                                                        msg_id="step2")

for key, value in current_unique_products.items():
    if key in unique_products:
        unique_products[key] += value
    else:
        unique_products[key] = value
```

Figure 7: computation Party 3 on Party2 data

After the dictionary item count values computation, all parties need to participate in the decryption process for the final output. The final product items will be broadcast to all participants in the protocol as shown in Figure 8. Meanwhile, the decryption process will be jointly completed, and then stored in the new dictionary. Other parties will receive the final encrypted dictionaries as shown in Figure 9 to decrypt locally. Then finally the maximum value item is computed and returns the item with the highest count. Here we can find the source code link *https://github.com/rakeshreddy06/multichain* [1].

```
await distributed_paillier.pool.broadcast(unique_products, msg_id="step3",
                                          handler_names=["client_8888",
                                                         "client_8889"])

decrypted_values = {}
for key, encrypted_value in unique_products.items():
    decrypted_value = await distributed_paillier.decrypt(encrypted_value)
    decrypted_values[key] = decrypted_value
max_key = max(decrypted_values, key=decrypted_values.get)
print(max_key)
```

Figure 8: broadcasting to other parties

```
final_ciphertext = await distributed_paillier.pool.recv( handler_name: "client_8890",
                                                         msg_id="step3")

decrypted_values = {}
for key, encrypted_value in final_ciphertext.items():
    decrypted_value = await distributed_paillier.decrypt(encrypted_value)
    decrypted_values[key] = decrypted_value

max_key = max(decrypted_values, key=decrypted_values.get)
print(max_key)
```

Figure 9: decryption protocol

6 Communications

In this SMPC protocol, we set up a local pool on my machine. This communication module uses asynchronous functions for sending and receiving messages. The pool contains a server, which listens for incoming data from other parties in the network, and clients for each party in the network. HTTP clients can be identified by an IP address.

In the figure below, we configure the server port and add HTTP clients to the server. The servers are

configured using the command line arguments, which are shown in Figure 11.

```python
def setup_local_pool(server_port: int, others: List[Tuple[str, int]]) -> Pool:
    pool = Pool()
    pool.add_http_server(server_port)
    for client_ip, client_port in others:
        pool.add_http_client(
            name: f"client_{client_port}", client_ip, client_port
        )
    return pool
```

Figure 10: Local pool participants

```python
parser.add_argument(
    *name_or_flags: "--party",
    type=int,
    help="Identifier for this party. This should be different for all scripts but should be in the "
        "set [0, ..., nr_of_parties - 1].",
)

parser.add_argument(
    *name_or_flags: "--nr_of_parties",
    type=int,
    help="Total number of parties involved. This should be the same for all scripts.",
)
```

Figure 11: Arguments Parser

7 Computational Analysis

It is found that in this implementation of a secure multiparty Paillier cryptosystem, the relationship between key length and execution time is not strictly linear. The results show that the elapsed time for key lengths of 56, 128, and 256 bits is relatively small, with the execution time ranging from

approximately 2.9 to 4.8 seconds. However, there is a significant increase in execution time when the key length is increased to 512 and 1024 bits, with the elapsed time jumping to 31.4 and 62.3 seconds, respectively.

The observed trend suggests that the execution time grows exponentially as the key length increases beyond a certain threshold. This behavior can be attributed to the increased computational complexity associated with larger key sizes in the Paillier cryptosystem. As the key length doubles from 256 to 512 bits, the execution time increases by a factor of more than 10. Similarly, doubling the key length from 512 to 1024 bits results in a nearly two-fold increase in execution time. These findings highlight the trade-off between security and computational efficiency in the implementation of a secure multiparty Paillier cryptosystem, where longer key lengths provide enhanced security but come at the cost of increased execution time.

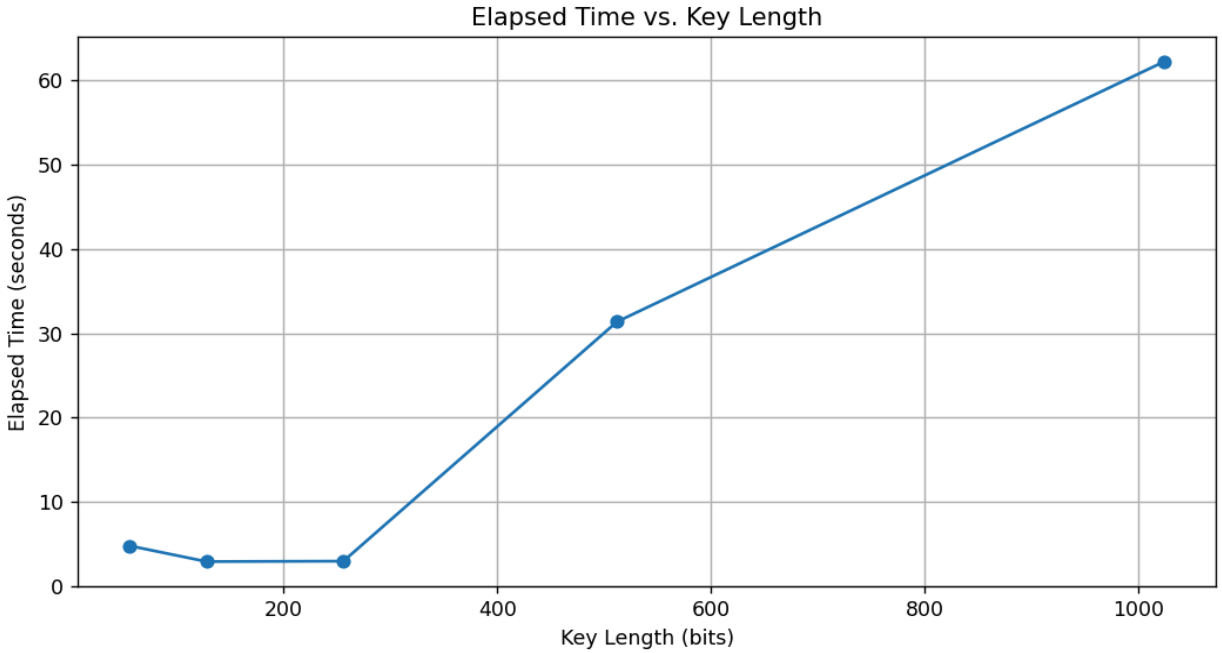

Figure 12: key size vs elapsed time

8 Results

The result we received from the SMPC protocol is that toothpaste is the most frequently bought item among all items found in convenience stores, department stores, and pharmacy stores. This finding is based on the collective data from multiple retailers, providing a comprehensive view of consumer purchasing habits across different store types. As we can see, the protocol is receiving messages from the other two parties and computing the result. These results are computed by all parties, and everyone can view the output, which can provide valuable insights without exposing each party's input data to the others. The SMPC protocol ensures that the privacy of each party's data is maintained throughout the computation process. By leveraging this secure multi-party computation approach, businesses can collaborate and gain a deeper understanding of market trends and customer preferences while preserving the confidentiality of their datasets. This innovative technology opens up new possibilities for data-driven decision-making and strategic planning in various industries.

As you can see node 8890 is communicating protocol with 8888 and 8889 in Figure 13.

```
2024-05-03 00:52:47,471 - tno.mpc.communication.httphandlers - INFO - Received message from 127.0.0.1:8888
2024-05-03 00:52:47,472 - tno.mpc.communication.httphandlers - INFO - Received message from 127.0.0.1:8889
2024-05-03 00:52:47,477 - tno.mpc.communication.httphandlers - INFO - Received message from 127.0.0.1:8889
2024-05-03 00:52:47,478 - tno.mpc.communication.httphandlers - INFO - Received message from 127.0.0.1:8888
The most frequently bought item is :  Toothpaste
```

Figure 13: Final result

9 Summary

In this project, the team implemented a secure multiparty computation (SMPC) model to determine the most frequently bought item by customers across different retailer datasets, including convenience stores, department stores, and pharmacies. The model assumes semi-honest parties and utilizes advanced cryptographic techniques such as Paillier homomorphic encryption, Shamir's secret sharing, and Threshold Cryptography to ensure secure computations and resilience against compromised parties. The TNO-MPC library was used to facilitate the implementation of the SMPC protocol.

The project involved setting up a local pool for participating members, transforming datasets, and encrypting item counts using the Paillier encryption scheme. The convenience store party initiated the process by sending its encrypted dictionary to the department store party, which computed its encrypted items with the received data. The process continued with the pharmacy store party, resulting in a secure multiparty computation protocol. Finally, all parties participated in the decryption process to obtain the final output, revealing that toothpaste was the most frequently bought item among all participating retailers. The project demonstrates the feasibility and effectiveness of SMPC techniques in enabling collaborative computations while preserving data privacy.

References

[1] *https://github.com/rakeshreddy06/multichain*

[2] *https://github.com/TNO*