Assignment 2: Secure Multiparty Computation

Adrian Walczak

December 2023

1 Introduction

In the paper we're going to explore concepts of Secure Multiparty Computation (SMC). To do so, we are going to define a problem that can be solved with PSI protocols. Then we will test execution time and data sent using:

- the naive hashing protocol
- the Diffie-Hellman-based PSI protocol of [4]
- the OT-based PSI protocol of [1]

To do so, we are going to use PSI protocols tool [2] and Wireshark.

2 Data

2.1 Datasets

We are going to consider scenerio with 2 datasets where the first one has data about users that clicked on an add and the second dataset has informations about products ordered in online shop. The identityfier is an e-mail address. All the data were generated. The emails were generated using script provided by PSI tool and the other data with my code which you can find in file 'generate_data.py'.

Table 1: clicked.csv

e-mail	addID	date,time	timeLookinForAdd
Manfred.Kuhlmann@uwr.edu.pl	8	2023-09-22 18:50:38	5.93294

Table 2: bought.cvs

e-mail	date,time	price	numberOfProducts
Heinz-Werner.Bauer@yahoo.com	2023-01-27 02:35:29	43.57	8

2.2 Motivation

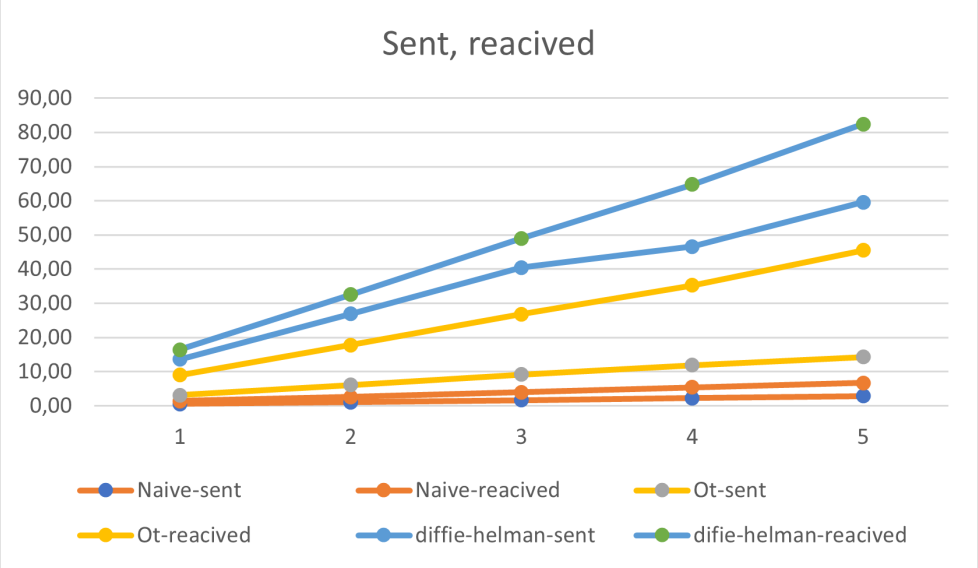
We can use this data to explore the results of the advertising campaign, checking which ads were the most effective and which ones captured the most attention.

2.3 Creating subsets

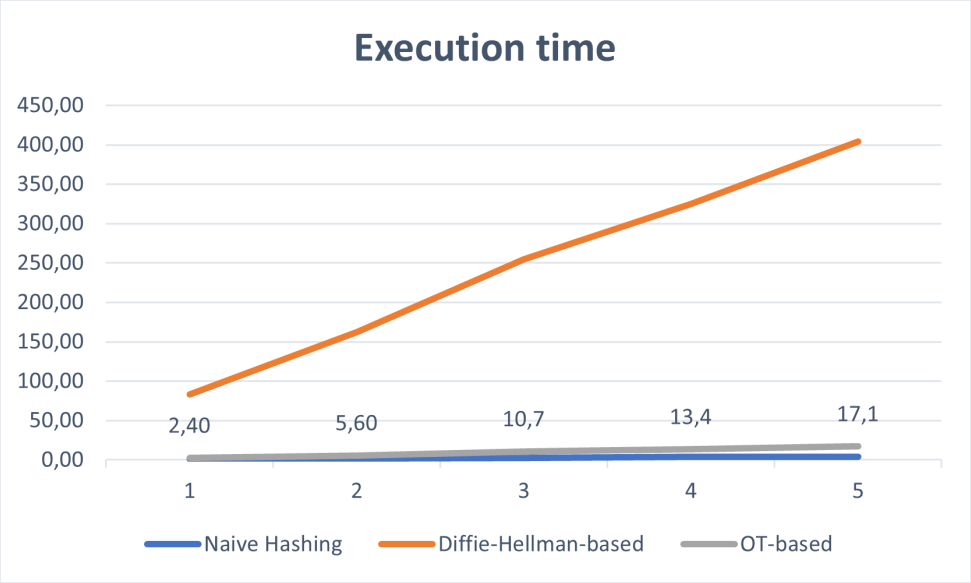
We are going to create 5 subsets having: 1: 20,2: 40,3: 60,4: 80,5: 100% of the full data size which is 200 000 rows in clicked.csv and 100 000 in ordered.csv. The subsets include rows number 1,2..x of original dataset where x is subset size. The python script for generating these subsets is located in the file 'split_data.py'."'

3 Benchmarks

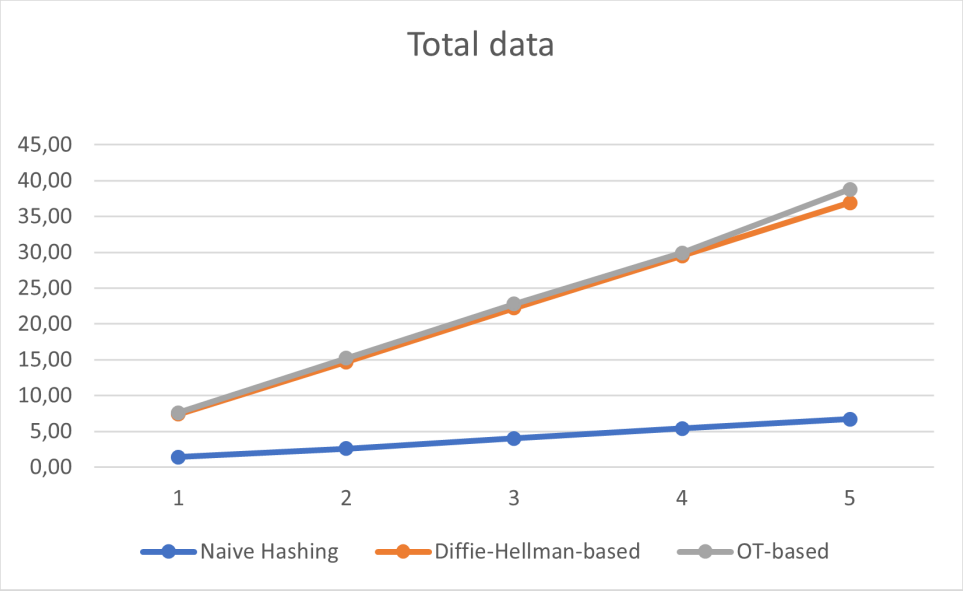
3.1 Data sent and reacived



3.2 Execution time



3.3 Total data



3.4 Observations

Note: In the observations below, I will use "A" to represent party 0 and "B" to represent party 2.

- Naive Hashing Results: The Naive hashing protocol emerges as the fastest. In this protocol, parties agree on a hash function and use it to hash elements. Subsequently, B sends the hashes to A, and A compares them. However, a challenge arises when B lacks entropy. For instance, A may discover that it possesses all elements of B, contradicting the concept of PSI protocols (providing no extra information apart from the intersection).

- in OT-based protocol : Let's try to understand he difference between data sent and reacived. In step when A is checking if element belongs to set of B, B has to send back n masks, where n is number of B elements. [3] It ends up with complexity O(n*m) where m is number elements of A. However after applying optimalization with hashing elements to buckets and just comparing elements that are in the same bucket. After the optimalization we have O(nlogn). n=max(n,m)

- In the OT-based protocol: Let's try to understanding the difference between the data sent and received. During the step where A checks if an element belongs to the set of B, B must send back n masks, where n is the number of elements in B set [3]. This results in a complexity of O(n*m), where m is the number of elements in A. However there is an optimization involves hashing elements into buckets and only comparing elements within the same bucket. Post-optimization, we achieve O(n log n), where n equals max(n, m). The optimalization decrease number of masks that we have to sent in single "Private set Inclusion" step, but still B needs to send them what results in the difference.

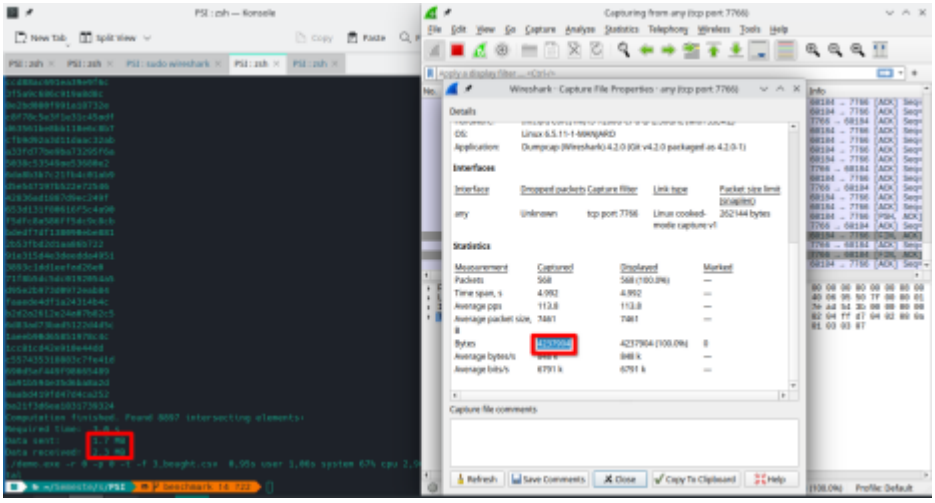
- Diffie-hellman protocol: The execution time is very long comparing to others. I discovered that this delay is mostly a consequence of hashing. One implementation I came across involved hashing all elements until they became primitive roots modulo p. The algorithm also has to deal with large numbers. However in in small datasets (around 1000 elements) is very effective.

3.5 Wireshark

The main focus of the assignment was to observe three pieces of information: data sent, data received, and execution time for each protocol with different sizes. All of these can be obtained directly from the PSI tool. However, we can also experiment with Wireshark to delve more deeply into information such as the types of requests sent and the total number of packages.
To get more information:

- Install wireshark

- Type 'sudo wireshark' in terminal - the app's window should appear
- Choose what kind of requests you want to see (any to see all of them
- In the "...using this filter" text area type: 'tcp port 7766' and click enter Note: By default the PSI tools works on localhost with port 7766. To change it you can use flag -a or manually change in the .cpp files that are in directory PSI/src/mains
- Run the PSI tool (tons of requests appeared, right?)
- After executing the protocol you can also check Capture-¿Check File Properties



References

[1] `http://eprint.iacr.org/2016/930`. 2016.

[2] `https://github.com/rscmendes/PSI/tree/benchmark`.

[3] `https://www.youtube.com/watch?v=6Wo-MFd4Uuw&t=510s&ab_channel=USENIX`.

[4] C. Meadows. *A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party.* C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In IEEE S&P'86, pages 134–137. IEEE, 1986. 1986.