Study about Secure Multiparty Computation

Privacy and Security - Assignment 2

Departamento de Engenharia Informática
Universidade de Coimbra
José Donato
donato@student.dei.uc.pt

***Index Terms*—Secure Multiparty Computation, Blockchain, Enigma, Private Set Intersection, OT, Naive-Hashing, Diffie-Hellman, Server-aided, COVID-19.**

I. INTRODUCTION

In a world where more and more applications are becoming data-driven, i.e., our input data dictates how the app works, needs for security and privacy emerge exponentially. Lot of applications related to big data, cloud computing, machine learning, artificial intelligence, etc, need huge amount of data to perform well, data that in the majority of the times is highly sensitive. Therefore, we must seek technologies that are privacy-preserving. Secure Multiparty Computation (SMPC or SMC) is a possible solution to this problem. However, as in many other scenarios, security and privacy has some costs (mainly performance-related). In this paper I will do a quick study about Secure Multiparty Computation. The paper is divided into two main parts: a first section II more theoretical where I talk about the basics of Secure Multiparty Computation II-A, its problems II-B, and how Blockchain is a possible solution to those II-C. As we will see, due to performance and security issues, blockchain is not the perfect solution. Trying to find a suitable solution, in Subsection II-D, a blockchain-based platform named Enigma is presented. To conclude this section, in II-E several applications of SMPC are listed.

In a second part, in Section III I present a more practical section where I study several different Private Set Intersection protocols that use Secure Multiparty Computation. I also perform some benchmarking tests between them and present real world-case scenario where Private Set Intersection can be useful in sections III-D and III-E, respectively.

Finally, in Section IV I conclude the paper.

II. SECURE MULTIPARTY COMPUTATION WITH BLOCKCHAIN

A. Introduction - What is SMPC?

Secure Multiparty Computation (SMPC) is as simple as do some kind of computation without revealing any users' input or output. The main requirement of SMPC is that private information must remain secret between the parties involved in the computation, i.e., each party must only have access to its input and never to others parties input (unless the other party gives them permission).

SMPC born with the problem of two millionaires proposed by [1]. In this problem, two millionaires were trying to discover which one was richer without disclosing their wealth to each other.

A more simple example to illustrate how SMPC can work is the following one [2]:

- imagine a simple scenario where there are three different employees (three different parties) trying to compute the average salary between the three but without revealing their own salary. An naive but effective approach would be to each party divide its salary into three different numbers (eg. for a 1000$ salary it could be divided into 500, 700, -200) and sends one for each party (including himself). After the three parties do the same process, each party ends up with three different numbers meaningless by themselves alone (important that they need to seem random for each party). After each party sums their three numbers and join with the other parties sum (of course, dividing by three, the number of parties) we get the average without disclosing any salary.

B. Problems in SMPC

In SMPC we have five main requirements:

1) privacy - inputs secret to others all the time
2) correctness - final result is correct when everyone is honest
3) fairness - corrupted party cannot get its output and deny other parties to get their outputs
4) process should not be interrupted
5) corrupted adversaries (someone who tries to break the protocol) should be detected

The problems against this protocol are any disruption of these requirements. However, keeping fairness due to the fact that participants may be dishonest is the most important among those five. In addition, as we are gonna see in the second part of this paper in Section III, as in many other scenarios, security and privacy are achieved at the expense of performance (in this scenario most related to scalability). Therefore, we must seek not only secure but efficient SMPC solutions.

C. What is blockchain and how it can help?

The following motivation is also applicable to the next subsection II-D since the solution Enigma is based in a blockchain.

Centralization is becoming an increasing problem day by day. Even if centralizing data has performance or money benefits, the disadvantages outweigh those. It centralizes the power and increases the possibility of corruption, inequality and abuse of power. The lack of transparency regarding data privacy (normally related to manipulation, surveillance or even data breaches) can lead up to catastrophic events. Solutions to decentralize this are necessary and although they may not be as efficient as centralized ones, work must be done. Bitcoin and other technologies using blockchains make us hopeful. With such technologies we can develop decentralized applications without a centralized party with full control of the participants and their data. Transparency, irrefutable record of activities and rewards for honest behaviors are some of the advantages we can see on a blockchain.

Blockchain is a distributed database with some kind of agreement. It is a decentralized peer-to-peer network persistent with transparency and auditability. Also, its data structure are data blocks in chain, i. e., in sequential order. By adding an hash of the previous block to the current one makes the blockchain immutable making it impossible for attackers to act dishonestly without being detected (all operations can be inserted into the blocks, which are distributed among the different peers in the network).

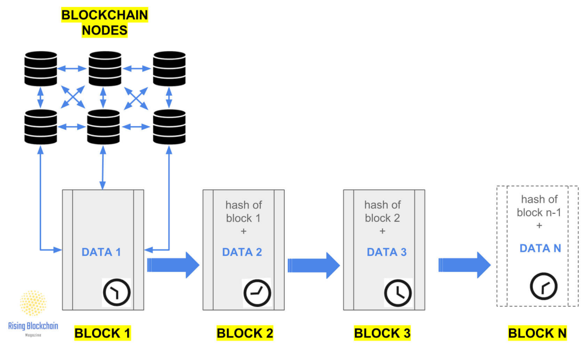


Fig. 1: Blockchain structure
Image taken from [4].

These characteristics makes blockchain a possible solution to SMPC that can solve both trust (fairness) and security problems we have seen in the last subsection. Also, with some tweaks, we can have a solution that while maintaining security, is also efficient (this topic is talked in more detail in the next subsection II-D).

With a blockchain we can have an agreement in decentralized scenarios without the need of a trusted third party.

Parties receive rewards for acting honestly while malicious ones are penalized if they are dishonest. Therefore, in such system, parties are encouraged to be honest. Even if parties still choose to be dishonest, since everything in blockchain is auditable and non-tamperable, malicious actions stay logged in the blockchain so we can know who is acting malicious and penalize them. Also, with the help of smart contracts (code that runs automatically when something happens in the blockchain), we can automate this step (rewarding or penalizing the parties).

However, we do not want to have everything public in the blockchain (as it happens in Bitcoin, for example), i.e., private data cannot flow fully exposed through every block in the chain [5]. Regarding SMPC, some data should remain off-chain (i.e., not visible in the blockchain where all parties can see it). Also, having all data distributed over several nodes (redundancy) can bring performance problems.

In the next subsection II-D we will see a viable and efficient solution to perform SMPC called Enigma.

D. Enigma - a possible solution to bitcoin blockchains limitations

Bitcoin and other related blockchains have some problems when we think about SMPC: they are directly related to cryptocurrency, all the data is public on the ledger and performance-issues may arise.

That is why Enigma was born [5]: a decentralized computation platform based on blockchain. Combines both blockchain, an off-chain storage and computation.

Off-chain is basically tasks that happen outside the blockchain but a reference (encrypted or not) about them is stored in the blockchain. When the task is to store some data, normally a reference to the data is stored on the blockchain but not the data itself (because sometimes this data is too big). Also, off-chain storage happens when the data it-self requires the ability to be changed or deleted [6] (this last point is important in our scenario because parties should have the possibility to remove their sensitive data when they wish).

It provides incentives and efficiency in order make Enigma secure and a viable solution.

As it is based on a blockchain, Enigma is also a distributed peer-to-peer network that enables different parties to join and run computations while keeping each party inputs private to each other.

Enigma is a highly optimized secure multiparty computation algorithm and it is guaranteed by verifiable secret-sharing scheme (in the image 2 we can see an illustration of this scheme).

It is important to understand what a secret-sharing scheme is: similarly to the example about the average salary presented before in subsection II-A, in secret-sharing (also known as secret splitting) we have a secret splitted between all the computation parties called a share. The share on its own has no usage but when combined together the secret can be reconstructed. This scheme makes possible to perform secure computations between multiple parties with Enigma.

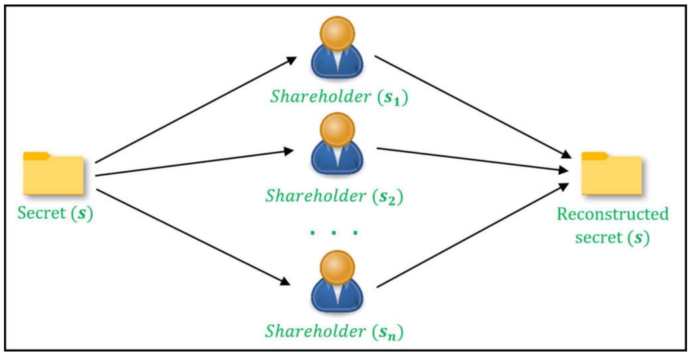


Fig. 2: Secret Sharing Scheme
Image taken from [7].

Some important aspects about Enigma:

- it uses a blockchain to manage access control and tamper-proof log of events
- with security deposits (explained later in this subsection) and incentive fees provides corrections and fairness of the system
- there is no need for a trusted third party (in other schemes, the presence of a centralized third party ruin everything since they can be the ones acting maliciously) - autonomous control of personal data
- users can share their data with privacy (using cryptography) and the access control to the data is managed by the blockchain

With Enigma we can achieve decentralized computation with guaranteed privacy. With Privacy by design we can develop end to end decentralized applications without trusted third parties and achieve two characteristics that were not possible before:

- Private: Enigma uses SMPC and data queries are computed in a distributed way. The data is spit between different nodes and those nodes compute functions together without leaking information to other nodes. In any point no single party has access to data in its entirety, they only have access to a piece of it that by itself seems completely random to them.
- Scalable: unlike in bitcoin, in Enigma, computations and data storage are not replicated in every node. They are stored in the off-chain resulting in less redundancy and, consequently, more power for computation. A simple application is again the example with the salary where each party wants to know its position in the group but do not want to disclose their salary to the others.

About Enigma's design, as I said before, it is connected to a blockchain and off-loads the private and intensive computations to an off-chain network. This computations that happen outside the blockchain provide anonymity for the parties. The transactions that occur are governed by the intrinsic properties of the blockchain and because of that we have access control with digital signatures and programmable permissions. This model results in two different parts:

1) Enigma is used for private and computationally intensive tasks. It ensures privacy and correctness.
2) By itself Enigma cannot provide fairness so that is where the blockchain enters. It provides access control and incentives to ensure correctness and fairness.

This model result in a system reliable, secure, fair and efficient.

After the computations, the proofs of correct execution are stored on the blockchain that can be further audited. This way, we can detect if parties are being honest or not.

Enigma also introduces the term of *private contracts* that they claimed [5] to be more private smart contracts since their state is not publicly shared.

To sum up, I will structure the Enigma's components:

1) Storage: it is a distributed hash table that happens on the off-chain and it is accessible through the blockchain (it stores a reference to the data but not the data itself). Private data is encrypted before sent to the storage and access control enters the scene at this time when the data owner allows his data to be shared between other parties.
2) Private computation: is done using Enigma to execute code without leaking the raw data to any node while ensuring correct execution. Correct execution is guaranteed by deposit fees. Once a party enters in this system, they are required to deposit a fee. If the parties act honestly, they are rewarded. In the other hand, if they act maliciously, Enigma takes their deposit and distribute between the honest nodes. Dishonest nodes are prevented to participate if their account balance is below a minimum threshold and are removed from the network if they do not deposit a fee again.

I will not go into details, but because Enigma uses the algorithm SPDZ, guarantees security even if there are dishonest nodes [8].

Another important feature of Enigma is the network reduction: the goal is to maximize the computational power of the network. Choosing randomly between the nodes with best reputation (measured by their publicly validated actions) and load-balancing from the network, the network is fully used since it will not choose the nodes that are being used (because of the load balancing requirement).

Also, when developing application with Enigma, developers can use the private keyword to specify private objects ensuring that any computation involving those remain secure and private. It is important to remember because the data is not publicly available on the blockchain, only a reference to them.

Finally, the image below represents the comparison between unsecure algorithms, naive SMPC algorithms and optimized SMPC algorithm (Enigma):

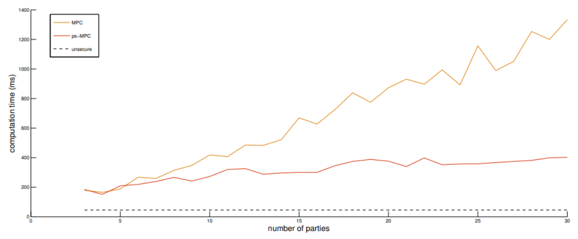


Fig. 3: Comparison between different SMPC algorithms Image taken from [5].

We can easily see that unsecure options remain constant with increasing number of parties but, unsurprisingly, they should not be considered since they are unsecure. Other SMPC algorithms increase exponentially as the number of parties increase. Fortunately, Enigma shows good results even when the number of parties increase. This can make us hopeful for the future.

E. Conclusion - countless possibilities

In this last subsection of this first part, I would like to conclude with some possible applications of such solutions (like Enigma). I collected some of the most interesting from [5]:

- Data marketplace: if the users know that their data is being securely and privately processed (eg. using Enigma), customers will be more willing to sell access it.
- Secure backend: prevent data breaches.
- Internal Segmentation: if the data is protected, companies can prevent rogue employees.
- n-factor Authentication: biometrics data can be saved and computed on enigma ensuring that only the data-owner has access to it.
- Internet of Things (IoT): with the increase usage of this type of technology, a new problem related to privacy emerged. Using Enigma, this problem can be solved and the data processed by IoT devices can remain private and secure.
- Crypto-bank: gives the possibility to this type of bank appear where users can use their banks to invest, make loans, etc without publicly revealing their private details.
- Blind e-voting: as the name suggests, vote anonymously.

Also, it is important to note that using such technology would be a good approach to be compliant with different regulations such as General Data Protection Regulation (GDPR).

III. Report - Study on SMPC PSI

A. Introduction

Private Set Intersection (PSI) can be explained as when two parties P1 and P2 with sets X and Y want to discover the intersection between the sets without revealing any information that is not in the intersection.

Common usages of such algorithm could be relationship path discovery in social networks, botnet detection, testing human genomes, proximity testing or cheater detection in online games [11].

Normally, adding security to an insecure solution (as we will see with PSI) makes them inefficient with more than two orders of magnitude of overhead. Therefore, PSI solutions that guarantee security are slower than tools that does not provide any. However, as we will see in this section, there are already solutions that provide acceptable results and we need them to replace the unsafe solutions that are currently widely in use.

In this section I will perform a practical study about PSI using the tool [9].

I start by doing a quick introduction about the four different PSI protocols in Subsection III-B. Following, in Subsection III-C with two different datasets (two different but relevant contact lists) I use the four protocols to understand them better and perform some comparisons.

In section III-D, I use the tool `psi.exe` provided by [9] to perform some benchmarking tests between the protocols and give my two cents' worth.

Finally, in Subsection III-E, I explain a real-world scenario where PSI could be useful and use the most effective protocol to calculate the intersections in the datasets I made artificially using [10].

All files used and obtained in this section can be found in this google drive url.

B. Comparison between four PSI protocols

As said before, the tool provided in [9] uses four different PSI protocols. I tried to collect some information from the following references [12] [13] [11] to explain explain them:

1) **Naive Hashing**: we hash all elements and instead of comparing them in plaintext, we compare only the hashes. This solution has the overhead of hashing everything but after that is fast since we just compare the hashes just like any insecure algorithm compares the values in clear text resulting in a very efficient solution. Of course this solution is insecure. If the input domain is small or has low entropy, i.e., easy to predict an attacker can easily bruteforce to calculate for all hashes likely to be an input and compare with the hashes.
2) **Server-aided**: relies on the use of a third party that makes the computational resources available as a service. Tries to solve the problem of non-fairness and non-scalable PSI algorithms but relies too much on the honesty of the third party. This solution is against everything we talked in the previous section II.
3) **Diffie-Hellman-based**: first PSI protocol with linear communication complexity. It uses public key encryption. As expected if Elliptic-Curve Cryptography (ECC) is used it is faster than using RSA. It basically allows for two parties to verify if their preferences match using public-key encryption. We can already see that this solution will not be as efficient as desired (due to public-key encryption known performance limitations).
4) **OT-based**: uses oblivious transfer and symmetric encryption with special optimizations (hashing into spe-

cific bins to reduce the number of comparisons, more about this at the end of this subsection) resulting in a viable (and efficient) solution for PSI with SMPC.

As suggested in the assignment, I performed some basic tests with pre-generated emails (emails_alice and emails_bob) to compare the different protocols. Also, I generated 10000, 100000 and 1000000 emails with the tool`emailgen.py` provided in [9] to see how the protocols behave when the number of elements increase.

In order to further analyze the protocols, I sniffed the packets exchanged with Wireshark. In [9] implementation, all four protocols rely on a TCP connection. They start with a packet SYN to start the connection, exchange messages using a specific flag of TCP to push data named PSH and finish the connection with packet FIN. Of course, all packets have another packet associated called ACK that acknowledges if the packet was received by the partner.

In Appendix A there are four different figures with the packets exchanged and the respectively diagrams for each one of the protocols. I put out all the packets that I thought were not relevant to the protocol's operation.

In PSI problem, the challenge is to compute the intersection with the fewest comparisons. Normally, we would need square of N (set size) comparisons but this results in a high number of cryptographic operations in secure solutions (however, as we will see, OT-based solution is promising).

In the naive-hashing solution presented in the figure 9 we have the sets 1,2 and 2,3. Both parties calculate the hashes of their sets and sent to each other before closing the collection (in packets 24 and 25). After that, each party compares the hashes of the partner to their own and calculate the intersections. This solution is the most used because of its efficiency although we have already seen that it is not secure.

Server-aided solution in figure 10 the inputs were 1,2 and 2,3,4. Their sizes are sent from the clients to the server as soon as they are connected. Right after the clients send their hashed datasets to the server. When the server receives them it computes the intersection and sends the number of intersection and the intersected elements to each client.

Analysing the source code in [9] we can see that Diffie-Hellman-based solution uses Elliptic-curve cryptography instead of RSA for better performance. As we will see, this solution is efficient to small datasets but as the size increases the efficiency degrades exponentially. To understand the implementation, parties' inputs were 1,2 and 2,3. As we can see in figure 11 the sizes of the datasets are exchanged in packets 4 and 14. Then, server sends its dataset after hashed and publicly encrypted. The client returns its dataset hashed and encrypted. The server encrypts the received client dataset with server's key and sends to the client. It is important to note that the computation is only performed on the client side. So, in the last step, the client has both datasets and can calculate the intersections.

In OT-based protocol, oblivious transfer is used. In order to otimize oblivious transfer, hashing can be used. We take advantage of this right when we hash our dataset. The items are hashed into a certain bin. When comes to the phase to compute the intersections, only hashes in the same bin are compared. Since there are less elements in a bin there will be less comparisons decreasing the number of necessary computations. In order to ensure that parties cannot get information (from empty bins for example), bins can be padded with dummy items. Finally, in figure 12 we can see more relevant packets exchanged. However, I did not understand the exchanges used in this implementation of this protocol. First, both parties exchange their sets sizes (packets 4 and 14). Then, in packets 24 and 26 I think the size of the bins that will be used are exchanged. As in Diffie-Hellman-based the computation is also done only on the client side after the client receives the last packet from the server (packet 33).

C. Explanation about the chosen dataset and tests

In this subsection I used my phone contacts and the contacts of a family member of mine. I had to clean the dataset since it had too much unnecessary data and the numbers had different formats (some had +351, others not). Also, I anonymized the datasets to only include the first name of the person in both datasets. One sample row of the dataset is:

Name	Phone
João	912-345-678

The function to clean the dataset was a simple python script stored in this url. Using any of the four protocols would easily calculate the intersection between the two contact lists. A sample output is the following:

```
Computation finished. Found 27 intersecting elements
    :
Alberto,912-801-522
Jo o,912-124-614
Rita,910-999-182
Andr ,912-652-328
Manel,917-528-563
Lu s,912-718-030
Jos ,916-660-427
Ant nio,918-663-338
Manuela,916-118-488
Rui,918-529-325
Alda,916-648-219
Lu sa,912-413-679
Josefina,913-993-266
Tiago,915-818-207
Tatiana,917-594-335
Alexandre,917-591-567
Alexandrina,919-539-990
Jo o,914-788-701
Nuno,934-268-653
Diana,912-564-520
Jo o,968-748-261
Jo o,914-893-288
Jo o,915-579-154
Jo o,917-300-854
Jo o,962-720-125
Jo o,918-963-016
Jo o,911-033-775
```

Just because any of the protocols reach the desired goal, it does not mean that any tool is suitable for the usecase. Of course the naive hashing is insecure (explained in Subsection III-A) and should not be used despite its low overhead.

Wireshark Statistics for contacts dataset				
Algorithm	0	1	2	3
Packets	30	34	36	37
Time span (s)	8.925	20.015	47.737	1.273
Average (packers p s)	3.4	1.7	0.8	29.1
Average packet size	235	300	927	1720
Bytes	7046	10184	33376	63654
Average bytes/s	786	508	699	49k
Average bits/s	6,315	4,070	5593	399k

Even if the server-aided approach uses a honest third-party, the same hashes as the naive approach are also sent over the internet making it possible for an attacker to sniff them. Also, in this second algorithm, the time it needs to calculate the intersection is also too much. In my opinion, this solution is also non-viable.

This leaves us to the second and third algorithms: Diffie-Hellman-based and OT-based, respectively. Since the second algorithm is based on public key encryption, its overhead will always be superior to solutions that use symmetric encryption (such as OT-based). This higher overhead makes this solution completely unusable when the number of elements increases. In the subsection III-D, we will see that when the number of elements increases, the algorithm struggles a lot to get the job done, i.e., calculate the intersection between the two datasets.

This last algorithm, although it uses more bandwidth (easily seen in the table above: bigger packet size and number of bytes) is faster than other algorithms (with the exception of native hashing but even though in the performance subsection we will see that they are close). Therefore, in my opinion and basing myself in the tests I have presented so far, the best algorithm from a security vs. cost point of view the third algorithm (OT-based) is the goto approach for Private Set Intersection with SMPC. In the following subsection III-D I will perform some benchmarking tests to confirm my assumptions.

D. *Performance Benchmarking of PSI protocols*

As said before, I used the tool `psi.exe` provided by [9] to perform some benchmarking tests against the three protocols. It was unfortunate that the tool do not support the tests for server-aided protocol, however, in the first subsection II-A I perform some comparisons between all four protocols.

Performance Benchmarking					
Alg. 0 - Set Size	5k	10k	100k	1000k	10000k
Required Time (s)	0.01	0.02	0.14	1.72	21.36
Data sent (MB)	0.04	0.09	0.95	9.54	104.90
Data received (MB)	0.04	0.09	0.95	9.54	104.90
Alg. 2 - Set Size	5k	10k	100k	1000k	10000k
Required Time (s)	16.02	32.25	320.80	3228.61	/
Data sent (MB)	0.33	0.66	6.58	65.80	/
Data received (MB)	0.18	0.35	3.54	35.29	/
Alg. 3 - Set Size	5k	10k	100k	1000k	10000k
Required Time (s)	1.33	1.39	2.76	13.9	/
Data sent (MB)	0.15	0.28	2.88	28.63	/
Data received (MB)	0.38	0.75	7.34	73.25	/

The cells marked with / are due to the fact that the tool closed prematurely (probably due to the lack of resources of the virtual machine).

All the data to calculate the following graphics can be found in this url.

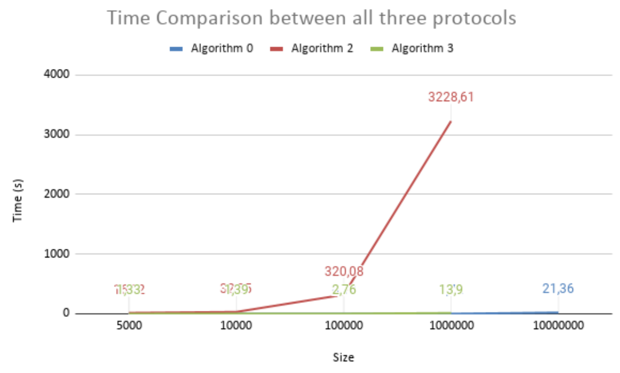


Fig. 4: Time comparison between three algorithms

In terms of required time, we clearly see that algorithm 2 that uses public key cryptography takes way too long making it a solution non viable for the majority of the cases. In the other hand, the times in algorithm 3 (OT-based) seem interesting. I did an additional graphic comparing only this algorithm and the naive hashing:

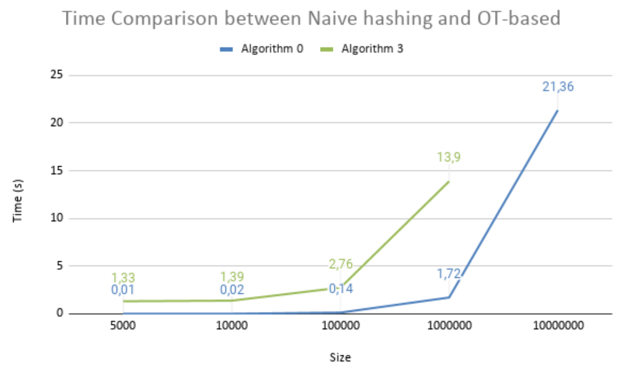


Fig. 5: Time comparison between two algorithms

Naive hashing wins, but not for much. These results makes OT-based solution a very viable approach without missing security/privacy.

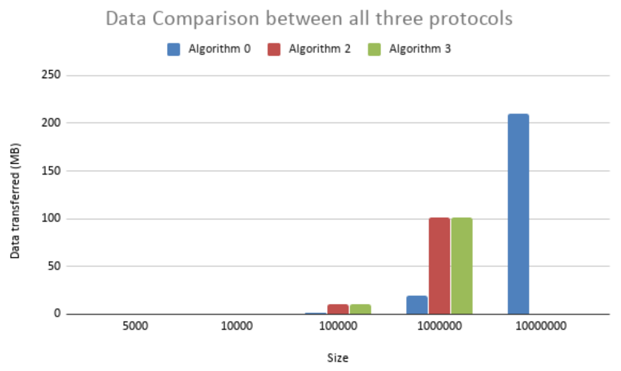


Fig. 6: Data comparison between three algorithms

We can clearly see that these solutions that enable privacy (algorithm 2 and 3) need lot more data to be transferred (comparing to naive hashing). In my opinion, this is not serious and is a trade-off that we have to accept.

The following image taken from [14] presents results similar to those obtained with the tests performed:

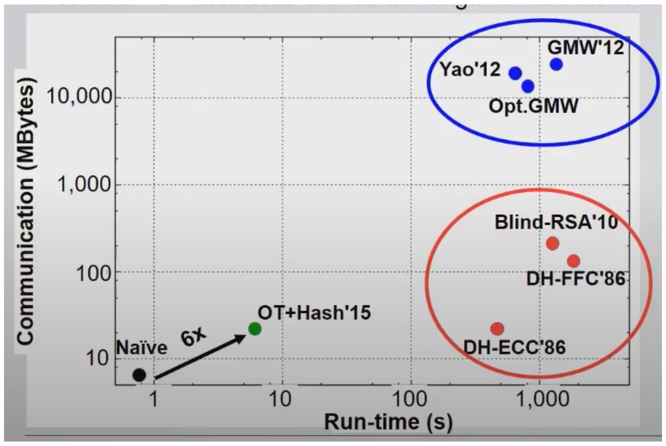


Fig. 7: Protocols comparison taken from [14].

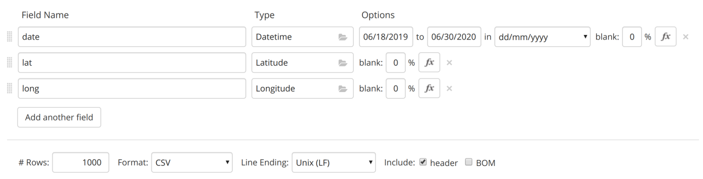
The black point in the bottom-left corner is the solution that uses naive-hashing and is by far the most efficient one but the less secure. The OT-based comes close with only one order of magnitude of overhead. The red circle represents the Diffie-Hellman-based protocol and we can see that the one that uses Elliptic-Curve Cryptography (ECC) is a viable solution when the set sizes are not too big.

E. Real-world example where PSI can be useful - COVID-19 intersection

Due to COVID-19 pandemic, a new flood of applications emerged to do contact-tracing and even localization-tracing. Of course such applications are important but we need to make sure those use privacy-preserving technologies.

As so, I generated random datasets containing the positions of a certain person during the previous fourteen days (given that 14 is the maximum number of days for COVID-19 incubation period according to [15]). Therefore, if we have background that given two different parties one of those is infected, we can use PSI algorithms to calculate the intersection between the two sets of positions of those parties, i.e., if they had been in the same place on the same day.

To generate the datasets, I used [10]:



Field Name	Type	Options
date	Datetime	06/18/2019 to 06/30/2020 in dd/mm/yyyy blank: 0 % *fx*
lat	Latitude	blank: 0 % *fx*
long	Longitude	blank: 0 % *fx*

Add another field

Rows: 1000 Format: CSV Line Ending: Unix (LF) Include: [✓ header] BOM

Fig. 8: Mockaroo fields. Image captured on [10].

By default, mockaroo generates latitudes and longitudes with a lot of decimal cases (eg. lat: 38.6395893, long: -8.9690651) that could result in no intersection between the datasets. Therefore, I made another simple script in python (can be find in this url) to remove the decimal cases in both latitude and longitude and replace them by `.0`. A sample row in this dataset is (latitude, longitude, date):

Latitude	Longitude	Date
47.0	124.0	28/06/2020

With both datasets prepared, we can use the PSI algorithm to calculate the intersections. Following the last subsection, I chose, as expected, the third algorithm (OT-based, -p 3 in the tool [9]). The output is the following:

```
Hashing 1000 elements with arbitrary length into
    into 8 bytes
Client: bins = 1200, elebitlen = 54 and maskbitlen =
     64 and performs 1200 OTs
Computation finished. Found 31 intersecting elements
    :
49.0,14.0,1/1/2020
-6.0,107.0,1/16/2020
15.0,120.0,1/8/2020
-7.0,108.0,1/8/2020
-7.0,108.0,1/8/2020
7.0,124.0,1/29/2020
-7.0,-35.0,1/10/2020
41.0,-8.0,1/22/2020
41.0,-8.0,1/22/2020
48.0,2.0,1/2/2020
49.0,17.0,1/28/2020
41.0,-8.0,1/16/2020
45.0,-72.0,1/31/2020
39.0,117.0,1/17/2020
49.0,18.0,1/14/2020
59.0,17.0,1/21/2020
-6.0,106.0,1/1/2020
-6.0,106.0,1/1/2020
10.0,123.0,1/22/2020
-7.0,112.0,1/7/2020
60.0,29.0,1/2/2020
59.0,15.0,1/11/2020
-7.0,108.0,1/18/2020
40.0,19.0,1/12/2020
59.0,54.0,1/19/2020
41.0,-8.0,1/31/2020
41.0,-8.0,1/6/2020
-6.0,111.0,1/28/2020
```

```
-7.0,112.0,1/3/2020
-8.0,113.0,1/22/2020
14.0,121.0,1/10/2020
```

By itself, the output does not say much. It only says that both people had been in the same place in the same day 31 times but because the data is completely artificial the output is not relevant.

However, this was a simple but good example of an application of PSI where without revealing all the 1000 positions of both parties, we can see where they intersect themselves.

IV. Conclusion

In this paper I showed that achieving security and privacy in computations is hard and costly. In my opinion this is the main reason why unsecure solutions are still being used. We seen in this paper that solutions with privacy and security require more data, are more complex and have more latency (although the gap is closing).

As computers are getting faster and faster we need to put performance in the background and focus more on security until we get secure alternatives.

It is not just negative points, we also seen that solutions like Enigma seen in Section II and OT-based PSI seen in Section III are promising and in addition to ensuring security and privacy, they achieve results similar to unsafe solutions.

References

[1] Yao, Andrew C. "Protocols for secure computations." 23rd annual symposium on foundations of computer science (sfcs 1982). IEEE, 1982.

[2] (2020, June 29). *What is Secure Multiparty Computation — inpher*. Retrieved from https://www.inpher.io/technology/what-is-secure-multiparty-computation

[3] Zhong, Hanrui, et al. "Secure multi-party computation on blockchain: An overview." International Symposium on Parallel Architectures, Algorithms and Programming. Springer, Singapore, 2019.

[4] (2020, June 29). *1. THE BASICS OF BLOCKCHAIN & CRYPTOCURRENCIES What is the Blockchain technology and how it works?* Retrieved from https://risingblockchain.com/what-is-the-blockchain-how-it-works/

[5] Zyskind, Guy, Oz Nathan, and Alex Pentland. "Enigma: Decentralized computation platform with guaranteed privacy." arXiv preprint arXiv:1506.03471 (2015).

[6] (2020, June 29). *Why new off-chain storage is required for blockchains*. Retrieved from https://www.ibm.com/downloads/cas/RXOVXAPM

[7] taabishm2. (2020, June 29). *Implementing Shamir's Secret Sharing Scheme in Python - GeeksforGeeks*. Retrieved from https://www.geeksforgeeks.org/implementing-shamirs-secret-sharing-scheme-in-python/

[8] Baum, Carsten, Ivan Damgård, and Claudio Orlandi. "Publicly auditable secure multi-party computation." International Conference on Security and Cryptography for Networks. Springer, Cham, 2014.

[9] (2020, June 29). *bluetrickpt/PSI: Implementations of Private Set Intersection Protocols*. Retrieved from https://github.com/bluetrickpt/PSI

[10] (2020, June 30). *Mockaroo - Random Data Generator and API Mocking Tool — JSON / CSV / SQL / Excel*. Retrieved from https://www.mockaroo.com/

[11] Pinkas, Benny, Thomas Schneider, and Michael Zohner. "Scalable private set intersection based on OT extension." ACM Transactions on Privacy and Security (TOPS) 21.2 (2018): 1-35.

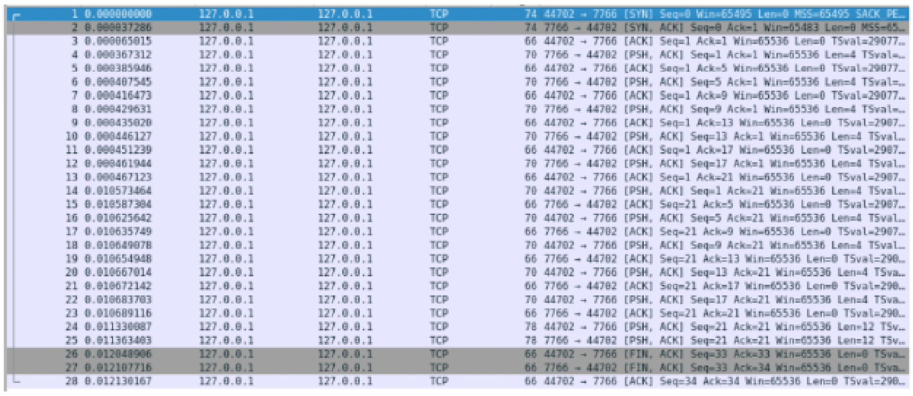
[12] Kamara, Seny, et al. "Scaling private set intersection to billion-element sets." International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2014.

[13] 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.

[14] (2020, July 6). *Solving Private Set Intersection via Cuckoo Hashing: Benny Pinkas, Bar-Ilan University, Israel - YouTube*. Retrieved from https://www.youtube.com/watch?v=iXopZ7A7dM0

[15] (2020, June 29). *Coronavirus Incubation Period (COVID-19) - Worldometer*. Retrieved from https://www.worldometers.info/coronavirus/coronavirus-incubation-period/

APPENDIX



```
┌  1 0.000000000  127.0.0.1  127.0.0.1  TCP  74 44702 → 7766 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PE…
   2 0.000037286  127.0.0.1  127.0.0.1  TCP  74 7766 → 44702 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65…
   3 0.000065015  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=29077…
   4 0.000367312  127.0.0.1  127.0.0.1  TCP  70 7766 → 44702 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=…
   5 0.000385946  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=29077…
   6 0.000407545  127.0.0.1  127.0.0.1  TCP  70 7766 → 44702 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=4 TSval=…
   7 0.000416473  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=1 Ack=9 Win=65536 Len=0 TSval=29077…
   8 0.000429631  127.0.0.1  127.0.0.1  TCP  70 7766 → 44702 [PSH, ACK] Seq=9 Ack=1 Win=65536 Len=4 TSval=…
   9 0.000435020  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=1 Ack=13 Win=65536 Len=0 TSval=2907…
  10 0.000446127  127.0.0.1  127.0.0.1  TCP  70 7766 → 44702 [PSH, ACK] Seq=13 Ack=1 Win=65536 Len=4 TSval…
  11 0.000451239  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=1 Ack=17 Win=65536 Len=0 TSval=2907…
  12 0.000461944  127.0.0.1  127.0.0.1  TCP  70 7766 → 44702 [PSH, ACK] Seq=17 Ack=1 Win=65536 Len=4 TSval…
  13 0.000467123  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=1 Ack=21 Win=65536 Len=0 TSval=2907…
  14 0.010573464  127.0.0.1  127.0.0.1  TCP  70 44702 → 7766 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=4 TSval…
  15 0.010587304  127.0.0.1  127.0.0.1  TCP  66 7766 → 44702 [ACK] Seq=21 Ack=5 Win=65536 Len=0 TSval=2907…
  16 0.010625642  127.0.0.1  127.0.0.1  TCP  70 44702 → 7766 [PSH, ACK] Seq=5 Ack=21 Win=65536 Len=4 TSval…
  17 0.010635749  127.0.0.1  127.0.0.1  TCP  66 7766 → 44702 [ACK] Seq=21 Ack=9 Win=65536 Len=0 TSval=2907…
  18 0.010649078  127.0.0.1  127.0.0.1  TCP  70 44702 → 7766 [PSH, ACK] Seq=9 Ack=21 Win=65536 Len=4 TSval…
  19 0.010654948  127.0.0.1  127.0.0.1  TCP  66 7766 → 44702 [ACK] Seq=21 Ack=13 Win=65536 Len=0 TSval=290…
  20 0.010667014  127.0.0.1  127.0.0.1  TCP  70 44702 → 7766 [PSH, ACK] Seq=13 Ack=21 Win=65536 Len=4 TSva…
  21 0.010672142  127.0.0.1  127.0.0.1  TCP  66 7766 → 44702 [ACK] Seq=21 Ack=17 Win=65536 Len=0 TSval=290…
  22 0.010683703  127.0.0.1  127.0.0.1  TCP  70 44702 → 7766 [PSH, ACK] Seq=17 Ack=21 Win=65536 Len=4 TSva…
  23 0.010689116  127.0.0.1  127.0.0.1  TCP  66 7766 → 44702 [ACK] Seq=21 Ack=21 Win=65536 Len=0 TSval=290…
  24 0.011330087  127.0.0.1  127.0.0.1  TCP  78 44702 → 7766 [PSH, ACK] Seq=21 Ack=21 Win=65536 Len=12 TSv…
  25 0.011363403  127.0.0.1  127.0.0.1  TCP  78 7766 → 44702 [PSH, ACK] Seq=21 Ack=21 Win=65536 Len=12 TSv…
  26 0.012048906  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [FIN, ACK] Seq=33 Ack=33 Win=65536 Len=0 TSva…
  27 0.012107716  127.0.0.1  127.0.0.1  TCP  66 7766 → 44702 [FIN, ACK] Seq=33 Ack=34 Win=65536 Len=0 TSva…
└ 28 0.012130167  127.0.0.1  127.0.0.1  TCP  66 44702 → 7766 [ACK] Seq=34 Ack=34 Win=65536 Len=0 TSval=290…
```

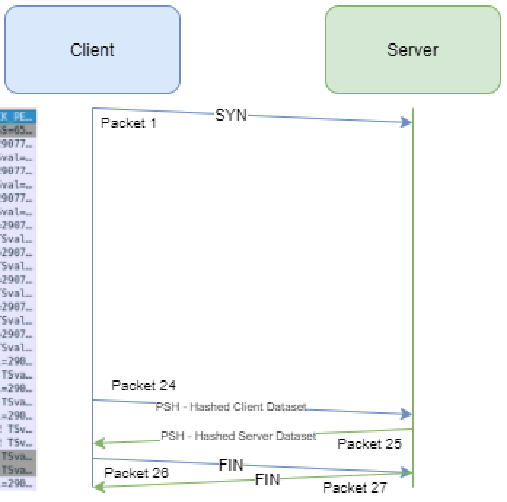
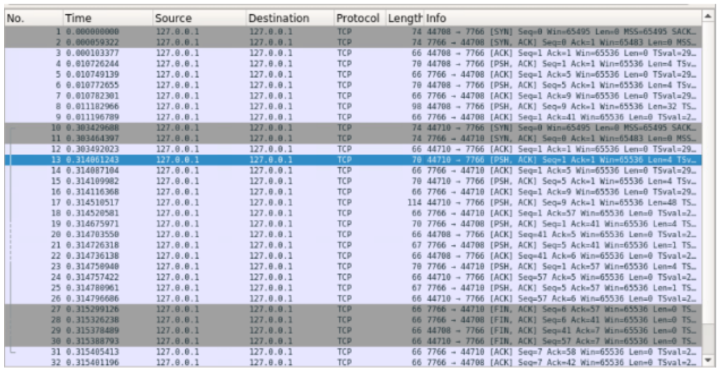


Fig. 9: Naive-hashing exchanged packets and protocol diagram



```
No.  Time         Source     Destination  Protocol  Length Info
   1 0.000000000  127.0.0.1  127.0.0.1    TCP        74 44708 → 7766 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK…
   2 0.000059322  127.0.0.1  127.0.0.1    TCP        74 7766 → 44708 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS…
   3 0.000103377  127.0.0.1  127.0.0.1    TCP        66 44708 → 7766 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=29…
   4 0.010726244  127.0.0.1  127.0.0.1    TCP        70 44708 → 7766 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSv…
   5 0.010749139  127.0.0.1  127.0.0.1    TCP        66 7766 → 44708 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=29…
   6 0.010772655  127.0.0.1  127.0.0.1    TCP        70 44708 → 7766 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=4 TSv…
   7 0.010782301  127.0.0.1  127.0.0.1    TCP        66 7766 → 44708 [ACK] Seq=1 Ack=9 Win=65536 Len=0 TSval=29…
   8 0.011182966  127.0.0.1  127.0.0.1    TCP        98 44708 → 7766 [PSH, ACK] Seq=9 Ack=1 Win=65536 Len=32 TS…
   9 0.011196789  127.0.0.1  127.0.0.1    TCP        66 7766 → 44708 [ACK] Seq=1 Ack=41 Win=65536 Len=0 TSval=2…
  10 0.303429688  127.0.0.1  127.0.0.1    TCP        74 44710 → 7766 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK…
  11 0.303464397  127.0.0.1  127.0.0.1    TCP        74 7766 → 44710 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS…
  12 0.303492023  127.0.0.1  127.0.0.1    TCP        66 44710 → 7766 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=29…
  13 0.314061243  127.0.0.1  127.0.0.1    TCP        70 44710 → 7766 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSv…
  14 0.314087104  127.0.0.1  127.0.0.1    TCP        66 7766 → 44710 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=29…
  15 0.314109982  127.0.0.1  127.0.0.1    TCP        70 44710 → 7766 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=4 TSv…
  16 0.314116368  127.0.0.1  127.0.0.1    TCP        66 7766 → 44710 [ACK] Seq=1 Ack=9 Win=65536 Len=0 TSval=29…
  17 0.314510517  127.0.0.1  127.0.0.1    TCP       114 44710 → 7766 [PSH, ACK] Seq=9 Ack=1 Win=65536 Len=48 TS…
  18 0.314520581  127.0.0.1  127.0.0.1    TCP        66 7766 → 44710 [ACK] Seq=1 Ack=57 Win=65536 Len=0 TSval=2…
  19 0.314675971  127.0.0.1  127.0.0.1    TCP        70 7766 → 44708 [PSH, ACK] Seq=1 Ack=41 Win=65536 Len=4 TS…
  20 0.314703550  127.0.0.1  127.0.0.1    TCP        66 44708 → 7766 [ACK] Seq=41 Ack=5 Win=65536 Len=0 TSval=2…
  21 0.314726318  127.0.0.1  127.0.0.1    TCP        67 7766 → 44708 [PSH, ACK] Seq=5 Ack=41 Win=65536 Len=1 TS…
  22 0.314736138  127.0.0.1  127.0.0.1    TCP        66 44708 → 7766 [ACK] Seq=41 Ack=6 Win=65536 Len=0 TSval=2…
  23 0.314750940  127.0.0.1  127.0.0.1    TCP        70 7766 → 44710 [PSH, ACK] Seq=1 Ack=57 Win=65536 Len=4 TS…
  24 0.314757422  127.0.0.1  127.0.0.1    TCP        66 44710 → 7766 [ACK] Seq=57 Ack=5 Win=65536 Len=0 TSval=2…
  25 0.314780961  127.0.0.1  127.0.0.1    TCP        67 7766 → 44710 [PSH, ACK] Seq=5 Ack=57 Win=65536 Len=1 TS…
  26 0.314796686  127.0.0.1  127.0.0.1    TCP        66 44710 → 7766 [ACK] Seq=57 Ack=6 Win=65536 Len=0 TSval=2…
  27 0.315299126  127.0.0.1  127.0.0.1    TCP        66 7766 → 44710 [FIN, ACK] Seq=6 Ack=57 Win=65536 Len=0 TS…
  28 0.315326238  127.0.0.1  127.0.0.1    TCP        66 7766 → 44708 [FIN, ACK] Seq=6 Ack=41 Win=65536 Len=0 TS…
  29 0.315378489  127.0.0.1  127.0.0.1    TCP        66 44708 → 7766 [FIN, ACK] Seq=41 Ack=7 Win=65536 Len=0 TS…
  30 0.315388793  127.0.0.1  127.0.0.1    TCP        66 44710 → 7766 [FIN, ACK] Seq=57 Ack=7 Win=65536 Len=0 TS…
  31 0.315405413  127.0.0.1  127.0.0.1    TCP        66 7766 → 44710 [ACK] Seq=7 Ack=58 Win=65536 Len=0 TSval=2…
  32 0.315401196  127.0.0.1  127.0.0.1    TCP        66 7766 → 44708 [ACK] Seq=7 Ack=42 Win=65536 Len=0 TSval=2…
```

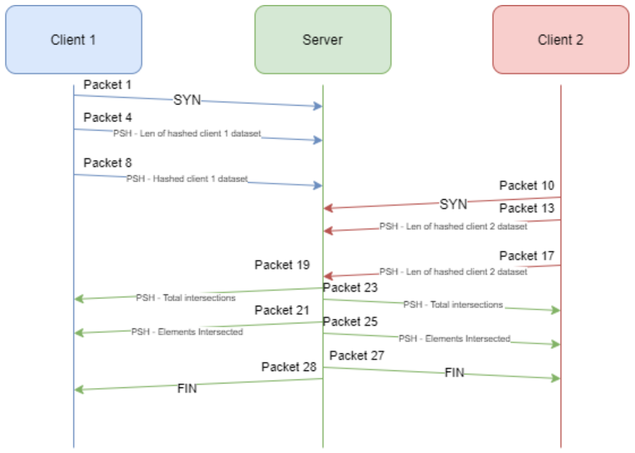
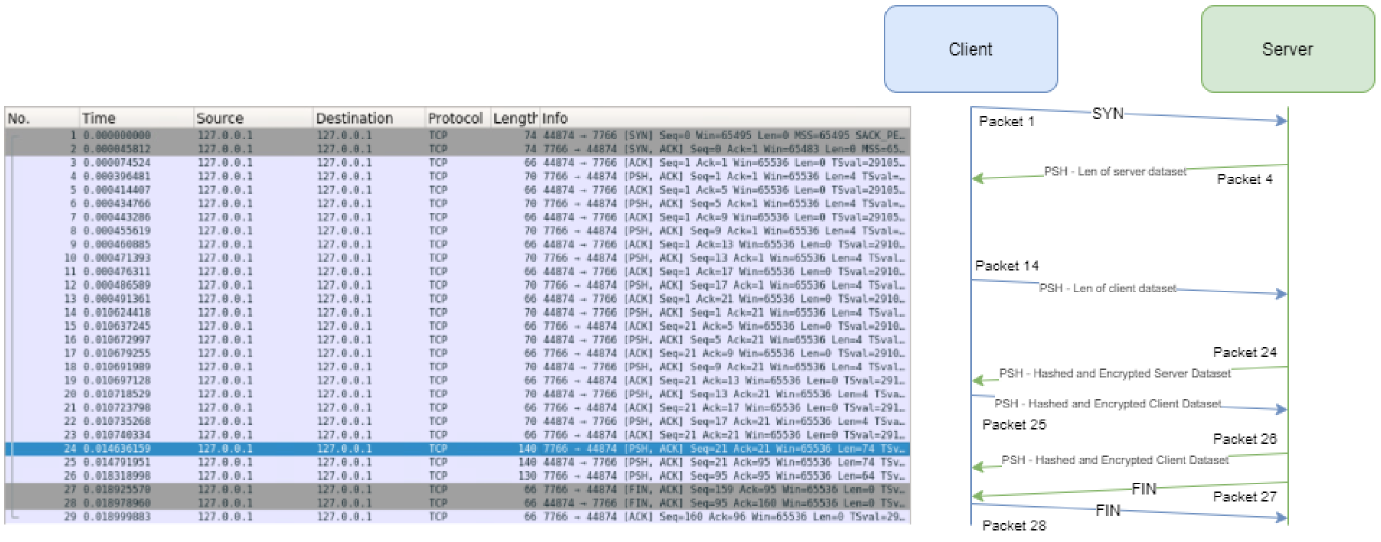

Client 1
Server
Client 2
Packet 1
SYN
Packet 4
PSH - Len of hashed client 1 dataset
Packet 8
PSH - Hashed client 1 dataset
Packet 10
SYN
Packet 13
PSH - Len of hashed client 2 dataset
Packet 17
PSH - Len of hashed client 2 dataset
Packet 19
Packet 23
PSH - Total intersections
PSH - Total intersections
Packet 21
Packet 25
PSH - Elements Intersected
PSH - Elements Intersected
Packet 27
Packet 28
FIN
FIN

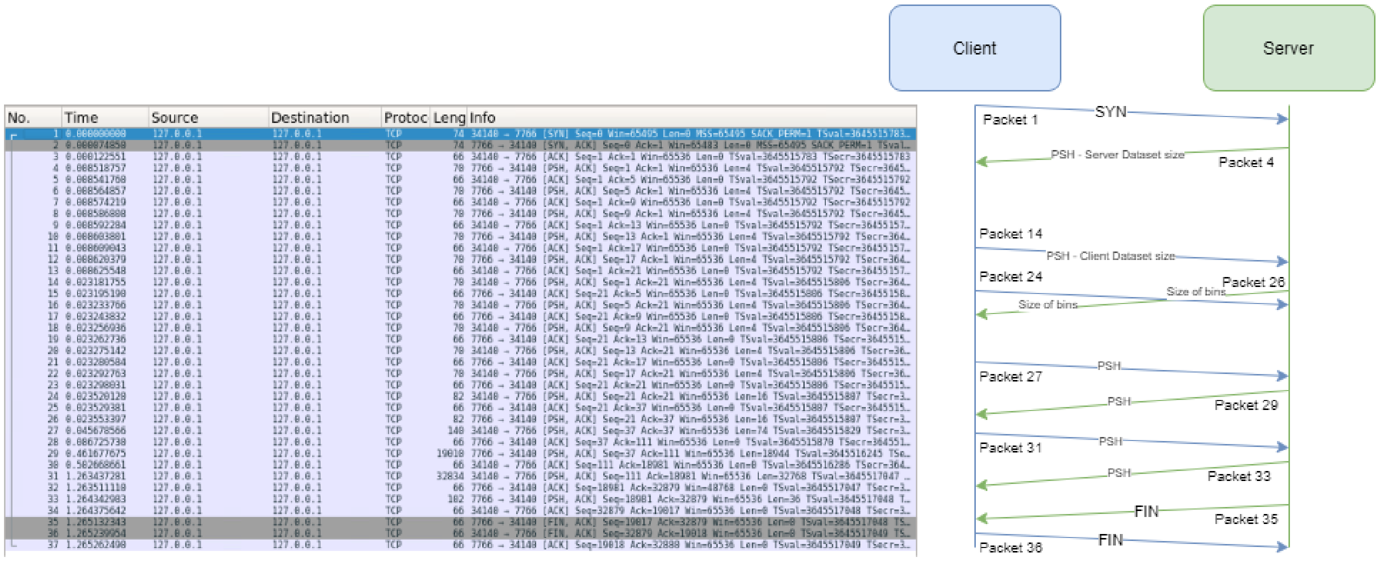
Fig. 10: Server-aided exchanged packets and protocol diagram



```
No.   Time         Source      Destination  Protocol  Length  Info
    1 0.000000000  127.0.0.1   127.0.0.1    TCP         74 44874 → 7766 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PE…
    2 0.000045812  127.0.0.1   127.0.0.1    TCP         74 7766 → 44874 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65…
    3 0.000074524  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=29105…
    4 0.000396481  127.0.0.1   127.0.0.1    TCP         70 7766 → 44874 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=…
    5 0.000414407  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=29105…
    6 0.000434766  127.0.0.1   127.0.0.1    TCP         70 7766 → 44874 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=4 TSval=…
    7 0.000443286  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [ACK] Seq=1 Ack=9 Win=65536 Len=0 TSval=29105…
    8 0.000455619  127.0.0.1   127.0.0.1    TCP         70 7766 → 44874 [PSH, ACK] Seq=9 Ack=1 Win=65536 Len=4 TSval=…
    9 0.000460885  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [ACK] Seq=1 Ack=13 Win=65536 Len=0 TSval=2910…
   10 0.000471393  127.0.0.1   127.0.0.1    TCP         70 7766 → 44874 [PSH, ACK] Seq=13 Ack=1 Win=65536 Len=4 TSval…
   11 0.000476311  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [ACK] Seq=1 Ack=17 Win=65536 Len=0 TSval=2910…
   12 0.000486589  127.0.0.1   127.0.0.1    TCP         70 7766 → 44874 [PSH, ACK] Seq=17 Ack=1 Win=65536 Len=4 TSval…
   13 0.000491361  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [ACK] Seq=1 Ack=21 Win=65536 Len=0 TSval=2910…
   14 0.010624418  127.0.0.1   127.0.0.1    TCP         70 44874 → 7766 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=4 TSval…
   15 0.010637245  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [ACK] Seq=21 Ack=5 Win=65536 Len=0 TSval=2910…
   16 0.010672997  127.0.0.1   127.0.0.1    TCP         70 44874 → 7766 [PSH, ACK] Seq=5 Ack=21 Win=65536 Len=4 TSval…
   17 0.010679255  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [ACK] Seq=21 Ack=9 Win=65536 Len=0 TSval=2910…
   18 0.010691989  127.0.0.1   127.0.0.1    TCP         70 44874 → 7766 [PSH, ACK] Seq=9 Ack=21 Win=65536 Len=4 TSval…
   19 0.010697128  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [ACK] Seq=21 Ack=13 Win=65536 Len=0 TSval=291…
   20 0.010718529  127.0.0.1   127.0.0.1    TCP         70 44874 → 7766 [PSH, ACK] Seq=13 Ack=21 Win=65536 Len=4 TSva…
   21 0.010723798  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [ACK] Seq=21 Ack=17 Win=65536 Len=0 TSval=291…
   22 0.010735268  127.0.0.1   127.0.0.1    TCP         70 44874 → 7766 [PSH, ACK] Seq=17 Ack=21 Win=65536 Len=4 TSva…
   23 0.010740334  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [ACK] Seq=21 Ack=21 Win=65536 Len=0 TSval=291…
   24 0.014636159  127.0.0.1   127.0.0.1    TCP        140 7766 → 44874 [PSH, ACK] Seq=21 Ack=21 Win=65536 Len=74 TSv…
   25 0.014791951  127.0.0.1   127.0.0.1    TCP        140 44874 → 7766 [PSH, ACK] Seq=21 Ack=95 Win=65536 Len=74 TSv…
   26 0.018318998  127.0.0.1   127.0.0.1    TCP        130 7766 → 44874 [PSH, ACK] Seq=95 Ack=95 Win=65536 Len=64 TSv…
   27 0.018925570  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [FIN, ACK] Seq=159 Ack=95 Win=65536 Len=0 TSv…
   28 0.018978960  127.0.0.1   127.0.0.1    TCP         66 44874 → 7766 [FIN, ACK] Seq=95 Ack=160 Win=65536 Len=0 TSv…
   29 0.018999883  127.0.0.1   127.0.0.1    TCP         66 7766 → 44874 [ACK] Seq=160 Ack=96 Win=65536 Len=0 TSval=29…
```

Client
Server
SYN
Packet 1
PSH - Len of server dataset
Packet 4
Packet 14
PSH - Len of client dataset
Packet 24
PSH - Hashed and Encrypted Server Dataset
PSH - Hashed and Encrypted Client Dataset
Packet 25
Packet 26
PSH - Hashed and Encrypted Client Dataset
FIN
Packet 27
FIN
Packet 28

Fig. 11: Diffie-Hellman-based exchanged packets and protocol diagram



```
No.   Time         Source      Destination  Protoc  Leng  Info
    1 0.000000000  127.0.0.1   127.0.0.1    TCP       74 34140 → 7766 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3645515783…
    2 0.000074858  127.0.0.1   127.0.0.1    TCP       74 7766 → 34140 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval…
    3 0.000122551  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3645515783 TSecr=3645515783
    4 0.008518757  127.0.0.1   127.0.0.1    TCP       70 7766 → 34140 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=3645515792 TSecr=3645…
    5 0.008541760  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=1 Ack=5 Win=65536 Len=0 TSval=3645515792 TSecr=3645515792
    6 0.008564857  127.0.0.1   127.0.0.1    TCP       70 7766 → 34140 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=4 TSval=3645515792 TSecr=3645…
    7 0.008574219  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=1 Ack=9 Win=65536 Len=0 TSval=3645515792 TSecr=3645515792
    8 0.008586808  127.0.0.1   127.0.0.1    TCP       70 7766 → 34140 [PSH, ACK] Seq=9 Ack=1 Win=65536 Len=4 TSval=3645515792 TSecr=3645…
    9 0.008592284  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=1 Ack=13 Win=65536 Len=0 TSval=3645515792 TSecr=36455157…
   10 0.008603881  127.0.0.1   127.0.0.1    TCP       70 7766 → 34140 [PSH, ACK] Seq=13 Ack=1 Win=65536 Len=4 TSval=3645515792 TSecr=364…
   11 0.008609043  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=1 Ack=17 Win=65536 Len=0 TSval=3645515792 TSecr=36455157…
   12 0.008620379  127.0.0.1   127.0.0.1    TCP       70 7766 → 34140 [PSH, ACK] Seq=17 Ack=1 Win=65536 Len=4 TSval=3645515792 TSecr=364…
   13 0.008625548  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=1 Ack=21 Win=65536 Len=0 TSval=3645515792 TSecr=36455157…
   14 0.023181755  127.0.0.1   127.0.0.1    TCP       70 34140 → 7766 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=4 TSval=3645515806 TSecr=364…
   15 0.023195198  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=21 Ack=5 Win=65536 Len=0 TSval=3645515806 TSecr=36455158…
   16 0.023233766  127.0.0.1   127.0.0.1    TCP       70 34140 → 7766 [PSH, ACK] Seq=5 Ack=21 Win=65536 Len=4 TSval=3645515806 TSecr=364…
   17 0.023243832  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=21 Ack=9 Win=65536 Len=0 TSval=3645515806 TSecr=36455158…
   18 0.023256936  127.0.0.1   127.0.0.1    TCP       70 34140 → 7766 [PSH, ACK] Seq=9 Ack=21 Win=65536 Len=4 TSval=3645515806 TSecr=364…
   19 0.023262736  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=21 Ack=13 Win=65536 Len=0 TSval=3645515806 TSecr=3645515…
   20 0.023275142  127.0.0.1   127.0.0.1    TCP       70 34140 → 7766 [PSH, ACK] Seq=13 Ack=21 Win=65536 Len=4 TSval=3645515806 TSecr=36…
   21 0.023280584  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=21 Ack=17 Win=65536 Len=0 TSval=3645515806 TSecr=3645515…
   22 0.023292763  127.0.0.1   127.0.0.1    TCP       70 34140 → 7766 [PSH, ACK] Seq=17 Ack=21 Win=65536 Len=4 TSval=3645515806 TSecr=36…
   23 0.023298031  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=21 Ack=21 Win=65536 Len=0 TSval=3645515806 TSecr=3645515…
   24 0.023520128  127.0.0.1   127.0.0.1    TCP       82 34140 → 7766 [PSH, ACK] Seq=21 Ack=21 Win=65536 Len=16 TSval=3645515887 TSecr=3…
   25 0.023529381  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=21 Ack=37 Win=65536 Len=0 TSval=3645515887 TSecr=3645515…
   26 0.023553397  127.0.0.1   127.0.0.1    TCP       82 7766 → 34140 [PSH, ACK] Seq=21 Ack=37 Win=65536 Len=16 TSval=3645515887 TSecr=3…
   27 0.045678566  127.0.0.1   127.0.0.1    TCP      140 34140 → 7766 [PSH, ACK] Seq=37 Ack=37 Win=65536 Len=74 TSval=3645515829 TSecr=3…
   28 0.086725738  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=37 Ack=111 Win=65536 Len=0 TSval=3645515870 TSecr=364551…
   29 0.461677675  127.0.0.1   127.0.0.1    TCP    19010 7766 → 34140 [PSH, ACK] Seq=37 Ack=111 Win=65536 Len=18944 TSval=3645516245 TSe…
   30 0.502668661  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=111 Ack=18981 Win=65536 Len=0 TSval=3645516286 TSecr=364…
   31 1.263437281  127.0.0.1   127.0.0.1    TCP    32834 34140 → 7766 [PSH, ACK] Seq=111 Ack=18981 Win=65536 Len=32768 TSval=3645517047 …
   32 1.263511118  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=18981 Ack=32879 Win=48768 Len=0 TSval=3645517047 TSecr=3…
   33 1.264342983  127.0.0.1   127.0.0.1    TCP      102 7766 → 34140 [PSH, ACK] Seq=18981 Ack=32879 Win=65536 Len=36 TSval=3645517048 T…
   34 1.264375642  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [ACK] Seq=32879 Ack=19017 Win=65536 Len=0 TSval=3645517048 TSecr=3…
   35 1.265132343  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [FIN, ACK] Seq=19017 Ack=32879 Win=65536 Len=0 TSval=3645517048 TS…
   36 1.265239054  127.0.0.1   127.0.0.1    TCP       66 34140 → 7766 [FIN, ACK] Seq=32879 Ack=19018 Win=65536 Len=0 TSval=3645517049 TS…
   37 1.265262490  127.0.0.1   127.0.0.1    TCP       66 7766 → 34140 [ACK] Seq=19018 Ack=32880 Win=65536 Len=0 TSval=3645517049 TSecr=3…
```

Fig. 12: OT-based exchanged packets and protocol diagram