Assignment 2

Secure Multiparty Computation for Privacy in Practice

Tecnologias de Reforço da Privacidade

May 20, 2025

Guilherme Coutinho, up202108872

Leandro Costa, up202408816

Group D

Contents

2 Experimentation with Protocols for Private Set Intersection

2.4 Security Limitations of the Naive Hashing PSI Protocol

The naive hashing protocol for Private Set Intersection (PSI) operates by computing the SHA-256 hash of each input element and truncating the output to the 6 most significant bytes (48 bits) before exchanging these hashes between the two parties. While this approach is computationally simple and provides basic functionality, it presents considerable security vulnerabilities that make it unsuitable for real-world applications involving sensitive data.

First and foremost, truncating a cryptographic hash function like SHA-256 to only 48 bits drastically reduces the search space, making brute-force attacks feasible with modern hardware. An adversary who intercepts the exchanged hashes, as clearly observable in the captured packets via Wireshark, can efficiently compute and match all possible hash values for a given input domain. This vulnerability is exacerbated by the fact that the hashing process is deterministic and unsalted: identical inputs always result in the same truncated hash, regardless of the session or participant. This enables replay and re-identification attacks, where an adversary can determine whether a known element was included in another party's input set across multiple PSI executions.

Moreover, truncation increases the risk of hash collisions. Two distinct inputs may produce identical 6-byte prefixes, leading to false positives in the intersection and compromising both correctness and privacy. Since the hashes are sent in plaintext, there is no protection against passive eavesdroppers, and no mechanism to prevent malicious participants from exploiting the protocol.

2.5 Naive hashing protocol against AppList2

The results of this intersection were saved in the file *intersection5.txt*, which contained the following outcome:

```
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

Listing 1: Output of the Intersection

Was started a new capture in wireshark and using the list of apps AppList1.csv, run the naive hashing protocol against AppList2.csv.

Using the filter **tcp.port == 7766**, it is possible to filter the packets to display only those that correspond to this specific intersection, rather than all the services running on the local machine.

Statistics

Measurement	Captured	Displayed
Packets	73	25 (34.2%)
Time span, s	44.131	0.018
Average pps	1.7	1367.2
Average packet size, B	131	88
Bytes	9541	2196 (23.0%)
Average bytes/s	216	120 k
Average bits/s	1729	960 k

Figure 1: Captured file Properties

This screenshot illustrates that this intersection involves 25 packets totaling 2196 bytes.

2.6 Server-aided protocol

```
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
```

```
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

Listing 2: Output of the Intersection

The screenshot below shows the Wireshark capture of the communication between the two clients and the server during the execution of the server-aided PSI protocol.

Statistics

Measurement	Captured	Displayed
Packets	39	33 (84.6%)
Time span, s	3.389	3.389
Average pps	11.5	9.7
Average packet size, B	114	102
Bytes	4446	3364 (75.7%)
Average bytes/s	1311	992
Average bits/s	10 k	7940

Figure 2: Captured packets

Were captured 33 packets and 3364 bytes. We will use the same filter for this and the next protocols.

2.7 Privacy issues that arise from the use of a server as a third-party

In the server-aided PSI protocol (option -p 1), both clients transmit their hashed input elements to a centralized third-party server, which performs the intersection computation on their behalf. As seen in the server output, the elements received from each client are 128-bit hashes, which are the same deterministic truncated hash values used in the naive protocol (step 5). This output confirms that the server has complete visibility over the hashed inputs of both clients.

This raises significant privacy concerns. Although the inputs are hashed, the use of fixed, truncated, and deterministic hashes allows a semi-honest or malicious server to carry out dictionary or brute-force attacks to potentially reverse these hashes, especially if the domain of inputs (example, common app names or email addresses) is small or predictable. Furthermore, because the server sees both sets entirely, it can compute not

just the intersection, but also the entire union or differences between the client datasets, violating the core PSI goal of mutual privacy.

The protocol assumes the server is trusted not to misuse this data. However, in real-world scenarios where such trust is not guaranteed, this model introduces an unacceptable risk of information leakage. In contrast, modern PSI protocols such as those based on oblivious transfer or Diffie-Hellman avoid giving any single party full access to both input sets. Therefore, while the server-aided model is computationally efficient and easier to deploy, it offers weak privacy guarantees and should be avoided in settings requiring strong confidentiality of input data.

2.8 Diffie-Hellman-based PSI protocol

```
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

Listing 3: Output of the Intersection

Statistics

Measurement	Captured	Displayed
Packets	30	30 (100.0%)
Time span, s	0.081	0.081
Average pps	371.8	371.8
Average packet size, B	190	190
Bytes	5714	5714 (100.0%)
Average bytes/s	70 k	70 k
Average bits/s	566 k	566 k

Figure 3: Captured packets

This intersection involved 30 packets totaling 5714 bytes.

2.9 OT-based PSI protocol

```
Hashing 34 elements with arbitrary length into into 7 bytes
Insertion not successful for element 30!
Client: bins = 41, elebitlen = 51 and maskbitlen = 56 and performs 41
    OTs
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

Listing 4: Output of the Intersection

Statistics

Measurement	Captured	Displayed
Packets	167	166 (99.4%)
Time span, s	2.618	1.001
Average pps	63.8	165.8
Average packet size, B	380	381
Bytes	63482	63322 (99.7%)
Average bytes/s	24 k	63 k
Average bits/s	193 k	506 k

Figure 4: Captured packets

This intersection involved 166 packets totaling 63322 bytes.

2.10 Number of captured packets & Total captured bytes

The following table illustrates the number of captured packets and total bytes exchanged for each PSI protocol during execution, as measured using Wireshark. These values provide insight into the communication cost of each protocol and help assess the trade-off between security and performance.

Table 1: Number of captured packets and total bytes for each PSI protocol

Protocol	**Captured Packets**	**Total Bytes Captured**
Naive Hashing (p = 0)	25	2196 Bytes
Server-aided (p = 1)	33	3364 Bytes
Diffie-Hellman-based (p = 2)	30	5714 Bytes
OT-based (p = 3)	166	63322 Bytes

The naive hashing protocol (p = 0) is simple and lightweight in terms of communication, as it requires only a few small packets to be exchanged. However, it is fundamentally insecure because it exposes the hashes of all elements, making them vulnerable to brute-force or dictionary attacks.

The server-aided protocol (p = 1) enhances scalability by offloading computation but raises privacy concerns since it relies on a semi-trusted third-party server that has access to the full set of inputs from both clients. Although the communication cost is slightly higher than that of the naive approach, the primary drawback lies in the trust assumptions required.

In contrast, the Diffie-Hellman-based PSI (p = 2) provides stronger security guarantees based on established cryptographic principles and protects the inputs from both parties during computation. This comes with a significant increase in communication overhead due to the need to exchange multiple large cryptographic messages.

Lastly, the OT-based PSI (p = 3) achieves the best balance of security and performance. It ensures that only the sender learns the intersection of sets, without disclosing any other information. While it incurs the highest communication costs in terms of total bytes and packets, this overhead is justified by its strong security guarantees and practical scalability for large datasets.

In summary, while protocols (0) and (1) are lightweight, they present unacceptable privacy risks in many situations. Protocols (2) and (3), though more expensive regarding bandwidth, offer significant privacy protections aligned with secure multiparty computation principles, making them the preferred choice for real-world applications where privacy is essential.

3 Benchmarking Private Set Intersection Protocols

After running the excutable file *./psi.exe* multiple times, the following *stats.csv* file was made in a way to register all the following data:

```
Set Size,Protocol,Time(ms),Sent(Bytes),Received(Bytes)
10000,0,100,104858,104858
100000,0,1000,1048576,1048576
500000,0,8200,5033165,5033165
1000000,0,9800,9961472,9961472
10000,2,10600,734003,419430
100000,2,66900,6912000,3670016
200000,2,136000,13841203,7444889
500000,2,338300,18454937,34406400
1000000,2,675700,37077606,69011046
1000,3,300,0,104857
5000,3,300,419430,104857
10000,3,300,314572,838861
50000,3,500,3879731,1363149
100000,3,600,7654605,3040870
200000,3,800,5976883,15414067
500000,3,1600,14942208,38377881
1000000,3,2400,29949952,76860621
```

Listing 5: Output of the Intersection

Based on this data, we were able to create the following plots. In Figure 5, we compare the execution time of three PSI protocols: 0 (naive hashing), 2 (Diffie-Hellman-based), and 3 (OT-based) across varying input set sizes. Protocol 0 consistently performs the fastest, with minimal variation in execution time. In contrast, protocol 2 experiences a significant increase in execution time as the set size grows, which indicates poor scalability. On the other hand, protocol 3 maintains relatively low execution times even with larger input sets, demonstrating better computational efficiency compared to protocol 2.

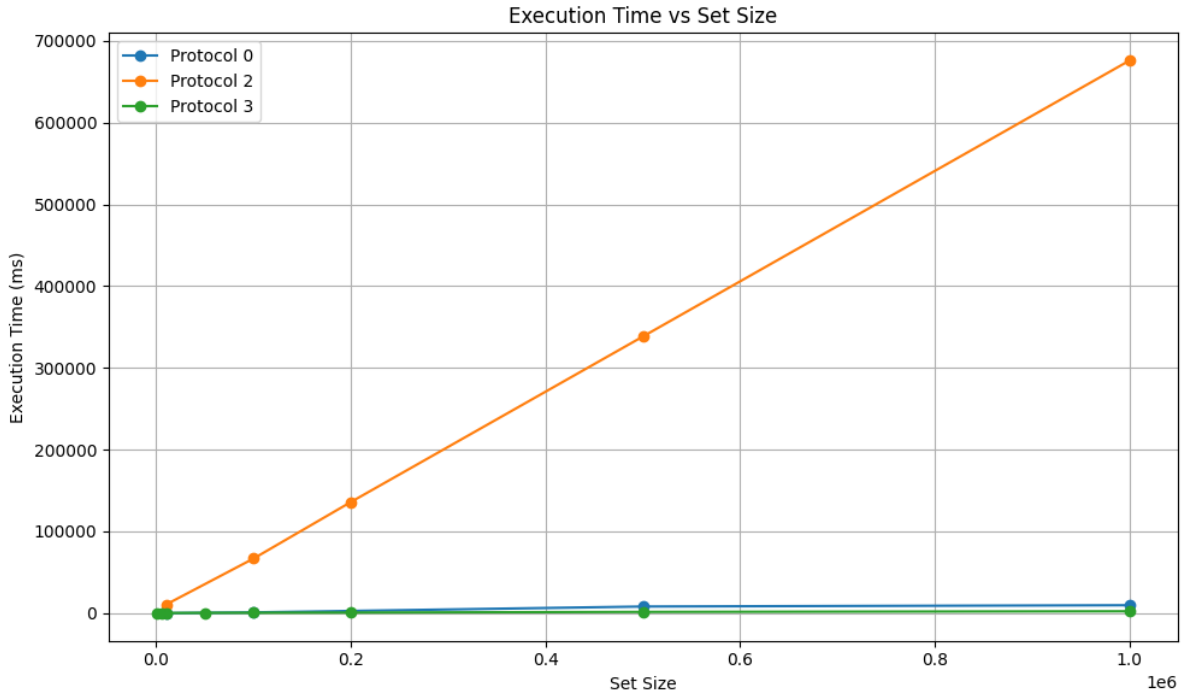

Figure 5: Execution Time vs Set Size

The following plot, from Figure 6, illustrates the total amount of data exchanged (both sent and received) by each protocol as a function of the input set size. Protocol 0 exchanges the least amount of data, while protocols 2 and 3 incur significantly higher communication overhead, which increases linearly with the size of the input set. Protocol 3 shows a slightly higher total data exchange than protocol 2, reflecting the additional cost associated with stronger privacy guarantees.

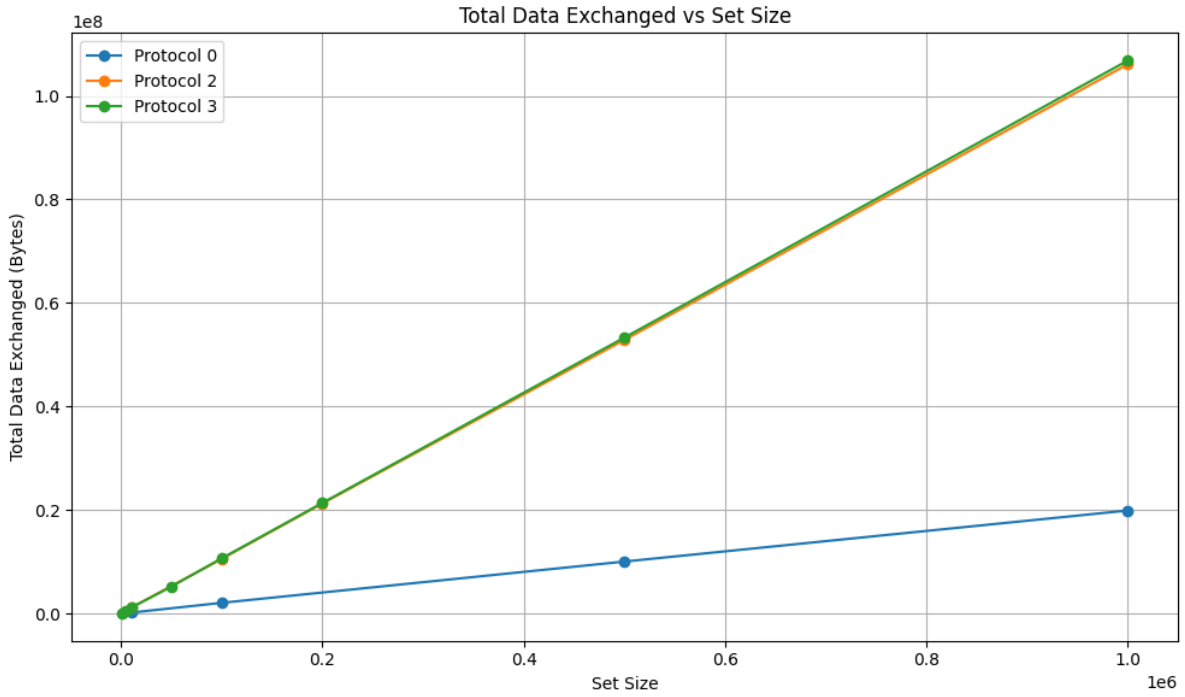

Figure 6: Total Data Exchanged vs Set Size

The benchmarking results reveal the trade-offs involved in designing PSI protocols. Protocol 0 is primarily suitable for testing or educational use, while Protocol 2 might not be feasible for large datasets due to its high computational demands. In contrast, Protocol 3 stands out as the most practical solution for real-world applications.

4 Secure intersection protocols to another dataset of your choice

To illustrate a realistic application of PSI, we simulated a scenario where two parties compare password sets while ensuring privacy.

We used the Rockyou password dataset, publicly available through the SecLists project. This dataset contains millions of real-world leaked passwords and is widely used in password security research and penetration testing. For our private set intersection experiments, we extracted subsets from this dataset to simulate realistic password usage scenarios.

Based on the performance and privacy evaluation conducted in previous steps, we chose to apply Protocol 3 for our final experiments. Protocol 3 demonstrated significantly better scalability when handling large datasets and provided stronger privacy guarantees compared to the naive and server-aided approaches.

We chose to use 10 million passwords from the dataset as our larger sub-dataset and a small_dataset.txt that contains 10,000 passwords, with 500 of them guaranteed to be found in big_dataset.txt.

```
wget https://github.com/danielmiessler/SecLists/raw/master/Passwords/
    Leaked-Databases/rockyou.txt.tar.gz

head -n 10000000 rockyou.txt > breach_leak_10M.txt

head -n 500 big_dataset.txt > shared.txt
tail -n +10000001 rockyou.txt | head -n 9500 > unique_small.txt
cat shared.txt unique_small.txt > small_dataset.txt
```

Listing 6: Command lines used for dataset preparation

To optimize the execution of terminal code, the following script was made to help the benchmark.

```
#!/bin/bash

# === CONFIG ===
SCENARIO="$1"
```

```
ROLE="$2"
RECEIVER_FILE="$3"
SENDER_FILE="$4"
OUTPUT_CSV="$5"

if [ $# -ne 5 ]; then
    echo "Usage: $0 <Scenario> <Role> <Receiver_File> <Sender_File> <Output_CSV>"
    exit 1
fi

echo "[INFO] Starting $SCENARIO with big dataset as $ROLE"

# === TEMP FILES ===
RECEIVER_LOG=receiver.log
SENDER_LOG=sender.log
RECEIVER_TIME=receiver_time.txt
SENDER_TIME=sender_time.txt

# === RUN PSI ===
if [ "$ROLE" == "Receiver" ]; then
    /usr/bin/time -v -o $RECEIVER_TIME ./demo.exe -r 0 -p 3 -f "$RECEIVER_FILE" > $RECEIVER_LOG &
    sleep 1
    /usr/bin/time -v -o $SENDER_TIME ./demo.exe -r 1 -p 3 -f "$SENDER_FILE" -a 127.0.0.1 > $SENDER_LOG
else
    /usr/bin/time -v -o $RECEIVER_TIME ./demo.exe -r 0 -p 3 -f "$SENDER_FILE" > $RECEIVER_LOG &
    sleep 1
    /usr/bin/time -v -o $SENDER_TIME ./demo.exe -r 1 -p 3 -f "$RECEIVER_FILE" -a 127.0.0.1 > $SENDER_LOG
fi

# === TIME PARSING ===
convert_to_seconds() {
    local t="$1"
    IFS=: read -r h m s <<< "$(echo "$t" | awk -F: '{ if (NF==3) print $1":"$2":"$3; else if (NF==2) print "0:"$1":"$2; else print "0:0:"$1
```

```
    }')"
38      echo | awk -v h="$h" -v m="$m" -v s="$s" 'BEGIN { print (h * 3600)
    + (m * 60) + s }'
39  }
40
41  TIME_R=$(convert_to_seconds "$(grep "Elapsed (wall clock)"
    $RECEIVER_TIME | awk '{print $8}')")
42  TIME_S=$(convert_to_seconds "$(grep "Elapsed (wall clock)" $SENDER_TIME
    | awk '{print $8}')")
43  TOTAL_TIME=$(echo "$TIME_R + $TIME_S" | bc)
44
45  # === CPU & MEMORY ===
46  CPU_R=$(grep "Percent of CPU" $RECEIVER_TIME | awk '{print int($8)}')
47  CPU_S=$(grep "Percent of CPU" $SENDER_TIME | awk '{print int($8)}')
48  CPU_MAX=$((CPU_R > CPU_S ? CPU_R : CPU_S))
49
50  MEM_R=$(grep "Maximum resident" $RECEIVER_TIME | awk '{print int($6 /
    1024)}')
51  MEM_S=$(grep "Maximum resident" $SENDER_TIME | awk '{print int($6 /
    1024)}')
52  MEM_MAX=$((MEM_R > MEM_S ? MEM_R : MEM_S))
53
54  # === INTERSECTION ===
55  INTERSECT=$(grep -hoP 'Found \K[0-9]+' $RECEIVER_LOG $SENDER_LOG | head
    -n 1)
56
57  # === HEADER CHECK ===
58  if [ ! -f "$OUTPUT_CSV" ]; then
59      echo "Scenario,Role,Time(s),CPU(%),Memory(MB),Intersection Count" >
    "$OUTPUT_CSV"
60  fi
61
62  # === APPEND TO CSV ===
63  echo "$SCENARIO,$ROLE,$TOTAL_TIME,$CPU_MAX,$MEM_MAX,$INTERSECT" >> "
    $OUTPUT_CSV"
64  echo "[INFO] Done. Data saved to $OUTPUT_CSV"
65
66  # === CLEANUP ===
```

```
rm -f $RECEIVER_LOG $SENDER_LOG $RECEIVER_TIME $SENDER_TIME
```

Listing 7: Benchmark script

The "Receiver" / "Sender" argument indicates which terminal contains the larger dataset during data intersection.

The following example shows the execution of a benchmark where the "Sender" holds the large dataset (breach_leak_10M.txt) and the "Receiver" holds the smaller dataset (user_passwords.txt).

```
./run_psi_benchmark.sh "Test 6" "Sender" user_passwords.txt
   breach_leak_10M.txt psi_benchmark.csv
```

Listing 8: Code execution for an intersection where the "Sender" holds the larger dataset.

This made it possible to register the data in the following CSV file:

```
Scenario,Role,Time(s),CPU(%),Memory(MB),Intersection Count
Test 1,Receiver,39.10,0,4268,500
Test 2,Sender,29.79,0,1217,500
Test 3,Receiver,39.74,0,4268,500
Test 4,Sender,21.83,0,1218,500
Test 5,Receiver,33.33,0,4268,500
Test 6,Sender,21.67,0,1217,500
```

Listing 9: Performance results recorded from benchmark runs using Protocol 3

The total of 500 shared passwords were found between the two datasets, confirming the expected intersection.

During the execution of the PSI protocol, we found that switching the roles of sender and receiver significantly affects both memory usage and execution time, as illustrated in the following plots. When the larger dataset is assigned to the sender, the receiver's resource usage decreases.

Figure 7: PSI Execution Time by role

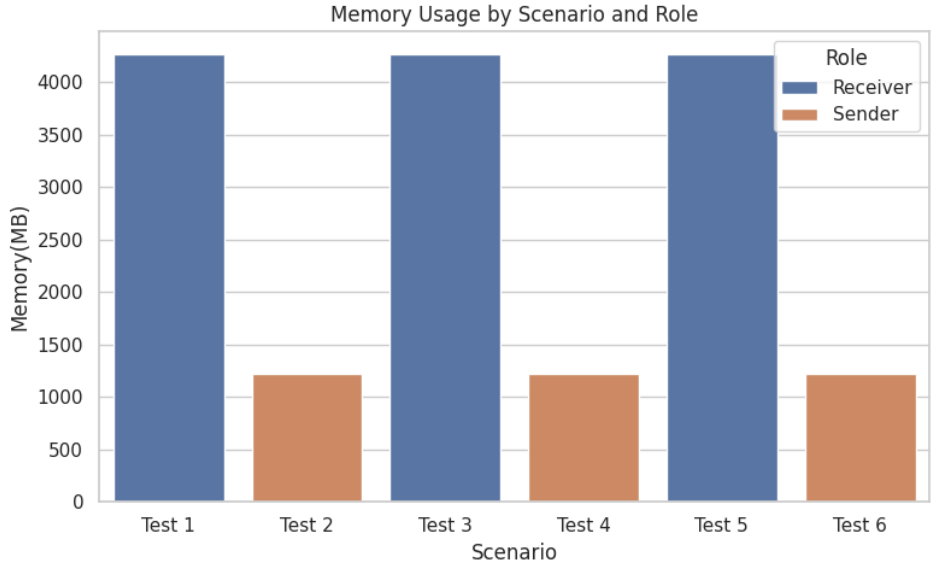

Figure 8: Memory Usage by role

Our tests demonstrate that this configuration consistently resulted in lower memory consumption on the receiver side and improved overall execution time, as visualized in Figures 7 and 8. In practical applications, this can be advantageous when the receiver operates in a constrained environment (e.g., a mobile or embedded device).

By carefully choosing the PSI protocol and optimizing data flow between the parties, we demonstrated that it is possible to achieve the goal of securely and efficiently intersecting sets in real-life scenarios, while meeting all the requirements of Step 4.