GokondaPSI: A simple P2P PSI Interface

Kevin Lu
kevin_lu2@brown.edu

Madhav Ramesh
madhav_ramesh@brown.edu

1 OVERVIEW

GokondaPSI will provide an interface for two nodes to exchange the intersection of two sets, one of them each containing a set. It will satisfy the requirements of Private Set Intersection (PSI) in the sense that no party reveals more information beyond the result of the intersection. GokondaPSI provides a CLI interface to perform PSI-CA using DDH. While we originally planned on a peer to peer design, we ended up going with a client-server design. The server maintains a local set of strings in volatile memory. The client can send a PSI request after which the PSI protocol ensues and the client can calculate the cardinality of the intersection. The only information either parties get which they did not start with is the hashed and exponentiated elements of the other party's set. This information does not reveal the elements of the other party's set in polynomial time under the DH-assumption and the random oracle model. Thus our implementation stisfies the requirements for PSI, and correctly implements PSI-CA. Communication between parties is encrypted with AES and tagged with HMAC.

2 PREREQUISITE KNOWLEDGE

The goal of PSI is for two parties to exchange the intersection of their sets without revealing information aside from the intersection. PSI-CA is a simple extension, where the cardinality of the intersection is exchanged. A common protocol to implement PSI is DDH based PSI which introduces 3-4 rounds of communication between two parties (P_1, P_2), based off of a partial implementation of PSI-sum from 3.1 of [1]

(1) Some preliminary setup involves both parties generating a random exponent k on an integer ring modulus q.
(2) P_1 hashes each element of their set then exponentiates their set with their private exponent k_1, producing $H(s_1)^{k_1}$, then sends this to P_2.
(3) P_2 hashes each element of thier set then exponentiates thier set with thier private exponent k_2, producing $H(s_2)^{k_2}$. P_2 also exponentiates the set recieved from P_1 in step 1, producing $H(s_1)^{k_1 k_2}$. P_2 then sends both sets to P_1
(4) P_1 exponentiates P_2's set, producing $H(s_2)^{k_1 k_2}$ then computes the intersection of $H(s_1)^{k_1 k_2}$ and $H(s_2)^{k_1 k_2}$
(5) PSI-CA will then just compute the size of the intersection producing the cardinality

3 DIFFICULTIES

Most of the difficulties faced were logistical in the sense that neither of us had experience with Cmake, leading to a large amount of time trying to solve build errors. The solution then was to just take our existing logic for PSI, and place it on top of an already functioning project. We chose PIR to be the foundation, and just replaced the logic inside `CloudClient::DoSend()` with P_2's logic, and `AgentClient::DoRetrive()` with $P_1's$ logic. This produces a client-server architecture as opposed to the P2P archtecture originally desired.

4 INTERFACE

4.1 Cloud

The cloud acts as P_2 in the protocl described in (2). The user can intialize the cloud client with the executable `./pir_cloud <port> <d> <d>` the `d` parameters are useless artifacts from PIR, we suggest just putting 4. Once the executable is running, the user can interact with the interface using the REPL. There's two main commands

- `insert <value>`, the user can insert values into the cloud's local set (volatile memory)
- `get`, the local set is printed to the CLI

4.2 Agent

The agent acts as P_1. The user can initialize the agent client with the executable `./pir_agent <address> <port> <d> <d>` Similarly the `d` parameters are useless but must be the same as the ones chosen for the cloud. There's one main command for the agnet REPL,

- `get <set>`, this will initate the protocol with the server, using the `<set>` given as P_1's set.

5 EXPERIMENTS

We conducted a few basic tests to check with PSI-CA was working correctly, an example is shown below with a basic set of numbers
Cloud:

```
$ ./pir_cloud 9000 4 4
$ insert 6
$ insert 5
$ insert 4
$ insert 3
$ get
6 5 4 3
```

Agent:

```
$ ./pir_agent localhost 9000 4 4
$ get 1 5 2 3$
2
```

the intersection of $\{6, 5, 4, 3\}$ and $\{1, 5, 2, 3\}$ is $\{5, 3\}$, thus the cardinality is 2. Thus our implementation produces the correct result in this case

6 REPO GUIDE

Our repository has two branches, the main branch and the asdf branch. The main branch contains our first attempt, most of the logic is in src, src-include/messages.cxx. This is an attempt at P2P PSI. The asdf bracnh contains our working implementation of PSI-CA on top of PIR.

REFERENCES

[1] "On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality", "https://eprint.iacr.org/2019/723.pdf",