intro
background
proposal/approach
results
appendices

1. intro
   (a) what this paper is
   (b) who was involved
2. background
   (a) what is cryptography?
   (b) what is secure multiparty computation?
   (c) what is the problem we are trying to solve
   (d) what is hamming distance?
   (e) what is OT? COT? (2,2)-threshold encryption
   (f) what solutions already exist (polynomial scheme)
3. proposal
   (a) basic scheme
   (b) fully secure scheme
   (c) n sets of binary strings, if $d_h(X_1, X_2)$ is 0, they are equal for intersection
4. results
   (a) implementation of basic scheme
   (b) reasons fully secure didnt work
   (c) graph1 hamming distance vs throughput
   (d) graph2 hamming distance vs time
   (e) graph3 time vs n vs set size
5. appendecies
   (a) elgamal
   (b) elgamal prover
   (c) basic
   (d) fully secure
   (e) files to generate and plot csv
      i. testing to basic
      ii. basic documentation of code
      iii. makefile
      iv. make sure all is commented
      v. gitignore gch and swp and anything else?

1 Introduction

We propose a method of securely computing set intersections among parties using a hamming distance protocols as a comparator function.

2 Background

What exactly is cryptography? Cryptography is the practice and study of secure communications in the presence of adversaries. It is using Mathematics to secure information. Cryptography is very new and also very old. Julius Caesar used to encrypt his messages he deemed of military significance using the *Caesar Cipher*, which shifted every letter over by three. The field in which Dr. Rasheed and I studied is called *Secure Mulitparty Computation.* In a given system of n players, each player P_i has a secret input x_i. The players want to compute some $f(x_1, x_2, ..., x_n)$ while revealing no information about their inputs. A real world example would be if you have three co-workers, and they want to find out who has the highest salary without revealing their salaries to each other. This means we have $n = 3$ players, and $f(x_1, x_2, x_3) = max(x_1, x_2, x_3)$. A good MPC protocol satisfies two properties:

1. Input Privacy - No information about the players inputs should be able to be inferred during the execution of the protocol. The only information that should be inferred is whatever could have been seen by seeing the output of the function alone.

2. Correctness - No player or players who may deviate from the protocol should be able to force honest parties to output an incorrect result.

The problem we are trying to solve in this research is to construct a MPC protocol such that each player P_i has an input x_i that is some finite set. and $f(x_1, x_2, ..., x_n) = \bigcap_{i=1}^{n} x_i$.

This has direct application. Consider the case where n robots wish to communicate. There are eleven frequency channels on which they may decide to send information. Not every robot can always access every channel, but they need to find out which channels they can communicate on. Each players input would be the set of frequencies that they are eligible to communicate on. In the case that a robot is comprised, no information of the other robots is compromised. Even if $n - 1$ robots are compromised, no information is revealed about the nth robot.

Hamming Distance is a measurement between two bitstrings. It represents the number of substituitions required to make two numbers identical. For example, if $X = 1001$ and $Y = 1101$, then $d_h(X, Y) = 1$. Not related to this research, hamming distances have a lot of unique properties. They form a metric space on $\{0, 1\}^n$ (also known as a hamming space). A hamming space can be represented as a Q_n graph with where the hamming distance between any two elements is the shortest walk between their representative two vertices.

They have many applications in coding theory, graph theory, cryptography, and information theory.

Oblivious Transfer (OT) is a protocol type in which a sender transfers one of many pieces of information, but is oblivious as to what piece has been transferred.

Threshold homomorphic cryptosystems are systems that in order to decrypt an encrypted message, require the work of several parties is required. If there are n parties, and atleast t of them must aid in the decryption, then we call this a (t, n)-threshold scheme. As a more layman example, consider having n parties, and some $n-1$ degree polynomial $\Sigma_{i=1}^{n-1} a_i x^i$ where a_i is an element of some finite field and $\langle a_1, a_2, ..., a_n \rangle$. is our message to decrypt (our shared secret). We give each player some unique point that is a solution to our polynomial. Clearly if they work together, by methods of Lagrange interpolation they can solve for the coefficients of our polynomial since they have n points points and the polynomial is defined as having degree $n-1$. However this requires all n parties. Even if $n-1$ parties converse, there will be many possible solutions in our field. In the real numbers, given some polynomial of degree $n-1$, and if you don't have n or more points, then there exist infinite solutions for your coefficients of your polynomial. This can be considered a (n, n)-threshold scheme, and its called Shamir's Secret sharing scheme. (CITE)

3 Proposal

We propose a different solution than in (cite polynomial guy). Given n sets, $x_1, ..., x_n$, we compute the intersections of two sets at a time. We compare each element of one set to every element of the other set. If the hamming distance of two elements compared is zero, this implies that they are identical, so they are added to a new temporary set. This temporary set is is then added to the sets to compute the intersection of. If is it the case that some $x_i \cap x_j = \emptyset$ for $i < j$, then we simply take x_i as it has higher precedence. For the secure hamming distance method, we borrow (?) two protocols from CITE and CITE. known as the basic scheme and the fully secure scheme. They are both secure against different types of adversaries. The basic scheme is secure against semi-honest (passive) adversaries. This means we can assume adversaries cooperate to gather information and do not deviate from the protocol specification. The fully secure scheme is secure against malicious (active) adversaries. In this case they may deviate from the protocol specification and attempt to cheat, as the design of the protocol ensures its a futile endeavor.

4 Results

Due to some issues, The fully secure scheme does not run with out running out of memory. This will continue to be worked on as time progresses. The basic scheme is fully implemented and we have recorded and presented some information based off its efficiency

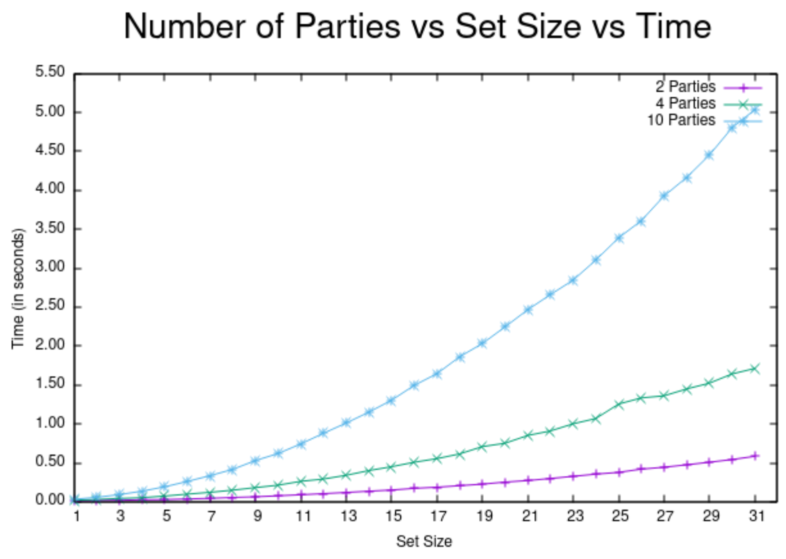

Figure 1: Time in seconds it takes to compute intersection of sets of different sizes. Shown here for 2, 4, and 10 parties

Notice that string size appears to have no difference on time, so this runs in $O(1)$ time, and the set size is a simple combinatorial algorithm in $O(n\log(n))$, nesting these we see that the SOMETHINGSOMETHING for the basic scheme is $O(n\log(n))$

5 Appendices

test of latexing a source code file

```
#include <string>
#include <gmpxx.h>
#include <gmp.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <omp.h>
#include <fstream>
#include "basic.h"
using namespace std;
/*int main(void){

    ifstream in1("input1.txt");
    ifstream in2("input2.txt");
    ofstream out1("output1.txt");

    vector<string> input1;
    vector<string> input2;

    string test;
    while(getline(in1,test)){
        input1.push_back(test);
    }
    string test2;
    while(getline(in2,test2)){
        input2.push_back(test2);
    }
    vector<string> output1;
    for(int i = 0; i < input1.size(); i++){
        for(int j = 0; j < input2.size();j++){
            vector<unsigned long> temp1 = string_to_vector(input1[i]);
            vector<unsigned long> temp2 = string_to_vector(input2[j]);

            if(basic_scheme(string_to_vector(input1[i]),string_to_vector(
                input2[j])) == 0){
                output1.push_back(input2[j]);
            }

        }
    }
    for(int k = 0; k < output1.size();k++){
        out1 << output1[k] << endl;
    }
}
*/
int main(void){
    int n = 10;
    ifstream in("inputn.txt");
    ofstream out("output.txt");
    vector<vector<string>> input(n);
    string line;
    int i = 0;
    while(getline(in,line)){
        if(line == "----"){
            i++;
        }
        else{
            input[i].push_back(line);
        }
    }
    vector<string> output;
    for(int k = 0; k < input.size() - 1; k++){
        for(int i =  0; i < input[0].size(); i++){
            for(int j = 0; j < input[1].size(); j++){
                vector<unsigned long> temp1 = string_to_vector(input[0][i]);
                vector<unsigned long> temp2 = string_to_vector(input[1][j]);
                if(basic_scheme(temp1,temp2) == 0){
                    output.push_back(input[1][j]);
                }
            }
        }
        if(output.size() != 0){
            input.push_back(output);
        }
        else{
            input.push_back(input[0]);
        }
        input.erase(input.begin(),input.begin()+1);
        output.erase(output.begin(),output.end());
    }
    for(int k = 0; k < input[1].size(); k++){
        out << input[1][k] << endl;
    }

}
```

References