Secure Multi-Party Computation Voting

Ian McClean
18idmm@queensu.ca
Queen's University
Kingston, Ontario, Canada

Elections Canada
Incorrect name or address?
Check your registration at elections.ca.
Nom ou adresse incorrects?
Vérifiez votre inscription à elections.ca.
POSTES CANADA POST
3122778

Figure 1: A ballot for an Elections Canada federal election

ABSTRACT

This paper aims to detail a procedure for Secure Multi-Party Computation Voting.

First we will introduce the motivation for this project and the guidelines for the voting procedure in section 1.

Then we will introduce the current voting procedure for Canadian federal elections, introduce the concept of Secure Multi-Party Computation, and introduce relevant literature on the topic of Secure Multi-Party Computation Voting in section 2.

Next, we will briefly discuss the tools and libraries used to make this project possible in section 3.

Then in the main portion of this paper, we will discuss the implementation of the voting procedure, as well as discussing the mathematics and computer science behind the procedure in section 4. We will also discuss potential attacks on the voting procedure.

Finally, we will make concluding remarks in section 5.

1 INTRODUCTION

The current voting process for most countries, including Canada, is to have each voter write their vote on a paper ballot before trusted ballot counters manually count each vote. This process is cumbersome and open to potential vulnerabilities. Secure Multi-Party Computation offers a solution to these problems. The goal of this project create a computer system that uses Secure Multi-Party Computation to implement an electronic voting system.

The system will be created to ensure multiple requirements, such as the integrity of each vote, the anonymity of each voter, and the inability for a voter to prove who they selected. This system will be written to allow users to simulate the entire voting process, including recording votes, tallying the votes, and presenting the final election results.

We were unable to find any other projects that have implemented Secure Multi-Party Computation during the research portion of the project.

2 BACKGROUND AND RELATED WORK

2.1 Canadian Voting Procedure

In Canada, the current voting procedure is as follows. Each voter goes to their local polling station and submits a ballot, which carries the name of the candidate they have chosen to vote for. The ballot is placed into a ballot box alongside the ballots of everyone else at the polling station. Once every voter has had a chance to vote, the ballots are tallied by election workers. The polling station then reports to then reports the total number of votes for each candidate to Elections Canada, who upon receiving the tallies from each polling station, determines the final count of how many votes each candidate received.

This procedure has multiple issues, including potential vulnerabilities. Firstly, the election process is lengthy. After the polls close it can take many hours, days or even weeks to tally all the ballots. For example, in the 2000 United States presidential election, it was over five weeks after election day that George W. Bush was declared the winner over his opponent Al Gore. [1]

Secondly, the procedure is open to mistakes and/or malicious actions. A tired ballot worker could incorrectly read a ballot and add a tally to the wrong candidate. Worse, a malicious ballot worker could intentionally add a tally for their candidate of choice.

Thirdly, the procedure exposes unnecessary information that could be used to derive secret information about voters. In Canada, voters are divided into geographical areas called ridings, and vote for their riding's Member of Parliament (MP). However, there is usually more than one polling station per riding. Consider the hypothetical scenario where a riding has multiple polling stations, a smaller one of which is assigned exclusively to voters from the small town of Murphyville. While tallying the ballots, any ballot worker at this polling station will be able to pick up on the voting trends of the people of Murphyville, something which is meant to be entirely completely anonymous.

2.2 Secure Multi-Party Computation

Secure Multi-Party Computation is a field of computer science with the goal of taking secret data held by multiple people, and computing statistics on the data without anybody having to divulge their secret information.

2.3 Average Salary Example

Consider the following example, sourced from the textbook *Applied Cryptography, Second Edition.* [6]

Suppose you are at a dinner party with your wealthy friends. You and your friends decide you'd like to know the average salary of the group. However, each attendant is shy, and would not want any person other than themselves to know their salary. Luckily, using Secure Multi-Party Computation, there is a solution to this problem.

Let N be the number of people at the party, $P_1 \dots P_N$ be the people at the party, and $S_1 \dots S_N$ be their respective salaries. In this case, the secret data each person holds is their salary, and the statistic they want to compute is the average salary.

To begin, P_1 thinks of a large, secret random number x. Grabbing a calculator, P_1 adds together $x+S_1$ in the calculator, and passes it to P_2. Upon receiving the calculator P_2 sees only the number $x+S_1$, and since he does not know x, he cannot derive any information about the salary of P_1. P_2 then adds their own salary to the calculator, resulting in the sum of $x + S_1 + S_2$. P_2 then passes the calculator to P_3, who again cannot derive any information about the salaries of P_1 or P_2. This process continues until P_N adds his salary to the calculator, leaving a sum of $x + \sum_{i=1}^{N} S_i$. Finally, the calculator is returned to P_1 who subtracts his secret number x, then divides by N to result in the final number on the calculator being the average salary, $\frac{1}{N} \sum_{i=1}^{N} S_i$.

2.4 Fear not, vote truthfully

As it turns out, voting is also a problem that can be solved with Secure Multi-Party Computation. Each person has a piece of private data, namely which candidate they wish to vote for. The goal of an election is to calculate statistics on this private data, namely the total number of votes each candidate received.

The majority of this project is based on the paper *Fear not, vote truthfully: Secure Multiparty Computation of score based rules,* [2] a paper from January 2022 that outlines an algorithm that uses Secure Multi-Party Computation to compute the winner of an election. For brevity, this paper may be referred to as FNVT below. During the research stage of the project, it became apparent that there is no publicly available implementation of the algorithm outlined in the paper.

This project implementation differs from the voting procedure set out in FNVT in a few key ways. Firstly, the FNVT algorithm can be used for many different methods of voting. For example, in one method voters are given a fixed number of votes that they may distribute as they see fit among any number of candidates. Another method has voters select either "Approve" or "Disapprove" for each candidate, and the candidate with the most "Approve" votes is selected. In this project, the voting method used is Plurality, a voting method where each voter selects one preferred candidate, and the candidate with the most selections is declared the winner. This is the voting method currently used in Canada.

Secondly, the voting procedure in FNVT ensures that the final result of the election is the relative ranking of candidates, in order from most votes to least votes. In Canada, the number of votes each candidate received is published. This information is secretive by design in the FNVT procedure. This project modifies the FNVT algorithm to ensure that the final output is the number of votes each candidate receives.

Thirdly, the FNVT procedure has a method that can determine if a voter cast an illegal ballot and disqualify their ballot from the election. This procedure is beyond the scope of this project and is not included. In section 4.5, we outline a potential vulnerability this introduces to the project.

3 TOOLS AND LIBRARIES USED

The project is written entirely using python3, using no tools other than the built-in python libraries. The project has includes some mathematical functions that are used without, such as `nextPrime(n)` which computes the smallest prime number that is less than or equal to n, and `reconstructConstant`, a function integral to the project that when given N points $P_i = (x_i, y_i)$ such that $x_i \neq x_j$ for $i \neq j$, can find $f(0)$, where `f(x)` is the unique $N-1$ degree polynomial that intersects all N points P_i. This function is explained in more detail in section 4.1.

4 IMPLEMENTATION

First, we will go over the mathematical fundamentals that make this voting procedure possible in sections 4.1 and 4.2. Then we will look at the details of the voting protocol in section 4.3. Finally, we will go over the implementation of the project in section 4.4.

4.1 Polynomial Interpolation

Suppose we have a finite field F, unique points $x_0, x_1 \dots x_N \in F$, and potentially non-unique points $y_0, y_1 \dots\ y_N \in F$.

The interpolation theorem states that there exists a unique polynomial $p \in F[x]$ of degree less than or equal to N such that $p(x_i) = y_i$ for all integers $0 \leq i \leq N$. [3]

Suppose that for some field F we have an unknown polynomial $p \in F[x]$. Suppose we also know $N+1$ points

$$\{(x_i, y_i) \in F^2\}_{i=0}^{N}$$

such that $p(x_i) = y_i$.

To derive the equation for p, we will use Lagrange polynomials, which were discovered by Euler. For the given set of points

$$\{(x_i, y_i) \in F^2\}_{i=0}^{N}$$

we define

$$l_i(x) = \prod_{\substack{0 \leq i \leq N \\ i \neq j}} \frac{x - x_j}{x_i - x_j}$$

In the above formula, since both x_i and x_j are constants, we can see that $l_i(x)$ is a polynomial of degree less than or equal to N. Also note that

$$l_i(x_m) = \begin{cases} 1 & m = i \\ 0 & m \neq i \end{cases}$$

Thus we can construct the polynomial

$$h(x) = \sum_{i=0}^{N} l_i(x) y_i$$

Since all $l_i(x)$ are polynomials of degree less than or equal to N, so too is $h(x)$. Note also that

$$h(x_i) = y_i$$

Thus by the interpolation theorem, $p(x) = h(x)$.

Thus given the $N + 1$ points

$$\{(x_i, y_i) \in F^2\}_{i=0}^{N}$$

we have a closed form to construct the unique polynomial p of degree less than or equal to N, that intersects all the given points.

If we only are interested in the constant coefficient of the polynomial, we can compute the simpler formula

$$p(0) = \sum_{i=0}^{N} l_i(0) y_i = \sum_{i=0}^{N} \prod_{\substack{0 \le i \le N \\ i \neq j}} \frac{0 - x_j}{x_i - x_j} y_i = \sum_{i=0}^{N} \prod_{\substack{0 \le i \le N \\ i \neq j}} \frac{x_j}{x_i - x_j} y_i$$

This exact formula is implemented in the project and shown in Listing 1, which uses the above formula to derive and return $p(0)$ for the unique polynomial $p \in \mathbb{Z}_p[x]$.

4.2 Secret Sharing

The basis of this voting protocol is made possible by Shamir's Secret Sharing, first discovered by Ad Shamir in 1979 [7].

Shamir outlines a protocol whereby a secret-holder, who possesses a secret $a \in F$ for a finite field F can be broken up into $D \in \mathbb{Z}^+$ shares, and can be revealed when at least $D' \in \{1 \ldots D\}$ of the shares are combined, where D and D' are chosen by beforehand by the secret-holder.

First the secret-holder selects parameters $D \in \mathbb{Z}$ and $D' \in \{1 \ldots D\}$. These two parameters, along with the field F are shared publicly.

Then, the secret-holder generates a random polynomial $p \in F[x]$ such that $\deg(p) < D'$ and $p(0) = a$.

Finally, the secret-holder generates D shares $s_d = (d, p(d))$ for $d \in \{1 \ldots D\}$.

If anyone is able to posses some D' shares, they are able to compute the secret $a = p(0)$ using the formula introduced in section 4.1.

4.3 Voting Protocol

We consider an election where there are N voters, $\vec{V} = (V_1 \ldots V_N)$ that are voting for M candidates, $\vec{C} = (C_1 \ldots C_M)$. The voting process is assisted by D talliers, $\vec{T} = (T_1 \ldots T_D)$.

Before the election, all parties must agree on parameters $D' \in \{1 \ldots D\}$, and a prime p such that $p \geq N$.

Each voter $V_n \in \vec{V}$ selects some candidate $C_r \in \vec{C}$ to vote for.

Then, for each candidate $C_m \in \vec{C}$, the voter V_n generates a random polynomial $p_{n,m} \in \mathbb{Z}_p[x]$ such that $\deg(p_{n,m}) \leq D'$. The voter must choose the constant coefficient of p such that

$$p_{n,m}(0) = \begin{cases} 1 & m = r \\ 0 & \text{otherwise} \end{cases}$$

Listing 1: An implementation of the function reconstructConstant

```
def reconstructConstant(pairsOfPoints,
    polynomialDegree, p):
    if len(pairsOfPoints) !=
        polynomialDegree + 1:
        raise ValueError("Degree␣must␣be␣one
            ␣less␣than␣number␣of␣input/
            output␣pairs")

    total = 0
    for j in range(len(pairsOfPoints)):
        yj = pairsOfPoints[j][1]

        prod = 1
        for m in range(len(pairsOfPoints)):
            if m == j:
                continue

            xm = pairsOfPoints[m][0]
            xj = pairsOfPoints[j][0]

            # Add p to the diff, since
                python does not correctly
                compute the modulus of
                negative numbers
            diff = xm - xj + p
            prod *= (xm * pow(diff, -1, p))
                % p

        total += (yj * prod) % p

    return total % p
```

Finally, to each tallier $T_d \in \vec{T}$, the voter shares a share vector

$$\vec{b}_{d,n} = \begin{pmatrix} p_{n,1}(d) \\ \vdots \\ p_{n,M}(d) \end{pmatrix}$$

Now, we move on to the job of each tallier. Each tallier $T_d \in \vec{T}$ takes their share vectors $\vec{b}_{d,n}$ and computes the tally vector

$$\vec{t}_d = \sum_{n=1}^{N} \vec{b}_{d,n}$$

and shares $\vec{t}_d$ publicly.

Note that

$$\vec{t}_d = \sum_{n=1}^{N} \vec{b}_{d,n} = \begin{pmatrix} \sum_{n=1}^{N} p_{n,1}(d) \\ \vdots \\ \sum_{n=1}^{N} p_{n,M}(d) \end{pmatrix}$$

If we look at the m^{th} index of this vector

$$(\vec{t}_d)_m = \sum_{n=1}^{N} p_{n,m}(d)$$

we can see that $s_d = (d, (\vec{t}_d)_m)$ is a share to the polynomial $p_m = \sum_{n=1}^{N} p_{n,m}$.

Finally, each voter, or anyone else who wishes to compute the results of the election, for each candidate $C_m \in \vec{C}$ can use the formula introduced in section 4.1 to calculate $p_m(0)$ using any D' shares s_d. $p_m(0)$ is the number of votes for candidate C_m.

4.4 Python Implementation

The proceeding protocol was implemented in python, and is available on GitHub. [5]

The program is broken down into three main actors, which all have their own files.

`Voter`, to represent a voter. This actor is used to create a vote-share to send to the talliers, given a candidate to vote for. It contains the function `generateBallotShareVectors()`. This function takes as input election parameters such as `numCandidates`, `numTalliers` and `p`, a prime number which determines which field $\mathbb{Z}_p$ to use. It also takes the parameter `candidateToVoteFor`. The function takes all these parameters and computes a valid vote-share for each tallier.

`Tallier`, to represent the talliers in the election. It contains the function `tally()`, a simple function that takes as input all the vote-shares given to it from the voters, and outputs the sum.

`ResultsComputer`, to represent anyone interested in computing the results of the election. It contains the function `getTotalVotes()`. This function is used to compute the results of the election, outputting a vector

$$\begin{pmatrix} c_1 \\ \vdots \\ c_M \end{pmatrix}$$

where c_m is the number of votes given to candidate t_m.

This function takes as input `tallyVectorPairs`, the pairs $(d, \vec{t}_d)$ that are published by the talliers. It also takes as input the election parameters `numCandidates`, `numTalliers`, and `p`.

The project code also contains a `tests` directory, that contains many unit tests to test the individual functions. It also contains a large integration test that ensures the entire voting process works and is able to correctly report the number of votes for each candidate. Adding unit tests to each new function accelerated development, as it ensured each individual part of the code was working, reducing debugging time.

Finally, the code contains a `mock_election` folder, used to simulate a mock election. This allows anybody to run a fully-working demo of the code. The mock election operates in a basic input loop, with options as displayed below.

```
Options:
0. Exit
1. Set election parameters.
2. Create votes.
3. Compute vote-shares.
4. Generate tallies.
5. Compute election results.
```

The options allow the user to simulate the entire election, starting with picking the parameters such as the number of voters, the number of talliers, and the number of candidates to select from.

The user can then create input votes, simulating which candidate each voter V_n selects.

Next, using the votes that were just computed, the user can simulate each voter computing their vote-share which is to be sent to the talliers.

Then the user can simulate the role of the talliers, summing all the vote-shares given to them and publishing the results.

Finally, the user can use the summation of the vote-shares to compute the results of the election, which are displayed to the user in vector form.

For each option, the outputs are saved to a file so the user may view the results of each intermediate step. Additionally, this allows a user to use their own data to compute later steps. For example, a user, who does not have the original list of votes, could give the program a list of vote-shares, which the program could then use to compute the winner of the election.

4.5 Attacks

We will outline a number of potential attacks and how our protocol mitigates them.

The first possible attack is that a tallier T_d may choose to act maliciously and instead of accurately tallying the share vectors, can generate it's own share-vectors and effectively cast all their own ballots. However, since computing the results of the election only requires D' of the talliers, it will become apparent that the results of the election are different when including T_d in the tallying vs. excluding them. If the results of the voting are different based on which talliers are used to compute the final sum, a voter can be sure that a tallier has either made a mistake or maliciously modified the ballots. A voter may even determine which tallier has tampered with the vote, by computing the final tally using many different subsets of talliers. If a voter notices that including or excluding a tallier affects the final vote, they can be sure that that specific tallier has made a mistake or maliciously impacted the vote.

In this way, the only way to tamper with the election is if all D talliers collude together. The risk of this happening decreases as we can increase the parameter D as needed before the election.

Another possible attack is for talliers to collude to discover which candidate a particular voter selected. If D' talliers collude to conspire against a voter $V_n \in \vec{V}$, they can combine D' shares $\vec{b}_{d,n}$, and compute the secret of who V_n voted for. Again, the risk of this happening decreases as we can increase the parameter D' as needed before the election.

One final attack is that a malicious voter may create an invalid polynomial. Suppose that voter $V_n \in \vec{V}$ wants to maliciously affect the outcome of the election to support candidate $C_r \in \vec{C}$. The voter may create an invalid polynomial such that

$$p_{n,m}(0) = \begin{cases} q & m = r \\ 0 & \text{otherwise} \end{cases}$$

and in this way effectively cast q votes for candidate C_r.

The talliers can all agree on the number of voters that participated in the election, and can publish this information. Thus upon calculating the final results of the election, any tallier or voter will notice that the total number of votes does not equal the total number of voters, and realize there is a problem. This makes it so that a malicious voter will be detected and the election can be deemed invalid.

However, this still makes it so that any voter that wishes to sabotage the election can make it so that a valid election can never be held. They can simply cast an invalid ballot every time the election is held, making it so every election is invalid, and the entire election procedure is blocked. As it turns out the previously mentioned paper *Fear not, vote truthfully: Secure Multiparty Computation of score based rules,* [2] outlines a method where in the case of an invalid election, the talliers can determine exactly who the malicious voter is and redo the tallying process, excluding the malicious voters ballot. However, this is beyond the scope of this particular project.

5 CONCLUDING REMARKS

The paper *Fear not, vote truthfully: Secure Multiparty Computation of score based rules* mentions that there is great difficulty in implementing any such protocol in real governmental elections. [2] The main issue lies with convincing legislators and voters of the benefits of the protocol as opposed to a regular paper ballot voting scheme. The main benefit of the paper ballot scheme is that all voters are able to easily understand how the process works, and can be convinced that their vote will be counted fairly, accurately and securely. This protocol offers no such benefit. There is a great deal of requisite knowledge in mathematics and computer science required to understand the proofs set out in this paper. If the average voter or legislator will be unable to understand that this voting scheme is secure and produces the correct result, they will likely be unwilling to adopt it.

Perhaps a better use case for this voting procedure is when large organizations rather than people have to vote. Certainly a large organization contains the necessary talent such that somebody on the team will be able to understand the voting method. For example, the London Inter-Bank Offered Rate, or LIBOR, is an interest rate that global banks use to lend money to each other. [4] LIBOR is determined each day by major banks voting on their preferred interest rate. The voting procedure implemented in this project seems a perfect fit for this use-case. Knowing the preferred interest rate of a bank can signal that bank's economic outlook, meaning it is important that other banks do not know how any individual bank voted. Additionally, the global nature of the voting, and the need to run a new election each day makes a digital election scheme necessary.

This project has been incredibly rewarding to work on. Research of this project has revealed many different exciting use cases for Secure Multi-Party Computation Voting. This field is in its infancy; the main paper that this project was based on was only written in January 2022. [2] The future of Secure Multi-Party Online voting will be exciting and fruitful as new discoveries and implementations are made.

REFERENCES

[1] 2010. Congress certifies George W. Bush winner of 2000 elections. https://www.history.com/this-day-in-history/congress-certifies-bush-winner-of-2000-elections

[2] Lihi Dery, Tamir Tassa, and Avishay Yanai. 2021. Fear not, vote truthfully: Secure Multiparty Computation of score based rules. *Expert Systems with Applications* 168 (2021), 114434. https://doi.org/10.1016/j.eswa.2020.114434

[3] Jeffrey Humpherys and Tyler J Jarvis. 2020. *Foundations of applied mathematics, volume 2.* Society for Industrial & Applied Mathematics, New York, NY.

[4] Julia Kagan. 2022. Libor: What the London interbank offered rate is, how it's used. https://www.investopedia.com/terms/l/libor.asp

[5] Ian McClean. 2022. *Secure Multi-Party Computation Voting.*

[6] Bruce Schneier. 1995. *Applied cryptography* (2 ed.). John Wiley & Sons, Nashville, TN.

[7] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22 (1979), 612–613.

Received 9 December 2022