Threshold voting protocol

Vlada Petrusenko, Liza Horokh, Pranjal Srivastava, Nyx Theos Haile

May 13, 2025

Abstract

In this paper, we describe a secure threshold voting protocol. Our protocol only reveals whether the candidate got a threshold k number of votes, without revealing an exact number of votes and preserving voters' anonymity. As a first step, we authorize all the users as valid ones by a decentralized endorsement system, which prevents parties to generate a lot of fake identities. In further steps, we assume that adversaries are honest, but curious, and at least two voters are not malicious. The described algorithm is secure under the DDH assumption against a poly-time adversary. The protocol does not require communication between voters, but assumes the existence of a public server and authorized access to data on it. Complexity of voting protocol is $O(n^2 \cdot k)$, where n is the number of voters and k is the threshold. Complexity is comparable to the alternative schemes like BGW, but does not require pairwise communication.

1 Introduction

Electronic voting has gotten more common in various settings - from decentralized blockchain applications (e.g. DAO - Decentralized Autonomous Organization), to national elections. This popularity of electronic voting applications dictates a set of accuracy, security and privacy expectations. While a balance between them is often a tradeoff [2], both are immensely important in the context of elections.
The goal of this project is to design a protocol that would provide a layer of anonymity while being able to successfully determine the result of the election. More specifically, this project shows a protocol, that given some threshold k returns solely the information about whether the candidate got more or less than k votes, without revealing the actual number. This is useful in context when one wants to provide privacy to the candidates and only reveal the purpose of the elections - who is the winner - not what the vote difference was. Additionally, this scheme leaks absolutely no information about the voters' choice, which is an important consideration, that guarantees the voters that their vote will not be used against them. To reiterate, our protocol achieves fairness of the election, while maintaining anonymity objectives.

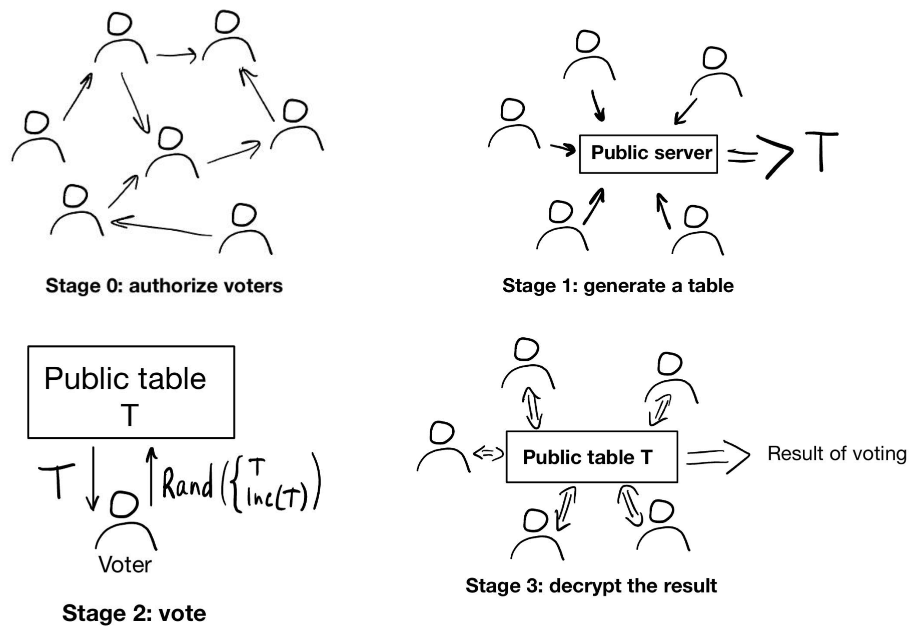

Figure 1: Steps of the protocol

As per our construction, we assume parties which are honest but curious, however, could generate fake accounts. We show how to filter out fake accounts in the Decentralized Authentication step, and afterwards - how to compute Threshold Voting result without pairwise communication. This construction is secure under the Decisional Diffie-Hellman (DDH) assumption against a polynomial-time adversary.

Our protocol can be roughly split into four stages (see Figure 1).

- As a stage 0, we need to authorize voters which have real identities. This includes assuming that there isn't a trusted authentication authority, and identifying parties that are generating fake identities.
- For stage 1, we assume that all remaining participants are honest but curious, and initialize a voting table. The table looks random to an arbitrary observer and encodes initial state of voting. It is hosted on a public server that is trusted to host encrypted data.
- For stage 2, each voter updates the table independently. Two voters cannot vote at the same time, and there is a public track of table history (which doesn't reveal any information about who voted for what). Each

voter can either do operation ”Vote(T)” or ”Not Vote(T)” and updated table looks indistinguishable to any observer.

- For the last stage, all voters collaborate to decrypt the resulting value. That being said, they don’t need to communicate to each other, they just have to decrypt their shares and post them publicly.

The remainder of this paper expands on the specifics of implementation of each of these steps, along with security analysis and performance evaluation.

2 Background and Prior Work

2.1 ElGamal

ElGamal protocol is an asymmetric encryption algorithm that secures messages using a public-private key pair and randomness, making it hard to decrypt without the private key. It relies on the difficulty of the discrete logarithm problem for security.[1]

2.2 BGW

A protocol used to carry out multiparty computations. We have used it as a basic comparison for our scheme as it is the most used multiparty computation scheme. Additive shares provide the same complexity.

2.3 Additive Shares

In multi-party-communication, we say that n people hold an additive share of some value v if they each have some personal value (usually secret), such that the sum of values is v.

2.4 LHE

A linearly holomorphic encryption scheme is one where one can perform linear operations on ciphertexts. For our purposes, we chose pseudo-encryption with linear properties, but for improving security against quantum adversaries, we can base our security on the linear property of LWE public key encryption schemes. It might decrease performance because of the range cipher text size, but would improve the hardness of the assumption.

2.5 Min-node cut

The minimum node cut of a graph is the smallest set of nodes whose removal disconnects the graph. In cryptographic protocols or network security contexts, this concept helps analyze communication bottlenecks or adversarial resilience. The NetworkX library provides an implementation of this algorithm, useful for

evaluating connectivity properties in graph-based models of distributed protocols. [3]
This is a useful idea that we are using for the authentication step of the algorithm.

3 Authentication Protocol

Probabilistic Trust Propagation under Adversarial Mislabeling in Sparse Random Graphs

Introduction

To address the challenge of distributed voting we must first address the challenge of distributed authentication over a permissionless network. In this setting, honest parties must discover all other honest parties, prune out malicious parties, and reach a consensus on the final honest set. We assume that the adversaries are computationally bounded, but they have the ability to generate false identities (minions), and forge connection tests.
Our approach utilises properties of sparse random graphs along with a probabilistic (potentially zero-knowledge) verification function triplet to label edges and propagate along trust paths. We show that in $O(\log n)$ rounds with $O(\log n)$ pairwise communication, honest parties converge w.h.p., on the honest core when the adversary controls up to $1 - 1/2c$ nodes.

3.1 Definitions and assumptions

We assume the existence of some verification function triple

$$C(j) \to \rho, \quad R(j, \rho) \to r, \quad V(j, \rho, r) \to \{0, 1\}$$

such that given two voters i and j, i issues a challenge to j, and

$$\Pr[V(j, \rho, R(j, C(j))) = 1] = 1$$

when j is honest (irrespective of i) and

$$\Pr[V(j, \rho, R(j, C(j))) = 1] = \varepsilon$$

when j is is not a legitimate party. No repetition is possible, the result of a connection test on i, j is constant.
We refer to this function triple as a connection test.
Each voter has the ability to sign and verify signed messages with a known public/private keypair.
We model the voters as a graph, where nodes represent voters and each edge represents a connection attempt between two voters.
Finally, we define two kinds of adversaries:

- **Impostors** are malicious privileged actors. They can perform valid authentications under C, R, V and are indistinguishable from honest actors by any isolated test. Impostors are allowed to perform any actions maliciously at any time.

- **Minions** are malicious nodes created by impostors. Unlike adversaries, they do not have valid keys, and thus only pass the verification with probability ε. Among other minions, they may fail or pass at their discretion. They also have the freedom to perform any actions maliciously, such as dropping messages or falsely accusing any other nodes.

3.2 Minion Detection via Accusation Voting

The protocol proceeds in asynchronous rounds, which we model as an interval over which every honest node performs a constant number of actions. As such, it is round-agnostic, but we use the round as measure of progress. In each round, every honest node i performs the following actions:

1. i samples $c \log n$ nodes from the network (without replacement). This sampling takes place over the protocol, so nodes will not be replaced even between rounds.

2. For each chosen j, i initiates the connection test and sets the result to be w_{ij} Similarly, j performs the same test with i. Thus, each contact yields a potentially asymmetric verdict for both endpoints.

3. i broadcasts a signed copy of w_{ij} and their connection status.

Over the rounds, i collects enough information to identify whether there exists at least one node-disjoint trusted-path from i to j. Any node j for which $T_i(j) = 0$ is declared untrusted, otherwise j is temporarily trusted.
Among the trusted set, honest nodes are a connected component. However, since minions cannot pass tests on any honest-honest link, all true honest nodes are mutually reachable by trusted paths. Any impostor must be isolated from this component on at least one cut, due to false accusations, and thus they will be untrusted by some honest node. By exchanging connected-component labels, the honest nodes reach consensus on the honest core.

3.3 Security

When the adversary controls at most $1 - \frac{1}{2c}$ nodes, where c is the parameter used in node sampling, honest nodes will still have at least one disjoint honest-only path to each other. Since w_{ij} is locally determined by the connection test and verified by signatures, no node can falsify a testimony from another node.
Impostors, even if they pass all tests on some edges, cannot cross every cut in the honest component, so they eventually fail $T_i(j) = 1$.

3.4 Runtime

Both the full graph and the honest subgraph are Erdős–Rényi graphs. The full graph has average degree $\Theta(\log n)$, diameter $O(\frac{\log n}{\log \log n})$ w.h.p., and $\Theta(\log n)$ edge-disjoint path between random pairs.

Thus, w.h.p., the discovery phase propagates through the graph in $O(\log n)$ rounds.

3.5 Future Work

We hope to present a formalised version of this work in the future :P

4 Formal objectives of the voting protocol

The scheme consists of an algorithm

- Let's say that total number of voters is n and threshold is k, λ is a security parameter
- $S_v(\lambda) \rightarrow (s_i, pk_i)$
  generator of the voters' public/secret keys pair, run by voters individually
- $S(\lambda, n, k) \rightarrow A$
  protocol to each party to generate the part of the public table and combine them
- $Inc(A) \rightarrow A'$
  protocol that increments the total number of votes
- $Rand(A) \rightarrow A'$
  protocol that re-randomizes the table without changing the number of votes
- $Eval(A) \rightarrow \{0, 1\}$
  protocol for each player to decrypt part of the table and combine them to return 1 $\iff$ # of votes $\geq$ k

4.1 Correctness

$Eval(V) = 1 \iff$ candidate C received $\geq k$ votes.

4.2 Security

- mask $mal = (m_1, \dots m_n) : m_i = 1 \iff$ voter i is malicious
- $v^a = (v_1^a \dots v_n^a), v_i^a = 1 \iff$ voter i voted for the candidate and $\sum_{i=1}^n v_i^a = a$
- $I(v^x)$ is all the information available to malicious parties
- Then $I(v^a) \cong I(v^b), \forall v^a, v^b : a, b < k$ and $v^a \vee mal = v^b \vee mal$
- Optional: $I(v^a) \cong I(v^b), \forall v^a, v^b : a, b \geq k$ and $v^a \vee mal = v^b \vee mal$

4.3 Security assumptions

Decisional Diffie–Hellman (DDH), polynomial time honest, but curious adversary, at least two honest voters.

5 Voting Protocol

5.1 Setup

To construct the voting protocol, we use chosen fixed threshold k and public parameters p, g, such that p is a large prime number and g is a root element from $\mathbb{Z}_p$. To guarantee perfect correctness and simplify some calculations, we will choose $p = 2 \cdot q + 1$ for some other large prime q.
Additionally, we define a polynomial T, such that:

$$T_0(x) = r \cdot x(x-1)...(x-k+1)$$

for some randomness r. If we want to reveal a number of votes above the threshold, set $r = 1$. Steps that are required to hide a number of votes above the threshold will be marked as Op.
And for $0 < i \leq k$, T is defined recursively as

$$T_i(x) = T_{i-1}(x+1) - T_{i-1}(x) \tag{1}$$

Note that $T_0(x) = 0$ if $0 \leq x < k$ and $T_0 \neq 0$ otherwise.

5.2 Initialization

5.2.1 Generating shares, $S_v(\lambda)$

Let's generate each voters' private and public key pair as following:
$S_v(\lambda) = (sk, pk)$, where sk is a randomly chosen number $1 \leq sk \leq p-2$ and $pk = g^{sk} \mod p$.

Those keys will be used for the pseudo-encryption scheme

$$Enc(sk, m) = g^{m \cdot sk} \mod p$$

$$Dec(sk, c = g^{sk \cdot m}) = c^{\frac{p}{sk} \mod (p-1)} \mod p = g^m \mod p$$

Let's notice that our pseudo-encryption scheme has following properties:

$$Enc(sk, m_1 + m_2) = Enc(sk, m_1) \cdot Enc(sk, m_2) \mod p \tag{2}$$

$$Enc(sk, c \cdot m_1) = Enc(sk, m_1)^c \mod p \tag{3}$$

$$Enc(sk, m_1 - m_2) = Enc(sk, m_1) \cdot Enc(sk, m_2)^{p-2} \mod p \tag{4}$$

5.2.2 Combining shares, $S(\lambda, n, k)$

To initialize voting for n voters, each uses their secret key sk_i to encrypt starting shares $B_{i,j} = \frac{1}{n} T_j(0)$ for $j = 0 \ldots k$. Each party then posts their encrypted shares to public server to form a public table A:

$$A = \begin{bmatrix} Enc(sk_1, \frac{1}{n}T_0(0)) & Enc(sk_1, \frac{1}{n}T_1(0)) & \dots & Enc(sk_1, \frac{1}{n}T_k(0)) \\ Enc(sk_2, \frac{1}{n}T_0(0)) & Enc(sk_2, \frac{1}{n}T_1(0)) & \dots & Enc(sk_2, \frac{1}{n}T_k(0)) \\ \vdots & \vdots & \ddots & \vdots \\ Enc(sk_n, \frac{1}{n}T_0(0)) & Enc(sk_n, \frac{1}{n}T_1(0)) & \dots & Enc(sk_n, \frac{1}{n}T_k(0)) \end{bmatrix} =$$

$$= \begin{bmatrix} g^{sk_1 \cdot \frac{1}{n}T_0(0)} & g^{sk_1 \cdot \frac{1}{n}T_1(0)} & \dots & g^{sk_1 \cdot \frac{1}{n}T_k(0)} \\ g^{sk_2 \cdot \frac{1}{n}T_0(0)} & g^{sk_2 \cdot \frac{1}{n}T_1(0)} & \dots & g^{sk_2 \cdot \frac{1}{n}T_k(0)} \\ \vdots & \vdots & \ddots & \vdots \\ g^{sk_n \cdot \frac{1}{n}T_0(0)} & g^{sk_n \cdot \frac{1}{n}T_1(0)} & \dots & g^{sk_n \cdot \frac{1}{n}T_k(0)} \end{bmatrix}$$

Every column i represents $T_i(cur\_num\_votes)$, and since initially there were 0 votes, during initialization it represents $T_i(0)$.

Then, at any point of the voting process A stores encoded values of $T_j(x = current\_number\_of\_votes)$ as follows:

$$A = \begin{bmatrix} Encoding(T_0(x)), ..., Encoding(T_k(x)) \end{bmatrix} = \begin{bmatrix} g^{sk_1 \cdot B_{1,0}} & g^{sk_1 \cdot B_{1,1}} & \dots & g^{sk_1 \cdot B_{1,k}} \\ g^{sk_2 \cdot B_{2,0}} & g^{sk_2 \cdot B_{2,1}} & \dots & g^{sk_2 \cdot B_{2,k}} \\ \vdots & \vdots & \ddots & \vdots \\ g^{sk_n \cdot B_{n,0}} & g^{sk_n \cdot B_{n,1}} & \dots & g^{sk_n \cdot B_{n,k}} \end{bmatrix}$$

such that $\sum_{i=1}^{n} B_{i,j} = T_j(x)$.

5.3 Voting process

When the person wants to vote, they go one of the two trajectories, depending on whether they want to vote for or against.
If they want to vote for, they transform the table $A \to Rand(Inc(A))$, and if they want to vote against, they transform the table $A \to Rand(A)$. Increment keeps track of the correct vote count, and Re-randomization ensures that nobody can know what was the vote - it looks random and completely indistinguishable to any other party.

5.4 Incrementing the vote, $Inc(A)$

$Inc(A)$ protocol: if the previous number of votes is v, modify table A such that it represents $T_j(v+1)$. To do that, we notice that by recursive property 1 to update value T_j we can calculate: $T_j(v+1) = T_j(v) + T_{j+1}(v)$ (let us notice that T_k is constant and does not need to be updated). Using 2 of the pseudo-encryption used, we can get $A'_{i,j} = Enc(sk_i, B'_{i,j}) = A_{i,j+1} \cdot A_{i,j}$. The property of the shares is still preserved since $\sum_i B'_{i,j} = \sum_i B_{i,j} + B_{i,j+1} = T_j(v) + T_{j+1}(v) = T_j(v+1)$.

To summarize:

$$A' = \begin{bmatrix} A_{1,0} \cdot A_{1,1} & A_{1,1} \cdot A_{1,2} & \dots & A_{1,k} \\ A_{2,0} \cdot A_{2,1} & A_{2,1} \cdot A_{2,2} & \dots & A_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,0} \cdot A_{n,1} & A_{n,1} \cdot A_{n,2} & \dots & A_{n,k} \end{bmatrix}$$

$$= \begin{bmatrix} g^{sk_1(B_{1,0}+B_{1,1})} & g^{sk_1(B_{1,1}+B_{1,2})} & \dots & g^{sk_1 B_{1,k}} \\ g^{sk_2(B_{2,0}+B_{2,1})} & g^{sk_2(B_{2,1}+B_{2,2})} & \dots & g^{sk_2 B_{2,k}} \\ \vdots & \vdots & \ddots & \vdots \\ g^{sk_n(B_{n,0}+B_{n,1})} & g^{sk_n(B_{n,1}+B_{n,2})} & ... & g^{sk_n B_{n,k}} \end{bmatrix}$$

Note that any voter doesn't know the current vote count v, but knows that combining columns would increment the vote count by exactly 1.

5.5 Re-randomization, $Rand(A)$

Then, regardless of whether the user chose to increment the vote or not, they need to re-randomize the table after voting to hide the voting process. Assuming that the state of the table is:

$$A = \begin{bmatrix} Enc(sk_1, B_{1,0}) & Enc(sk_1, B_{1,1}) & \dots & Enc(sk_1, B_{1,k}) \\ Enc(sk_2, B_{2,0}) & Enc(sk_2, B_{2,1}) & \dots & Enc(sk_2, B_{2,k}) \\ \vdots & \vdots & \ddots & \vdots \\ Enc(sk_n, B_{n,0}) & Enc(sk_n, B_{n,1}) & \dots & Enc(sk_n, B_{n,k}) \end{bmatrix}$$

Generate random values $c_{i,j}, i = 1 \dots n, j = 0 \dots k$ such that $\sum_{i=1}^{n} c_{i,j} = 0$. Let's notice that the only requirement for A being a valid encoding of T is that the sum of the shares $B_{i,j}$ in the column equals the corresponding value of the required polynomial. This means that if $(B_{i,j}, i = 1 \dots n, j = 0 \dots k)$ is valid set of shares, then set $(B_{i,j} + c_{i,j}, i = 1 \dots n, j = 0 \dots k)$ is also a valid representation. This means that we can re-randomize the table using property 3 of our encryption scheme, since $pk_i = Enc(sk_i, 1)$ we can generate $Enc(sk_i, c_{i,j}) = pk_i^{c_{i,j}}$ and then using property 2 update share $A'_{i,j} = Enc(sk_i, B_{i,j} + c_{i,j}) = Enc(sk_i, B_{i,j}) \cdot Enc(sk_i, c_{i,j}) = A_{i,j} \cdot pk_i^{c_{i,j}}$. So after re-randomization we will obtain matrix A':

$$A' = \begin{bmatrix} A_{1,0} \cdot pk_1^{c_{1,0}} & A_{1,1} \cdot pk_1^{c_{1,1}} & \dots & A_{1,k} \cdot pk_1^{c_{1,k}} \\ A_{2,0} \cdot pk_2^{c_{2,0}} & A_{2,1} \cdot pk_2^{c_{2,1}} & \dots & A_{2,k} \cdot pk_2^{c_{2,k}} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,0} \cdot pk_n^{c_{n,0}} & A_{n,1} \cdot pk_n^{c_{n,1}} & \dots & A_{n,k} \cdot pk_n^{c_{n,k}} \end{bmatrix}$$

$$= \begin{bmatrix} g^{sk_1(B_{1,0}+c_{1,0})} & g^{sk_1(B_{1,1}+c_{1,1})} & \dots & g^{sk_1(B_{1,k}+c_{1,k})} \\ g^{sk_2(B_{2,0}+c_{2,0})} & g^{sk_2(B_{2,1}+c_{2,1})} & \dots & g^{sk_2(B_{2,k}+c_{2,k})} \\ \vdots & \vdots & \ddots & \vdots \\ g^{sk_n(B_{n,0}+c_{n,0})} & g^{sk_n(B_{n,1}+c_{n,1})} & \dots & g^{sk_n(B_{n,k}+c_{n,k})} \end{bmatrix}$$

Op: if we want to provide an optional security guarantee 4.2, on the randomization step, each voter chooses a random number r and multiplies all values in the table by r using property 3 of pseudo-encryption. Then the table would be encoding of $r \cdot T_0(v), r \cdot T_1(v) \ldots r \cdot T_k(v)$. This does not change the Inc function property and decryption of results. By introducing randomness, we ensure that $T(v)$ does not reveal v for $v \geq k$.

5.6 Calculating the result of voting, $Eval(A)$

Let's notice that after each person submitted their vote and re-randomize the table, if total number of votes for candidate is v, then the final state of the table is:

$$A = [Encoding(T_0(v)), Encoding(T_1(v)), \ldots Encoding(T_k(v))]$$

$$= \begin{bmatrix} Enc(sk_1, B_{1,0}) & Enc(sk_1, B_{1,1}) & \ldots & Enc(sk_1, B_{1,k}) \\ Enc(sk_2, B_{2,0}) & Enc(sk_2, B_{2,1}) & \ldots & Enc(sk_2, B_{2,k}) \\ \vdots & \vdots & \ddots & \vdots \\ Enc(sk_n, B_{n,0}) & Enc(sk_n, B_{n,1}) & \ldots & Enc(sk_n, B_{n,k}) \end{bmatrix}$$

such that $\sum_{i=1}^{n} B_{i,0} = T_0(v)$. So to get the result of the voting, each voter i publishes share equal to preudo-decryption 5.2.1 of $A_{i,0}$ $s_i = Dec(sk_i, A_{i,0}) = Dec(sk_i, Enc(sk_i, B_{i,0})) = g^{B_{i,0}}$.
After we got share of each voter, we calculate $r = \prod_{i=1}^{n} s_i = g^{\sum_{i=1}^{n} B_{i,0}} = g^{T_0(v)}$. By construction of 5.1 p, g and T_0, $T_0(v)$ does not have divisors greater than v, so $T_0(v) \neq 0 \rightarrow p - 1 = 2 \cdot q \nmid T_0(v) \rightarrow r = 1 \iff T_0(v) = 0 \iff v < k$, so result of $Eval(A)$ is $r == 1?$.

6 Analysis and Evaluation

6.1 Security

We will first prove security guarantees in the setting where the adversary which controls $t \leq n - 2$ honest-but-curious parties is computationally bounded and cannot solve discrete-log type problems. We will then describe how the algorithm can be tweaked such that the protocol is secure even if the adversary has access to a quantum computer. We will also achieve security with abort if the adversarial parties are malicious and do not respond as per protocol.

6.1.1 Honest-But-Curious non-quantum adversary

The first key idea behind the proof is that to an $t < n - 1$ collaborating parties, a column in table A looks computationally indistinguishable from random conditioned on Decisional-Diffie-Hellman and hence leaks no information. The other key idea is any two columns (either in the same table or across tables) look independent of each other due to the re-randomization step. This means that even with access to all data, $t < n - 2$ parties will not gain any information about the voting process.

We now formalize both of these steps. We will first show that to $t < n-2$ parties, a column looks indistinguishable from random. For convenience, we provide a few definitions before formalizing what this means.

For any choice of secrets $sk = (sk_1, \cdots, sk_n)$, let

$$\text{exp-enc}(sk) := (g^{sk_1}, \cdots, g^{sk_n})$$

For some choice of secrets sk and column $C = \left[g^{sk_1 b_1}, \cdots, g^{sk_n b_n}\right]$, we define

$$\text{rep}(C) = b_1 + \cdots + b_n.$$

Informally $\text{rep}(sk, C)$ is the value the entire column encrypts. Define D to be the uniform distribution over all possible pairs $(\text{exp-enc}(sk), C)$, and let D_i be the uniform distribution over pairs $(\text{exp-enc}(sk), C)$ such that $\text{rep}(sk, C) = i$. Our claim is that even with an oracle which responds with t different elements of sk, we have $D_b \overset{c}{\cong} D$.

Assume for the sake of contradiction that this is false. Since additional oracle access cannot harm the adversary, assume that the adversary gains access to all of the last $n-2$ elements of sk. We need to show that

$$\text{Unif}(\{(g^{sk_1}, g^{sk_2}), [g^{sk_1 b_1}, g^{sk_2 b_2}] \mid b_1 + b_2 = b\}) \overset{c}{\cong} \text{Unif}(\{(g^{sk_1}, g^{sk_2}), [g^{sk_1 b_1}, g^{sk_2 b_2}]\})$$

The result for $b = 0$ will imply the result for all b as the adversary can multiply $g^{sk_2 b_2}$ by $g^{-sk_2 b}$ to reduce the general case to the special case $b = 0$. However, the special case for $b = 0$ requires the adversary to ascertain with non-negligible advantage over random whether some tuple (g^a, g^b, g^c, g^d) satisfies $ad = bc$ which is known to be equivalent to Differential-Diffie-Hellman (DDH). This concludes the proof of the first key-idea, that a column does not leak any information about the value it encrypts.

We now show the second key idea, that any two revealed columns are mostly independent. More formally, we show that for any column C in column A, such that $\text{rep}(sk, C) = e$, C is uniformly distributed among columns such that $\text{rep}(sk, C) = e$. The reason behind this is due to the re-randomization step. The values $(B_{1,j}, B_{2,j}, \cdots, B_{n,j})$ are an additive share of $\sum_i B_{i,j}$. We know that when a uniformly random additive share of 0 is added to any given additive share, the additive share gets randomized. Thus, after re-randomization step, the column is randomized.

We now prove a corollary that will help us finish. Consider $l = \text{poly}(n)$ and tuple $(e_1, \cdots, e_l) \in F_p^l$. For brevity, let $ske = \text{exp-enc}(sk)$. We claim

$$\text{Unif}(\{ske, C_1, C_2, \ldots, C_l\} \mid \forall i, \text{rep}(ske, C_i) = e_i) \overset{c}{\cong} \text{Unif}(\{ske, C_1, C_2, \ldots, C_l\})$$

This corollary follows from a standard hybrid argument, but for completeness we quickly outline a proof. Consider the intermediate distributions $E_{l'} = \text{Unif}(\{ske, C_1, C_2, \ldots, C_l\} \mid \forall i < l', \text{rep}(sk, C_i) = e_i$ By the triangle inequality, that the adversarial advantage between the distributions E_0 and E_l in our corollary is at most l times the maximum adversarial advantage between $E_{l'}$ and $E_{l'+1}$. From $D_{e_{l'+1}} \overset{c}{\cong} D$, we can show the adverary gains at most poly(n)negl(n) advantage between E_0 and E_l, or that the two distributions are computationally equivalent.

Finally we show that for any possible set of votes by non-adversarial parties which would result in the same value of T_0(#total votes), the adversary cannot distinguish the view it observes from any other set of votes by non-adversarial parties. Suppose for contradiction that the adversary could do so. Suppose these two sets of votes are $v_1, \ldots, v_n$ and $w_1, \ldots, w_n$. Let v'_i and w'_i represent the partial sums $v_1 + \cdots + v_i$ and $w_1 + \cdots + w_i$. The adversary can apparently with non-negligible advantage distinguish between a set of nk uniformly random columns representing $T_i(v'_j)$ and a set of nk uniformly random columns representing the $T_i(w'_j)$. However, this is a direct contradiction to our corollary!

This proves that apart from the value of T_0(#total votes) which is revealed at the end. No other information is revealed as desired.

6.1.2 Quantum Adversaries

The protocol as specified is not secure against quantum adversaries, as quantum adversaries are capable of solving discrete-log and DDH. However the key idea is that instead of using $\text{Enc}(sk_i, b_i) = g^{sk_i b_i}$, we can use any other linearly homomorphic encryption scheme for which DDH is quantum-secure.

6.1.3 Malicious Parties

We also outline how the protocol can be modified to achieve security against malicious parties whose communication does not follow the protocol. For this, we require that each apart from communicating the new value of table A' given table A, each party also provide a zero-knowledge proof that either $A' \in \mathit{Rand}(A)$ or $A' \in \mathit{Inc}(\mathit{Rand}(A))$. One downside of this approach, however, is that the addition of zero-knowledge proofs decreases the speed by a large constant factor.

6.2 Complexity and Comparision with BGW

Note that our voting protocol essentially is a MPC to compute the function

$$f(v_1, \ldots, v_n) = \begin{cases} 0 & \text{if } \sum v_i < k \\ 1 & \text{if } \sum v_i \geq k \end{cases}$$

We know that the BGW algorithm provides one way to compute the above function. We compare the performance of our algorithm against BGW in terms of security as well as time and communication complexity.

6.2.1 Security

The BGW algorithm provides statistical security. Even a computationally unbounded adversary is unable to gain information about the actions of other parties. In contrast, our algorithm only functions against computationally bounded adversaries. However, this is a reasonable safety assumption in practice.

The main security advantage of our algorithm is that it is secure against $n-2$ colluding parties, whereas BGW is only safe against $n/2$ honest-but-curious colluding parties, and $n/3$ malicious colluding parties.

6.2.2 Time and Communication Complexity

For simplicity, we consider operations in F_p as taking $O(1)$ time and elements of F_p as taking $O(1)$ space. This keeps the comparision fair, as both our algorithm and BGW rely on computations done in F_p. The smallest circuit that can compute whether the sum of votes is at most k seems to require at least k AND gates. The simplest BGW circuit we could find which calculates this function uses k AND gates. While circuit lower bounds are hard to prove, there does not seem to be any quicker approach. This means that BGW computation of the fucntion f will have total time and communication complexity of $O(n^\omega k)$ and $O(n^2 k)$ each respectively, where ω is the matrix-multiplication constant.

Our algorithm requires n consecutive steps, each with $O(nk)$ operations, so our algorithm has a total time and communication complexity of $O(n^2 k)$ respectively. Thus, in terms of raw computation power required, our algorithm slightly out-performs BGW.

Another important comparision is that our algorithm does not require pairwise communication between voters, whereas BGW requires secure pairwise communication between each pair of voters for each AND gate.

6.2.3 Summary

We present a quick summary of the comparision across security and complexity..

	BGW	our scheme
Communication Cost	$O(n^2 k)$	$O(n^2 k)$
Time-Complexity	$O(n^\omega k)$	$O(n^2 k)$
Pairwise Communication	Needed	Not needed
Power of Adversary	unbounded	poly-time
#Colluding parties	$n/2$	$n-2$

7 More Application and Future Work

7.1 Approve/Reject/No opinion

Another use of our scheme is a voting protocol, where we can vote for the candidate, vote against the candidate or abstain. We want to check whether the candidate got more approval votes than rejection. To make this vote, we can use different starting polynomial $T_0(x) = (x+1)\dots(x+n)$, where n is the number of voters. The main property of this polynomial is $T_0(x) = 0 \iff x < 0$. Then we construct the same recursive polynomials $T_i(x) = T_{i-1}(x+1) - T_{i-1}(x)$ and table as in the setup section 5.1. Voting protocol would be applying $Inc(A), Dec(A), Rand(A)$, where $Inc, Rand$ are the same operations described in the voting section 5.3. The voter applies Inc if they want to support the candidate, and $Decr$ if they want to reject the candidate. In the end, they apply $Rand$ function. Then at any point of the vote our table would encode $T(current\_vote\_support - current\_vote\_against)$.

7.1.1 Rejecting the candidate $Decr(A)$

To implement decrementing polynomials T_i, we notice that T_n is a constant. By the recursive property 1 we can notice that $T_i(x-1) = T_i(x) - T_{i+1}(x-1)$. So to get new table A' we start from $j = n$ of the table and go to $j = 0$ using linear property of our encryption, update each entry to be $A'_{i,j} = A_{i,j} - A'_{i,j+1}, A'_{i,n} = A_{i,n}$.

$$A' = \begin{bmatrix} A_{1,0}\cdot(A'_{1,1})^{p-2} & A_{1,1}\cdot(A'_{1,2})^{p-2} & \dots & A'_{1,n} = A_{1,n} \\ A_{2,0}\cdot(A'_{2,1})^{p-2} & A_{2,1}\cdot(A'_{2,2})^{p-2} & \dots & A'_{2,n} = A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,0}\cdot(A'_{n,1})^{p-2} & A_{n,1}\cdot(A'_{n,2})^{p-2} & \dots & A'_{n,n} = A_{n,n} \end{bmatrix}$$

$$= \begin{bmatrix} g^{sk_1(B_{1,0}-B'_{1,1})} & g^{sk_1(B_{1,1}-B'_{1,2})} & \dots & g^{sk_1 B_{1,n}} \\ g^{sk_2(B_{2,0}-B'_{2,1})} & g^{sk_2(B_{2,1}-B'_{2,2})} & \dots & g^{sk_2 B_{2,n}} \\ \vdots & \vdots & \ddots & \vdots \\ g^{sk_n(B_{n,0}-B'_{n,1})} & g^{sk_n(B_{n,1}-B'_{n,2})} & \dots & g^{sk_n B_{n,n}} \end{bmatrix}$$

So if $A = [Encoding(T_0(v)), Encoding(T_1(v))\dots Encoding(T_n(v))]$, then $A' = [Encoding(T_0(v-1)), Encoding(T_1(v-1))\dots Encoding(T_n(v-1))]$

7.1.2 Summary

$S, S_v, Eval$ algorithms are the same as in threshold voting.
The complexity of this voting scheme is equivalent to the threshold voting for $k = n, O(n^3)$ and the security proofs and guarantees are the same. Because the threshold is $O(n)$ and the number of voters is expected to be large, the scheme

provides much worse complexity than the equivalent BGW but does not require voters to communicate with each other and protect against larger number of malicious parties.

7.2 Winning candidate

Another use of this scheme is to determine the winning candidate (or the ranking of candidates), without revealing the number of votes they got. More formally, we have k candidates and each person wants to vote for one of them. Winning candidate is the one which got mot votes. To implement this voting, we will use previous scheme to approve/reject/no opinion (ARN) voting. We will come up with the table of ARN voting tables

$$W = \begin{bmatrix} 0 & A_{1,2} & \dots & A_{1,k} \\ A_{2,1} & 0 & \dots & A_{2,k} \\ \\ \vdots & \vdots & \ddots & \vdots \\ A_{k,1} & A_{k,2} & \dots & 0 \end{bmatrix}$$

Where $A_{i,j}$ is a ARN voting table to determine whether candidate i got more votes than candidate j. Each voter uses the same pair of private keys to encrypt their share. Each table stores encoding of difference of votes for candidate i and j. Then to vote for the candidate i, voter apply $Inc(A_{i,j}), Decr(A_{j,i}), \forall 0 \leq j \leq k$. Let's notice that table $A_{i,j}$ would encode $0 \iff i$ lost voting to j. This means that if we sum the tables (apply property 2 for each pair of corresponding entries) and then decrypt first columns of the sum, it would be equal to $0 \iff$ person won against all other candidates (or if acidentally $\sum_{j=1}^{k} T_0^{i,j}(\#votes\_for\_i - \#votes\_for\_i) = 0$, but this happens with neglidgible probability). So

$$\sum_{j=1}^{k} A_{0,j} = \begin{bmatrix} \prod_{j=1}^{k} Enc(sk_1, B_{1,0}^{i,j}) & \prod_{j=1}^{k} Enc(sk_1, B_{1,1}^{i,j}) & \dots & \prod_{j=1}^{k} Enc(sk_1, B_{1,n}^{i,j}) \\ \prod_{j=1}^{k} Enc(sk_2, B_{2,0}^{i,j}) & \prod_{j=1}^{k} Enc(sk_2, B_{2,1}^{i,j}) & \dots & \prod_{j=1}^{k} Enc(sk_2, B_{2,n}^{i,j}) \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{j=1}^{k} Enc(sk_n, B_{n,0}^{i,j}) & \prod_{j=1}^{k} Enc(sk_n, B_{n,1}^{i,j}) & \dots & \prod_{j=1}^{k} Enc(sk_n, B_{n,n}^{i,j}) \end{bmatrix}$$

$$= \begin{bmatrix} Enc(sk_1, \sum_{j=1}^{k} B_{1,0}^{i,j}) & Enc(sk_1, \sum_{j=1}^{k} B_{1,1}^{i,j}) & \dots & Enc(sk_1, \sum_{j=1}^{k} B_{1,n}^{i,j}) \\ Enc(sk_2, \sum_{j=1}^{k} B_{2,0}^{i,j}) & Enc(sk_2, \sum_{j=1}^{k} B_{2,1}^{i,j}) & \dots & Enc(sk_2, \sum_{j=1}^{k} B_{2,n}^{i,j}) \\ \vdots & \vdots & \ddots & \vdots \\ Enc(sk_n, \sum_{j=1}^{k} B_{n,0}^{i,j}) & Enc(sk_n, \sum_{j=1}^{k} B_{n,1}^{i,j}) & \dots & Enc(sk_n, \sum_{j=1}^{k} B_{n,n}^{i,j}) \end{bmatrix}$$

$$= [Encoding(\sum_{j=1}^{k} T_0^{i,j}), Encoding(\sum_{j=1}^{k} T_1^{i,j}), \dots Encoding(\sum_{j=1}^{k} T_n^{i,j})]$$

If we want to get full ranking, we can run $Eval$ on each $A_{i,j}$ to get pairwise comparison. Whole algorithm runs in $O(n^3k^2)$, because we use k^2 treshold voting with treshold n. For BGW we would need to find maximum (or sort the array for full ranking) which would require $O(n^2 polylog(k)k)$ operations.

7.3 Future work

In this project we proposed alternative ways to evaluate non-linear max functions on combined secret inputs using only linear operations. This approach reduces complexity of multi-party computation and eliminates the need in FHE or pairwise communication. Future work might involve exploring other potentially useful functions which could be computed using polynomials (some examples include finding whether number belongs to fixed set) and expanding possible size of each share (by constructing different recursive polynomials we can allow $Inc(m)$ operation, which increments current encoded polynomial by any m instead of 1).
Another direction to work towards is reducing size of $Encoding(T)$. While currently we require each person to encode their share separately, we can try to come up with combined encryption of one share $Enc(sk_1, sk_2, \dots sk_n, T_0)$. One possible way to do so, is to divide our input into 3 shares $B_{0,i}, B_{1,i}, B_{2,i}$: $B_{0,i} + B_{1,i} + B_{2,i} = T_i(v)$. Then instead of each person having their own public key, we simulate encryption using uniform secret key $SK = (SK_0, SK_1, SK_2) = (\prod_{i=1}^n sk_{0,i}, \prod_{i=1}^n sk_{1,i}, \prod_{i=1}^n sk_{2,i})$. To do so, we can notice that

$$Enc(sk_1 \cdot sk_2, m) = Enc(sk_1, m)^{sk_2} \tag{5}$$

$$\text{if}(sk_1, pk_1)\text{is a valid pair} \rightarrow (sk_1 \cdot sk_2, pk_1^{sk_2}) \text{ is also valid pair} \tag{6}$$

$$Dec(sk_1 \cdot sk_2, c) = Dec(sk_1, c)^{\frac{p}{sk_2} \mod p-1} \tag{7}$$

This means that each voter can apply their own encryption "on top" of previous one after voting. In addition to maintaining the table, we will simulate shared public key $PK = (PK_0 = g^{\prod_{i=1}^n sk_{0,i}}, PK_1 = g^{\prod_{i=1}^n sk_{1,i}}, PK_2 = g^{\prod_{i=1}^n sk_{2,i}})$. Idea is that after m people voted, table would look like

$$A = \begin{bmatrix} Enc(\prod_{i=1}^m sk_{0,i}, B_{0,0}) & Enc(\prod_{i=1}^m sk_{0,i}, B_{0,1}) & \dots & Enc(\prod_{i=1}^m sk_{0,i}, B_{0,k}) \\ Enc(\prod_{i=1}^m sk_{1,i}, B_{1,0}) & Enc(\prod_{i=1}^m sk_{1,i}, B_{1,1}) & \dots & Enc(\prod_{i=1}^m sk_{1,i}, B_{1,k}) \\ Enc(\prod_{i=1}^m sk_{2,i}, B_{2,0}) & Enc(\prod_{i=1}^m sk_{2,i}, B_{2,1}) & \dots & Enc(\prod_{i=1}^m sk_{2,i}, B_{2,k}) \end{bmatrix}$$

$$= \begin{bmatrix} g^{(\prod_{i=1}^m sk_{0,i}) \cdot B_{0,0}} & g^{(\prod_{i=1}^m sk_{0,i}) \cdot B_{0,1}} & \dots & g^{(\prod_{i=1}^m sk_{0,i}) \cdot B_{0,k}} \\ g^{(\prod_{i=1}^m sk_{1,i}) \cdot B_{1,0}} & g^{(\prod_{i=1}^m sk_{1,i}) \cdot B_{1,1}} & \dots & g^{(\prod_{i=1}^m sk_{1,i}) \cdot B_{1,k}} \\ g^{(\prod_{i=1}^m sk_{2,i}) \cdot B_{2,0}} & g^{(\prod_{i=1}^m sk_{2,i}) \cdot B_{2,1}} & \dots & g^{(\prod_{i=1}^m sk_{2,i}) \cdot B_{2,k}} \end{bmatrix}$$

$$PK = [g^{\prod_{i=1}^m sk_{0,i}}, g^{\prod_{i=1}^m sk_{1,i}}, g^{\prod_{i=1}^m sk_{2,i}}]$$

Inc function can be run the same way as described in the 5.3 section. During re-randomization step, after adding shares, we can update the table by adding our secret key by applying properties $5, 6$. More formally:

$$Rand(A) = \begin{bmatrix} (A_{0,0} \cdot PK_0^{c_{0,0}})^{sk_{0,m+1}} & (A_{0,1} \cdot PK_0^{c_{0,1}})^{sk_{0,m+1}} & \dots & (A_{0,k} \cdot PK_0^{c_{0,k}})^{sk_{0,m+1}} \\ (A_{1,0} \cdot PK_1^{c_{1,0}})^{sk_{1,m+1}} & (A_{1,1} \cdot PK_1^{c_{1,1}})^{sk_{1,m+1}} & \dots & (A_{0,k} \cdot PK_1^{c_{1,k}})^{sk_{0,m+1}} \\ (A_{2,0} \cdot PK_2^{c_{2,0}})^{sk_{2,m+1}} & (A_{2,1} \cdot PK_2^{c_{2,1}})^{sk_{2,m+1}} & \dots & (A_{2,k} \cdot PK_2^{c_{2,k}})^{sk_{2,m+1}} \end{bmatrix}$$

$$= \begin{bmatrix} Enc(\prod_{i=1}^{m+1} sk_{0,i}, B'_{0,0}) & Enc(\prod_{i=1}^{m+1} sk_{0,i}, B'_{0,1}) & \dots & Enc(\prod_{i=1}^{m+1} sk_{0,i}, B'_{0,k}) \\ Enc(\prod_{i=1}^{m+1} sk_{1,i}, B'_{1,0}) & Enc(\prod_{i=1}^{m+1} sk_{1,i}, B'_{1,1}) & \dots & Enc(\prod_{i=1}^{m+1} sk_{1,i}, B'_{1,k}) \\ Enc(\prod_{i=1}^{m+1} sk_{2,i}, B'_{2,0}) & Enc(\prod_{i=1}^{m+1} sk_{2,i}, B'_{2,1}) & \dots & Enc(\prod_{i=1}^{m+1} sk_{2,i}, B'_{2,k}) \end{bmatrix}$$

$$PK' = [PK_0^{sk_{0,m+1}}, PK_1^{sk_{1,m+1}}, PK_2^{sk_{2,m+1}}]$$

To evaluate the result, each party applies decryption under their secret keys to first column of the table. After each party did so, they can obtain result of the voting.

While security of this optimization is yet to be proven, potentially it can reduce amount of computation needed to $O(n \cdot k)$ since size of table is now $O(k)$ and does not depend on n. Additionally, this scheme would be flexible to unknown amount of voters, because voter applies their encryption during voting.

One more direction for future work is to provide security guarantees against malicious voters and server. Voters can produce signature of the table to ensure that server does not modify it and attack ZK-proof to verify that voting was done correctly. While ZK-proofs traditionally cause high constant overhead, it might be possible to design specific ZK-proofs for our application since both the Inc and Rand operations are fairly elementary.

8 Acknowledgments and Work Distribution

We greatly appreciate support of 6.5610 team for providing feedback and guiding our project. Special thanks to our TA Lila Chen, professor Henry Corrigan-Gibbs, and professor Yael Kalai.

As per work distribution, here are the main contributions of each team member:

- Vlada Petrusenko - contributed to the design and analysis of both systems, including research of related ideas. They are also the main author of the writeup for section 1, second half of 5.
- Liza Horokh - contributed the most to the design and analysis of the threshold voting system, including many technical design ideas. They are also the main author of the writeup for sections 4, 7, and first half of 5.
- Pranjal Srivastava - contributed the most to the design and analysis of the threshold voting system, originating with the finalized version of the technical design. They are also the main author of the writeup for section 6.

- Nyx Theos - contributed the most to the design and analysis of the distributed authentication system. They are also the main author of the writeup for section 3.

References

[1] T. ElGamal (1984) *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms* `https://link.springer.com/chapter/10.1007/3-540-39568-7_2`

[2] Ahamed et. al (2024) *Accuracy-Privacy Trade-off in the Mitigation of Membership Inference Attack in Federated Learning* `https://arxiv.org/abs/2407.19119`

[3] Networkx, *Min-node-cut implementation* `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.connectivity.cuts.minimum_node_cut.html`

[4] 6.6510 Lecture notes (especially on additive shares and LHE the LWE based encryption)

[5] `https://crypto.stanford.edu/pbc/notes/crypto/voting.html`