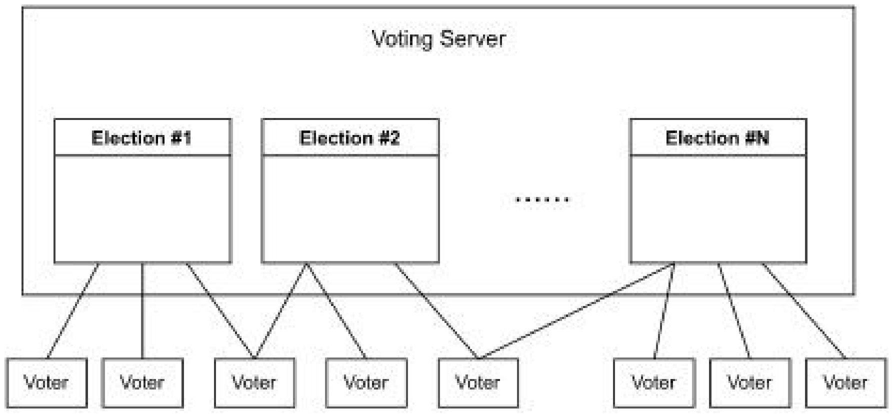
(FTC) Online Electronic Voting – Part I. RPC Interface (due 3/27)

The goal of the project is to develop a remote electronic voting system. With the system, a registered user can hold a vote and collect ballots from registered users that might be located at different locations. At the end of a vote, the system will count the ballots and announce the results. As our university consists of multiple campuses, the remote e-voting system will be essential to our daily operations.

Here we define the operations that should be supported by the online electronic voting system. The overall architecture of the e-Voting system is shown in the following figure. The voting server has a list of ongoing elections. Each voter (client) can participate in multiple elections. A voter casts the ballots, and the server will count the ballots at the end of each election.



This part of the project will implement the communication interface between the clients and the server.

For the communication between the voting server and the voting clients, we will use Google gRPC. gRPC is a Remote Procedure Call (RPC) framework. Assume that program X and program Y run on two separate machines that are connected by the network. The RPC framework allows program X to invoke functions in program Y. The saves the troubles of network programming. A good starting point for learning gRPC is gRPC Basic Tutorial.

Following are the RPC APIs you need to implement for the voting server and the voting client.

We use Google's Protocol Buffers to serialize/deserialize the messages.

```
syntax = "proto2";
package voting;
import "google/protobuf/timestamp.proto";
```

A. Local Server API

```
RegisterVoter(Voter) returns (Status)
UnregisterVoter(VoterName) returns (Status)
```

A.1. Message Formats

```
message Voter {
    required string name = 1;
    required string group = 2;
    required bytes public_key = 3;
}
```

```
message VoterName {
    required string name = 1;
}
```

```
message Status {
  required int32 code = 1;
}
```

B. RPC APIs

```
service eVoting {
    rpc PreAuth (VoterName) returns (Challenge);
    rpc Auth (AuthRequest) returns (AuthToken);
    rpc CreateElection (Election) returns (ElectionStatus);
    rpc CastVote (Vote) returns (VoteStatus);
    rpc GetResult(ElectionName) returns (ElectionResult);
}
```

B.1. Message Formats

```
message Challenge {
  required bytes value = 1;
}
```

```
message Response {
  required bytes value = 1;
}
```

```
message AuthRequest {
    required VoterName name = 1;
    required Response response = 2;
}
```

```
message AuthToken {
  required bytes value = 1;
}
```

```
message Election {
    required string name = 1;
    repeated string groups = 2;
    repeated string choices = 3;
    required google.protobuf.Timestamp end_date = 4;
    required AuthToken token = 5;
}
```

```
message ElectionStatus {
    required int32 code = 1;
}
```

```
message Vote {
  required string election_name = 1;
  required string choice_name = 2;
  AuthToken  token = 3;
}
```

```
message ElectionName {
  required string name = 1;
}
```

```
message VoteCount {
  required string choice_name = 1;
  required int32  count  = 2;
  AuthToken token = 3;
}
```

```
message ElectionResult {
    required int32 status = 1;
    repeated VoteCount count = 2;
}
```

At this moment, you only need to make sure the RPC interfaces (client can make calls to server) work correctly. You do not need to worry about the program logics behind the APIs at this moment.

Please prepare the following for submission

1. Put your code on GitHub
2. Prepare a README describing how to build and run your code
3. Write a report that cover the design, implementation, and evaluation