Homomorphic Vote Tally

CSC Project

2019/2020

Monday, October 28, 2019

Version 1.1

Introduction

The goal of this project is training the students in the use of public key cryptography and homomorphic encryption.
With that purpose the students should develop an online voting system using i) public key cryptography for voter authentication; ii) homomorphic encryption for vote tally, and iii) Shamir's secret sharing to reveal the results.

Usually an e-Voting application requires a number of communication and fault-tolerant features. However, for the purpose of this project there is no such requirements.

The project has several actors performing a set of operations. Each operation can be implemented by a simple application that reads inputs from the command line and writes its outputs in files.

Workflow of the voting

The voting system should have the following actors:

1. The administrator that sets everything;
2. The voter;
3. The ballot box;
4. The tally official;
5. Counter;
6. The trustees.

Administrator

The **administrator** is not a process or any kind of activities it is just the user running a set of tools to set things up. His goals are:

1) Generate a root CA certificate and private key;
2) Install the root certificate in the tally official app;
3) Generate a certificate for every voter (e.g. with openssl)
4) Generate the election key - a special homomorphic key pair (e.g. using Microsoft SEAL library, see below)
5) Install on each voter app:

    a. The root CA certificate
    b. The voter private key and certificate
    c. The election public key
6) Split the election private key using Shamir's secret sharing, distribute each of the shares by the trustees, and erase the private key.
7) Assigns a weight to each voter, encrypts it with the election public key and publishes the list of encrypted weights.

Some of these operations can be performed directly using the openssl command. There is no need to implement those operations, just specify the command line needed to perform the operation.

Voter

The **voter** may vote multiple times but only the last one counts. The voter runs a command line application that:

1) Reads the voter's intentions from the command line;
2) Encrypts the vote using the election public key and Microsoft seal library;
3) Signs the vote using the voter's private key, using the libcryp library (see below);
4) Cast the vote and sending it to the ballot box.

The voter application should allow malformed ballots. In particular, it should allow sending more votes then candidates, both in terms of total amount of votes, as in terms of distribution of those votes by more entities then candidates. It should allow the usage of an incorrect election public key and of an incorrect signing certificate. This is relevant for the purpose of testing the correction of the cryptographic validations.

Ballot Box

The **ballot box** may be a process running on a computer, a webservice with access to a database, an email box, or just a simple file. The goal of the ballot box is to collect votes sent by voters. Usually a ballot box requires a set of security mechanism but in this mockup system it has none.

Tally official

The **tally** official's goal is to choose which votes to count and send it to the counter. The tally executes the following actions for each vote in the ballot box:

1) Check the signature of the vote, if signature fails remove the vote from the tally;
2) Check if there is another vote in the tally from the same voter with a date previous to the current, if so discards the vote otherwise replaces the vote in the tally
3) Computes homomorphically the checksum for each vote and adds it to an accumulator (see below)
4) Compute homomorphically the result of the election (see below)
5) Sends the election results and the checksum accumulator to the counter

Counter

The goal of the counter is to check the checksum accumulator and announce the election results. Usually the decryption of the election results is performed by a threshold scheme that decrypts the results without revealing the election private key, but for simplicity the counter will get from each trustee their share of the election private key and rebuild it using Shamir's secret sharing.

With the election private key, the counter decrypts the checksum accumulator and verifies its validity. If every vote is valid it decrypts the elections results.

Trustees

The trustees only purpose is to store the private key share for the period of the election

Ballot Format

There are many voting systems, from the standard direct democratic elections, where each voter casts one vote for one of the available candidates, to the write-ins ballot elections, where a voter writes the name of someone (every one is a candidate).

The intended vote system is a ranked and weighted voting system. In a ranked voting system, the voter has a number of votes that she may cast to each candidate. For example, the voter may have 10 votes to distribute by 10 candidates. She may cast the 10 votes on one of the candidates or distribute the votes for some of the candidates. In a weighted voting system each voter has a different number of votes to cast, i.e. it is not a democratic election.

The format of the ballot should be a vector with one position per candidate, containing an integer. The sum of every integer in a ballot is called the checksum of the ballot and should not exceed the number of votes that each voter has.

To simplify the voting process for the voter. Each voter is given the same number of votes; one for each candidate. The weight of the voter is assigned by the administrator and used in the vote tally by multiplying each position in the ballot vector by the weight of the voter, before adding it to a vote counter.

This process also simplifies the checksum verification. The election is declared correct if the checksum accumulator is equal to the number of candidates times the number of votes.

Libraries and references

For homomorphic encryption use the Microsoft SEAL library available here:
https://github.com/Microsoft/SEAL

For public key encryption in C, use, for instance, the libcrypto available here:
https://github.com/openssl/openssl

You may find example code to sign and verify documents with libcrypto here:
https://wiki.openssl.org/index.php/EVP_Signing_and_Verifying

For Shamir's secret sharing you may use the libsss available here:
https://github.com/dsprenkels/sss/blob/master/README.md