Subject 6: E-voting based on Secure Multiparty Computation

Ethan Zouzoulkowsky
Alexis Martins De Carvalho
Romain De Javel De Villersfarlay
Jean Bou Raad

The process of voting online with a platform that provides secure, private and reliable infrastructure is hard to achieve. Hence, multiple voting methods were implemented to achieve that challenge and one of them is based on secure multiparty computation.

Overview of SMC

Secure multi-party computation (MPC) is a subfield of cryptography that deals with enabling multiple parties to jointly compute a function over their private inputs without revealing any information about their inputs to each other. The goal of MPC is to provide a way for multiple parties to collaboratively perform a computation, while ensuring that no party can learn anything about the input of any other party.

In traditional computation, all parties share their inputs with a trusted third party who computes the function and returns the result. However, in MPC, there is no trusted third party, and all parties must work together to compute the result. To achieve this, MPC uses cryptographic protocols that allow parties to share information and perform computations on that information without revealing any details about the input.

Hence, MPC relies mainly on the honesty of each party. To reflect the possibility of a dishonest party, there are multiple types of security setups:

- Semi-Honest (Passive): meaning that no attacker will directly interfere with the protocol but may listen to the communications.
- Malicious (Active): an attacker may try to cheat during the process. The result of the computation will be incorrect but privacy remains intact.

Under the hood, MPC can be implemented for these setups with a number of parties to compute an output.
Firstly, there are protocols that rely on two-party computation, for most, they are implemented with those two methods:

- Yao's Garbled Circuits: in this method, the two parties construct a "garbled circuit" that represents the function they want to compute. The garbled circuit is constructed in such a way that the parties can evaluate it without revealing any information about their inputs. The parties use a combination of encryption and randomization to create the garbled circuit, and then they exchange information to evaluate the circuit and obtain the result.
- Fully Homomorphic Encryption (FHE): FHE is a powerful encryption scheme that allows computations to be performed on encrypted data. In 2PC, the two parties encrypt their inputs using FHE, then perform computations on the encrypted data to

obtain an encrypted result. They can then decrypt the result to obtain the final output without ever revealing their inputs to each other.

Yao's Garbled Circuits are efficient and can handle a wide range of functions, but they can be complex to construct and require a lot of communication between the parties. FHE, on the other hand, is very flexible and can handle any function, but it is computationally expensive and can be slow in practice.

On the other hand, there exist multiparty protocols that rely on secret sharing. Each party has a share of the data that is later merged to get the output of a function. The most commonly used schemes are the Shamir secret sharing scheme and the additive secret sharing. Those protocols can be reliable with up to half of the parties being half-dishonest and a third of them actively cheating during the computation.

The multiparty protocols present the advantage of scalability with reasonable performance.

Detailed description of the chosen SMC protocol

We have chosen to implement our e-voting with a Shamir based protocol. The Shamir protocol was created in 1979 by Adi Shamir.

The protocol features information-theoretic security, meaning that an attacker with knowledge of the secrets cannot compute the real value whatever means he has. The protocol is also lightweight with many customizations possible.

The protocol relies on the Lagrange interpolation theorem to share its secrets with different parties. The main idea behind the protocol is that the points of a polynomial function can be given and split to different parties with the coefficient of degree being the secret. Each point isolated is not enough to build the polynomial function back. But when a party has at least half of the points generated, it is possible to combine the different values to find the function back.

$$l_i = \frac{x - x_0}{x_i - x_0} \times \ldots \times \frac{x - x_{i-1}}{x_i - x_{i-1}} \times \frac{x - x_{i+1}}{x_i - x_{i+1}} \times \ldots \times \frac{x - x_{k-1}}{x_i - x_{k-1}}$$

$$f(x) = \sum_{i=0}^{K-1} y_i l_i(x)$$

There is a minimal number of points needed to be able to retrieve the secret. However, with that method, it is possible to create new parties and generate additional points. Also, by changing the polynomial function dynamically, it is possible to create an additional layer of security as long as the part holding the secret remains intact.

$$l_0 = \frac{x - x_1}{x_0 - x_1} = \frac{x - 3}{1 - 3}$$

$$l_1 = \frac{x - x_0}{x_1 - x_0} = \frac{x - 1}{3 - 1}$$

$$f(x) = y_0 l_0 + y_1 l_1$$

$$f(x) = 80\left(\frac{x - 3}{-2}\right) + 110\left(\frac{x - 1}{2}\right)$$

$$f(x) = -40x + 120 + 55x - 55$$

$$f(x) = 15x + 65$$

The implementation of this protocol is lightweight. Indeed, computing an image of function is relatively fast compared to most encryption schemes. The performance critical part of the protocol is the sharing and retrieval of each point in a secured manner, mostly done through an encrypted communication.

Criteria and process for selecting the SMC library

We first searched for libraries that could fit our usage, meaning being able to hold two servers or more. We need to compare the security of each of their protocols, their advantages and weaknesses.

We ended up using MPyC after comparing the following libraries:

Library Name	Language	Advantages	Weaknesses
OblivC	C	Simple and lightweight language	Low-level, particular syntax to learn
MpyC	Python	Shamir implementation with high level of abstraction and highly configurable	Limited types. No native client / server structures. The script is considered to be the client.
Sharemind	C++	Entire implementation of e-voting	Open-source but proprietary with limited OS support (Debian)
MP-SPDZ	C++ / Python	Large protocol support	Hard to install, complex to use
JIFF	Javascript	Web-oriented, multiparty protocol	Project is not being actively supported
ABY	C++	Gates based	Only two party

			protocol

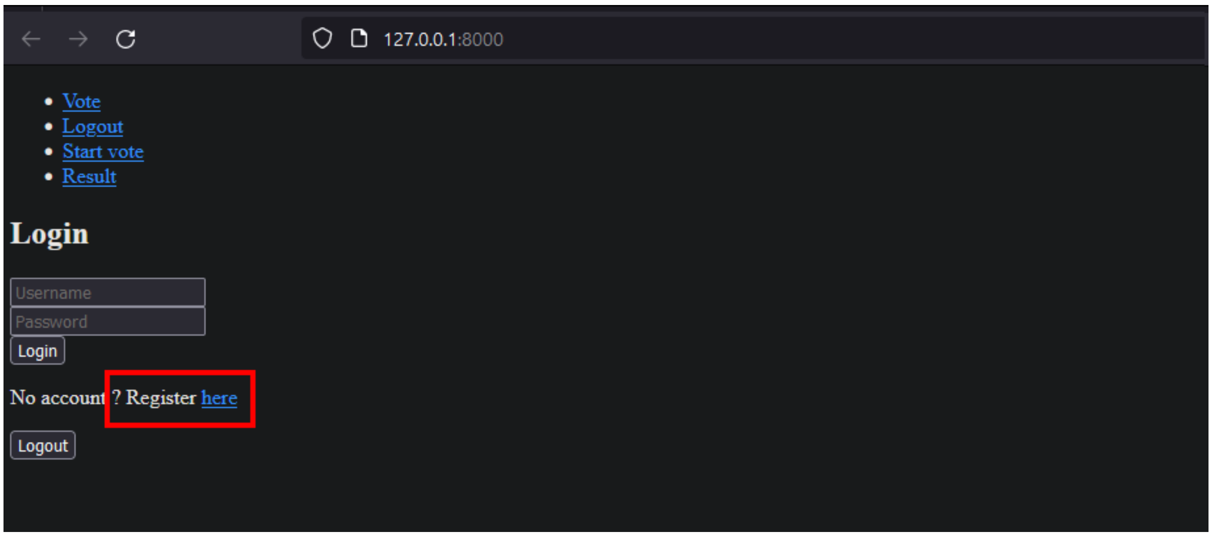
MPyc is a python package which implements the Shamir protocol to exchange secrets and compute an output. It is still in its early stages of development but features a large set of configuration options. Moreover, the library does not rely directly on any other external library making it extremely lightweight and easy to use.
Finally, the library supports SSL encryption for its communication between servers.

Experiment & Results

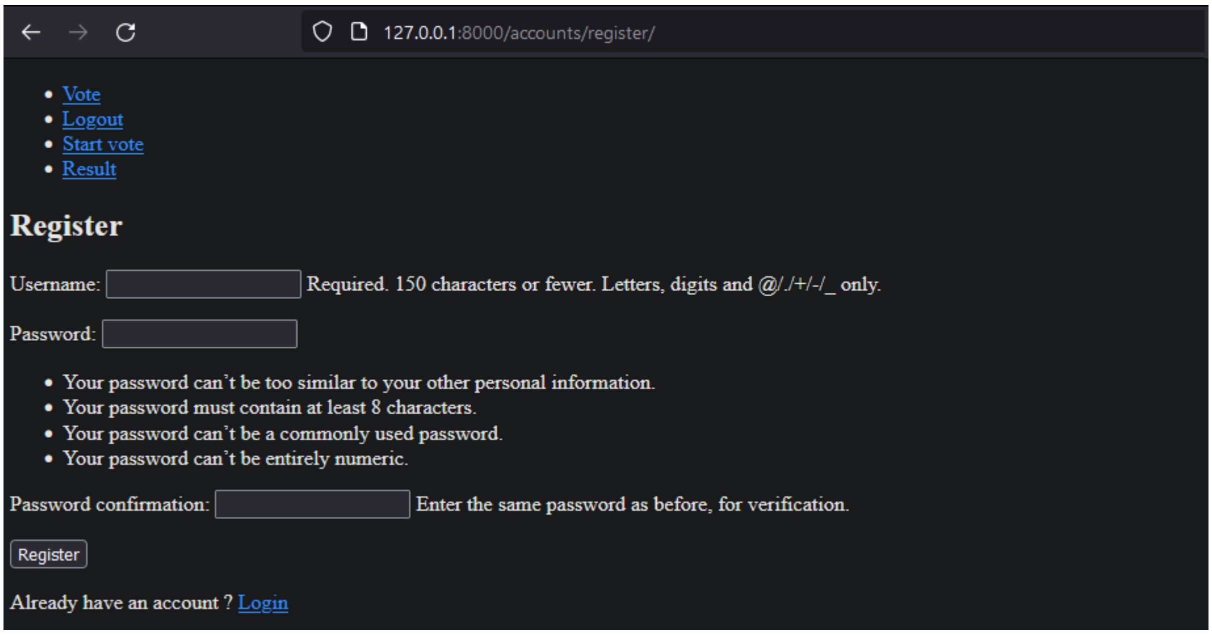
For this project, we built a voting system which includes a frontend for users to register, login, and send their votes. Behind it, the backend handles the votes by making security checks (for duplicate votes) and holds virtually count until we tally the votes.

Each voter has an account in our database, when we initialize the voting system, we associate to each of them a random identifier generated by the MPyC library.

We can registrate a set of voters to our database by clicking on the register button:



The voter can then enter its username and password:


127.0.0.1:8000/accounts/register/

- Vote
- Logout
- Start vote
- Result

Register

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

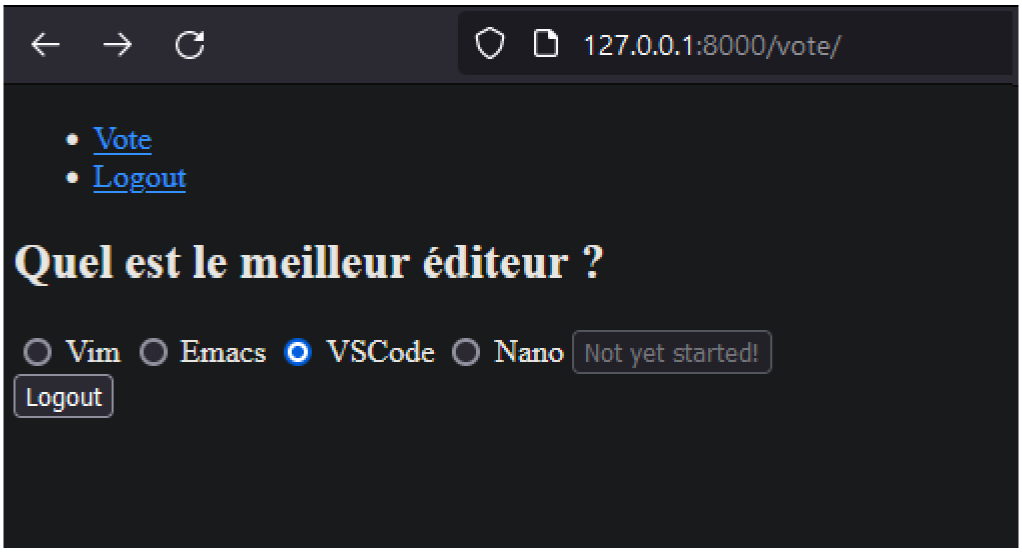
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Register

Already have an account ? Login

After registration, the voter can log in:


127.0.0.1:8000/vote/

- Vote
- Logout

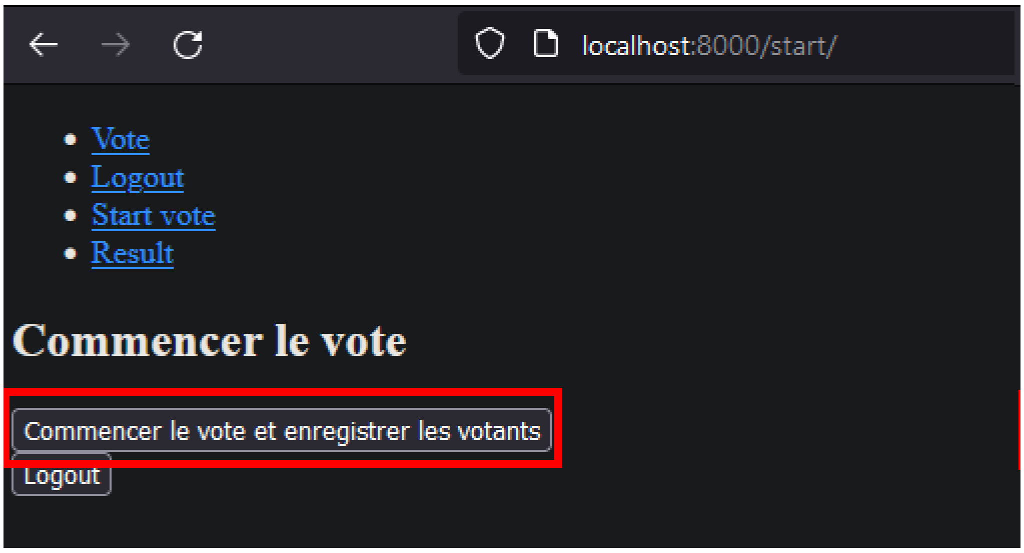
Quel est le meilleur éditeur ?

Vim Emacs [✓ VSCode] Nano Not yet started!

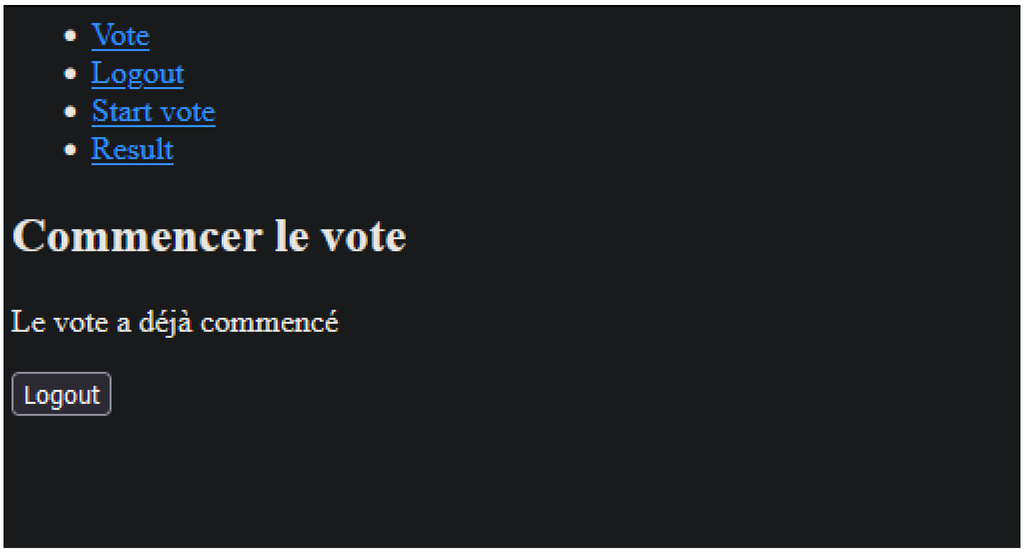
Logout

For the moment, the user cannot vote because the session has not yet been started by an administrator.

We can then login using an administrator account:



An administrator can start the vote and stop it and get the result. When started, the MPC registers a list of registered users for the vote. Any new user trying to vote will see its vote not taken into account. Once the vote has been started, the administrator cannot vote as he has the power to modify the outcome easily by stopping the vote at the time of his choice.

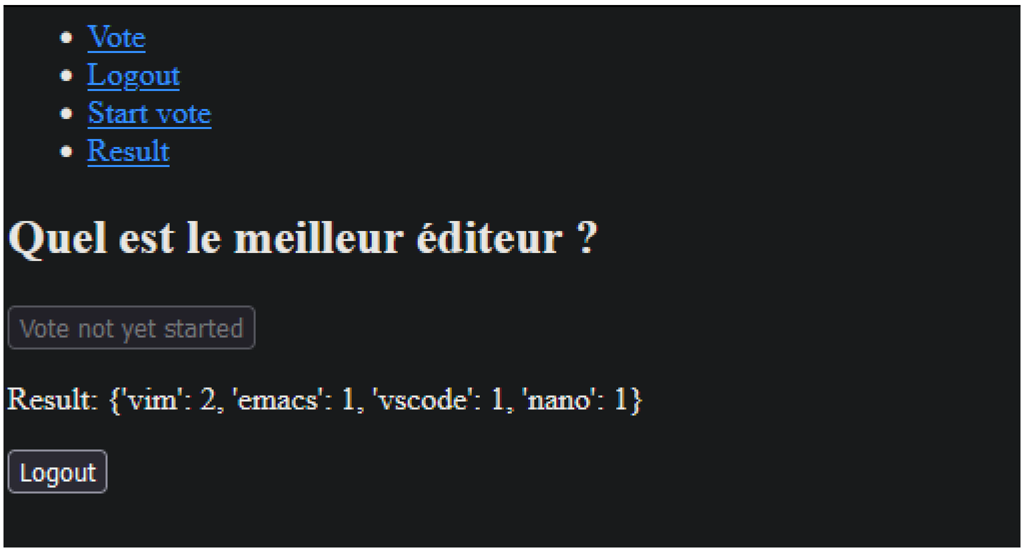


The voters on the portal can make his choice and send it to the backend which registers a vote and his identifier in two different arrays. For this test, we created 5 accounts before the beginning of the vote, one of each will vote for each of the four choices. The last one will vote for vim. Before the tally, two of the voters try to cheat by double voting. It is possible because the server does not hold direct knowledge of who voted. Instead, the multiparty computation system has a branching condition that will nullify invalid votes.

One array holds the list of people who already voted independently from the vector of votes for which a value in the list represents the total amount of vote for each candidate. The independence of each list is the best way to keep privacy intact while ensuring that no client can cheat the system by voting two times.

During the vote registration process, we also sanitize the input by making sure that it is within the accepted range. Any invalid value will be ignored.

At the end of the voting window, an administrator can tally the votes and verify that the count of votes equals the number of identifiers in the list. Finally, we can validate the result and display the winner on the voting page.



Quel est le meilleur éditeur ?

```
Result: {'vim': 2, 'emacs': 1, 'vscode': 1, 'nano': 1}
```

We can see the count is correct for each of our options, meaning that the vote was correctly taken into account.

In a semi-honest setting, the privacy of each vote is kept intact unless half of the parties are corrupted. By having control on the servers and clients locally, we have the guarantee that no 3rd party is actively listening to our encrypted communications.

Challenges faced during the project

The MPC library was hard to integrate to the web framework because of its asynchronous nature. We used the Django framework for our web services, it does not directly support asynchronous operations. Hence, we had to find a way around those constraints while registering the votes.

The main challenge we faced during the project was on the matter of privacy while enhancing security and integrity of the vote. Giving user feedback without revealing sensitive secrets was complicated with this library in its early stages.

Finally, most projects and libraries implementing an MPC are either deprecated or hard to integrate into existing projects. Some of them require very specific environments, making them unsuitable even though they can be feature complete like ShareMind SDK.

Recommended future steps for enhancing the project

The project does not include a fully secured chain of communication for the votes. There is still a transition phase between the vote being received and the vote being accounted for in the result. In that phase, an attacker may retrieve the result. However, if an attacker could retrieve information on the server, then the security of the whole voting system may be compromised.

To improve the project, we could encrypt the voting selection process using a key generated at the beginning of the vote or have a native client on the web client to keep the votes within the MPC system.