**BASIC GUIDE TO E-VOTING SYSTEM**

We have made a secure e-voting protocol, with many cryptographic primitives and network communication. The code is written in Python, and it imports several libraries such as hashlib, os, random, secrets, socket, pickle, struct, sys, threading, time, uuid, cryptography, Crypto, and phe. The purpose of this code is to simulate a client-side system for a secure electronic voting system.

The code starts by importing the necessary libraries such as hashlib, os, random, secrets, socket, pickle, struct, sys, threading, time, uuid, cryptography, Crypto, and phe. Next, it defines the host address for the server as '127.0.0.1' using the HOST.

The class **Administrator** is used to create an administrator object for an election system. The code generates keys for collectors and the administrator, initiates an election, and starts an administrator server.

Here is a brief explanation of the code:

1. In the **\_\_init\_\_** method, the code generates an election ID, which returns a string of 16 bytes. The election ID is then assigned to a class variable called **election\_id**.
2. The code generates public and private keys for collectors and the administrator using the **rsa.generate\_private\_key** function, which returns an instance of a private key. The public keys are then extracted from the private keys using the **public\_key()** method.
3. The code writes the private keys for collectors and the administrator to **.pem** files.
4. The code calculates a hash of the public key for each collector using the **hashlib.sha256** function, and assigns the result to class variables called **collector1\_key\_hash** and **collector2\_key\_hash**.
5. The code converts the public keys for collectors and the administrator to byte strings using the **public\_bytes** method, and assigns the results to class variables called **collector1\_pk**, **collector2\_pk**, and **admin\_pk**.
6. The code assigns the lengths of collector hosts to class variables called **collector1\_host\_length** and **collector2\_host\_length**.
7. The code starts an administrator server using a separate thread and the **admin\_server** method. The server listens on port 8000.
8. Encryption of all the information

**Note code contains much more details then listed here this is only very basic overview of the whole system. Please look code if you have doubts.**

The class **Collector**, has big part in voting system. The class has several attributes:

* **election\_id**, which is a string that presumably identifies the election being conducted.
* **M** and **N**, which are integers, where **M** is set to 2 and **N** is set to 5.
* **voters\_info**, which is an empty list that will presumably be populated with information about voters.
* **paillier\_public\_key**, which set to the public key for the Paillier encryption scheme

The class has a constructor method, **\_\_init\_\_**, which takes an argument **collector\_index**. The constructor loads two PEM-encoded private keys from files named "data/collector1\_private\_key.pem" and "data/collector2\_private\_key.pem", respectively, and sets the **private\_key** attribute to the appropriate one based on the value of **collector\_index**. It also sets the **pk** attribute to the corresponding public key, and sets several other attributes that appear to be used in network communication. Specifically, **self.other\_C\_host\_length** is set to the length of a variable **HOST**, contains the hostname or IP address of another machine in the network. **self.other\_C\_host** is set to **HOST**, and **self.other\_C\_port** is set to a value that depends on **collector\_index**. The **client\_thread** and **server\_thread** variables are then initialized with two **threading.Thread** objects, which run the **collector\_client** and **collector\_server** methods in separate threads.

The **collector\_server** method creates a **socket** object using the **AF\_INET** and **SOCK\_STREAM** constants and binds it to a particular port (8001 for **collector\_index** 1, and 8002 for **collector\_index** 2). If the **bind** operation fails, the method prompts the user to enter a different port number and tries again. The server then enters an infinite loop, accepting connections from clients and spawning a new thread to handle each connection. Each connection is handled by the **collector\_controller** method.

The **collector\_controller** method first receives data from the client. If the first element of this object is the bytes **b'\x05'**, the method prints a message indicating that it received a message from the administrator, and if **collector\_index** is 2, it closes the connection. Otherwise, if **collector\_index** is 1, it initialize Paillier encryption. It then returns from the method, so that it can be re-entered once the Paillier encryption is set up.

If the first element of the received object is **b'\x08'**, the method decrypts several pieces of information from the object, including a key hash, an election ID, and a voter ID. The public key is used to compute a hash, which is then hashed again using BLAKE2b hash and SHA-256 to produce the **key\_hash** attribute. The method then creates a message to be sent to the voter, which includes several pieces of information, such as the **key\_hash**, the **election\_id**, and **N**. The message is then encrypted using the cipher with a random 256-bit symmetric key, which is then encrypted using the voter's public key using the Paillier encryption scheme.

The class **Voter** defines the attributes of a voter, such as the election ID, the public key of the collector servers, the host and port of the collector servers, and other parameters. The class has an init method that initializes these attributes.

In the **init** method, a private key is generated for the voter using the **rsa.generate\_private\_key** method from the cryptography library. The private key is saved. The method then generates a payload that contains the message type, key hash, and voter ID. The payload is signed using the private key, and the RSA signature is added to the payload. The resulting message is then sent to the **Admin** server using the socket library.

The code then receives a response from the **Admin** server, which contains the election ID, the public key of the collector servers, the host and port of the collector servers, and other parameters. These parameters are stored in the corresponding attributes of the Voter class.

Overall, the code implements a secure e-voting protocol that uses various cryptographic techniques to ensure the privacy and integrity of the voting process. The Voter class represents a client-side system that interacts with the **Admin** server and the collector servers to cast the vote securely.

The V**ote** that takes in five arguments: **admin\_host**, **admin\_port**, **voter\_private\_key**, **collector\_public\_key**, and **collector\_address**. The purpose of the function is to send an encrypted message from a voter to a collector over a TCP/IP connection.

The key associated with the voter's private key is retrieved done using the **public\_key()** method of the **voter\_private\_key** object. The resulting public key is then serialized using the DER format.To ensure the integrity of the message being sent, a hash of the **serialized\_key** is computed using BLAKE2b hash and SHA-256 and stored in a **key\_hash** variable.

The **message** variable is then created by concatenating a few different elements. Specifically, it contains a byte representing the message type (**0x03**), the **key\_hash**, the **election\_id**, and the **voter\_id**. The **voter\_id** is converted to a four-byte integer and packed in big-endian byte order using the **to\_bytes()** method.

The next step is to sign the **message** using the voter's private key. This is done using the **sign()** method of the **voter\_private\_key** object. The signing operation uses the PSS padding scheme with a salt length of **padding.PSS.MAX\_LENGTH** and the BLAKE2b hash and SHA-256 hashing algorithm. The resulting signature is stored in a RAS **signature** variable.

The final message to send to the collector is created by concatenating a few different elements. Specifically, it contains a byte representing the message type (**0x08**), the **key\_hash**, the **election\_id**, the **voter\_id**, and the **signature**.

The **final\_message** is then encrypted using the collector's public key. The encryption operation uses the OAEP padding scheme with the BLAKE2b hash and SHA-256 hashing algorithm for both the mask generation function (MGF) and the data. The **label** parameter is set to **None**.

The resulting **encrypted\_message** is then sent to the collector over a TCP/IP connection. This is done using a **socket** object, which is created using the **socket.socket()** method with the **AF\_INET** and **SOCK\_STREAM** parameters. The **connect()** method is used to establish a connection to the collector's address and port, and the **sendall()** method is used to send the encrypted message. The connection is closed automatically when the **with** block ends.

**Note code contains much more details then listed here this is only very basic overview of the whole system. Please look code to grasp more knowledge.**

**Additional INFO**

A secret ballot system has been implemented for this project E-Voting System by adding the following steps to the code:

1. Before the vote is cast, the voter's identity is be verified to ensure that only eligible voters are allowed to cast their votes.
2. Once the voter's identity has been verified, the voter is given a public key that is used to identify their vote. The key is kept secret and not be linked to the voter's identity to ensure the anonymity of the vote.
3. The voter then casts their vote using the E-Voting System. The votes are encrypted and send to collector.
4. The encrypted vote is sent to the collector along with a message that includes the election ID and other required parameters.
5. The collector then decrypts the message and retrieves the encrypted vote. The collector should then validate the vote to ensure that it is a valid vote that can be counted.
6. Once the vote has been validated, the collector stores the encrypted data securely.
7. Once all the votes have been cast and validated, the collector counts the votes and announce the winner. The collector ensures data is not disclosed to anyone, including the election officials or candidates.