```
pragma solidity ^0.5.0;

contract BPS {
    //Track State of Voters Ballot
    enum State {
        NONE,
        START,
        VOTE_RECORDED,
        VOTE_SEALED,
        VOTE_AUDITED,
        VOTE_VERIFIED
    }
    //Track how many Proposals are in this voting round
    struct Proposal {
        uint voteCount;
    }
    struct  Votie {
        bool  voted;
        bytes e_vote;
        string vote;
        State state;
    }
    string public election_name;
    address chairperson;
    string private key_encrypt;
    mapping(address => Votie) public  Voter;
    mapping(address => uint256) public balances;

    // Create a new ballot with different proposals.
    constructor() public {
        chairperson = msg.sender;
        key_encrypt = 'b2IV8VAyokkkbdJmaPivPCRqwpPq6oBTHEmgLnGqVQqw=';
    }

    function join_Ballot () public payable{
      // require (msg.sender != chairperson);
        balances[msg.sender] += msg.value;
        Voter[msg.sender].voted = false;
        Voter[msg.sender].state = State.START;

    }
    modifier notVoted(){
```

```
    require((Voter[msg.sender].state != State.START) ||(Voter[msg.sender].state != 
State.VOTE_AUDITED)
        ||(Voter[msg.sender].state != State.VOTE_RECORDED) ||(Voter[msg.sender].state != 
State.VOTE_SEALED)||
        (Voter[msg.sender].state != State.VOTE_VERIFIED));
    _;
  }
  modifier isVoteStartorAudit(){
    require((Voter[msg.sender].state == State.START) ||(Voter[msg.sender].state == 
State.VOTE_AUDITED));
    _;
  }

  modifier isVoteRecorded(){
    require(Voter[msg.sender].state == State.VOTE_RECORDED);
    _;
  }
  modifier isVoteSealed(){
    require(Voter[msg.sender].state == State.VOTE_SEALED);
    _;
  }
  modifier isVoteVerified(){
    require(Voter[msg.sender].state == State.VOTE_VERIFIED);
    _;
  }

  function record_vote(bytes memory encoded_vote, address person) public 
isVoteStartorAudit{
    Voter[person].e_vote = encoded_vote;
    Voter[person].state = State.VOTE_RECORDED;
  }
  function seal_vote(string memory vote, address person) public isVoteRecorded{
    Voter[person].vote = vote;
    Voter[person].state = State.VOTE_SEALED;
  }
  function getAudit(address person) view public returns(bytes memory){return 
Voter[person].e_vote;}
  function audit_vote(address person) public {Voter[person].state = State.VOTE_AUDITED;}
  function authenticate(address person) public payable isVoteSealed{

    Voter[person].state = State.VOTE_VERIFIED;
  }
```

```
    function stateofVote(address person) view public returns (State){ return Voter[person].state;}
    function encrypt(string memory social) view public returns (bytes32){
        bytes32 temp = sha256(bytes(concatenateString(key_encrypt, social)));
        return temp;
    }
    function concatenateString(string memory a, string memory b) internal pure returns (string memory) {
        bytes memory aa = bytes(a);
        bytes memory bb = bytes(b);

        string memory ab = new string(aa.length + bb.length );
        bytes memory ba = bytes(ab);

        uint k = 0;
        for (uint i = 0; i < aa.length; i++){
            ba[k++] = aa[i];
        }
        for (uint i = 0; i < bb.length; i++){
            ba[k++] = bb[i];
        }

        return string(ba);
    }
}
```