SCHOOL OF ARTIFICIAL INTELLIGENCE

PROFFESSIONAL ELECTIVE 1:

APPLIED CRYPTOGRAPHY

MID-TERM PROJECT

B.TECH

CSE-AI(Semester-5)

BLOCK CHAIN BASED E-VOTING SYSTEM

Under the supervision of:

Dr.K.P Soman AND Dr.Sunil Kumar S

SUBMITTED BY

GROUP-8

SYKAM SUMANJALI(CB.EN.U4AIE21068)

SUDA HARI PRIYA(CB.EN.U4AIE21067)

R HEMA RADHIKA(CB.EN.U4AIE21050)

SOUVIK GORAIN(CB.EN.U4AIE21065)

ACKNOWLEDGEMENT:

We would like to express our sincere gratitude to our esteemed faculty for their invaluable guidance and support throughout our crypto project on the Blockchain-based E-Voting System. Your expertise and mentorship were instrumental in shaping our project's success. Your dedication to our learning and development has been a source of inspiration. Thank you for your unwavering commitment to our academic growth. We deeply appreciate your contributions to our team's journey.

TABLE OF CONTENTS:

ABSTRACT:

In an era marked by digitalization, the need for secure and transparent voting systems is paramount. This report presents the culmination of our efforts in developing a Blockchain-based E-Voting System using the SHA-256 encryption algorithm. The primary objective of our project is to provide a resilient and tamper-resistant platform for electronic voting. We delve into the fundamentals of blockchain technology and its suitability for the electoral process. Our implementation incorporates the SHA-256 algorithm for encryption, ensuring data integrity and confidentiality.

We discuss the system's architecture, highlighting its decentralized nature and immutability, which eliminates the risk of electoral fraud and manipulation. Moreover, we explore the user experience and security features, ensuring a user-friendly and secure voting process. Through testing and analysis, we demonstrate the robustness and reliability of our system.

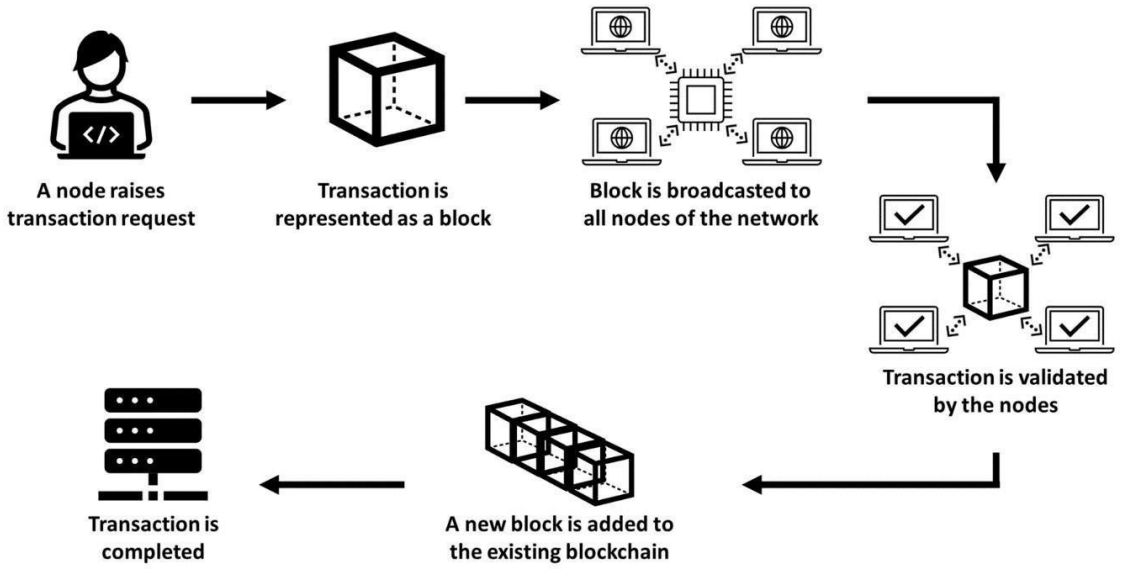
This report serves as a comprehensive guide to our Blockchain-based E-Voting System, showcasing its potential to revolutionize the electoral process, making it more accessible and trustworthy in the digital age.

BLOCKCHAIN:

A blockchain is "a distributed database that maintains a continuously growing list of ordered records, called blocks." These blocks "are linked using cryptography.

- Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data.
- Blockchain Technology first came to light when a person or group of individuals name **'Satoshi Nakamoto'** published a white paper on **"*BitCoin: A peer-to-peer electronic cash system*"** in 2008.
- The four key concepts of Blockchain are : Shared ledger, Permission, Smart contracts, Consensus.

1. **Shared ledger.** A shared ledger is an "append-only" distributed system of record shared across a business network.
2. **Permissions.** Permissions ensure that transactions are secure, authenticated, and verifiable.
3. **Smart contracts.** A smart contract is "an agreement or set of rules that govern a business transaction; it's stored on the blockchain and is executed automatically as part of a transaction."
4. **Consensus.** Through consensus, all parties agree to the networkverified transaction. Blockchains have various consensus mechanisms.

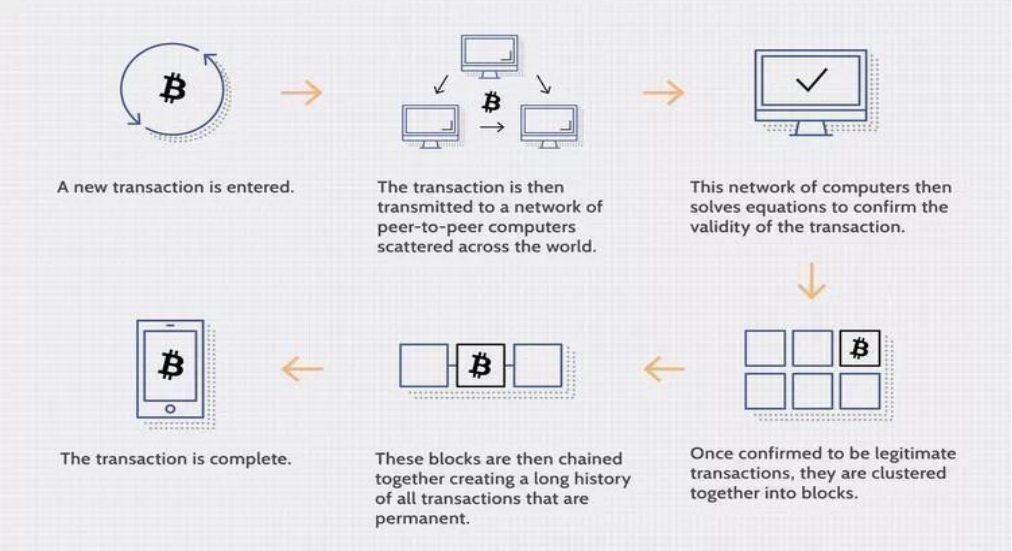


How does Blockchain Works?

- As each transaction occurs, it is recorded as a “block” of data.
- Those transactions show the movement of an asset that can be tangible (a product) or intangible (intellectual). The data block can record the information of your choice: who, what, when, where, how much and even the condition — such as the temperature of a food shipment.
- Each block is connected to the ones before and after it
- These blocks form a chain of data as an asset moves from place to place or ownership changes hands.
- The blocks confirm the exact time and sequence of transactions, and the blocks link securely together to prevent any block from being altered or a block being inserted between two existing blocks.
- Transactions are blocked together in an irreversible chain: a blockchain
- Each additional block strengthens the verification of the previous block and hence the entire blockchain.
- This renders the blockchain tamper-evident, delivering the key strength of immutability.
- This removes the possibility of tampering by a malicious actor — and builds a ledger of transactions you and other network members can trust.

Example of Blockchain (BITCOIN):

For example, on Bitcoin's blockchain, if you initiate a transaction using your cryptocurrency wallet—the application that provides an interface for the blockchain—it starts a sequence of events.

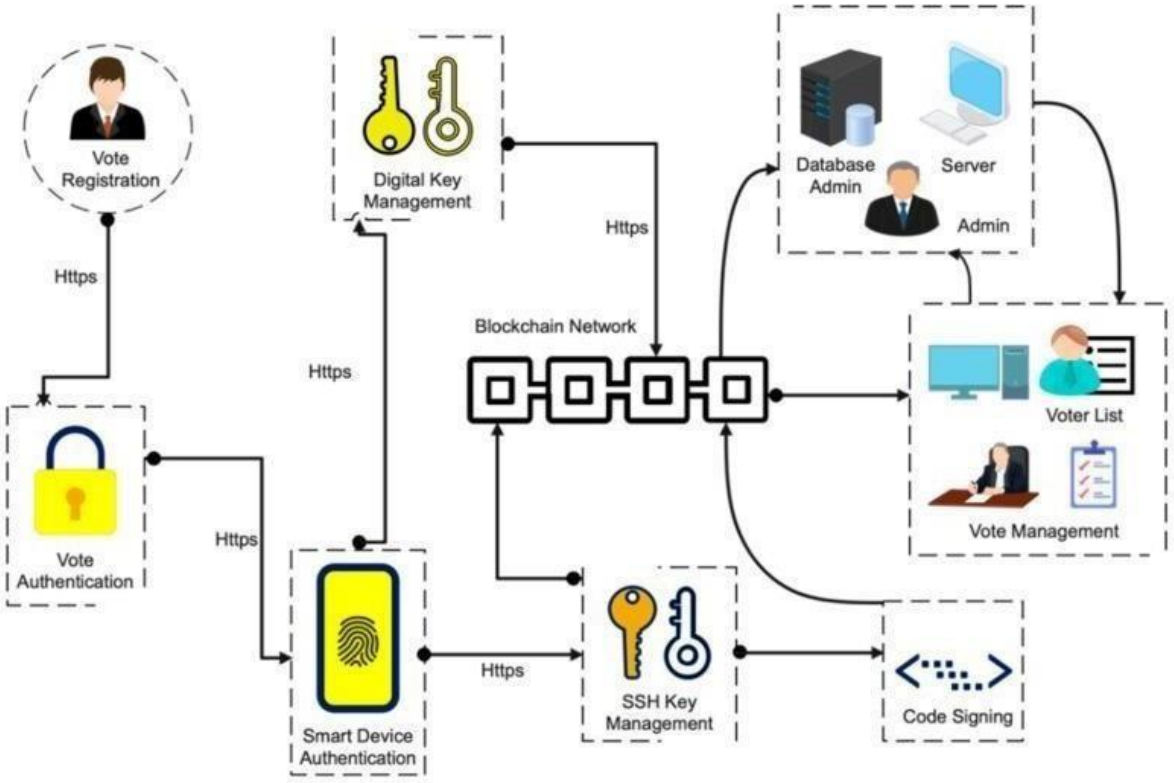


In Bitcoin, your transaction is sent to a memory pool, where it is stored and queued until a miner or validator picks it up. Once it is entered into a block

and the block fills up with transactions, it is closed and encrypted using an encryption algorithm. Then, the mining begins.

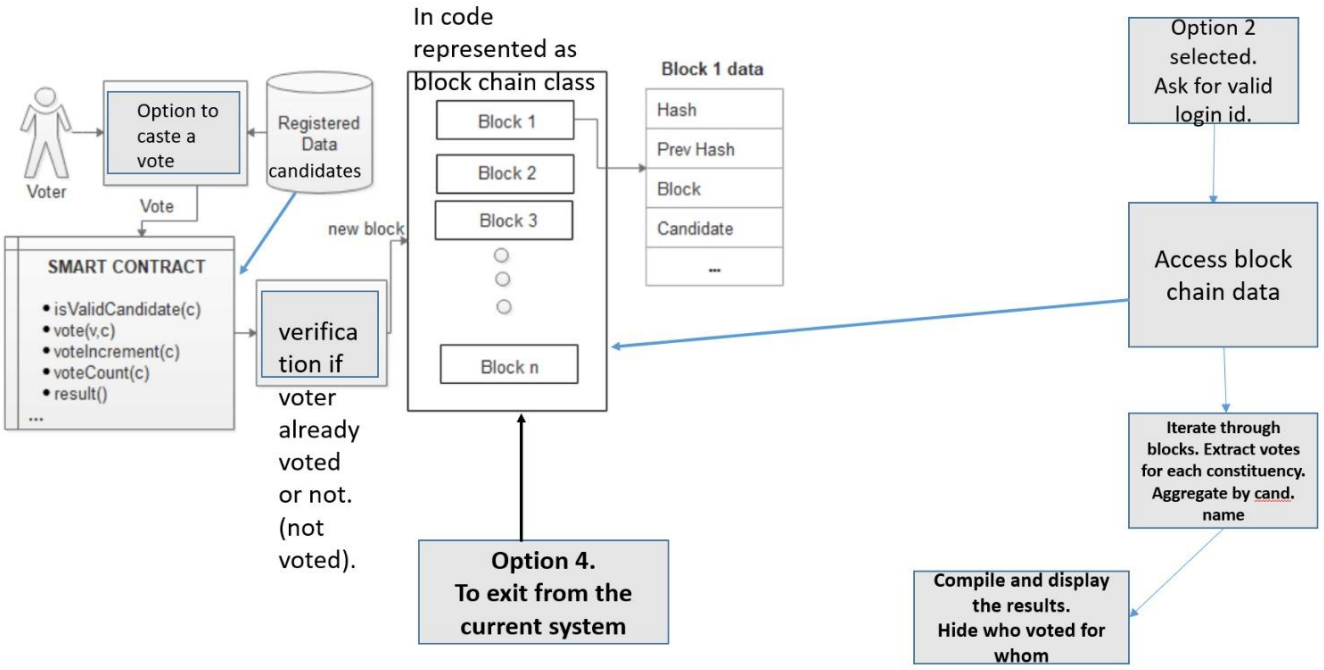
Blockchain Based E-Voting Systems:

Electronic voting is a voting technique in which votes are recorded or counted using electronic equipment. Electronic voting is usually defined as voting that is supported by some electronic hardware and software.

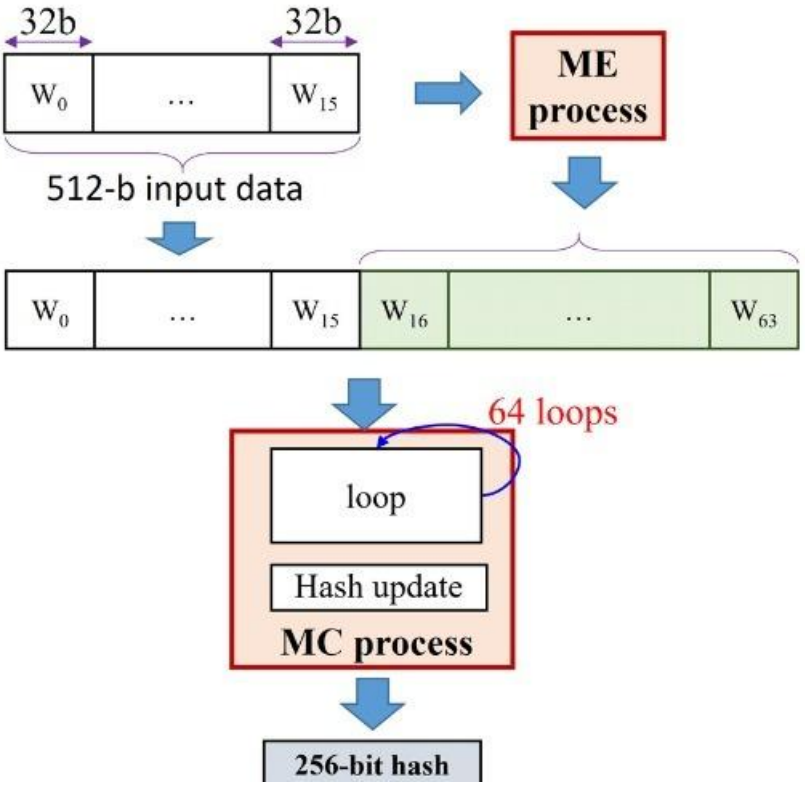


- Voter needs to enter his/her credentials in order to vote.
- All data is then encrypted and stored as a transaction. This transaction is then broadcasted to every node in network, which in turn is then verified.
- If network approves transaction, it is stored in a block and added to chain.

BLOCK DIAGRAMS:



SHA-256:



Secure Hash Algorithm (SHA-256):

A cryptographic hash function is a mathematical algorithm that takes an input (or message) and produces a fixed-size string of characters, which is typically a sequence of numbers and letters. The resulting output is called the hash value or hash code.

The key characteristics of a cryptographic hash function, including SHA-256, are as follows:

- **Deterministic:** For a given input, the hash function will always produce the same hash value.
- **Fast Computation:** Hash functions are designed to be computationally efficient, allowing them to quickly process data.
- **Preimage Resistance:** Given a hash value, it should be computationally infeasible to reverse the process and determine the original input.
- **Collision Resistance:** It should be highly unlikely for two different inputs to produce the same hash value.
- **Avalanche Effect:** A small change in the input data should result in a significantly different hash value.

One of the primary use cases of SHA-256 is in securing the integrity of data. It ensures that even a minor change in the input data will result in a substantially different hash value, making it a valuable tool for verifying the authenticity and integrity of information.

How SHA-256 Is used in Blockchain based E-Voting System?

In a blockchain-based e-voting system, SHA-256 is typically used in the following ways:

- **Data Integrity:**
  - When a voter casts their vote, their choice is represented as data. This data is hashed using SHA-256, creating a fixed-size string of characters that represents the unique fingerprint of the voter's choice.
  - This hash is stored on the blockchain to ensure the integrity of the vote.
  - If anyone tries to tamper with the vote data, the hash value would change, alerting the system to potential manipulation.

- **Block Validation:**
  - In a blockchain, data is grouped into blocks, and each block contains a reference to the previous block's hash. SHA-256 is used to calculate and store the hash of each block.
  - This ensures the immutability and integrity of the entire blockchain, making it extremely difficult for malicious actors to alter the historical record of votes.
- **Verification of Votes:**
  - When a voter wants to verify that their vote was counted correctly, they can hash their vote data using SHA-256 and compare it to the hash stored on the blockchain.
  - If the two hashes match, it means their vote was not altered or tampered with.
- **Identity Verification:**
  - In some cases, SHA-256 can be used to secure the identity of voters by hashing sensitive information like personal identification numbers (PINs) or biometric data.
  - The hash is stored on the blockchain rather than the actual data to protect voter privacy.

It's important to note that while SHA-256 is a crucial component in ensuring the security and integrity of a blockchain-based e-voting system.

CODE:

Client-Server Approach:

SERVER CODE:

```
from blockchain_voting import *
import socket
import threading
import json

# Server setup
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('localhost', 5555))
server.listen()
# Function to handle client connections
def handle_client(conn, addr):
 while True:
  data = conn.recv(1024).decode()
  if not data:
      conn.close()
      break
# Parse vote data
  try:
      vote = json.loads(data)
      #print(f'Received vote from client: {vote}')
  except json.JSONDecodeError:
      print("Failed to parse vote data.")
      continue
  voter_id = vote['voter_id']
  constituency = vote['constituency']
  candidate_id = vote['candidate_id']
   # Check if voter has already voted
  if voter_id in blockchain.voter_ids:
       conn.send('You have already casted a vote.'.encode())
       continue
   # Check if constituency is valid
  if constituency not in ['A', 'B', 'C']: # Add more constituencies as
needed
       conn.send('Invalid constituency. Please try again.'.encode())
       continue
   # Check if candidate is valid
  if candidate_id not in ['1', '2', '3']: # Add more candidates as needed
       conn.send('Invalid candidate ID. Please try again.'.encode())
       continue
   # Add vote to blockchain
  blockchain.new_vote(voter_id, vote)
  blockchain.increment_candidate_votes(constituency, candidate_id)
  blockchain.voter_ids.add(voter_id) # Mark voter ID as used
   # Confirm vote
  conn.send('Vote confirmed.'.encode())
  # wait until client does the work
```

```
  time.sleep(4)
  confirmation = 'Vote confirmed.'
  print(f'Sending confirmation to client: {confirmation}')
  conn.send(confirmation.encode())
   # Broadcast updated blockchain to all clients
  for client in clients:
       client.send(str(blockchain.chain).encode())
# List to keep track of all connected clients
clients = []
while True:
 conn, addr = server.accept()
 clients.append(conn)
 threading.Thread(target=handle_client, args=(conn, addr)).start()
```

CLIENT CODE:

```
import socket
import time
from blockchain_voting import *

# Client setup
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('localhost', 5555))
# Voting process
voter_id = input('Enter your voter ID: ')
constituency = voter_id[:1].upper()
if constituency not in ['A', 'B', 'C']: # Add more constituencies as
needed
  print('Invalid constituency. Please try again.')
else:
  candidates = {}
  if constituency == 'A':
      candidates = {
          '1': 'Candidate A',
          '2': 'Candidate B',
          '3': 'Candidate C'
      }
  elif constituency == 'B':
      candidates = {
          '1': 'Candidate X',
          '2': 'Candidate Y',
          '3': 'Candidate Z'
      }
  else: # Default to 'C' constituency
      candidates = {
          '1': 'Candidate D',
          '2': 'Candidate E',
          '3': 'Candidate F'
      }
  print(f'\nCandidates for constituency {constituency}:')
  for candidate_id, candidate_name in candidates.items():
      print(f'{candidate_id}: {candidate_name}')
  vote_casted = False
```

```
  while not vote_casted:
      candidate_id = input('Enter the candidate ID you want to vote for
(or type "exit" to finish): ')
      if candidate_id.lower() == 'exit':
          break
      if candidate_id in candidates:
          vote = {'voter_id': voter_id, 'constituency': constituency,
'candidate_id': candidate_id}
          vote_json = json.dumps(vote)
          # Send vote to server
          client.send(vote_json.encode())
          print('Vote sent to server. Waiting for confirmation...')
          confirmation = ''
          while not confirmation:
              chunk = client.recv(1024).decode()
              if chunk:
                  confirmation = chunk
              else:
                  break
          print(f'Confirmation received from server: {confirmation}')
          if confirmation == 'Vote confirmed.':
              print('Vote confirmed by server!')
              vote_casted = True
          else:
              print('Vote not confirmed by server. Please try again.')
      else:
          print('Invalid candidate ID. Please try again.')
# Receive updated blockchain
blockchain = client.recv(1024).decode()
print(f"Updated blockchain: {blockchain}")
client.close()
```

Block-chain Voting:

```
import hashlib
import json
import time

class Blockchain:
    def __init__(self):
        self.chain = []
        self.current_votes = []
        self.nodes = set()
        self.voter_ids = set()  # Set to keep track of used voter IDs
        self.candidates = {}  # {constituency: {candidate_id: {'name': candidate_name,
'votes': vote_count}}}
 # Genesis block
        self.new_block(proof=100, previous_hash='1')
def new_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'votes': self.current_votes,
            'proof': proof,
```

```
            'previous_hash': previous_hash,
        }
        self.current_votes = []
        self.chain.append(block)
        return block
def hash(self, block):
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
def proof_of_work(self, last_proof):
        proof = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1
        return proof
def valid_proof(self, last_proof, proof):
        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == '0000'
def new_vote(self, person_id, vote):
        self.current_votes.append({
            'voter_id': hashlib.sha256(str(person_id).encode()).hexdigest(),
            'vote': vote,
        })
        self.chain.append(self.current_votes)
def confirm_vote(self):
        confirmation = input('Do you want to cast your vote? (yes/no): ').lower()
        return confirmation == 'yes'
def increment_candidate_votes(self, constituency, candidate_id):
        if constituency not in self.candidates:
            self.candidates[constituency] = {}
if constituency == 'A':
            candidate_names = ['Candidate A', 'Candidate B', 'Candidate C']
        elif constituency == 'B':
            candidate_names = ['Candidate X', 'Candidate Y', 'Candidate Z']
        else:  # Default to 'C' constituency
            candidate_names = ['Candidate D', 'Candidate E', 'Candidate F']
 if candidate_id not in self.candidates[constituency]:
            self.candidates[constituency][candidate_id] = {'name':
candidate_names[int(candidate_id) - 1], 'votes': 1}
        else:
            self.candidates[constituency][candidate_id]['votes'] += 1
def election_commission_view(self):
        login_id = input('Enter your login ID: ')
        if login_id != '55':
            return 'Access denied. Invalid login ID.'
constituency = input('Enter the constituency to view results: ')
        if constituency not in self.candidates:
            return f'No votes recorded for constituency: {constituency}'
 votes = self.candidates[constituency]
        result = f'Total votes for constituency {constituency}:\n'
        for candidate_id, candidate_info in votes.items():
            result += f'Candidate {candidate_id}: {candidate_info["name"]} - Votes:
{candidate_info["votes"]}\n'
        return result
 def view_blockchain(self):
        return self.chain
# Example usage:
# Initialize blockchain
blockchain = Blockchain()
while True:
    print('\nOptions:')
    print('1. Cast Vote')
    print('2. Election Commission View')
    print('3. View Blockchain')
    print('4. Exit')
```

```
choice = input('Enter your choice (1/2/3/4): ')
if choice == '1':
        voter_id = input('Enter your voter ID: ')
        constituency = voter_id[:1].upper()
if voter_id in blockchain.voter_ids:
            print('You have already casted a vote. Exiting...')
            break
if constituency not in ['A', 'B', 'C']:  # Add more constituencies as needed
            print('Invalid constituency. Please try again.')
            continue
 candidates = {}
if constituency == 'A':
            candidates = {
                '1': 'Candidate A',
                '2': 'Candidate B',
                '3': 'Candidate C'
            }
        elif constituency == 'B':
            candidates = {
                '1': 'Candidate X',
                '2': 'Candidate Y',
                '3': 'Candidate Z'
            }
        else:  # Default to 'C' constituency
            candidates = {
                '1': 'Candidate D',
                '2': 'Candidate E',
                '3': 'Candidate F'
            }
 print(f'\nCandidates for constituency {constituency}:')
        for candidate_id, candidate_name in candidates.items():
            print(f'{candidate_id}: {candidate_name}')
 vote_casted = False
        while not vote_casted:
            candidate_id = input('Enter the candidate ID you want to vote for (or type
"exit" to finish): ')
            if candidate_id.lower() == 'exit':
                break
            if candidate_id in candidates:
                if blockchain.confirm_vote():
                    blockchain.new_vote(voter_id, {'constituency': constituency,
'candidate_id': candidate_id})
                    blockchain.increment_candidate_votes(constituency, candidate_id)
                    blockchain.voter_ids.add(voter_id)  # Mark voter ID as used
                    print('Vote casted successfully!')
                    vote_casted = True
                else:
                    print('Vote cancelled.')
            else:
                print('Invalid candidate ID. Please try again.')
elif choice == '2':
        print(blockchain.election_commission_view())
 elif choice == '3':
        print('Blockchain:', blockchain.view_blockchain())
 elif choice == '4':
        print('Exiting...')
        break
else:
        print('Invalid choice. Please try again.')
```

SHA - 256 :

```
import tkinter as tk
from tkinter.scrolledtext import ScrolledText
from tkinter.font import Font

from ctypes import windll
windll.shcore.SetProcessDpiAwareness(1)
from tkinter import messagebox
K = [
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe,
0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa,
0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb,
0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624,
0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb,
0xbef9a3f7, 0xc67178f2
]
def generate_hash(message: bytearray) -> bytearray:
    """Return a SHA-256 hash from the message passed.
    The argument should be a bytes, bytearray, or
    string object."""

    if isinstance(message, str):
        message = bytearray(message, 'ascii')
    elif isinstance(message, bytes):
        message = bytearray(message)
    elif not isinstance(message, bytearray):
        raise TypeError
# Padding
    length = len(message) * 8 # len(message) is number of BYTES!!!
    message.append(0x80)
    while (len(message) * 8 + 64) % 512 != 0:
        message.append(0x00)
message += length.to_bytes(8, 'big') # pad to 8 bytes or 64 bits
assert (len(message) * 8) % 512 == 0, "Padding did not complete properly!"
# Parsing
    blocks = [] # contains 512-bit chunks of message
    for i in range(0, len(message), 64): # 64 bytes is 512 bits
        blocks.append(message[i:i+64])
    # Setting Initial Hash Value
    h0 = 0x6a09e667
    h1 = 0xbb67ae85
    h2 = 0x3c6ef372
    h3 = 0xa54ff53a
    h5 = 0x9b05688c
    h4 = 0x510e527f
    h6 = 0x1f83d9ab
    h7 = 0x5be0cd19
    # SHA-256 Hash Computation
    for message_block in blocks:
```

```
# Prepare message schedule
        message_schedule = []
        for t in range(0, 64):
            if t <= 15:
                # adds the t'th 32 bit word of the block,
                # starting from leftmost word
                # 4 bytes at a time
                message_schedule.append(bytes(message_block[t*4:(t*4)+4]))
            else:
                term1 = _sigma1(int.from_bytes(message_schedule[t-2], 'big'))
                term2 = int.from_bytes(message_schedule[t-7], 'big')
                term3 = _sigma0(int.from_bytes(message_schedule[t-15], 'big'))
                term4 = int.from_bytes(message_schedule[t-16], 'big')
                # append a 4-byte byte object
                schedule = ((term1 + term2 + term3 + term4) % 2**32).to_bytes(4,
'big')
                message_schedule.append(schedule)
      assert len(message_schedule) == 64
        # Initialize working variables
        a = h0
        b = h1
        c = h2
        d = h3
        e = h4
        f = h5
        g = h6
        h = h7
        # Iterate for t=0 to 63
        for t in range(64):
            t1 = ((h + _capsigma1(e) + _ch(e, f, g) + K[t] +
                   int.from_bytes(message_schedule[t], 'big')) % 2**32)
           t2 = (_capsigma0(a) + _maj(a, b, c)) % 2**32
            h = g
            g = f
            f = e
            e = (d + t1) % 2**32
            d = c
            c = b
            b = a
            a = (t1 + t2) % 2**32
        # Compute intermediate hash value
        h0 = (h0 + a) % 2**32
        h1 = (h1 + b) % 2**32
        h2 = (h2 + c) % 2**32
        h3 = (h3 + d) % 2**32
        h4 = (h4 + e) % 2**32
        h5 = (h5 + f) % 2**32
        h6 = (h6 + g) % 2**32
        h7 = (h7 + h) % 2**32
    return ((h0).to_bytes(4, 'big') + (h1).to_bytes(4, 'big') +
            (h2).to_bytes(4, 'big') + (h3).to_bytes(4, 'big') +
            (h4).to_bytes(4, 'big') + (h5).to_bytes(4, 'big') +
            (h6).to_bytes(4, 'big') + (h7).to_bytes(4, 'big'))
def _sigma0(num: int):
    """As defined in the specification."""
    num = (_rotate_right(num, 7) ^
           _rotate_right(num, 18) ^
           (num >> 3))
    return num
def _sigma1(num: int):
    """As defined in the specification."""
    num = (_rotate_right(num, 17) ^
```

```
            _rotate_right(num, 19) ^
            (num >> 10))
    return num
def _capsigma0(num: int):
    """As defined in the specification."""
    num = (_rotate_right(num, 2) ^
           _rotate_right(num, 13) ^
           _rotate_right(num, 22))
    return num
def _capsigma1(num: int):
    """As defined in the specification."""
    num = (_rotate_right(num, 6) ^
           _rotate_right(num, 11) ^
           _rotate_right(num, 25))
    return num
def _ch(x: int, y: int, z: int):
    """As defined in the specification."""
    return (x & y) ^ (~x & z)
def _maj(x: int, y: int, z: int):
    """As defined in the specification."""
    return (x & y) ^ (x & z) ^ (y & z)
def _rotate_right(num: int, shift: int, size: int = 32):
    """Rotate an integer right."""
    return (num >> shift) | (num << size - shift)
def compute_hash(event=None):
   message = message_entry.get()
   hash_value = generate_hash(message)
   add_text_to_output(hash_value.hex())
#    messagebox.showinfo("SHA-256 Hash", hash_value.hex())
def add_text_to_output(text):
    output.configure(state="normal")
    output.delete("1.0", tk.END)
    output.insert(tk.INSERT, text)
    output.configure(state="disabled")
# Create the main window
root = tk.Tk()
root.title("SHA256 Generator")
root.geometry('500x500')
root.configure(bg='pink')
def_font = Font(weight='bold')
def_font.config(size=18)
# Create a label and an entry field for the message
message_label = tk.Label(root, text="Enter the message:", font=def_font, bg='pink')
message_entry = tk.Entry(root)
message_entry.bind("<Return>", compute_hash)
message_entry.focus_set()
# Create a button to compute the SHA-256 hash
hash_button = tk.Button(root, text="Compute SHA-256 Hash", command=compute_hash,
bg='orchid')
# create empty label for output
output_label = tk.Label(root, text="SHA256 Value:", font=def_font, bg='pink')
output = ScrolledText(root, width=40, height=40)
output.configure(state="disabled")
# Pack the widgets
message_label.pack(pady = 10)
message_entry.pack()
hash_button.pack(pady = 10)
output_label.pack()
output.pack(fill=tk.BOTH, side=tk.LEFT, expand=True, padx=10, pady=10)
# Run the main loop
root.mainloop()
```

Conclusion:

- In conclusion, our exploration of the blockchain-based e-voting system underscores the transformative potential of this technology in modernizing elections.
- We have seen how blockchain addresses the challenges of security, transparency, and accessibility that plague traditional voting systems.
- Key takeaways:
- Enhanced security through decentralization and cryptographic techniques.
- Unprecedented transparency, allowing voters to independently verify their votes.
- Increased inclusivity, accommodating all eligible voters.
- Efficient and real-time results tabulation.
- Strengthened trust in the electoral process.

References:

https://ieeexplore.ieee.org/document/9498566

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8434614/

https://www.sciencedirect.com/science/article/pii/S1319157822002221https://www.itm-conferences.org/articles/itmconf/pdf/2020/02/itmconf_icacc2020_03001.pdf

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.quora.com%2FWhat-is-the-difference-between-SHA-256-and-Scrypt&psig=AOvVaw2wNIT0EuARqhxi8N-rweGp&ust=1699265917501000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCLiVhtfSrIIDFQAAAAAdAAAAABAD