5

Blockchain Applications

The digital transformation of cities into intelligent hubs of connectivity, often termed 'Smart Cities,' is no longer a vision of the distant future but a rapidly unfolding reality. Interconnected systems like the Internet of Things (IoT), Intelligent Transportation Systems (ITS), and Smart Grids are at the heart of this transformation. These systems, while revolutionary, are not devoid of vulnerabilities. As cities become smarter, the security of their digital infrastructure becomes paramount. Blockchain technology is a decentralized ledger system that promises to strengthen the security framework of these smart city applications, which are given below.

- **Internet of Things (IoT)** encompasses a vast array of devices, from surveillance cameras to smart thermostats, all interconnected and communicating in real-time. While this interconnectivity offers unparalleled convenience and efficiency, it also opens up a Pandora's box of security challenges. Unauthorized access, data breaches, and device tampering are just a few of the myriad threats that loom large over IoT ecosystems. Blockchain, with its immutable records and consensus-driven validation processes, offers a robust shield against many of these threats, ensuring data integrity and thwarting unauthorized access.
- **Intelligent Transportation Systems (ITS)** are revolutionizing urban mobility, optimizing traffic flow, reducing congestion, and enhancing the overall transportation experience. However, the data-driven nature of ITS, which relies on continuous data exchange between vehicles, traffic lights, sensors, and more, is a potential goldmine for hackers. Blockchain can play a pivotal role here. By decentralizing and encrypting transportation data, blockchain can prevent malicious attacks, tampering, and unauthorized data access, ensuring a seamless and secure transportation network.
- **Smart Grids** represent the next frontier in energy management, allowing for dynamic energy distribution based on real-time demand and supply. Yet, they are susceptible to energy theft, unauthorized grid access, and data falsification. Blockchain's decentralized nature can transform the energy sector by ensuring transparent, tamper-proof energy distribution and consumption records. Moreover, blockchain can facilitate peer-to-peer energy trading, allowing consumers to sell excess energy back to the grid or other consumers, all in a secure environment.

5.1 SOLUTION FOR IOT APPLICATION

Using blockchain in IoT applications significantly enhances security by providing a tamper-resistant ledger that records all transactions, including sensitive messages, making it difficult for unauthorized entities to alter data. Blockchain also enables secure authentication and authorization processes, which further reduce the risks of device impersonation and data breaches. Additionally, by integrating blockchain into IoT applications, these systems are transformed from centralized to decentralized, eliminating the risk of a single point of failure—a particularly important benefit in IoT environments where devices are widely distributed. Furthermore, the immutability feature of blockchain ensures that information recorded from IoT devices is accurate and trustworthy, which is critical for applications such as smart cities and healthcare. The consensus algorithms commonly used in blockchain, such as PoW, PoS, and PBFT, are energy-intensive and introduce latency, resulting in low throughput. This makes them unsuitable for real-time IoT applications that require fast data processing and immediate responses. The energy and resource-intensive nature of these algorithms can be particularly challenging for the low-powered devices typically found in IoT applications. Therefore, we have proposed two security methods for IoT applications. First, BLISS, a blockchain based integrated security system for IoT applications, and second, SCAB-IoTA, a secure communication and authentication for IoT applications. The second method actually enhances the first method. In the first method, we apply smart contracts to create secure clusters of IoT devices so that they can communicate securely in a trusted environment. In the second approach, we introduced the angular distance to measure the distance if the device is movable. Apart from the angular distance, the SCAB-IoTA also uses the hybrid cryptosystem, i.e., ECC and AES, for secure authentication and communication. Before delving into the details of both schemes, we first define why we need blockchain in IoT, our motivation and key contributions, the system model, and the adversary model.

5.1.1 Motivation and Contribution

Security is the main issue that restricts the adoption of IoT in social life. However, many researchers have been working to make the IoT a more reliable and secure technology so that it can be entirely adopted in society to make human life more manageable and convenient. Since researchers and developers develop many schemes and methods, but due to IoT devices' constrained environment (i.e., low computational power, low energy), those techniques are not feasible. Thus, an efficient and secure information transmission scheme is required to exchange meaningful information in IoT applications. Blockchain technology is considered a powerful tool for enabling trusted interactions between various components in a decentralized way. Despite this, some challenges emerge during the integration of blockchain technology with IoT, such as high computation, storage, and communication costs. Therefore, focusing on secure communication, efficient authentication, and performance metrics, we proposed an efficient security scheme for secure communication and authentication that is based on a blockchain scheme called BLISS and SCAB-IoTA. The major contributions are as follows:

- The first methodology we have proposed is BLISS, a resource-friendly integrated security solution for IoT applications using blockchain technology and smart contracts. The smart contracts are to be implemented as a rule for cluster association and secure communication for IoT devices. We have created a testbed using 5 Raspberry Pi 4 modules and a computer system also referred as desktop. The Raspberry Pi 4 is used as an IoT device, and the desktop is a cluster head. Apart from the testbed, we have also deployed the scheme on the Sepolia Ethereum testnet, using Remix IDE for development

and MetaMask for network connectivity. This strategic shift to the Sepolia testnet allows us to better evaluate our system under conditions that closely mirror the Ethereum mainnet. Sepolia provides a realistic environment for testing the latency and overall performance of blockchain transactions, which is crucial for assessing the feasibility and efficiency of the BLISS scheme in a real-world scenario.

- The second methodology we have proposed is SCAB-IoTA, a blockchain-based lightweight authentication and secure communication scheme for IoT. We have designed an algorithm for the generation of a one-time session secret key for secure communication between devices. Furthermore, Unlike BLISS, the secure clusters are created based on the angular distance (AD), i.e., they are to be at a certain distance from the controller (called cluster radius R). Similar to BLISS, we have created a testbed for SCAB-IoTA using multiple Raspberry Pi devices and computer systems to show the feasibility of the proposed scheme.

- The performance of both the proposed schemes is measured based on the four parameters, i.e., computation cost, energy consumption, storage cost, and communication cost. We have also provided a comparative analysis with state-of-the-art schemes to show the performance improvement.

- We also provide a security analysis against various cyberattacks such as identification attacks, Sybil attacks, fake node injection attacks, tampering attacks, message replay attacks, man-in-the-middle attacks, botnet attacks, spoofing attacks, and message integrity attacks. The Random Oracle Model (ROM) is also adopted to show the system's security proof for authentication, integrity, and non-repudiation.

5.1.2 System Model

In the proposed mechanism, there are three types of devices: Blockchain Node (BN), Cluster Head (CH), and Cluster Member (M). The BN manages the record of data, which is shared with members of the cluster. All the CHs have an identical copy of the blockchain. The CH has high processing power and efficiency amongst all the cluster's devices (cluster members). Due to this property, the CH controls every cluster thus, we name it as the controller. The functions of the controller device are the authentication of candidates, block creation, and maintaining the blockchain. The controller plays an essential role in the candidate authentication process. The initiation of communication between two candidates is only possible when they belong to the same secure cluster. The IoT devices are smart devices, such as smart bulbs, smart speakers, specialized medical equipment, air-conditioners, and many others. Every cluster is created with a verification process so that no malicious devices can be a part of the cluster, and all the cluster members, referred to as candidates, can communicate with each other in a secure environment. The candidate first needs to send the join request to the controller. After authenticating the candidate, the controller sent some secret credentials to the candidate. So, only candidates having these credentials are called authentic devices. This exchange of information is called a request-response communication. All types of communication are secured with ECDSA (192-bit) and SHA256, which makes forging communication challenging. Every piece of information exchanged must be written into blocks to become a permanent record, including the candidate's authentication information. In the subsequent sections, we have briefly described all the phases. The used mathematical symbols are listed in Table 5.1. Figure 5.1 shows our scheme's framework and the interaction between components. There are three controller devices with ID Ctr_{11}, Ctr_{22}, and Ctr_{33} with cluster IDs Cst_1, Cst_2, and Cst_3, respectively. Ctr_{11} is connected with 4 candidate devices. The request to join the cluster or communicate with other candidate devices is shown in the dashed red arrow color, and

the blue arrow shows the response from the controller to the respective candidate device. We have used the ECDSA for public-key cryptography, and a 192-bit AES is used with secret-key S_K. The BN generates the key-pair (BK_{pub}, BK_{pri}) at timestamp T_1, and the controller generates the key-pair (Ctr_{pub}, Ctr_{pri}) at timestamp T_2. The public key of the BN (BK_{pub}) is shared with the controller because only the controller has the right to create the block and store it on the blockchain node. The public key of the controller (Ctr_{pub}) is shared with the candidate device. The secret key is the one-time session key, which is created and assigned by the controller to the candidate device (C_i).

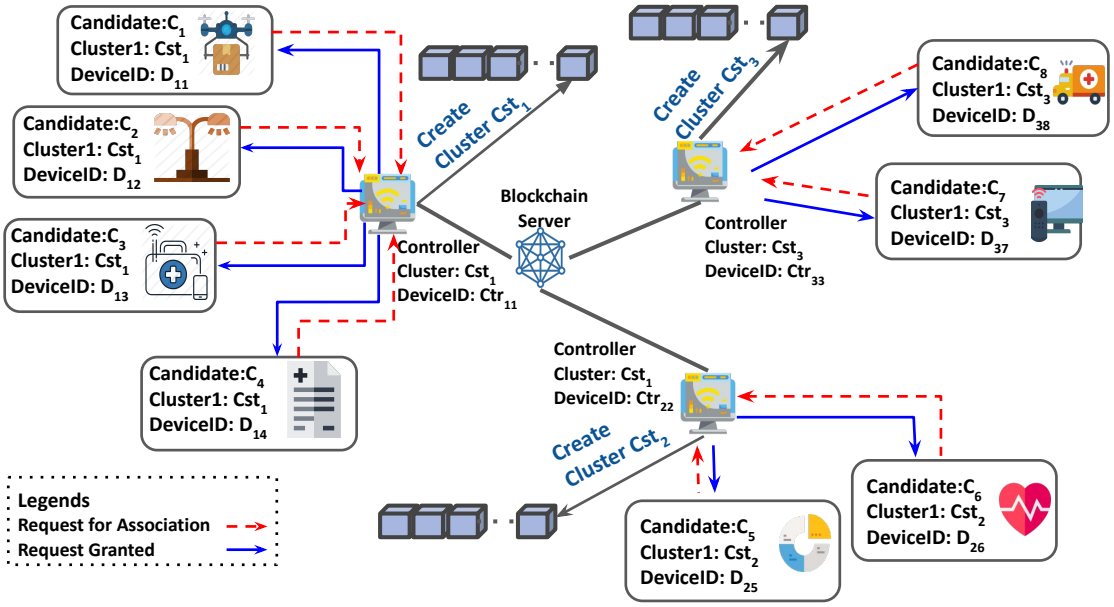


Figure 5.1: System model for IoT application

Table 5.1: Symbol table for BLISS and SCAB-IoTA

Symbol	Description	Symbol	Description
σ_x	Digital Signature of x	$H(x)$	Hash (SHA256) function for the input x
C_i	i^{th} candidate device	D_{ji}	ID of i^{th} candidate device in j^{th} cluster
C^i_{addr}	Unique address (HEX code) of i^{th} candidate device	Cst_i	i^{th} cluster device
Ctr_{id}	Controller's ID	$\triangle T$	Threshold time
C^i_{pub}, C^i_{pri}	Public-private Key-pair of candidate device $\forall$ i = 1,2,...	Ctr_{pub}, Ctr_{pri}	Public-private Key-pair of controller device
BK_{pub}, BK_{pri}	Public-private Key-pair of Blockchain Node	$Req[x]$	Request message with parameter x
$Res[y]$	Response message with parameter y	$M_1, M_2, ..$	Messages
$Tr_1, Tr_2, ...$	Transactions	$Mrt()$	A function representation to create Merkle root

5.1.3 Adversary Model

Adversary modeling is used to identify potential threats or weaknesses in a system and build countermeasures. With adversary modeling, defenders may see what defences are needed, the system's nature, the attacker's profile, and their intended assets. An adversary model is a set of assumptions and proof methods used to assess a test protocol's security. Our adversary model is based on the well-known Dolev-Yao (DY) adversary model [145]. The DY model can express all conceivable attacker actions in a compact manner. This aspect of the DY model makes it a widely used adversary model. In other words, the DY model con-

siders end-users as secure and trusted entities. In the proposed schemes, the information is shared between the candidate and controller over a public channel. Thus, we make some assumptions for the security and performance of the system according to the DY adversary model mentioned above. The properties of the adversary are given below.

1. The adversary $\mathbb{A}$ knows the communication channel is open, and it can compute the secret values if $\mathbb{A}$ has private credentials.
2. Consider an authorized controller performing some malicious activities, e.g., trying to put a fake transaction in the block or a fake block. However, smart contracts take care of this.
3. A legitimate candidate can act as an adversary $\mathbb{A}$ to make a DoS attack by sending multiple fake transactions. However, any transaction is included in the processing after the verification by smart contracts.
4. An adversary can compromise the confidentiality and integrity of the shared transactions in the communication channel. However, all communication between devices is secured using a digital signature (ECDSA) and the hash algorithm (SHA256). The private key is assumed to be unknown to the adversary; hence, an adversary cannot re-sign the transactions.
5. An adversary can not break the cryptographic algorithms (ECDSA) digital signature and one-way hash) and is considered secure.

5.1.4 BLISS: Blockchain-based Integrated Security System for Internet of Things (IoT) Applications

The BLISS is a blockchain-enabled secure storage and communication method to provide integrity to the information and authentication to the participating devices of IoT. BLISS has two phases: first, authentication and cluster creation, and second, candidate-to-candidate communication. A controller must authenticate all the candidates who want to join the cluster and want to interact with each other. All the controllers synchronize to maintain the consistency of the blockchain. Candidates need to be authenticated by a controller to join the cluster. The controller has the following functions: authentication of devices, generating genesis blocks, and mining new blocks. Candidate devices who wish to join the cluster and are waiting to be authenticated with the controller send an authentication request to the controller. After authenticating the device, the controller sends its public key and a signed token to the candidate. So, only those devices that have a signed token are called authentic candidates. Every exchanged information must be written into blocks to become a permanent record, including the candidate's authentication information. Each block contains records or critical information and the hash code of the previous block. The avalanche effect of a hash function states that a minimal change to the input could produce a significant change to the output. Therefore, tampering with information in any block is challenging or nearly impossible.

Authentication and Cluster Creation

The authentication of candidate devices and the creation of a secure cluster of candidate devices are discussed in this section. For the authentication of the device, the controller generates a signed token signed by its private key for every requesting device. As the controller's private key is unknown to anyone, only the controller can generate the token. Hence, no adversary can generate the token, which reduces the chances

of a fake controller device. The requesting candidate device receives the response from the controller after sending the cluster joining request. The candidate device retrieves the confidential information and keeps it for future communication. Figure 5.2 shows the authentication and cluster creation process. In the diagram, the red lines show the requesting line, and the blue line shows the response with the signed token. The process of authentication and cluster creation is described below. Before creating the request to join the

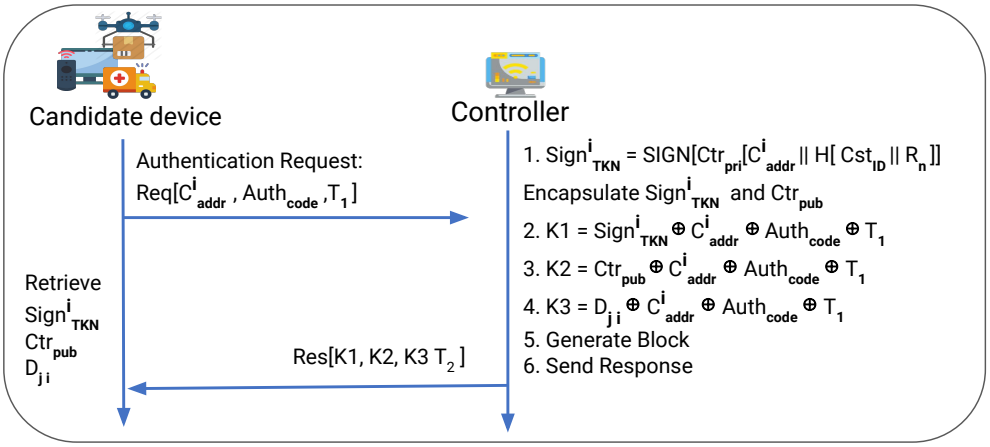


Figure 5.2: Device authentication process in BLISS

cluster, candidate C_i generates its public-private keypair (C^i_{pub}, C^i_{pri}) using the ECDSA as following:

Let G be the generator polynomial on the ECC curve, and select a finite integer number F_n.

1. First it select a pseudo-random integer int in the interval $[1, F_n - 1]$
2. Compute, $Q = int \times G$.
3. C_i's public key C^i_{pub} is Q; and C_i's private key C^i_{pri} is int.

After generating the key pair, it creates its unique address as

$$C^i_{addr} = H(C^i_{pub} \oplus F_n) \tag{5.1}$$

The unique address uniquely identifies every device, whether a candidate or a controller device. After generating the key pair and unique address, it calculates

$$Auth^i_{code} = H(C^i_{addr} \oplus n) \tag{5.2}$$

where $Auth^i_{code}$ is the authentication code, and n is the nonce value. Now, it's time to create the join request. To do this, C_i creates the request as

$$Req[C^i_{addr} || Auth^i_{code} || T_1] \tag{5.3}$$

The request contains the candidate address, $Auth^i_{code}$, and the current timestamp T_1. The $Auth^i_{code}$ is used to check the authenticity of the controller's response. It then sends this request to the controller. If it is the first request from any candidate device, the controller generates the cluster ID, Cst_{ID}

1. $R_n = Math.rand()$

2. Cst_{ID}= $H(R_n||Ctr_{pub})$

The cluster ID is the hash of the random number R_n and the public key of the controller, Ctr_{pub}. Using this method, the cluster ID is unique among all the clusters. After creating the cluster ID, it creates the signed token for the requesting candidate as

$$Sign^i_{TKN} = SIGN[Ctr_{pri}[C^i_{addr}||H[Cst_{ID}||T_2]]] \tag{5.4}$$

The signed token is used to verify whether the candidate belongs to the cluster or not because the token of one cluster is not valid for another cluster. Every cluster will generate a different signed token. This is how it is ensured whether the device belongs to the cluster or not. After creating the signed token, it computes two parameters $K1$ and $K2$ as,

1. $K1 = Ctr_{pub} \oplus C^i_{addr} \oplus T_1 \oplus Auth^i_{code}$
2. $K2 = Sign^i_{TKN} \oplus C^i_{addr} \oplus T_1 \oplus Auth^i_{code}$
3. $K3 = D_{ji} \oplus C^i_{addr} \oplus T_1 \oplus Auth^i_{code}$

The $K1$. $K2$, $K3$ are the intermediate variables used to communicate the controller's public key (Ctr_{pub}), signed token ($Sign^i_{TKN}$), and cluster membership ID (D_{ji}) to the requesting candidate device. The controller also generates the block as the authentication block, which can be used by the other controllers when the device leaves one cluster and joins another cluster. If it is the first block of the system, it generates the genesis block, where the hash of the previous block PBH is set to zero. Otherwise, get the hash of the previous block and generate the block as

$$block = createBlock[T_1, PBH, BH, H(C^i_{addr}), H(D_{ji})] \tag{5.5}$$

The block is appended to the blockchain and sent to the other controllers so that every cluster has a consistent copy of the blockchain. The controller sends $< K1, K2, K3, T_2 >$ to the requesting candidate C^i_{addr}. After receiving the response candidate, C_i first check $\triangle T \geq T_2 - T_1$. If true, it retrieves the secret information from the $K1$ and $K2$ as follows and stores them for future communication.

1. $Get[Ctr_{pub}] = K1 \oplus C^i_{addr} \oplus T_1 \oplus Auth^i_{code}$ = $(Ctr_{pub} \oplus C1_{addr} \oplus T_1 \oplus Auth_{code}) \oplus C1_{addr} \oplus T_1 \oplus Auth_{code}$
   = $(Ctr_{pub} \oplus C1_{addr} \oplus T_1) \oplus C1_{addr} \oplus T_1$
   = $(Ctr_{pub} \oplus C1_{addr}) \oplus C1_{addr}$
   = (Ctr_{pub})
2. $Get[Sign^i_{TKN}] = K2 \oplus C^i_{addr} \oplus T_1 \oplus Auth^i_{code}$ = $(Sign^i_{TKN} \oplus C1_{addr} \oplus T_1 \oplus Auth_{code}) \oplus C1_{addr} \oplus T_1 \oplus Auth_{code}$
   = $(Sign^i_{TKN} \oplus C1_{addr} \oplus T_1) \oplus C1_{addr} \oplus T_1$
   = $(Sign^i_{TKN} \oplus C1_{addr}) \oplus C1_{addr}$
   = $(Sign^i_{TKN})$
3. $Get[D_{ji}] = K3 \oplus C^i_{addr} \oplus T_1 \oplus Auth^i_{code}$

Now, candidate C_i and the controller are both connected. Similar process is followed for all the devices, i.e., $i = 1, 2, 3, ...$

To join the cluster, the candidate device must execute the smart contract defined in the *Algorithm* 10. The uniqueness of the candidate and controller device is verified using the smart contract. The joining is aborted if any of the rules are violated in the contract. The *Algorithm* 10 defines the contract for joining the cluster. This contract is executed on the controller's end. First, it checks the identity of the controller receiving the cluster joining request from the candidate device. Second, it checks the identity of the candidate device that sent the joining request, and the third condition is to check the signed token of the candidate, which the candidate has received from the controller before sending the joining request, i.e., the authentication step. The signed token is used to verify whether the candidate belongs to the cluster or not because the token of one cluster is not valid for another cluster. This token is signed using the private key of the cluster head, i.e., the controller. It is verified by the public key of the controller as $b \leftarrow VERIFY(Ctr_{pub}(Sign^i_{TKN}))$. If b is found true, it is considered a valid candidate. This contract self-executes as soon as any authentic candidate device sends the joining request.

Algorithm 10: Smart Contract to Join Cluster

Input: $C^i_{addr}, D_{ji}, Cst_{ID}$
Output: Device Validity
Function `joinCluster`$(C^i_{addr}, D_{ji}, Cst_{ID}, Sign^i_{TKN})$**:**

```
if isAvailable(C^i_addr) != true then
    Discard request;
else
    /* the C^i_addr is true for both, candidate and controller */
    if C_i = controller then
        if isAvailable(Cst_ID) != true then
            Discard request;
    else
        if C_i = candidate then
            if isAvailable(Cst_ID) != true then
                Discard request;
            if Sign^i_TKN.Verify = fail then
                Discard request;
            if isAvailable(D_ji) = fail then
                Discard request;
```

Key Distribution and Secure Communication

The key distribution phase is distributing the candidate device's public key to the other cluster members for communication. However, instead of directly sending a key to other devices, the candidate devices store it at the controller. The controller sends the candidate device's public key, say C_1, to the other candidate device, say C_2. The benefit of distributing public keys using this method is that only authenticated can-

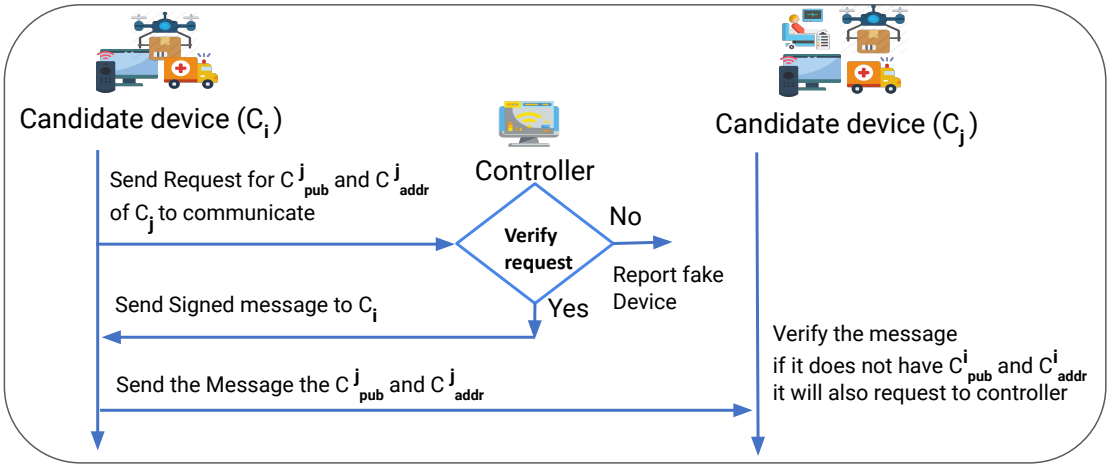


Figure 5.3: Candidate-to-candidate secure communication in BLISS

didate devices can get the key, reducing the possibility of key compromise. The key distribution phase is mandatory for the communication phase because 1) the sender's public key is required to verify the other device's incoming message. 2) to send the secret message to the other candidate device. All the candidate devices have the controller's public key during the authentication and cluster creation phase. The abstract of secure communication between two candidate devices is illustrated in Figure 5.3. The procedure for key distribution and communication is as follows.

Let candidates C_i and C_j want to communicate with each other. First, C_i sends request M_{s1} (a message which contains the request) to the corresponding controller to get public-key C^j_{pub} and address C^j_{addr} of C_j signed with private key C^i_{pri} as $\sigma_{M_{s1}} = SIGN(C^i_{pri}(M_{s1}))$, $Req[C^i_{addr}||D_{ji}||Sign^i_{TKN}||\sigma_{M_{s1}}||T_1]$. Here, $M_{s1} = get(C^j_{pub}$ and C^j_{addr} of $C_j)$. The controller receives the request and first checks whether it is a new request or an old one as $\triangle T \geq T_2 - T_1$, and if it is true, then the controller verifies the request as $b \leftarrow VERIFY(C^i_{pub}, M_{s1}, \sigma_{M_{s1}})$. It also checks the authenticity of C_i against the stored value of C^i_{addr}, D_{ji}, and $Sign^i_{TKN}$. If the signature is valid, the controller checks the identity of both the candidates C_i and C_j. The controller searches for the authenticity of both the candidate devices in the blockchain ledger. If the candidate is authenticated before the communication, then the candidate's public address and ID should already be present on the ledger. If both C^i_{addr} and C^j_{addr} are identified, the controller responds as follows. $M_{s2} = C^j_{pub} \oplus C^i_{addr} \oplus T_2$, $M_{s3} = C^j_{addr} \oplus C^i_{addr} \oplus T_2$, $\sigma_{M_{s2}} = SIGN(Ctr_{pri}(M_{s2}))$, and $\sigma_{M_{s3}} = SIGN(Ctr_{pri}(M_{s3}))$. After signing the response, send it to the requesting candidate, i.e., C_i, as follows. $Res[M_{s2}, M_{s3}, \sigma_{M_{s2}}, \sigma_{M_{s3}}, T_2, T_3]$.

After getting a response from the controller, candidate C_1 first checks whether the response was received in due time or an old response using $\triangle T \geq T_4 - T_3$ where T_3 is the request sending timestamp and T_4 is the received response timestamp. If the response is received within the threshold value, it is further processed; otherwise, it is discarded. After checking the timestamp, the received response's signature is verified, and if it is found to be true, retrieve the C^j_{pub} and C^j_{addr} as $Get(C^j_{pub}) = M_{s2} \oplus C^i_{addr} \oplus T_2$ and $Get(C^j_{addr}) = M_{s3} \oplus C^i_{addr} \oplus T_2$ }.

After retrieving the required values, i.e., C^j_{pub} and C^j_{addr}, C_i send message M_1 after signing it as σ_{M_1} = $SIGN(C^i_{pri}(M_1))$. The C_j receives the signed message, and first, it verifies using C^i_{pub}, and if it is not available, C_j requests from the controller to get it. Then check if $VERIFY(C^i_{pub}(M_1), \sigma_{M_1})$ is valid, then accept it; otherwise, reject it. All the candidates put the messages (treated as transactions) in the transaction pool TP. The controller takes the transactions from the TP as $TP \leftarrow Tr_1, Tr_2, ..., Tr_n$ to create the block on the regular interval. The Merkle root is calculated as the transaction from all the transactions taken from the pool. The Merkel root is added in the block's header, used to check the integrity of the transactions. To propose the block, the controller computes the Merkle root, the hash of the previous block PBH and the hash of the current block BH, and the T_3 as follows:

1. $Mrt = createHashTree(H(Tr_1), H(Tr_2), ..., H(Tr_n))$
2. $PBH = H(LB)$ {LB: last block of blockchain}
3. $BH = H(T_3, PBH, Mrt, H(Tr_1), H(Tr_2)...H(Tr_n))$
4. $block = createBlock[Mrt, BH, PBH | H(Tr_1), H(Tr_2)...H(Tr_n)]$

All the controllers agreed on this new proposed block and added it to the current blockchain. The *Algorithm* 11 illustrates the smart contract for two candidate devices' communication. For successful communication, the smart contract needs to be executed successfully. To summarize, depending on the device type, i.e., the controller or candidate device, the smart contract's rules are applied as follows:

1. **Controller** can build a single trusted cluster with a unique cluster identifier, which should not exist or be active in the blockchain network. The controller's role is to sign the token for the candidates and perform block generation and verification. The controller can not create the two clusters simultaneously, i.e., one controller can only be part of one cluster.
2. **Candidate** is associated with a single cluster only if it exists. Candidates can not belong to two separate clusters at the same time. All the candidates in the cluster require authentication from the controller before any communication. Every candidate must have a signed token signed by the controller.

Token and Key Revocation in BLISS

In the BLISS scheme, maintaining the integrity of both signed tokens and cryptographic keys is paramount. If either becomes compromised, i.e., token or keys, the scheme allows for their immediate revocation and replacement. Here, we present the mathematical description of the token and key revocation process.

Let $\mathbb{CTR}$ denote the set of all controllers, and $\mathbb{C}$ the set of all candidate devices within the BLISS network. Each candidate device $C_i \in \mathbb{C}$ is associated with a signed token $Sign^i_{TKN}$ issued by a controller $Ctr \in \mathbb{CTR}$ and a pair of cryptographic keys (C^i_{pub}, C^i_{pri}), (Ctr_{pub}, Ctr_{pri}) generated by all the legitimate candidate and controller device. The revocation process is illustrated as follows.

1. Upon the compromise of $Sign^i_{TKN}$ or (C^i_{pub}, C^i_{pri}), controller Ctr initiates the revocation process. The controller adds the compromised token or key to a revocation list, R_c for tokens and K_c for keys, both maintained on the blockchain ledger L.

Algorithm 11: Smart Contract for Communication

Input: D_{ji}, Cst_{ID}, $Sign^{i}_{TKN}$, $Sign^{j}_{TKN}$
Output: Device Validity
Function `clusterCommunication`(C^{i}_{addr}, D_{ji}, Cst_{ID}, $Sign^{i}_{TKN}$, $Sign^{j}_{TKN}$):

```
if isAvailable(D_ji(sender)) != true then
    if isAvailable(D_ji(receiver)) != true then
        Discard request;
/* Checking the ID of cluster whether it exists or not                    */
if isAvailable(Cst_ID) != true then
    Discard request;
if sender.Sign^i_TKN.verify = fail then
    if receiver.Sign^j_TKN.verify = fail then
        Discard request;
```

2. Verification post-revocation involves checking any token or key against R_c and K_c in L. A device is considered a valid member of the cluster if both its token and public key pass verification.

Traditional security systems rely on certificate revocation to maintain trust within the PKI framework. Revocation is managed using CRLs (Certificate Revocation Lists) or OCSP (Online Certificate Status Protocol), ensuring that any entity can verify the validity of a certificate. The BLISS scheme's revocation process mirrors these methods by maintaining a revocation list on the blockchain ledger for both signed tokens and cryptographic keys. Just as with CRLs, these lists can be checked by any participant in the network to verify the revocation status of a token or key.

The main differences lie in the medium of distribution and the technology used to implement these revocation lists. While traditional systems may use centralized servers or databases to store and distribute CRLs, the BLISS scheme utilizes blockchain technology's decentralized and immutable nature. This not only ensures that revocation lists are tamper-proof and always up-to-date across all nodes but also leverages the blockchain's inherent security and consensus mechanisms to enforce the revocation status network-wide. Thus, while the principles are similar, the implementation and underlying technologies differ, with BLISS adopting a blockchain-centric approach to revocation in contrast to the centralized models typically used in traditional PKI systems.

5.1.5 SCAB-IoTA: Secure Communication and Authentication for IoT Applications Using Blockchain

SCAB-IoTA is the second blockchain-based security solution for IoT applications. Following the similar architecture of BLISS, SCAB-IoTA enhances the performance and security compared with BLISS as well as the state-of-the-art schemes. Similar to BLISS, a different trust mechanism of cluster creation is employed in SCAB-IoTA. Before joining the cluster, every device must go through a secure authentication mechanism, and only authenticated devices can communicate with others. Unlike BLISS, we have used hybrid cryptography to provide a reliable security mechanism. The public-key cryptography is used for cluster member (D_{ji}) to cluster head (i.e., controller) communication, which is an ECDSA-based encryption/de-

cryption method. Secret-key cryptography is used for member-to-member ($M2M$) communication, which is an AES-based encryption/decryption method. In the cluster creation and member authentication phase, the clusters are created based on the angular distance (AD), i.e., the device that wishes to join the secure cluster first needs to be at a certain distance (called cluster radius R) from the controller. Once the device is in the range of the cluster, it sends a request (which contains device ID C_i) to join the cluster. The controller verifies the device and sends its public key Ctr_{pub} to the device. After getting the D_{ji} and Ctr_{pub}, the device becomes the cluster member. The controller also creates the block at the blockchain node containing the device ID of the requesting device for future authentication purposes. In the secret-key generation and M2M communication phase, the M2M communication is secured with AES encryption using secret-key S_K. Any member of the cluster who wants to communicate with another member must have the secret key generated. At first, The C_i with D_{ji} (i^{th} member of j^{th} cluster) sends a request for communication to C_k (K^{th} member of same cluster). C_k sends a second request to the controller to check for the authenticity of C_i. The controller verifies the member's identification through the blockchain node if the member's block exists, which means the member is authentic. Otherwise, a warning message is generated to the requesting device C_k. After verifying the C_i, the controller generates a one-time secret-key S_K and sends it to both the members, i.e., C_i and C_k. After getting a one-time secret key, members communicate using AES symmetric encryption.

Cluster Creation and Device Authentication

The clusters are created based on the angular distance (AD), i.e., the device that wishes to join the cluster to become a cluster member first needs to be at a certain distance (R) from the controller. Once the device is in the range, it sends a request (which contains device ID C_i) to join the cluster. The controller verifies the device and sends its public key Ctr_{pub} to the device. The device becomes the cluster member after getting Ctr_{pub} and the cluster membership ID (D_{ji}). The controller also creates the block at the blockchain node containing the device ID. *Algorithm* 12 defines how to calculate the distance between two coordinates. It takes two point coordinates as an argument and determines the distance. The device $Point_d$ is having coordinates (X_d, Y_d) and controller $Point_{Ctr}$ is having coordinates (X_{Ctr}, Y_{Ctr}).

Algorithm 12: Algorithm to Find Angular Distance

Input: P_d, P_{Ctr}: are the original coordinates, and $P_d^{'}$ is the coordinate after the movement, concerning their original coordinates.

Output: $d_{i,j}$: It is the angular distance between device and controller.

$P_d \leftarrow X_d, Y_d$; `/* Point coordinates of device */`

$P_{Ctr} \leftarrow X_{Ctr}, Y_{Ctr}$; `/* Point coordinates of controller */`

Function `AD`$(P_d, P_{Ctr}, P_d^{'})$**:**

$$\begin{bmatrix} X_d^{'} \\ Y_d^{'} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_d \\ Y_d \\ 1 \end{bmatrix}; \quad \texttt{/* calculate new coordinates of point } P_d \texttt{ */}$$

$d_{Ctr,d} \leftarrow |X_{Ctr} - X_d^{'}| + |Y_{Ctr} - Y_d^{'}|$;

The Cluster creation and device authentication process is illustrated in Figure 5.4, which is as follows:

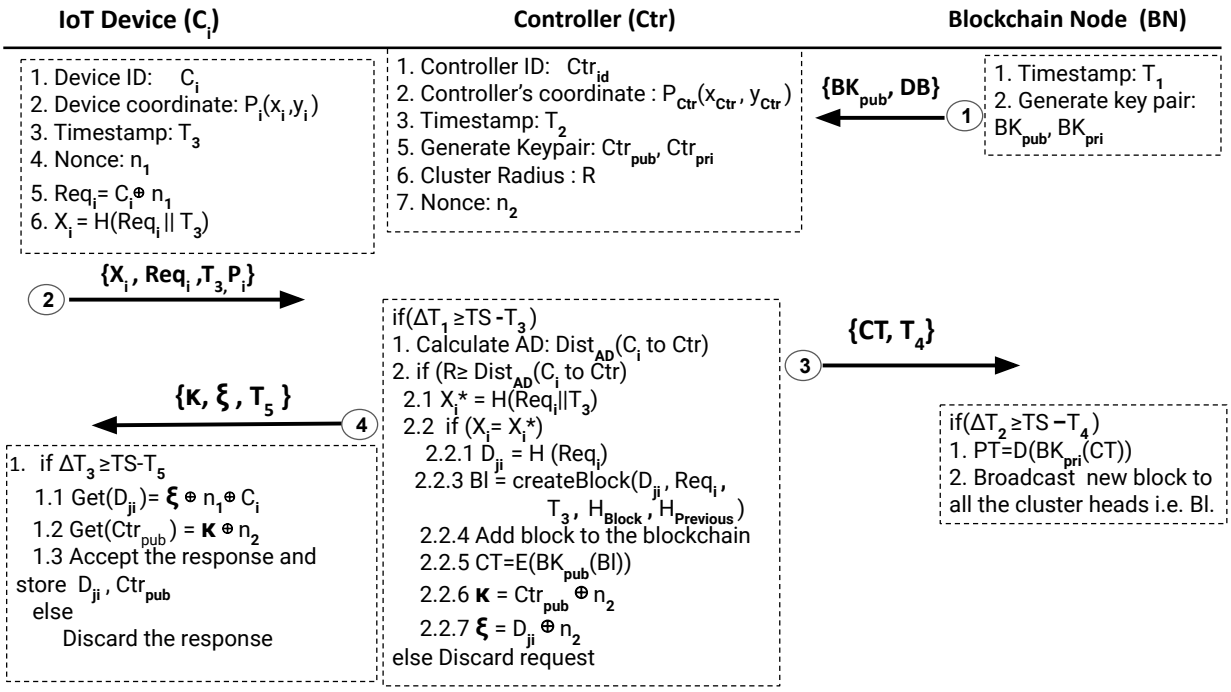


Figure 5.4: Cluster creation and device authentication

1. The device having device ID C_i, device coordinates $P_i(X_i, Y_i)$, timestamp T_3 and a nonce n_1. It create a request to join the cluster as $Req_i = C_i \oplus n_1$, $X_i = H(Req_i||T_3)$ and send $\langle X_i, Req_i, T_3, P_i \rangle$ to the controller.

2. After receiving the request, controller first check if $\triangle T_1 \geq TS - T_3$ is true, otherwise discard the request. If the condition holds true, the controller computes the distance using *Algorithm* 12. If the device is in the range of the controller, then it proceeds further and computes $X_i^* = H(Req_i||T_3)$, then check whether $X_i^* = X_i$. If so, then it does the computation as $D_{ji} = H(Req_i)$, where j is the cluster number. To propose the block, the controller computes the Merkle root, the hash of the previous block PBH and the hash of the current block BH, and the T_3 as follows:

   (a) $PBH = H(LB)$ {LB: last block of blockchain}
   (b) $BH = H(T_3, PBH, D_{ji}, Req_i)$
   (c) $block = createBlock[D_{ji}, Req_i, T_3, PBH, BH]$

   After that, it encrypts the created block using the public key BK_{pub} of the blockchain node as $CT = E(BK_{pub}(block))$ and sends $\langle CT, T_4 \rangle$ to the blockchain node to share with other controllers. Also, the controller computes κ and ξ as $\kappa = Ctr_{pub} \oplus n_2$, $\xi = D_{ji} \oplus n_2$ to share its public key, and device ID with requesting device. Then it sends $\langle \kappa, \xi, T_5 \rangle$ to the requesting device.

3. The blockchain node receives the encrypted new block from the controller. First, it checks whether or not $\triangle T_2 \geq TS - T_4$ is true. If true, then it decrypts the CT using its own private key BK_{pri} as $PT = D(BK_{pri}(CT))$, then it securely broadcasts this new block to all the controllers.

4. After receiving response from the controller, device C_i first calculate if $\triangle T_3 \geq TS - T_5$ is true or not. If true, then it retrieves the public key of the controller from the received response as $Get(D_{ji}) =$

$\xi \oplus n_1 \oplus C_i$, $Get(Ctr_{pub}) = \kappa \oplus n_2$ and store the D_{ji},Ctr_{pub} for future communication. The D_{ji} is now used as the main identity of the C_i for the communication. Other devices follow this procedure, and we create a secure cluster of devices where every device is authenticated from the controller.

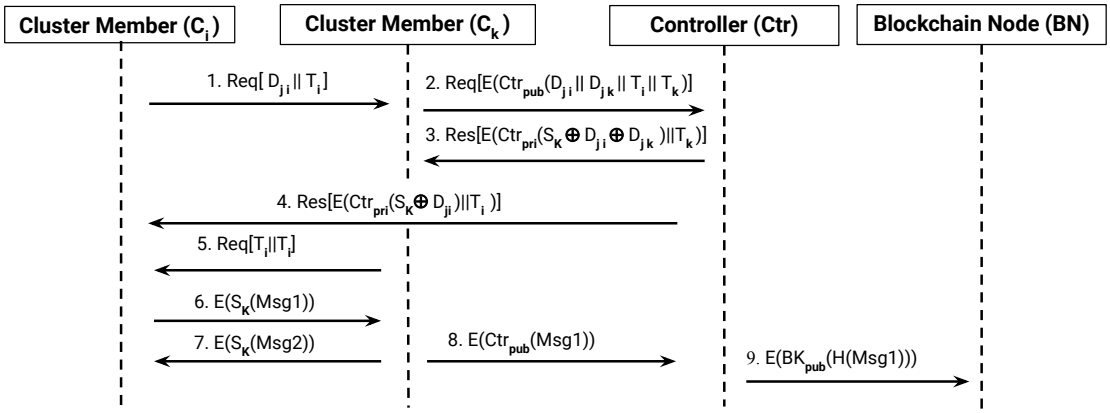


Figure 5.5: Secret-key share and M2M communication

Secret-key Generation and M2M Communication

Communication between two devices is secured using a 192-bit AES symmetric key cryptography. After receiving a request from the member device (D_{ji}) for communication, the controller generates the shared secret key. Communication between member devices and controllers is secured using the ECDSA asymmetric key cryptography. Member D_{ji} who wants to communicate with another member D_{jk} must have the secret key generated by the controller. The secret key generation process is explained in *Algorithm* 13, and the M2M communication is illustrated in Figure 5.5. The process of this phase is as follows:

1. Let, there are two devices C_i and C_k with cluster member IDs D_{ji} and D_{jk} who wants to communicate. Cluster member D_{ji} (i^{th} member in j^{th} cluster) sends a request as $Req[D_{ji}||T_i]$ to D_{jk} (K^{th} member of same cluster) for the communication.

2. Member D_{jk} receives the request and first checks if $\triangle T_1 \geq T_i^* - T_i$ holds; if yes, then, to check the authenticity of the member D_{ji} and one-time secret key generation, it sends a request to the controller as $Req[E(Ctr_{pub}(D_{ji}||D_{jk}||T_i||T_k))]$ and wait for the response.

3. After receiving a request from the D_{jk}, the controller first checks $\triangle T_2 \geq T_j^* - T_j$ is true, then it verifies both the member, i.e., $D_ji \& D_{jk}$ both, from the blockchain database. After verifying both the members first, it generates the one-time secret key S_K using *Algorithm* 13 and sends the response to both the members as $Res[E(Ctr_{pri}(S_K \oplus D_{ji} \oplus D_{jk} \oplus)||T_k)]$, $Res[E(Ctr_{pri}(S_K \oplus D_{ji})||T_i)]$ to D_{jk} and D_{ji} respectively. If the controller does not find either of the requesting members, a warning message is generated so other controllers are notified.

4. Member D_{ji} computes the received response and gets the one-time secret key S_K as $A = D(Ctr_{pub}($ $E(Ctr_{pri}(S_K \oplus D_{ji})||T_i)))$ and then check if $\triangle T_2 \geq T_i^* - T_i$ is true or not, if true then compute $Get(S_K) = A \oplus D_{ji}$ and store it for the communication. Similarly, the member D_{jk} also retrieves the one-time secret key S_K from the received response.

5. The D_{jk} sends a response to D_{ji} as $Res[T_i||T_j]$ as an acknowledgement, i.e., it is ready to accept the next message.

6. Now D_{ji} creates message $Msg1$ and encrypts it using a one-time secret key (using AES encryption technique) as $E(S_K(Msg1))$ and sends it to the D_{jk}.

7. D_{jk} receive the encrypted message and decrypt using the same key S_K, then it creates a message $Msg2$ and sends it to the D_{ji} after encrypt it as $E(S_K(Msg2))$. Similarly, the communication proceeds between two cluster members.

Algorithm 13: Secret-key-Generation

Input: Q_n, R_n
Output: S_k: It is the one-time secret-key
$Q_n \leftarrow$ randomNumber(); $R_n \leftarrow$ randomNumber();
Function `SKG(`Q_n, R_n`)`**:**
  if Q_n *is Prime* **then**
    if R_n *is Prime* **then**
      $N \leftarrow (Q_n - 1) \times (R_n - 1)$;
      $E \leftarrow$ randomNumber() mod 1000;
      $S_k \leftarrow N^E$; `/* The secret key` S_k `is calculated */`
      return S_k;
    else
      Call SECRET-KEY-GENERATION recursively;
  else
    Call SECRET-KEY-GENERATION recursively;

The *Algorithm* 13 is used to generate the secret key. In this algorithm, we first generate two random numbers, Q_n and R_n, and then check whether they are prime numbers. If they are prime numbers, then both numbers are subtracted by one and multiplied by each other. After that, we generated a random number and took mod with 1000 as $E \leftarrow randomNumber()\%1000$, then computed the secret key as $S_k \leftarrow N^E$, which is to be shared with the members. After getting a one-time secret key, both members communicate using symmetric key encryption.

5.1.6 Implementation Detail and Performance Analysis

We have used multiple Raspberry Pi 4 models to implement both schemes. We have also implemented the BLISS scheme on the Sepolia Ethereum testnet, using Remix IDE for development and MetaMask for network connectivity. The performance analysis of both schemes is done on four parameters, i.e., computational cost, energy consumption, storage cost, and communication cost.

Implementation Detail

In BLISS and SCAB-IoTA, we have used the Raspberry Pi 4 (RP4) model with a processor speed of 1.5GHz, memory of 4GB, and an operating system of Raspbian v5.10.17, a Debian-based OS. The Desktop PC has a

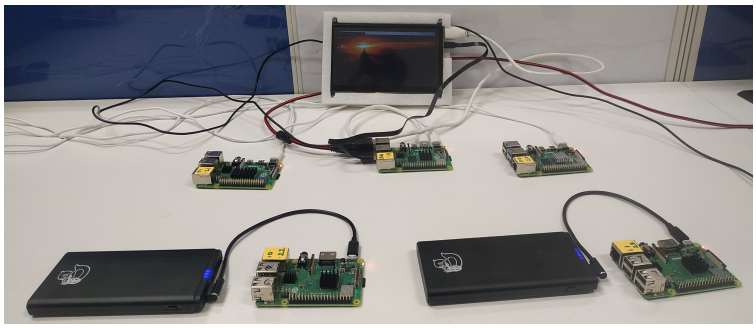


Figure 5.6: Testbed for one cluster of 5 Raspberry Pi model B devices

4.1GHz core i5 processor, 8GB of memory, and Windows 11 as an operating system. We have used Oracle Java 8 and NetBeans 11 IDE for the programming. The testbed consists of 5 Raspberry Pi devices and one computer system. This configuration was deliberately chosen to simulate a realistic and manageable IoT network that allowed us to monitor and accurately measure the system's performance. The testbed is shown in the Figure 5.6. We have used the Solidity programming language in BLISS to implement the smart contract. The functions that we have defined in this smart contract are *createDevicePool*, *createClusterPool*, *isAvailableDeviceID*, and *isAvailableClusterID*. We have also defined two events: *ClusterFound* and *DeviceFound*. These functions and events are necessary to auto-trigger the other two smart contracts. The cost was measured in terms of computational cost, energy consumption, storage cost, and communication cost at the scheme's operation, mainly during the initial authentication process of the devices in each cluster and during the execution of transactions. Power consumption was rigorously recorded throughout the experiment, especially during the cluster formation stage to capture the energy demands of setting up the network and during active communication between the devices to assess the ongoing operational efficiency. Apart from the testbed, we have also deployed the contracts on the Sepolia Ethereum testnet, using Remix IDE for development and MetaMask for network connectivity for the BLISS proposed scheme. This strategic shift to the Sepolia testnet allows us to better evaluate our system under conditions that closely mirror the Ethereum mainnet. Sepolia provides a realistic environment for testing the latency and overall performance of blockchain transactions, which is crucial for assessing the feasibility and efficiency of the BLISS scheme in a real-world scenario. This environment also exposes our contracts to more variable network conditions, including transaction delays and block processing times, which are essential factors in understanding the practical implications of deploying such a system. Using Remix IDE, we have developed and tested our smart contracts effectively, ensuring their integrity and robustness. MetaMask's integration has facilitated a seamless connection to the Sepolia testnet, allowing for efficient contract deployment and interaction. We utilized the Web3.js library to programmatically fetch and analyze block and transaction data from the Sepolia testnet to gather accurate and current statistics. This approach allowed us to meticulously compute the average block creation time and transaction delay over a significant period, ensuring the reliability of our findings. Based on our analysis, we deduced that the average block creation time on the Sepolia testnet is approximately 8-10 seconds. This slightly faster block time compared to the Ethereum mainnet, which averages around 13-15 seconds, is attributable to the lower network congestion typically experienced on the testnet. Additionally, we observed that transaction delays were generally shorter on Sepolia, reinforcing the suitability of this testnet for evaluating the BLISS scheme under less congested network conditions. These insights have been instrumental in enhancing our understanding of how the BLISS scheme performs in a dynamic blockchain environment and have been included in the revised manuscript. Below in Figure 5.7 shows the metamask interface, and Figure 5.8 shows the output of the contract JoinCluster.

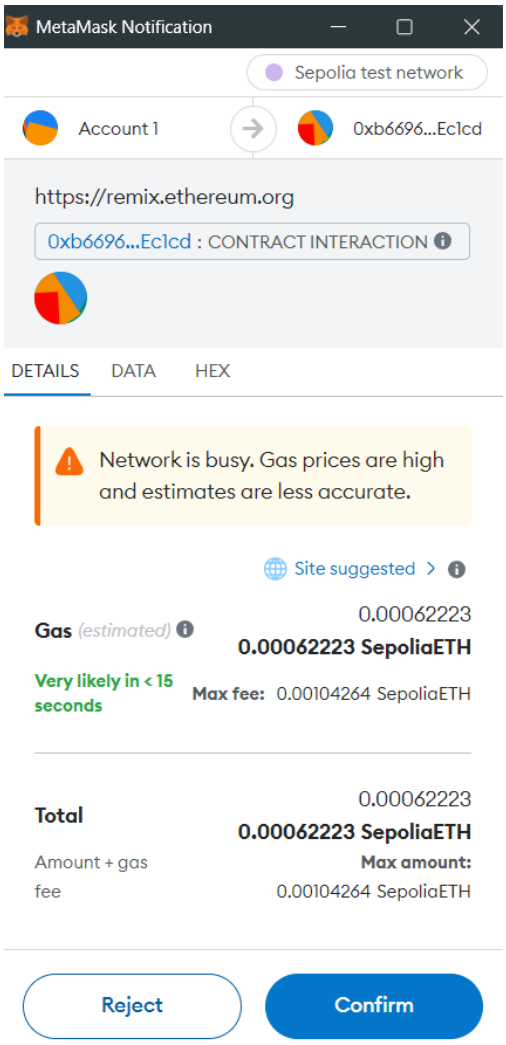


Figure 5.7: Metamask interface to connect with Ethereum testnet

Computational Cost and Energy Consumption

Computational time is the time taken by the proposed schemes for the execution of the whole framework. We have measured this cost in milliseconds (MS). The computational cost is denoted as T_{BLISS_ct} for BLISS and $T_{SCAB-IoTA_ct}$ for SCAB-IoTA. We illustrate the computational cost based on the operational cost for authentication, verification, and communication. Table 5.2 shows the execution time of different cryptographic functions used in the proposed schemes. Table 5.3 show the comparative computation cost of BLISS and SCAB-IoTA with state-of-the-art schemes. It is clear from Table 5.3 that the number of cryptographic operations in BLISS and SCAB-IoTA is lower than the state-of-the-art schemes; therefore, the computation cost is also lower in BLISS and SCAB-IoTA. Despite the lower computation cost in SCAB-IoTA, it is slightly higher compared with the BLISS scheme. The energy consumption is the energy consumed by the system to perform the operations of the scheme. In the execution process, the system requires some energy to carry out all the phases of the scheme. The energy consumption is calculated as $E_{rg} = T_{ct} * C_P$. Here, C_P is a CPU maximum power (10.88 W for a laptop and 2.4W for Raspberry Pi 4), and T_{ct} is a computational time.

```
status: 0x1 Transaction mined and execution succeed
transaction hash:
0x4904b196d7d61cd911be96ba81403a6844643b858158d644cb0697bf968ddd2b
block hash:
0x5e8bf8c4aa1b47e650b76b3c3a62b864aa9fddaf2498744abc9af6fcc5a4aa69
block number: 37 L
from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to: Common. isAvailableDeviceID(uint256)
0xddaAd340b0f1Ef65169Ae5E41A8b10776a75482d
gas: gas
transaction cost: 27927 gas
execution cost: 6495 gas
input: 0xff0 ... cabab L
decoded input:
           {
           "uint256 deviceId":
"687646025091532448527707505288253954804449848235"
           }
decoded output:
           {
           "0": "bool: true"
           }
logs: {
      "from": "0xddaAd340b0f1Ef65169Ae5E41A8b10776a75482d",
      "topic":
"0xa66d51529904c6cff23539df1e611d67015b0840db58d5a936da2dd7c56ccc15",
      "event": "DeviceFound",
      "args": {
            "0": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
            "owner": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2"
            }
      }
```

Figure 5.8: Output of the JoinCluster contract. The function call isAvailableDeviceID shows the device is found, mentioned under the decoded output

It is measured in Joules. Earlier, we had already calculated the computational cost. We calculated the energy consumption on two platforms, i.e., the computer system and Raspberry Pi 4 model. Table 5.3 shows the comparative result of energy consumption of BLISS and SCAB-IoTA compared with state-of-the-art schemes. It is very clear from the figure that in BLISS and SCAB-IoTA, energy consumption is reduced compared to state-of-the-art schemes.

Table 5.2: Individual computation time of different cryptographic functions for IoT testbed

Functions	**On Desktop**	**On Raspberry Pi 4GB**
SHA256 Hash function: $T_{H()}$	0.0008	0.005
ECC Crypto-system: T_{ECC}	0.053	1.6
ECDSA signature: T_{sign}	0.071	2.94
ECDSA verification: T_{veri}	0.065	2.86
point multiplication: $T_{mul(.)}$	0.117	5.0
AES Crypto-system 192-bit: T_{AES}	0.125	5.96
Modulo operation: $T_{mod(.)}$	0.067	2.0

Table 5.3: Computation cost (CC) and energy consumption (EC) of BLISS, SCAB-IoTA, compared with state-of-the-art schemes on Desktop and Raspberry Pi (RP)

Scheme	Computation Cost Equation	CC on Desktop	CC on RP4	EC on Desktop	EC on RP4
[101]	$4T_{ECC}+5T_{sign}+5T_{veri}$	0.892	35.4	9.705	84.96
[102]	$3T_{ECC}+6T_{sign}+6T_{veri}$	0.975	39.6	10.608	95.04
[108]	$16T_{H(.)}+5T_{ECC}+4T_{sign}+4T_{veri}$	0.822	31.28	8.94	75.07
[146]	$6T_{H(.)}+2T_{ECC}+4T_{sign}+4T_{veri}$	0.655	26.43	7.13	63.43
[147]	$8T_{H(.)}+5T_{sign}+5T_{veri}$	0.686	29.0	7.47	69.69
BLISS	$7T_{H(.)}+4T_{sign}+4T_{veri}$	0.550	23.23	5.99	55.75
SCAB-IoTA	$5T_{H(.)}+5T_{ECC}+2T_{AES}$	0.519	19.95	5.65	47.88

Storage and Communication Cost

Storage cost refers to the storage required for storing the required values of multiple variables used in the execution of the scheme, such as the secret key, hash code, and other parameters in all phases. On the other hand, communication cost refers to the storage consumed by the scheme based on the used variables for communication, such as hash code, ciphertext, and other parameters. The storage and communication cost is measured in Bytes. We have derived the equations for BLISS, SCAB-IoTA, and state-of-the-art schemes to compute the storage cost and communication cost shown in Table 5.4. The required storage for different cryptographic functions used in the schemes is as follows. A secret Key S_K requires 24B, ECC encryption/decryption S_{ECC} takes 48B, AES encryption/decryption S_{AES} takes 24B, a hash function $S_{H(.)}$ requires 32B, and ECDSA based signature S_{sign} and verification S_{veri} requires 10B. Table 5.4 represents the comparative storage cost of the BLISS, SCAB-IoTA with state-of-the-art schemes.

Table 5.4: Comparative storage and communication cost of BLISS, SCAB-IoTA with state-of-the-art schemes

Scheme	Storage Cost	Cost in Bytes	Communication Cost	Cost in Bytes
[101]	$10S_{H(.)}+1S_{ECC}+1S_{sign}$	378	$7S_{H(.)}+2S_{ECC}+2S_{sign}$	340
[102]	$4S_{H(.)}+2S_{ECC}+2S_{sign}+2T_k$	332	$2S_{H(.)}+2S_{sign}+2T_k$	212
[108]	$3T_{H(.)}+4S_{sign}+4T_k$	376	$3S_{sign}+2T_k+ConstantVar$	242
[147]	$3S_{H(.)}+2S_{sign}$	116	$4S_{H(.)}+2S_{sign}$	148
BLISS	$2S_{H(.)}+2S_{sign}$	74	$2S_{H(.)}+4S_{sign}$	94
SCAB-IoTA	$4S_{H(.)}+1S_{AES}+3S_{ECC}$	296	$1S_{H(.)}+2S_{AES}+3S_{ECC}$	224

5.1.7 Security Analysis

Here, we briefly explain how both the proposed schemes, BLISS and SCAB-IoTA, withstand multiple attacks. We have also defined how SCAB-IoTA ensures the authentication and integrity property of the security. Table 5.5 shows a comparative analysis of BLISS and SCAB-IoTA with state-of-the-art schemes. In this table, ✓means the scheme is protected, and × means the scheme is not protected. The proof description of all the attacks is defined below.

Lemma 9 (Authentication)**.** *BLISS secures the authentication of IoT devices and the integrity of data exchanges by employing digital signatures, specifically utilizing ECDSA and SHA256 algorithms. Under the chosen-message attacks, the probability of successfully forging a signature is negligible for any adversary* $\mathbb{A}$*. This is formalized by the conditions that for a given key pair* (C^i_{pub}, C^i_{pri}) *generated by* $KEYGEN(k)$*, and for any message* M *not in the set of signed messages* $M_1, M_2, \ldots, M_n$*, the verification function* $VERIFY(C^i_{pub}, M, \sigma) = True$ *implies a negligible success probability for* $\mathbb{A}$*, assuming* σ_M *is a signature on* M *that* $\mathbb{A}$ *computes without knowing* C^i_{pri}*. Thus, BLISS ensures a robust authentication protocol for IoT ecosystems.*

Table 5.5: Comparative security analysis of BLISS and SCAB-IoTA with state-of-the-art schemes

Types of attacks	[101]	[102]	[104]	[108]	**BLISS**	**SCAB-IoTA**
Identification	✓	✓	✓	✓	✓	✓
Message Integrity	✓	✓	✓	✓	✓	✓
Sybil Attack	✓	✓	✓	✓	✓	✓
Fake Node Injection	✓	✓	✓	✓	✓	✓
Tampering	✓	×	✓	×	✓	✓
Message Replay Attack	×	✓	✓	×	✓	✓
Man-in-Middle Attack	✓	✓	✓	✓	✓	✓
Spoofing Attack	✓	✓	✓	✓	✓	✓

Proof. We have used the digital signature based on the ECDSA and SHA256 because it is unalterable under any random attack. Let the message space be M, and the adversary is $\mathbb{A}$. The success possibility of adversary $\mathbb{A}$ is minimum or almost negligible if it satisfies the following computation.

Let $M_1, M_2, ...M_n$ are the set of messages chosen randomly from message space M and key-pair for candidate C_i (C^i_{pub}, C^i_{pri}) is generated using a function called $KEYGEN(k)$. Signature for the set of randomly chosen messages can be computed as $\sigma_{M_1} = SIGN(C^i_{pri}, M_1), ...,\ \sigma_{M_n} = SIGN(C^i_{pri}, M_n)$. $\mathbb{A}$ is given C^i_{pub} and $(M_i, \sigma_i)^n_{i=1}$ and output (M, σ_M). The success of $\mathbb{A}$ is possible if $VERIFY(C^i_{pub}, X, \sigma_X) = True$ and $X \notin (M_1, M_2, ..., M_n)$. However, the BLISS is strongly unalterable if this step is modified according to the following step. $VERIFY(C1_{pub}, X, \sigma_X) = True$ and $(X, \sigma_X) \notin (M_1, \sigma_{M_1}), (M_2, \sigma_{M_2}), ..., (M_n, \sigma_{M_n})$.

$\mathbb{A}$ could observe the communication channel as it is open. However, messages (M_n) from the C_i to C_j or from C_i to controller are signed $\sigma_{M_n} = SIGN(C^i_{pri}, M_n)$ and $VERIFY(C^i_{pub}, M_n, \sigma_{M_n})$ verified by C_j or controller before any further processing of the message. the $\mathbb{A}$ cannot reverse the transaction's hash, and it does not have the private key of the candidate or controller. Even if the $\mathbb{A}$ seizes the channel and obtains the signed messages, then fake IoT device authentication is impossible. □

Lemma 10 (Integrity)**.** *The BLISS ensures the data integrity of the shared messages between the candidate and controller.*

Proof. Let the message space be M, and the adversary is $\mathbb{A}$. The success possibility of adversary $\mathbb{A}$ to violate the integrity of the shared message is minimal or almost negligible if it satisfies the following computation. Let $M_1, M_2, ...M_n$ are the messages generated by $\mathbb{A}$, and it also know the public key C^i_{pub} of candidate C_i. The $\mathbb{A}$ somehow know signature for the set of randomly chosen messages $\sigma_{M_1} = SIGN(C^i_{pri}, M_1), ...,$ $\sigma_{M_n} = SIGN(C^i_{pri}, M_n)$. $\mathbb{A}$ is given C^i_{pub} and $(\sigma_{M_i})^n_{i=1}$ and output (M, σ_M). The success of $\mathbb{A}$ is possible if $VERIFY(C^i_{pub}, X,\ \sigma_X) = True$ and $X \notin (M_1, M_2, ..., M_n)$. However, the BLISS is strongly unalterable if this step is updated according to the following step. $VERIFY(C^i_{pub}, X, \sigma_X) = True$ and $(X, \sigma_X) \notin (M_1, \sigma_{M_1}), (M_2, \sigma_{M_2}), ..., (M_n, \sigma_{M_n})$.

BLISS uses signature and verification algorithms. Suppose a third party has access to the transactions and attempts to falsify them. It is only C_j who can decipher and verify M_1, a message signed with C_i's private key, which is sent to C_j. Private keys of both candidates and controllers can not be obtained by an $\mathbb{A}$. For example, if another candidate or controller's private key is needed, it is impossible for $\mathbb{A}$ to sign this transaction. Because of this, the system ensures the integrity of the transactions and the blockchain's immutability and tamper-proof record of the information. □

A. Identity Theft

Identity theft would necessitate an adversary $\mathbb{A}$ to steal the secret information of a candidate device. In BLISS and SCAB-IoTA, all secret information is stored on the blockchain in the form of hash codes, providing anonymity and preventing $\mathbb{A}$ from obtaining the actual identity of any device due to the computational infeasibility of reversing hash functions.

- **In BLISS:** Authentication is handled by the controller issuing a $Sign^i_{TKN}$, which is a digitally signed of C^i_{addr} and hash of the cluster ID and timestamp using the controller's private key Ctr_{pri}. For successful communication, a device must present $Sign^i_{TKN}$, its address C^i_{addr}, and an authentication code $Auth^i_{code}$, which are validated prior to communication. Assuming $\mathbb{A}$ attempts to impersonate a candidate C_j to send a forged message to C_i, it would require C^i_{addr} and C^j_{pri}. However, C^j_{pri} is confidential and $\mathbb{A}$'s requests for C^i_{addr} must be accompanied by a valid $Sign^i_{TKN}$, which $\mathbb{A}$ cannot fabricate without access to Ctr_{pri}. The controller's vigilant monitoring for anomalous behavior and the necessity of a valid $Sign^i_{TKN}$ for each request enables it to discard any attempts at identity theft by adversaries effectively. Therefore, the BLISS scheme's reliance on secure hash functions and strict authentication protocols ensures its security against identity theft.
- **In SCAB-IoTA:** Each member of the cluster is identified by the cluster head, which assigns its public key Ctr_{pub} for future communication as $\kappa = Ctr_{pub} \oplus n_2$, $\xi = D_{ji} \oplus n_2$, and the device which receives response do computation as $Get(D_{ji}) = \xi \oplus n_1 \oplus C_i$, $Get(Ctr_{pub}) = \kappa \oplus n_2$ and get the public key of the controller i.e., only the identified member can communicate to other members as well as from controller. Along with the public key, the controller generates the one-time secret key S_K for M2M communication, which is only assigned to identified members. Hence, our SCAB-IoTA withstands identification attacks.

B. Message Integrity

It is a kind of attack when the integrity principle of security is violated, i.e., when someone can modify the content of the message in transit. The integrity of the message is retained because the interaction between members is secured with a one-time secret key, i.e., encryption $E[S_K[Msg1]]$ and decryption $D[S_K[Msg1]]$, and interaction between the candidate and controller is secured with public-key encryption $E[Ctr_{pub}[Msg1]]$. The controller generates a one-time secret, so there is no secret key compromising.

C. Sybil Attack

In the face of Sybil attacks, where an adversary may attempt to subvert the network by creating multiple fake identities, the BLISS and SCAB-IoTA scheme enforces a strong authentication process.

- **In BLISS:** Each device within the network is assigned a unique set of credentials that include D_{ji}, C^i_{addr}, $Auth^i_{code}$, and C^i_{pub}. These credentials are securely recorded and maintained by the controller in each block. Smart contract rules are applied rigorously before allowing a device to join the cluster or to enable inter-device communication. These contracts, backed by lightweight cryptographic functions, are designed to authenticate each device's credentials, ensuring that each identity is verified and authenticated by the controller. This systematic approach to authentication, coupled with the security

provided by the blockchain ledger, renders the possibility of forging identities and executing a Sybil attack infeasible. The controller's role in uniquely identifying and verifying each cluster member constitutes a robust barrier against the threat posed by Sybil attacks. Hence, the BLISS scheme presents strong protection against Sybil attacks.

- **In SCAB-IoTA:** Each member device ID is stored at blockchain node $block = [D_{ji}, Req_i, T_3, BH, PBH]$, $CT = E(BK_{pub}(block))$. Hence, the device can have only one identity, and each member can have only one secret key at a given time for communication. All communicating messages are secured by the secret key S_K associated with this member ID D_{ji}. Moreover, all the participating members must be authenticated by the controller. Thus, an attacker cannot use fake identities.

D. Fake Node Injection Attack

A fake node injection attack in a blockchain-based IoT system occurs when an attacker introduces a malicious node into the network, aiming to disrupt the data flow or gain unauthorized access to sensitive information. The BLISS is designed to mitigate such security threats by ensuring that all participating nodes are authenticated and verified before they can communicate within the network. BLISS ensures that only authenticated candidate devices can join and communicate within a cluster, thus preventing fake node injection attacks.

The C^i_{pub} and C^i_{pri} is the public-private key pair of a candidate generated using ECDSA. The C^i_{addr} is the unique address of the candidate device, computed as $C^i_{addr} = H(C^i_{pub} \oplus F_n)$. The authentication code $Auth^i_{code}$ is given by $Auth^i_{code} = H(C^i_{addr} \oplus n)$, ensuring uniqueness due to the pre-image resistance of H. In the cluster creation phase, the controller issues $Sign^i_{TKN} = SIGN[Ctr_{pri}[C^i_{addr} H[Cst_{ID}||T_2]]]$. As the SIGN operation is secured by the private key Ctr_{pri}, $Sign^i_{TKN}$ cannot be replicated by an adversary. Furthermore, the smart contract mandates a valid $Sign^i_{TKN}$ for a candidate to join the cluster, verified by $b \leftarrow VERIFY(Ctr_{pub}, Sign^i_{TKN}, \sigma_{Sign^i_{TKN}})$. A fake node cannot participate in the cluster's communication without the correct $Sign^i_{TKN}$. Thus, by the properties of hash functions, ECDSA, and digital signatures, the BLISS scheme is provably secure against fake node injection attacks.

E. Tampering Attack

A *tampering attack* in a blockchain-based IoT system context refers to any malicious activity where an adversary intentionally alters, manipulates, or substitutes the data or devices within the system. This type of attack can manifest in various forms:

1. Altering the data stored on the blockchain to falsify records.
2. Physically tampering with IoT devices to change their behavior or output.
3. Manipulating the communication between nodes to inject or alter data.
4. Substituting legitimate devices with compromised ones to gain unauthorized access or disrupt the system's normal operation.

This alteration aims to disrupt the system's normal operation, introduce fraudulent data, or gain unauthorized access to system functionalities. Formally, a tampering attack is a function $T : N \rightarrow N'$, where N is the

set of legitimate nodes, and N' is the set of compromised nodes, such that the altered node's behavior or reported data deviates from the expected or genuine behavior. The proposed scheme BLISS ensures that it is protected from this attack. Below, we have described how BLISS is secured from tampering attacks. The authentication code $Auth^i_{\text{code}}$ is unique and securely bound to the device identity and nonce. The $Sign^i_{\text{TKN}}$ is a token signed by the controller's private key, which is unique to the controller and cannot be forged. The BLISS scheme ensures that every communication in the network is signed and verified using the public-private key pairs of the devices and controller. The integrity provided by ECDSA for signatures means that any modification of the data or the node identity will result in a signature verification failure, thus detecting any tampered message or device. Therefore, for all data d_i and nodes n_i, a verification function $\text{VERIFY}(C1_{\text{pub}}, d_i)$ returns false for tampered d_i. Authentication is guaranteed since each candidate device generates its own keys and unique address, and the controller verifies these during the join request, allowing only authenticated devices to join the network. Thus, a device C_i is authenticated if and only if $Sign^i_{\text{TKN}}$ is valid. Moreover, the signed token $Sign^i_{\text{TKN}}$ ensures non-repudiation, establishing that only the controller with the private key could have created it, making any message or command from the controller trusted and irrefutable. Hence, a verification function $VERIFY(Ctr_{\text{pub}}, Sign^i_{\text{TKN}})$ returns true if $Sign^i_{\text{TKN}}$ is valid. These cryptographic checks and balances effectively prevent tampering attacks since the system detects and rejects any unauthorized modifications to device identities, data, or communication.

F. Message Replay Attack

Both the BLISS and SCAB-IoTA schemes employ a unique transaction identifier and synchronized timestamps to prevent replay attacks. The authentication and cluster creation phase details how a request is formed as $Req[C^i_{addr}||D_{ji}||Sign^i_{TKN}||\sigma_{M_{s1}}||T_1]$, embedding the candidate's unique address C^i_{addr}, an identity D_{ji}, and a timestamp T_1 within each message. The timestamp T_1 ensures that each request is inherently tied to a specific point in time, enabling the network to distinguish between new and replayed messages. A message is considered valid only if the current network time T_{curr} minus T_1 does not exceed the defined interval ΔT. This protocol ensures that any message with a timestamp that does not conform to the temporal constraints is rejected, thereby preventing replay attacks. Moreover, the blockchain ledger records all transactions and, through network consensus, makes any replayed action detectable and preventable. Once an action is confirmed and recorded, it cannot be replayed, contributing to the security of the network against replay attacks. Therefore, by combining the unique message identifiers, the inclusion of synchronized timestamps, and the blockchain's immutability, the BLISS and SCAB-IoTA schemes ensure comprehensive protection against replay attacks, maintaining the integrity and security of the network's communications.

G. Man-in-The-Middle Attack

In the BLISS scheme, communication between devices and the controller, or between devices themselves, is secured using cryptographic signatures. Assume an adversary $\mathbb{A}$ has intercepted the communication and obtained the public key $C1_{\text{pub}}$ and address $C1_{\text{addr}}$ of a candidate device C_1. In the BLISS scheme, messages between devices or between a device and the controller are signed with the sender's private key, which $\mathbb{A}$ does not possess, thus preventing message forgery.

Furthermore, authentication in BLISS requires presenting a valid signed token $Sign^i_{\text{TKN}}$ to the controller, which is signed using the controller's private key and can only be verified with the public key Ctr_{pub}. Without the capability to produce a valid $Sign^i_{\text{TKN}}$, $\mathbb{A}$ cannot impersonate a legitimate device. Additionally, the integrity of messages is protected by including the timestamp T_2 in the message composition. This ensures

that even if $\mathbb{A}$ intercepts a message without the correct private key to sign a forged message, the network will reject the communication due to failed signature verification. Therefore, the BLISS scheme, through its use of digital signatures, secure authentication process, and timestamp verification, is protected against man-in-the-middle attacks, maintaining the confidentiality and integrity of the network's communications.

H. Spoofing Attack

In IoT systems, a *spoofing attack* occurs when an attacker disguises as a legitimate entity to gain unauthorized access to the system's resources or sensitive information. This involves the attacker presenting false credentials or manipulating communication protocols to mislead the system into accepting an illegitimate entity as authentic.

- **In BLISS:** For a spoofing attack to be successful, an attacker must forge $Sign^{i}_{\text{TKN}}$ to impersonate a legitimate device. Due to the collision resistance property of the hash function H, finding such an x that $H(x) = Sign^{i}_{\text{TKN}}$ is computationally infeasible. Furthermore, without the corresponding C^{i}_{pri}, no attacker can generate a valid signature that the BLISS controller would accept because $VERIFY(Ctr_{\text{pub}}, Sign^{i}_{\text{TKN}}) =$ true, if and only if $Sign^{i}_{\text{TKN}}$ was created with Ctr_{pri}. This is due to the security properties of the digital signature. Hence, the BLISS scheme, by the cryptographic strength of ECDSA and the hash function H, ensures that spoofing attacks are infeasible.
- **In SCAB-IoTA** Unlike the Sybil attack, the attacker spoofs the identity of a recognized device to gain access to his/her rights. As we have already mentioned in SCAB-IoTA, the controller does the authentication of participating devices and distributes the key for communication, i.e., send(κ and ξ), where $\kappa \leftarrow Ctr_{pub} \bigoplus n_2$, and $\xi \leftarrow D_{ji} \bigoplus n_2$. Hence, an attacker cannot spoof another member's identity since he needs to be authenticated by the controller.

5.2 SOLUTION FOR ITS APPLICATIONS

Road accidents are the 9^{th} biggest cause of death, which figures around 1.5 million people per annum globally [44]. 1.5 lakh people were killed, and 5 lakh people were injured in India, according to the road accident database report 2019-20 [45], [46]. It is found in the studies that the countries that do not invest in road safety could lose about 7 to 22% in potential per capita GDP growth in the coming years. In addition to life and economic loss, traffic jams cause the loss of fuel, time, and pollution. Therefore, any country needs to make decisions for road safety policies. The Intelligent Transportation System (ITS) is the right answer, and it is a rapidly growing concept in the field of computer science [148]. ITS aims to achieve road traffic efficiency by reducing road traffic-related problems. It informs users in advance with prior information about road traffic jams, accident zone areas, slow and fast speed zones, local convenience real-time running information. Moreover, it helps reduce travel time and enhance safety and comfort. The ITS aims to provide road safety and efficiency in terms of reducing time, cost, and pollution [149]. In ITS, blockchain technology has been utilized to enhance security and efficiency. Blockchain provides a secure communication framework for ITS applications, protecting against various cyber attacks, including man-in-the-middle attacks, impersonation attacks, device tampering, and Sybil attacks. The use of TPD (Tamper-Proof Devices) in conjunction with blockchain and SDN (Software-Defined Networking) ensures that critical information, such as vehicle identities and secret messages, cannot be tampered with. Implementing blockchain in ITS

transforms it into a decentralized application, which further reduces the risk of a single point of failure. In addition to decentralization, the secure framework and untampered information provided by blockchain enhance transparency in ITS, thereby increasing trust among participants in the transportation ecosystem, including vehicles, infrastructure, and service providers. Similar to IoT applications, the overhead introduced by blockchain, in terms of latency and low throughput, directly impacts the performance of ITS, where real-time information exchange and low-latency communication are critical. Moreover, integrating blockchain with ITS adds complexity to the system, particularly in terms of operational challenges. Therefore, it is necessary to design a suitable framework or protocol based on blockchain for ITS that addresses these integration issues, which includes the development of an appropriate consensus algorithm.

We have developed two schemes for ITS applications using blockchain technology. The first scheme is LBSV, a lightweight blockchain security protocol for secure storage and communication in SDN-enabled Internet of Vehicles (IoV), and the second is SmartCoin, a novel incentive mechanism for vehicles in ITS based on consortium blockchain. While distinct, these systems collectively represent blockchain's innovative use in ITS. LBSV and SmartCoin collectively showcase blockchain technology's potential to enhance the security and efficiency of ITS. While LBSV focuses on securing the IoV through SDN and blockchain integration, SmartCoin incentivizes honest vehicles, promoting a reliable and efficient ITS environment.

5.2.1 Motivation and Contribution

The fundamental issue faced by ITS is the use of a traditional centralized approach to provide security and privacy in the network, which is prone to a single point of failure. Indeed, secure communication depends on efficiency and the applied technique's fairness. Satisfying ITS network requirements, along with low computational cost, is another challenging problem faced by the research community. In LBSV and SmartCoin, we investigate a lightweight blockchain security protocol using SDN-based architecture to improve confidentiality and secure message dissemination. Secure message dissemination issues have been explored with the following question: *"How to devise a technique that can securely deliver a message in an untrustworthy ITS environment?"*. Centralized controlling issues have been explored to answer the question *"how to create a distributed and decentralized framework for ITS using blockchain technology"*. Further, this research is motivated by the challenge, *"how to improve the computational efficiency of blockchain and use a modified consensus algorithm to achieve secure communication"*. Moreover, it is quite challenging to handle storage requirements in the network. So, we need to explore *how to create lightweight blocks and small consensus mechanisms to overcome storage requirements*. ITS communication uses a tremendous amount of data and controls messages for the coordination and cooperation of vehicles. Specifically, the contributions of our proposed scheme are as follows:

- We have proposed a modified Practical Byzantine Fault Tolerance (mPBFT) consensus algorithm suitable for permissioned blockchain for SDN-enabled IoV network in LBSV. The mPBFT consensus scheme ensures fairness, i.e., it gives equal chances to all the miners to propose a block. An improved leader selection technique has been used to ensure fairness in the mPBFT consensus algorithm using a modular function.
- A hybrid SDN-based architecture is used in LBSV to reduce the controller's overhead and provide a distributed environment. Secure message dissemination is achieved through cryptographic hash functions and cryptographic digital signature techniques.

- To improve the network efficiency and maximum honest public participation, a novel incentive mechanism based on consortium blockchain is proposed in SmartCoin. A new digital cryptocurrency has been introduced and used as an incentive for vehicles called SmartCoins (STC); due to this, we also name the whole scheme as SmartCoin.
- SmartCoin also employed a proposed lightweight consensus algorithm (LCA) for transaction confirmation. SmartCoin provides the authentication of the participating vehicles and integrity and confidentiality of the vehicle information and the messages communicated.
- To show the efficiency of the LBSV and SmartCoin, the performance analysis is performed based on the computation cost, energy consumption, storage cost, and communication cost. We also provide the security analysis to show how the proposed scheme is resistant to various cyber-attacks, e.g., impersonation, botnet, Sybil, man-in-the-middle, message replay, and modification. We have also presented a lemma-proof of the security of Smartcoin for authentication, data integrity, double-spending, forking attacks, and fault tolerance.

5.2.2 Network Model

There are various components in the proposed mechanisms, i.e., vehicles, roadside units (RSU), controllers, and tamper-proof devices. As shown in Figure 5.9, RSUs and vehicles can communicate with each other using broadcast over the public channel using Dedicated Short Range Communication (DSRC) [150] protocol. We have divided the architecture into two layers. The consensus layer consists of SDN controllers and blockchain storage. Storing complete data in blockchain creates storage issues; we have used blockchain only to store tamper-proof data and create lightweight blocks. Each vehicle in the IoV layer needs to authenticate itself before involving in communication. The authentication process is carried out through the controller using a secure channel (e.g., Secure Socket Layer (SSL) or Transport Layer Security (TLS) protocol). Each vehicle is equipped with a Tamper Proof Device (TPD). Table 5.6 depicts the mathematical symbols that we have used in the methodology of both schemes. Below, we defined each component of the ITS network model.

- **Tamper Proof Device:** TPD is a specialized hardware component embedded in vehicles to securely store and manage sensitive cryptographic keys and other critical security parameters. These devices are designed to resist physical tampering, ensuring that even if an attacker gains physical access to the vehicle, they cannot extract or manipulate the stored keys. In the context of Intelligent Transportation Systems (ITS), TPDs play a crucial role in establishing secure communication channels between vehicles and infrastructure by safeguarding the integrity and confidentiality of cryptographic keys used for encryption, decryption, and digital signatures. A TPD is installed in the vehicle as a computational unit. It stores the vehicle's ID ID_v^i, user's ID ID_u^i, and nonce n_u. The TPD does the initial computation for the initialization phase. The hash (SHA256) is used to compute the hash code for the registration and communication phase. After the initialization phase, it generates the public and private key pair (V_{pub}^i, V_{pri}^i), which is used for the ECC encryption/decryption. It also stores the IDs ($ID_u^i, ID_v^i, ID_{RSU}, ID_c$) and a list of public keys of other vehicles Ls_k. Figure 5.10 illustrates the TPD with essential computations.

  While TPDs are crucial for safeguarding cryptographic keys and preventing unauthorized access at the device level, it is important to note that they cannot fully mitigate network-level threats such as man-in-the-middle attacks, replay attacks, or message tampering. These vulnerabilities highlight the

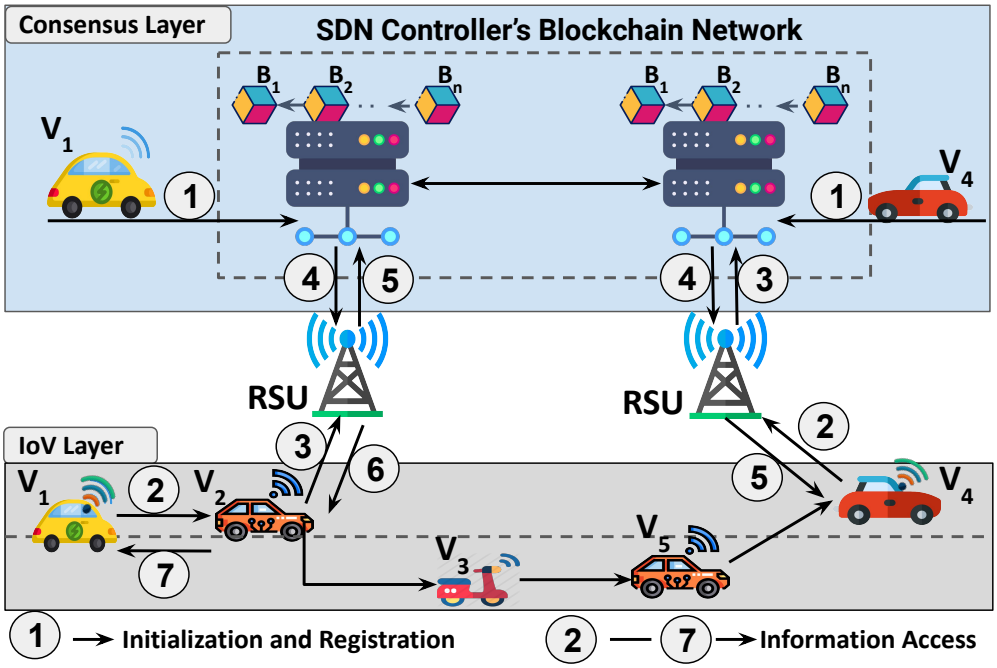


Figure 5.9: A basic network model of proposed ITS solution

necessity for additional security measures, such as those provided by the LBSV scheme, which are designed to secure communications across the network.

- **Vehicle:** A vehicle is the only mobile unit in the ITS network. It is equipped with TPD for communication with other components of the network, such as vehicles, RSUs, and controllers.
- **Roadside Unit (RSU):** The RSU is the first point of contact for the vehicle to interact with the controllers in the communication phase. Further, it is responsible for broadcasting updated information in the ITS network. RSUs also work as a bridge between the controller and vehicles. RSU exchanges network topology and other related information with the SDN controller. The data plane in SDN is used to manage both RSUs and vehicles.
- **Controller:** It is the high-processing entity of the ITS network and a trusted component. The SDN controller is connected to multiple controllers, and only the controller can add blocks in the blockchain and distribute them amongst other controllers. Controllers use a modified Practical Byzantine Fault Tolerance (mPBFT) consensus algorithm in LBSV and a lightweight consensus algorithm (LCA) in SmartCoin to run the blockchain functionalities.
- **SDN Operations:** SDN architecture provides a multitenant environment and offers flexibility to programmers to manage the increasing complexity of computing systems. In the context of ITS, heterogeneous communication introduces several security challenges that can lead to information leakage. By using SDN, we can separate application services from security services, thereby enhancing information security. The integration of blockchain technology with SDN can address the single point of failure typically associated with centralized systems, creating a decentralized computing environment. SDN architecture also offers a global overview of the network and facilitates management services such as protocol enabling, topological information, and communication setup.

Our proposed scheme utilizes the OpenFlow protocol for communication. LBSV incorporates a hybrid structure to facilitate communication between different components. The SDN controller is responsible for controlling network behavior and creating the blockchain. We assume that all vehicles are required to send information exchange messages to the control plane in SDN. The control plane then evaluates the optimum route to the destination as requested by the source vehicle. Each vehicle prepares its routing table based on the algorithm running at the controller plane. The controller recalculates the routing paths if an information packet fails to reach its destination. If consecutive failures occur for the same request, the route will be considered a failure. In the hybrid scenario, SDN does not bear all the communication overhead; instead, local computations are performed by the RSUs and the vehicles. The TPD in the vehicle functions as an OpenFlow switch, while the RSUs are used to exchange topological information with the SDN. Our configuration has three primary entities: the SDN controller, TPDs functioning as OpenFlow switches, and the SDN RSUs.

While SDN introduces centralized control for the efficient management of network operations, the LBSV scheme employs multiple SDN controllers to create a decentralized blockchain network. This decentralized approach eliminates any single point of failure, thereby enhancing the system's resilience. These SDN controllers collaborate to maintain the blockchain, ensuring secure communication and the transmission and authentication of messages across the network. The integration of SDN with blockchain technology, along with the use of multiple controllers, also allows for dynamic management of network resources, making the LBSV architecture well-suited to the dynamic and complex environment of ITS.

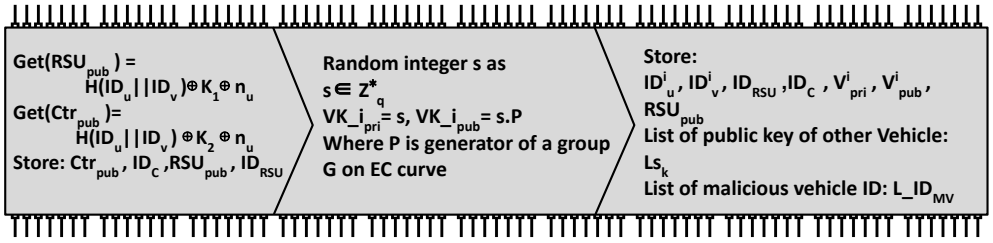


Figure 5.10: Representation of TPD with computation of different parameters.

Table 5.6: Symbol table for LBSV and SmartCoin

Symbol	Description	Symbol	Description
σ_x	Digital Signature of x	$H(x)$	Hash (SHA256) function for the input x
V_i	i^{th} Vehicle	ID_u^i	ID of i^{th} user
ID_v^i	ID of i^{th} vehicle	ID_{RSU}	ID of i^{th} RSU
ID_c	ID of i^{th} controller	Ctr_{id}	Controller's ID
$\triangle T$	Threshold time	V_{pub}^i, V_{pri}^i	Public-private Key-pair of vehicle $\forall$ i = 1,2,...
Ctr_{pub}, Ctr_{pri}	Public-private Key-pair of controller	RSU_{pub}, RSU_{pri}	Public-private Key-pair of RSU
Tr_1, Tr_2,....	Transactions	$Mrt()$	A function representation to create Merkle root hash

5.2.3 Assumptions

We have made some assumptions in the scheme. Before being part of the ITS network, a basic setup for the smart vehicle is needed. This setup is required for the necessary credentials for registration and communication. The manufacturing factory and user should perform some computation procedures.

1. The dedicated short-range communication (DSRC) protocol has been amended and renamed as wireless access in vehicular environments (WAVE). The WAVE protocol uses the IEEE 802.11p wireless communication standard. According to the standard, every vehicle disseminates messages related to road traffic in the time interval of 100–300 ms to other vehicles or RSUs. According to WAVE protocol, smart transportation's communication rate ranges from 6 to 27 Mbps, with the maximum broadcast coverage ranging up to 1,000 m.

2. A user U_i gets ID_v^i after purchasing a smart vehicle. Then U_i generates some random nonce n_u and computes $X_i = U_i \oplus ID_v^i \oplus n_u$ and sends $< U_i, X_i >$ to the factory through the secure channel [SSL/TLS].

3. The factory already knows the ID_v^i corresponding to the U_i. The factory receives the request, extracts the random nonce as $n_u = X_i \oplus ID_v^i \oplus U_i$, and computes $Y_i = H(X_i||n_u||n_f)$ with random nonce n_f. Further, The factory stores the Y_i in its secure storage. We assume that the factory database is highly secure. The factory also stores Y_i in the TPD and sends it to the user to install it in the vehicle. The factory's database is shared with the controller node of the ITS network to identify the authentic vehicles.

4. After installing the TPD in the vehicle ID_v^i, it generates the keypairs (V_{pub}^i, V_{pri}^i) using the ECC cryptographic algorithm. We also assume that TPD can calculate the hash (SHA256), digital signature, and verification of the messages.

5. The RSUs and controller node also generate their keypair (RSU_{pub}, RSU_{pri}) and (Ctr_{pub}, Ctr_{pri}) respectively using the ECC, and they share their public key with other vehicles and RSUs.

5.2.4 Adversary Model

We have used widely accepted Canetti and Krawczyk's (CK) adversary model for the proposed scheme [151]. The adversary model lists the assumption that is considered from the ITS network model. We assume that each vehicle is equipped with TPD and can communicate with RSUs and other vehicles in the range of TPD.

- Any vehicle/RSU/controller that acts like the adversary can send a false transaction/message on behalf of any legitimate vehicle/RSU/controller if it has all the necessary credentials (during initialization, registration, and communication phase).
- The adversary should know the private data to compromise security. To satisfy this condition, the calculated and received hashes of the intermediate variables have to match.
- A legitimate entity would act as an attacker if it tried to impersonate another entity using its credentials. Blockchain could be beneficial in this scenario to overcome impersonation along with the ECC and SHA256 cryptographic functions.
- A TPD is a tamper-proof physical device that stores the encrypted information and decrypts it using the user's private key using the ECC cryptographic technique. Thus, an adversary can not extract any information from TPD.
- Due to public channel message communication, the adversary can interrupt/modify/stop/delay the message dissemination and perform other malicious activities. The message authentication code and ECC provide authentication and integrity.

5.2.5 LBSV: Lightweight Blockchain Security Protocol for Secure Storage and Communication in SDN-enabled IoV

LBSV is a security protocol for storage and communication in IoV. IoV is a critical element of the ITS network, which has many modes of communication, such as vehicle-to-vehicle (V2V), vehicle-to-RSU (V2R), or infrastructure (V2I), RSU-to-SDN controller (R2C), and vehicle-to-an SDN controller (V2C). We have considered a two-layered network model in LBSV as shown in Figure 5.9, which are the IoV layer and consensus layer. The IoV layer comprises vehicles and RSU, and the consensus layer comprises SDN controllers. The SDN controllers create the blockchain network and follow the modified Practical Byzantine fault Tolerance (mPBFT) consensus *Algorithm* 14 for message verification and block generation. All the entities of the IoV network have the keypair for secure communication. Many blockchain-based solutions have been proposed for IoV networks to provide secure information storage and communication. However, monitoring the network effectively and focusing on the integration of SDN architecture [152] with blockchain technology is essential. Differentiating from the state-of-the-art techniques, LBSV tackles security-related issues such as secure storage, communication, and efficient computation. The conventional security solutions are not directly applicable to IoV architectures due to their high computation and storage demand. Therefore, we introduce a blockchain-based architecture to provide a security solution for ITS applications. The distributed nature of blockchain provides leak-proof security and ensures tamper-proof data. LBSV uses an mPBFT consensus algorithm to overcome the limitation in terms of consensus delay present in traditional consensus algorithms such as conventional PBFT, PoW, and PoS. Further, for the resource constraint device in IoV, we used lightweight cryptographic functions such as ECC and the SHA256 for secure communication, authentication, and data integrity. Further, The SDN controller is programmed to operate in a hybrid manner and provide a global network view. The SDN controller does not assert full control over the network. Instead, it communicates with the RSUs to exchange routing policies. Resource availability and routing control are also managed by the control plane of SDN [153]–[155].

LBSV consists of three phases, which are the initialization phase, the registration phase, and the communication phase. In the initialization phase, a vehicle interacts with the SDN controller for the registration process. This is the first requirement for a vehicle to be a part of the IoV. The vehicle communicates with the SDN controller over the secure channel and exchanges credentials for the next phase, i.e., registration. The vehicle generates its public-private key pair and gets the TPD in the registration phase. The SDN controller also creates a block for the vehicle registration and shares it with the other SDN controller of the network. Notably, in the registration phase, the vehicle gets its TPD for communication and registers its public key to distribute in the network. The initialization and registration phases are necessary to protect the IoV network from various attacks. Along with protection from attack, it also provides confidentiality, integrity, and authentication for the communication between vehicles, RSUs, and SDN controllers. After the successful registration phase, the vehicle is ready to be a part of the IoV layer and communicate with RSUs and other vehicles in the IoV network. In the communication phase, V2V communication takes place. The V2V communication could be direct communication or via RSU, as depicted in Figure 5.9. The detailed description of all these phases is described in the subsequent sections.

Initialization Phase

Every vehicle goes through an initialization phase before starting any communication with the entities. The initialization is carried out between the vehicle and the controller. Here, we assume the controller has a

public key and RSU IDs. As shown in Figure 5.11a shows the initialization process of the proposed scheme. The steps of the initialization process are as follows:

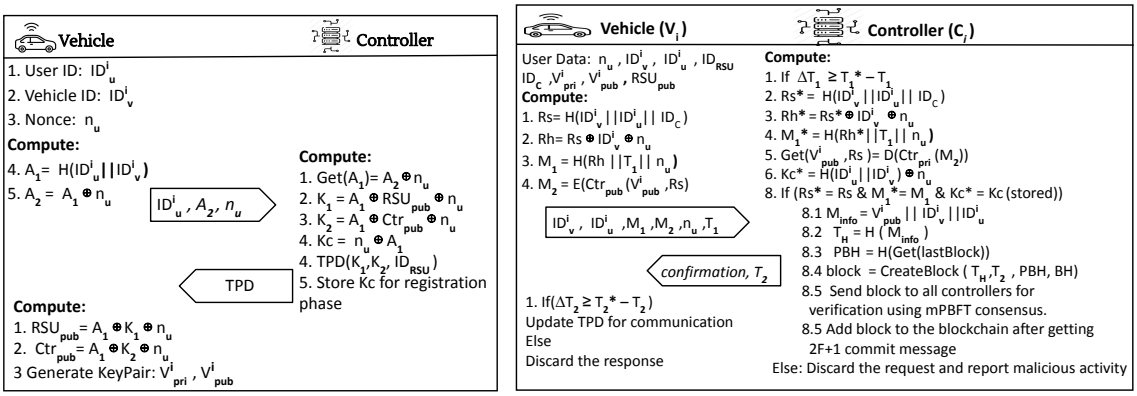
1. The user has its user ID, ID_u^i and vehicle ID, ID_v^i. It chooses a pseudo-random number called nonce n_u. Then it compute A_1 as $H(ID_u^i||ID_v^i)$ and A_2 as $A_1 \oplus n_u$. After that, the user sends $< ID_u^i, A_2, n_u >$ to the controller using a secure channel for getting the confidential credentials for future communication.

2. On receiving a request from the user, the controller computes A_1 from the received data as $A_2 \oplus n_u$. After computing A_1, it calculates K_1, K_2 and Kc as $A_1 \oplus RSU_{pub} \oplus n_u$, $A_1 \oplus Ctr_{pub} \oplus n_u$ and $n_u \oplus A_1$ respectively. Then, after storing the value of K and ID_{RSU} into the TPD, the controller also stores the value of Kc in its local repository against the vehicle ID for the registration phase.

3. After storing the K_1, K_2 and ID_{RSU} into the TPD, controller send $< TPD >$ to the user.

4. The user finds the public key of RSU and controller as $A_1 \oplus K_1 \oplus n_u$, $A_1 \oplus K_2 \oplus n_u$ respectively and stores it for future communication. The user also generates its key pair (V_{pub}^i, V_{pri}^i) for secure communication. Figure 5.10 shows the graphical representation of the TPD and the values stored in the TPD.

5. Apart from all this, the controller shares its public key (Ctr_{pub}) and ID (ID_C) with all the connected RSUs and other controllers over the secure channel. This communication between the controller and RSU is also secured with ECC and SHA256.



(a) Initialization **(b)** Registration

Figure 5.11: (a) Initialization phase of LBSV, (b) Registration phase of LBSV

Registration Phase

The registration phase includes two components, i.e., vehicles and the controller. In this phase, the vehicle has to register with the controller for secure communication. Figure 5.11b depicts the registration process of the LBSV, which is as follows:

1. Vehicle V_i computes Rs, Rh, M_1, M_2 in the same sequence in which they are mentioned. Then the vehicle V_i sends $< ID_u^i, ID_v^i, M_1, M_2, n_u, T_1 >$ to the controller.

2. The controller first checks $\triangle T_1 \geq T_2 - T_1$, if it holds true then compute Rs^*, Rh^*, M_1^*, $Get(V_{pub}^i, Rs) = D(Ctr_{pri}(M_2))$, Kc^*. Then controller check if $(Rs^* = Rs \ \& \ M_1^* = M_1 \ \& \ Kc^* = Kc(stored))$ is true then the controller computes M_{info}, T_H, PBH. After computing M_{info}, T_H, PBH controller creates the block as $block = CreateBlock(T_H, T_3, PBH, BH)$ and sends it to the other controller nodes for the verification of the block. It uses a mPBFT *Algorithm* 14 to consensus on blocks. When the controller receives $2N/3$ commit messages, it adds the block into the blockchain and sends the $< confirmation, T_3 >$ (confirmation of the registration) back to the vehicle V_i.

3. Vehicle V_i receive the response and check if $\triangle T_2 \geq T_3 - T_1$ is true or not. If it is true, then update the TPD for communication; otherwise, discard the response and send the request again.

Consensus Algorithm: mPBFT

The registration phase and communication phase use the consensus algorithm to propose the vehicle registration phase block. The algorithm works as follows. The SDN controllers in the IoV network share messages to commit a block. Malicious nodes (SDN controller) might broadcast a tampered block into the network; thus, the block validated (committed) by at least 2N/3 nodes is considered a valid block. Here, N is the number of total nodes (considering that 1N/3 are faulty nodes). The algorithm uses four states, i.e., NEW-ROUND, VERIFICATION, COMMITMENT, and APPEND. The state's workflow goes from NEW-ROUND to COMMITTED. The consensus algorithm introduces the leader election process so that every miner node in the network gets equal chances to propose the block. Due to the new leader election process, the consensus delay is minimized compared with the other consensus algorithms, such as Proof-of-Work, Proof-of-Stake, and conventional PBFT. Every node is given a fair chance to append its block to the blockchain. The details of mPBFT are described in *Algorithm* 14.

1. The first state is NEW-ROUND. In this state, the block proposer is selected from all the miner nodes (controllers). The new transactions (Tr_{New}) are appended in the transaction pool (TP) as $TP.append = Tr_{New}$ for processing. A new block proposer is selected based on the following equation

$$miner = R(mod)N \tag{5.6}$$

   Here, R is the number of rounds in the current instance, and N is the total node (including faulty nodes). Round R is the one complete block creation process from the block proposing to be appended to the blockchain. This procedure of selecting the block proposer is a uniform distribution of chances to propose the block, i.e., every miner node (controller) gets an equal chance to create the block. Furthermore, this process is very lightweight, resulting in low consensus delay.

2. After the block proposer is selected, it creates a transaction block using the $CreateBlock()$ function and broadcasts it for verification. The state is now changed from NEW-ROUND to VERIFICATION.

3. In the verification state, all the other miner nodes, except the leader node, verify the block, i.e., all the transactions included in the block are verified. If the total number of verification is greater than $2N/3$, a message for commitment is broadcast into the network, and the state is changed from VERIFICATION to COMMITMENT.

4. The commitment is nothing but a confirmation of the block to append into the blockchain. If the total number of commitment messages is more than $2N/3$, the state is changed from COMMITMENT to APPEND.

5. In the append state, the proposed block is appended into the blockchain by all the miner (controller) nodes. The state is changed from APPEND to NEW-ROUND.

Algorithm 14: Consensus Algorithm: mPBFT

Input: New Transactions: $Tr_i, i = 1,2,\ldots,n$, Nodes: N
Output: Blocks
Function `newRound(`Tr_{New}, R, N`):`
    node.state = NEW-ROUND;
    $TP.append = Tr_{New}$;
    $select_miner(R,N)$; `/* Function to select the miner who creates the block */`
    $miner = R \mod N$; `/* R: round number.  N: Total nodes */`
    if $node$ == $miner$ **then**
        $block = CreateBlock(TP.getAllTransactions)$;
        $broadcast_block(block)$;
        node.state = VERIFICATION;
Function `verification(`$block$`):`
    $verify_block(block)$;
    if $(count(message.verification) \geq 2N/3)$ **then**
        $broadcast_message(message.commitment)$;
        node.state = COMMITMENT;
Function `commitement((`$message.commitment$`):`
    $commit_pool.append(message.commitment)$;
    if $(count(message.commitment) \geq 2N/3)$ **then**
        node.state = APPEND;
Function `append(`$block$`):`
    $blockchain.append = block$;
    node.state=NEW-ROUND;

Communication Phase

The SDN controller manages the public key and is granted to the vehicles through RSU, who want to communicate with other vehicles. The SDN controller verifies the identity of every vehicle requested before granting a public key. Every vehicle maintains and updates one list Ls_k, which is the list of the authenticated vehicle's public key and vehicle ID. The vehicle does not need to take a public key from the RSU and SDN controller at all times. Figure 5.12 depicts the key exchange and communication process in LBSV. The public key is not directly shared with the other vehicle. Instead, it is communicated via an SDN controller into the network. There are two reasons behind this process. The first is to receive information only from identified vehicles. Secondly, if any vehicle sends misinformation, it can be traced immediately to disqualify the network and be reported as a malicious vehicle. Legitimate vehicles can be protected from malicious vehicles using this key exchange method. Specifically, the key exchange and communication process is described below:

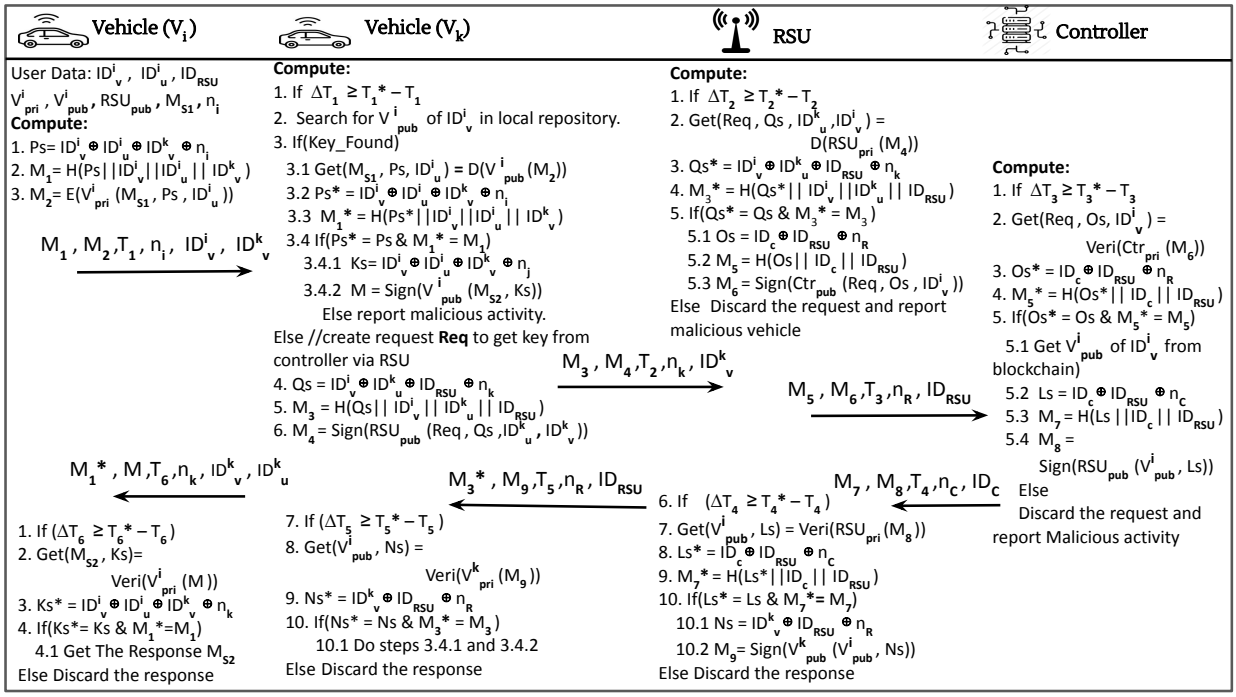


Figure 5.12: Communication phase of LBSV

1. Vehicle V_i generates the secret message M_{S1} and nonce n_i, and computes Ps, M_1, M_2. Vehicle V_i send $< M_1, M_2, T_1, n_i, ID^i_v, ID^k_v >$ to vehicle V_k.

2. Vehicle V_k receives the message from vehicle V_i. V_k first checks if $\triangle T_1 \geq T_1^* - T_1$ is true or not; if true, then proceed further. Otherwise, discard the response. If the condition holds true then vehicle V_k first look for the public key of vehicle V_i in local repository and do the following computation, $Get(M_{S1}, Ps, ID^i_u)$, Ps^*, M_1^* and then check if($Ps^* = Ps$ & $M_1^* = M_1$) is true or not, if true then do computation, Ks, M and response to the vehicle V_i. But if the key is not found at a local repository, the vehicle V_k sends a request to the controller via RSU to get the public key of vehicle V_i. The request is generated with Qs, M_3, M_4, and sends the $< M_3, M_4, T_2, n_k, ID^k_v >$ to RSU.

3. RSU receives the values and first it check if $\triangle T_2 \geq T_2^* - T_2$ is true or not. If it is true, then RSU does further computation else, it discards the request. If the above condition is true then RSU compute $Get(Req, Qs, ID^k_u, ID^i_v) = D(RSU_{pri}(M_4))$, Qs^*, M_3^*, then check if $(Qs^* = Qs$ & $M_3^* = M_3)$ is true then do further computation else report malicious activity. Then RSU does the following computation Os, M_5, M_6 and sends the $< M_5, M_6, T_3, n_R, ID_{RSU} >$ to the controller.

4. Controller receives the data and first checks if $\triangle T_3 \geq T_3^* - T_3$ is true, then do the following computation, $Get(Req, Os, ID^i_v) = D(Ctr_{pri}(M_6))$, Os^*, M_5^* and then check if $(Os^* = Os$ & $M_5^* = M_5)$ is true then go for further computation else discard the response and report malicious activity. If true then it compute $Get(V^i_{pub})$ of ID^i_v from blockchain, then Ls, M_7, M_8 and send the $< M_7, M_8, T_4, n_C, ID_C >$ to requesting RSU.

5. RSU receive the response and check if $\triangle T_4 \geq T_4^* - T_4$ is true then do the following computation, $Get(V^i_{pub}, Ls) = D(RSU_{pri}(M_8))$, Ls^*, M^*, then check if $(Ls^* = Ls$ & $M_7^* = M_7)$ is true then Ns, M_9 and RSU send the $< M_3^*, M_4, T_5, n_R, ID_{RSU} >$ to the vehicle V_j.

6. Vehicle V_k receive the data and check if $\triangle T_5 \geq T_5^* - T_5$ is true then vehicle compute the following, $Get(V_{pub}^i, Ns) = D(V_{pri}^k(M_9)), Ns^*$, then vehicle check if$(Ns^* = Ns \ \& \ M_3^* = M_3)$ is true then it compute Ks, M and send the $< M_1^*, M, T_6, n_j, ID_u^k, ID_v^k >$ to vehicle V_i.

7. Vehicle V_i receive the response and first check if $\triangle T_6 \geq T_6^* - T_6$ is true then do the following computation, $Get(M_{S2}, Ks) = D(V_{pri}^i(M), Ks^*$ and then check if $(Ks^* = Ks \ \& \ M_1^* = M_1)$ is true then get the response and act accordingly otherwise discard the response.

5.2.6 SmartCoin: A Novel Incentive Mechanism for Vehicles in Intelligent Transportation System Based on Consortium Blockchain

An ITS improves traffic safety and efficiency by enabling vehicles to generate and broadcast messages. However, trusting the received message's credibility in an untrusted environment or network is difficult. Additionally, there is no reward mechanism for the honest message generator. Hence, we proposed a novel incentive mechanism for vehicles called SmartCoin based on a consortium blockchain to improve safety and network efficiency and encourage more vehicles to participate in network improvisation. The proposed scheme aims to achieve social welfare, improved transportation, less traffic congestion, reduced road accidents, and a transportation network free from fraud messages. In the proposed scheme, vehicles give the rating to the message source vehicle based on the validity of the messages. Some SmartCoins are credited to the vehicle's account at the blockchain according to the message rating. Table 5.7 depicts different messages and their corresponding SmartCoins. The SmartCoins could be redeemed either at the fuel station, electric charge station for electric vehicles, or vehicle service station. Blockchain technology provides decentralization, immutability, and transparency to the ITS. The secure storage for the SmartCoin, traffic messages, and user information is achieved using blockchain. The proposed scheme encourages vehicles to participate and rewards them for their honesty. Vehicles rate the messages they receive, and based on that rating, some SmartCoins are credited to the vehicles' accounts in the blockchain. The scheme has two phases, i.e., the registration and communication phase. We have employed the consortium blockchain, where vehicles and RSU are the public nodes that generate, receive, and rate the messages but are not responsible for block mining. In contrast, the controller nodes make the private blockchain network, mine the blocks, and append them to the blockchain. We also explained the incentive mechanism of honest vehicles, which generates messages to improve network efficiency. We have defined the proposed Lightweight Consensus *Algorithm* (LCA) 15.

Message Types: In ITS, there are various types of traffic messages. Table 5.7 shows different types of messages and the corresponding value of SmartCoin (STC), which is to be credited into the vehicle account if messages are generated by the vehicles. The messages are arranged in priority order. $M1$ has the highest priority, so the SmartCoins are, and $M7$ has the lowest priority. The priority is assigned based on the emergency to take action. The Accident is a life-lost event, so it has the highest priority. Similarly, priority is assigned to other types of messages. The STC is the representation unit for SmartCoins like BTC in Bitcoin cryptocurrencies.

Registration Phase

For communication with other vehicles and RSUs, the vehicle must first register at the controller node. The communication channel is secure in the registration phase. The registration process is as follows:

Table 5.7: Types of messages and the corresponding SmartCoins

Symbol	Meaning	Credited SmartCoin (STC) into Vehicle/User Account
$M1$	Accident occurred	100 STC
$M2$	Accident area	50 STC
$M3$	School zone	30 STC
$M4$	Malicious information	20 STC
$M5$	Traffic jam	15 STC
$M6$	Fuel station	10 STC
$M7$	Road intersection	5 STC

1. First, vehicle V_i computes $A_i = (H(Y_i)||M_{req}||V^i_{pub})$ and $B_i = A_i \oplus ID^i_v$, and send the $< ID^i_v, B_i, T_1 >$ to the controller node.

2. The controller node first checks the $\Delta T_1 \geq T_2 - T_1$, here T_2 is the current timestamp of controller node. Then, if the condition is satisfied, the request is processed further. It computes $C_i = B_i \oplus ID^i_v$ and extracts the $H(Y_i)$. The controller node connects to the factory database, gets the Y_i corresponding to ID^i_v, and checks $H(Y_i)' \equiv H(Y_i)$.

3. If both $H(Y_i)$ matched, it compute, $D_i = H(C_i) \oplus (Ctr_{pub}||H(Y_i))$ and $E_i = H(C_i) \oplus (RSU_{pub}||H(Y_i))$. Controller node sends $< D_i, E_i, T_3 >$ to request vehicle V_i over the secure channel. After that, it creates the transaction as $Tr_1 =< (ID^i_v \oplus V^i_{pub}) >$. The same process followed for other vehicles as well. Hence, the controller has many transactions $< Tr_1, Tr_2, ..., Tr_n >$. The controller now creates the block after broadcasting the transactions to other controller nodes in the network using a digital signature. The other nodes verify this transaction and keep it in their transaction pool.

4. To create the block, controller node compute, $Mrt()$= $H(H(Tr_1), H(Tr_2), ..., H(Tr_n), BH, PBH$ and create the block $Block = createBlock(B_{ID}, Mrt(), PBH, BH, T_2, < H(Tr_1), H(Tr_2), ..., H(Tr_n) >)$. The controller node sends B_i to other nodes for verification. After verification, $Block$ is appended to the blockchain ledger by all the controller nodes so that the copy of the blockchain is identical to all the nodes.

5. The controller nodes also share the $< ID^i_v, V^i_{pub}$ with all connected RSUs using signature and verification, so that the vehicles get the public key of other vehicles from RSUs, i.e.,$\sigma = SIGN(Ctr_{pri}, (ID^i_v, V^i_{pub}))$, $VERIFY(Ctr_{pub}, (ID^i_v, V^i_{pub}), \sigma)$.

6. The requesting vehicle receives $< D_i, E_i, T_3 >$ from controller node. Then it checks $\Delta T_2 \geq T_4 - T_3$. Then, also check $H(A_i) \equiv H(C_i)$ and $H(Y_i) \equiv H(Y_i)$, and if both found true, then it extracts Ctr_{pub} and RSU_{pub} and store them into TPD.

Communication Phase and Credit of SmartCoin into Vehicle Account

As explained in the registration phase, the vehicle's public key, RSU, and controller node are shared. The wireless access in vehicular environments (WAVE) protocol is being used for communication in the ITS. Figure 5.13 shows the communication of V2V, V2R and R2N. While running on the road, the vehicle observes the events mentioned in Table 5.7 and broadcasts them into the network, either V2V or V2R. The communication channel in this phase is open. The communication can be defined as follows:

1. Vehicle V^i observe the event, e.g., M_i. After that, the vehicle sends this message to either another vehicle or RSU, whichever is in the communication range. Vehicle computes $\sigma_{M_k} = SIGN(V^i_{pri}, M_k)$ for $k = 1, 2, ..., 7$ and sends to either RSU or another vehicle. In any case, i.e., RSU or vehicle received a message is verified using the vehicle's public key as $VERIFY(V^i_{pub}, M_k, \sigma_{M_k})$. If the output is true, it is checked whether the received message is genuine or fake. If it is genuine, a one (1) rating is given; otherwise, a zero (0) rating is given.

2. The response to the received message is as follows. If genuine message is found then $\sigma_{(M_k||1)} = SIGN(V^j_{pri}, (M_k||1))$, and if fake message is found then $\sigma_{(M_k||0)} = SIGN(V^j_{pri}, (M1||0))$. the same response is also sent to the RSU (response signed by RSUs public key RSU_{pub}) so that the genuine rating can be recorded.

3. If the positive rating is more than 50%, then the SmartCoin corresponding to the message (mentioned in table 5.7) is credited into the vehicle's account and stored in the blockchain. For example in case of message type $M1$, which we have taken, the 100 STC as:
SmartCoin:1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA (The address of vehicle account)
amount=100
label=warning message
message=accident occurred
A transaction is created and added to the UTXO (unspent transaction list) of the vehicle account so that these coins can be redeemed at the fuel station or service station. The UTXO of the vehicle account looks like

```
[
    {
        "txid" : "9ca8f969bd3ef......c309acbae2c14ae3",
        "SmartCoin" : "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA",
        "scriptPubKey" : "76a91407bdb51....c9c13d1fd288ac",
        "Script"    :   "188192f....481f9fa77d116f39222a",
        "amount" : 100,
        "confirmations" : 200
    }
]
```

Lightweight Consensus Algorithm

In SmartCoin, the controller node makes the private blockchain, and the vehicles and RSU work as the public network to help authenticate the generated information. Thus, it is called consortium blockchain. The controller nodes mine the block, and the consensus algorithm is used to propose the block into the blockchain, which contains the network traffic messages and SmartCoins. Though the controller node network is a private blockchain network with only trusted controller nodes, there is no need to apply any competition amongst the nodes for block proposals like PoW. Therefore, we apply the Round Robin scheme, where every controller node proposes a block in the given time quantum δT. All the nodes are arranged in order, and every node is allowed to propose a block when δT of the previous block has elapsed. The *Algorithm* 15 works as follows for block creation and verification.

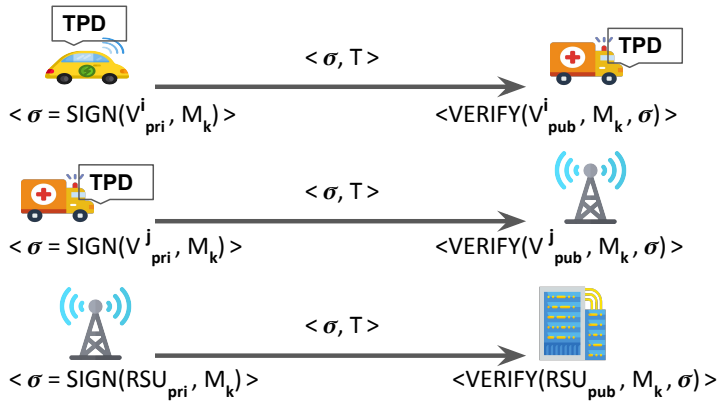


Figure 5.13: Communication between Vehicle, RSU, and Controller Node

1. The public nodes vehicles and RSUs are the traffic messages originator that generates the message based on the type of condition (See Table 5.7) and put them in the *transaction_pool* at the controller node.

2. The controller node assembles all the transactions from the *transaction_pool* and verifies them with the help of public node participants, i.e., vehicle. The recipient controller node then broadcasts these verified transactions to other controller nodes. After that, the selection block proposer calls the function *createHashTree* to calculate the *Mrt* of these transactions.

3. Next, the block proposer controller node also calculates the *PBH* using the *getHash* function along with the current timestamp T.

4. Finally, the block proposer controller node gathers all the calculated parameters, which are *Mrt*, *PBH*, T and *nonce* and generates the block. A *createBlock*() function is used to generate the function. After generating the block, the controller node broadcasts it to the network for verification. The block generator signs a block.

5. Other than the proposer node, other nodes are responsible for verifying the newly generated block, which is first verified using the generator's public key. To validate that the generated node contains legitimate information/transactions, first, they calculate the *Mrt* from the *transaction_pool* and check whether the calculated block matches the received block. If the match is correct, the *block_status* is set as *valid*; otherwise set as *invalid*. The *node.status* is also set as bad, so the node is not considered for the next block proposer.

6. The block proposer controller node confirms the new block if it receives at least $3N/4$ *valid* response where N is the total node. This means that if 75% of the node verifies, the block is considered the valid node.

5.2.7 Implementation Environment and Performance Analysis

This section explains the proposed scheme's implementation environment and performance analysis compared with state-of-the-art schemes. The performance analysis includes computation cost, energy consumption, storage cost, communication cost, and consensus delay. We have implemented the scheme using

Algorithm 15: Lightweight Consensus Algorithm

Input: Transaction: $Tr_1, Tr_2, ...Tr_n$
Output: block
$TP.append = Tr_i$; `/* append all the new transactions in the transaction pool */`
Function `blockGenerate(`Tr_{New}`):`
  if *node* == *proposer* **then**
    $Mrt = createHashTree(TP.getAllTransactions)$;
    $T = Timestamp$;
    $nonce = random_number$;
    $PBH = getHash(previous_block)$;
    $BH = getHash(Mrt, T, PBH, nonce)$;
    $block = createBlock(Mrt, T, nonce, PBH, BH)$;
    $sign_block(block)$;
Function `blockVerification(`*block, nonce*`):`
  if *node != proposer* **then**
    $receive_block(block, nonce)$;
    $Mrt = createHashTree(TP.getAllTransactions)$;
    $block* = createBlock(Mrt, T, nonce, PBH, BH)$;
    if $block == block*$ **then**
      $block_status(valid)$;
    else
      $block_status(not_valid)$;
      $node.status = bad$;
  if $(count(block_status == valid) \geq 3N/4)$ **then**
    $blockchain.append(block)$;

Raspberry Pi devices and produced results that are promising and convincing. Below, we have explained the analysis in detail.

Implementation Environment

The performance assessment for LBSV and SmartCoin is conducted using the Raspberry Pi 4 and Windows OS-based Desktop. The hardware configuration of Raspberry Pi is as follows, Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz processor, 2 GB memory, standard 40-pin GPIO header, and Linux-based Noobs v3.3.1 operating system. The hardware configuration of the Windows desktop system is Intel i5 processor 9th Generation, 8 GB RAM, 512 GB SSD secondary storage, Windows 11 operating system, 4 GB Graphics card of NVIDIA GeForce GTX 1650. Java programming is used to script blockchain consensus algorithms and cryptographic functions. Netbeans IDE 11 (64-bit) and Java 8 (Oracle Java SE, 64-bit) frameworks are used to perform the experiment. We have considered Raspberry Pi, 2GB as the lightweight device (i.e., OBU and RSU) and Desktop System as the controller node.

Table 5.8: Individual computation time of different cryptographic functions in the testbed environment

Functions	On Desktop	On Raspberry Pi 2GB
SHA256 Hash function: $T_{H()}$	0.0008	0.045
ECC Crypto-system: T_{ECC}	0.053	3.9
ECDSA signature: T_{sign}	0.071	6.13
ECDSA verification: T_{veri}	0.065	5.63
point multiplication: $T_{mul(.)}$	0.117	11.0
AES Crypto-system 192-bit: T_{AES}	0.125	12.71
Modulo operation: $T_{mod(.)}$	0.067	5.0

Computational Cost and Energy Consumption

Computational time is the time taken by the simulator for the execution of the whole framework. The measuring unit is milliseconds (MS) and represented as T_{ct}. We illustrate the computational cost (also known as execution time) based on the operational cost for authentication, verification, and communication. The different cryptographic functions used such as $\oplus$: bitwise XOR, T_{AES}: AES encryption/decryption 192-bit, T_{ECC}: ECDSA encryption/decryption, $T_{H()}$: SHA 256 hash function, T_{sign}: ECDSA signature, T_{veri}: ECDSA verification, $T_{mul(.)}$: point multiplication, $T_{mod(.)}$: Mod operation (Assuming that XOR and concatenation operations take negligible time compared to other operations). Table 5.8 shows the execution time of different cryptographic functions. Table 5.9 shows the comparative computation cost of SmartCoin and LBSV with state-of-the-art schemes. The energy consumption is the energy consumed by the system to perform the operations of the scheme. In the execution process, the system requires some energy to carry out all the phases of the scheme. The energy consumption is calculated as $E_{rg} = T_{ct} * C_P$, where C_P is a CPU maximum power (10.88 W for a laptop and 2.4W for Raspberry Pi 4) and T_{ct} is a computational time. It is measured in Joules We have calculated the computational cost in the previous section. Here, we calculate the energy consumption. We calculated the energy consumption on two platforms, i.e., Desktop and Raspberry Pi. Table 5.9 shows the comparative analysis of LBSV and SmartCoin energy consumption with state-of-the-art schemes. It is very clear from the table that energy consumption is reduced in LBSV and SmartCoin compared to state-of-the-art schemes.

Table 5.9: Computation cost (CC) and energy consumption (EC) of LBSV and SmartCoin compared with state-of-the-art schemes on Desktop and Raspberry Pi (RP)

Scheme	Computation Cost Equation	CC on Desktop	CC on RP4	EC on Desktop	EC on RP4
[156]	$10T_{H(.)} + 4T_{sign} + 4T_{veri}$	0.552	47.44	6.01	113.85
[157]	$8T_{H(.)} + 3T_{ECC} + 3T_{sign} + 3T_{veri}$	0.572	47.80	6.24	114.72
[110]	$5S_{H(.)} + 3T_{sign} + 3T_{veri}$	0.548	47.35	5.96	113.64
[158]	$5T_{H(.)} + 2T_{sign} + 2T_{veri}$	0.276	23.74	3.01	57.00
[120]	$14T_{H(.)} + 2T_{sign} + 2T_{veri} + 1T_{mul(.)}$	0.400	35.15	4.35	84.36
LBSV	$15T_{H(.)} + 2T_{ECC} + 2T_{sign} + 2T_{veri}$	0.390	31.99	4.24	76.56
SmartCoin	$6T_{H(.)} + 4T_{sign} + 4T_{veri}$	0.410	35.05	4.46	84.12

Storage Cost and Communication Cost

Storage cost refers to the storage required for storing the values such as secret key, hash value, and other parameters in all phases. The measuring unit for storage cost is bytes and represented as S_{sc}. Generally, a Key S_k requires 24B (B=Bytes), ECC encryption/decryption S_{ECC} takes 48B, AES encryption/decryption S_{AES} takes 24B, a hash function (SHA-256 bit) $S_{H(.)}$ takes 32B, and ECDSA based signature S_{sign} and verification S_{veri} requires 10B. Table 5.9 and Figure 5.14a represent the comparative storage cost of the LBSV and Smartcoin with state-of-the-art schemes. Communication cost refers to the storage required for storing the

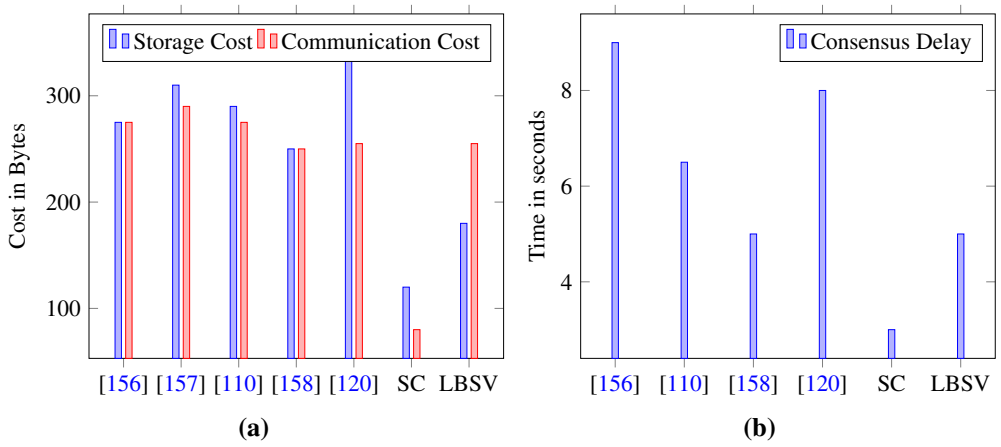


Figure 5.14: (a) Storage and communication cost of SmartCoin (SC) and LBSV with state-of-the-art scheme (b) Consensus delay of SmartCoin (SC) and LBSV with state-of-the-art scheme

Table 5.10: Comparative storage and communication cost of LBSV and SmartCoin with state-of-the-art schemes

Scheme	Storage Cost	Cost in Bytes	Communication Cost	Cost in Bytes
[156]	$8S_{H(.)}+2S_{sign}$	275	$8S_{H(.)}+2S_{sign}$	275
[157]	$6S_{H(.)}+3S_{ECC}+2S_{sign}$	310	$5S_{H(.)}+3S_{ECC}+2S_{k}$	290
[110]	$8S_{H(.)}+4S_{sign}$	290	$7S_{H(.)}+5S_{sign}$	275
[158]	$6S_{H(.)}+6S_{sign}$	250	$6S_{H(.)}+6S_{sign}$	250
[120]	$10S_{H(.)}+1S_{sign}+S_{mul(.)}$	350	$7S_{H(.)}+1S_{sign}+S_{mul(.)}$	255
LBSV	$3S_{H(.)}+2S_{Sign}$	116	$6S_{H(.)}+2S_{ECC}$	240
SmartCoin	$3S_{H(.)}+2S_{sign}+1S_{ECC}$	164	$3S_{H(.)}+3S_{sign}$	126

communicating variables such as hash code, ciphertext, and other parameters. The measuring unit is bytes and represented as C_{cc}. The operations and the variables that have been used in communication between candidates and controllers are a hash function, ECDSA based digital signature. The storage requirement for different cryptographic operations is mentioned in the previous section. Table 5.10 and Figure 5.14a represent the comparative storage cost of the LBSV and SmartCoin with state-of-the-art schemes.

Consensus Delay

The LBSV used the mPBFT, and SmartCoin used the LCA consensus algorithm for block creation and verification. A fair leader selection is very important because if only fewer nodes are selected repeatedly, it will attract the attackers and behave like a centralized system. For a fully decentralized system, all the nodes should have an equal chance to propose a block. Therefore, we employ the simple yet effective leader selection process to elect a leader to propose the block. Due to this mechanism, the consensus delay of the LBSV and SmartCoin is minimized. Furthermore, in SmartCoin, the message communication overhead in LCA is less than that of LBSV. Therefore, the SmartCoin shows the lowest consensus delay. The state-of-the-art schemes also use different consensus algorithms for block verification and validation such as delegated proof-of-stake (DPoS) used in [158], in [110], PoW and PBFT are used. In [120], PBFT is used for message verification under the hyperledger fabric framework. Among all the consensus algorithms used by all these schemes, the mPBFT and LCA perform block creation, verification, and validation in the

least time, which is much less than the others. Figure 5.14b shows a comparative analysis of LBSV and SmartCoin with state-of-the-art schemes.

SDN Computational Efficiency

We also calculated the computational efficiency of the SDN controller. An SDN controller's computational efficiency can be defined as the time taken to respond to a communication request and blockchain creation process. Figure 5.15 represents the computational efficiency of our proposed scheme. It can be evident from the figure that the computational cost is meager in the low-dense scenario because of the small chaining of the blockchain. Due to fewer communicating parties, verifying the blocks and the creation of the blockchain is easy. In the higher denser scenario, the computation cost increased by a small margin. This is a small price to pay for the benefits we are getting from the proposed protocol.

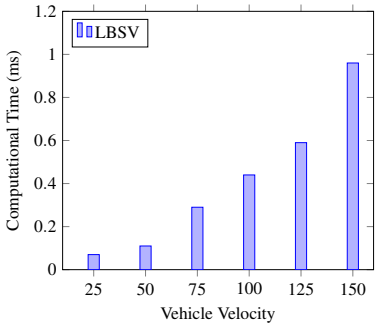


Figure 5.15: Computational efficiency of SDN

5.2.8 Security Analysis

In this section, we explain the security analysis of LBSV and SmartCoin based on the cryptographic and blockchain aspects. Theorems 1, 2, and 3 claim the cryptographic aspect, and Lemmas 11, 12, and 13 claim the blockchain aspect. Before proving the blockchain aspect, we have also provided the attacker scenario for more clear understanding.

Attacker Scenario

The attack scenario is extended compared to Nakamoto's consideration. Here, we add a finite-time attack model, i.e., a time gap exists for the possible attack. Other than the honest miners, there are also some adversaries or attackers. The honest miners tend to the honest longest chain, and the attacker tends to the fraudulent chain. To launch the double-spending attack, the attacker targets the transaction to pay for goods and services (such as in SmartCoin, the services are a service station, fuel station, and an electric vehicle charging station). A target transaction is a transaction made by an attacker to transfer cryptocurrency from the attacker to the victim. Let, at time t, the last block of the honest chain is generated. At the time $t+i$, the attacker broadcasts the fake transaction in the network to be included in the next block, puts it into the honest chain, and simultaneously creates the fork in the honest chain, which derives from the last block secretly. This secret fork is referred to as a fraudulent chain containing the attacker's fake transactions. A block confirmation time is when the victim is waiting before providing any service (which should be minimal in real-time applications like IoV). Let N_{BC} be the number of blocks chosen to be a block confirmation time.

The fraudulent chain (a fork in the honest chain) is made public once it is longer than the honest chain after the block confirmation time. The SmartCoin withstands double-spending attacks, fork attacks and fault tolerance, and we proved this in the following lemma considering the above attack scenario.

Theorem 1. *The SmartCoin supports the authenticity of the data shared between TPD and the controller node.*

Proof. The digital signature is secured under random message attack if for the message (transaction of SmartCoin) space M and all adversaries $\mathbb{A}$ the success possibility of $\mathbb{A}$ is negligible. Let, $M1, M2, ...Mn$ are the uniformly chosen messages at random from message space M and $KEYGEN(k)$ generates the private-public keypair (V^i_{pub}, V^i_{pri}). Let the adversary compute the digital signature of above mentioned chosen messages as $\sigma_{M1} \leftarrow SIGN(V^i_{pri}, M1), \ldots \sigma_{Mn} \leftarrow SIGN(V^i_{pri}, Mn)$. The adversary $\mathbb{A}$ is given V^i_{pub} and $(Mi, \sigma_{Mi})^n_{i=1}$ and output (M, σ_M). Adversary $\mathbb{A}$ is succeeded if $VERIFY(V^i_{pub}, X, \sigma_X) = True$ and $X \notin (M1, M2, ..., Mn)$. The scheme is strongly protected under a random message attack if the following condition is satisfied. $VERIFY(V^i_{pub}, X, \sigma_X) = True$ and $(X, \sigma_X) \notin \{(M1, \sigma_{M1}), (M2, \sigma_{M2}), ..., (Mn, \sigma_{Mn})\}$. In SmartCoin, an adversary $\mathbb{A}$ could observe the communication channel. However, messages (M) and transactions (Tr) from the TPD are signed $\sigma_{Tr/M} \leftarrow SIGN(V^i_{pri}, tr/M)$ and verified $VERIFY(V^i_{pub}, Tr/M, \sigma_{Tr/M})$ by each receiver before further process. An adversary cannot calculate the hash of the transaction. Even if the adversary hijacks the channel and gets the signed messages, the corresponding verification cannot be done. At the same time, blockchain guarantees the authenticity of stored information. □

Theorem 2. *The SmartCoin supports data integrity of the network traffic messages, user information, and transactions.*

Proof. SmartCoin employs the signature and verification algorithm. Suppose an adversary is able to know the transactions and try to forge them. In that case, the adversary again has to sign the transaction, which requires the private key of the vehicle or controller node whose transaction is to be modified by an adversary, which is not possible. Hence, the scheme provides the integrity of the transactions—furthermore, the blockchain records the information in the form of a block, which is immutable and tamper-proof. Therefore, the integrity of the information contained in the block can be assured. The attack is possible only if the adversary successfully acquires 51% of the legitimate nodes. □

Theorem 3. *The SmartCoin supports non-repudiation of the network traffic messages, user information, and transactions.*

Proof. With the help of proof of origin (PoO) and mandatory proof of receipt (PoR) [159], it can be justified that SmartCoin supports non-repudiation of network traffic messages in the ITS network. A mutual non-repudiation is achieved by encompassing the mandatory PoO and PoR both. The PoO must associate the originator's identity with exchanged information to not deny it after generating the message. Similarly, the PoR is also associated with the recipient's identity. The association in both must be undeniable and unforgeable. We have achieved both properties using the ECDSA-based digital signature. The proof can be described as follows.

$$PoO = Sign(V^i_{pri}[\mathbb{O}_V, V, RSU, M_i], T_1), \forall i = 1,2,3,4,5,6,7 \tag{5.7}$$

$$PoR = Sign(RSU_{pri}[\mathbb{O}_{RSU}, RSU, V, H(PoO)], T_2) \tag{5.8}$$

where $\mathbb{O}_V, \mathbb{O}_{RSU}$ indicates that this structure is represented as PoO and PoR, respectively. V and RSU are the originator and recipient (i.e., Vehicle and RSU), respectively. T_1, T_2 are the timestamps and M_i is the type of message where i varies from 1 to 7. In the PoO and PoR, the identity of vehicles and RSUs are associated, which is explained in the registration and communication phase of the SmartCoin scheme. Thus, the above two equations clearly show that the proposed scheme supports non-repudiation. It is very important to show it because it helps identify the malicious vehicles that try to broadcast fake messages. Additionally, the immutability of the signature assures the non-repudiation of the transaction. The controller node puts all the transactions in the block and publishes them in the blockchain. The vehicle can obtain the SmartCoin corresponding to its unique ID and private key. Once the information is published into the blockchain, it can not be modified. Hence, the SmartCoin withstands non-repudiation. □

Lemma 11. *The SmartCoin protocol is protected from double-spending attacks. A double-spending attack is possible if there exists a double-spending attack time $t_{dsa} \in [0, \infty]$ such that, in this duration, the fraudulent chain length is longer than that of honest chain length.*

Proof. It is termed double spending when someone spends the same amount of money twice. Each node in the network verifies every transaction in the digital cryptocurrency, allowing for multiple copies of the transaction to be produced. The transaction is then re-broadcast into the network prior to final transaction confirmation. Double-spending attack is a serious issue for blockchain systems such as Bitcoin, Bitcoin-NG, and others. Double-spending can be understood from the process described below. (Note: UTXO is the unspent output transaction with respect to the cryptocurrency). Let M be a victim, and Λ be an adversary, and the double-spend attack has happened when N_{BC} have validated within a period of time of t_{dsa}.

Let, at time t = 0, a M receives a transaction Tr from Λ. The Tr is an unspent output transaction issued from a UTXO. The fair Miners mine the block transparently, and Λ mine secretly, it includes Tr; however, Λ does not include Tr, but another transaction (fake transaction) Tr' conflicting with Tr. Also, when N_{BC} have mined, M sends goods to Λ. The Λ has mined the blockchain greater than the legal one, it sends his blockchain to the network. Thus, the legitimate transaction Tr is removed from the blockchain.

The double-spending attack is possible if t_{dsa} is in $[0, \infty]$ range. However, the block confirmation time is very low in SmartCoin, and the consensus delay is also very low (in milliseconds). Hence, the t_{dsa} is not possible. To produce a block, just the leader node is permitted. Since this eliminates the chance of rebroadcasting the same transaction, SmartCoin eliminates the potential of rebroadcasting the same transaction in the network. □

Lemma 12. *The SmartCoin protocol prevents fork generation. A fork in the blockchain is possible if $t_{fork} \in [0, \infty]$ such that, in this duration, the fraudulent chain length is longer than that of honest chain length.*

Proof. The difference between the number of nodes mined by the malicious node and the fair node to confirm a transaction should be < 0, i.e., $t_{fork} < 0$. Additionally, to lower the vulnerability probability of the blockchain network, the threshold $\varepsilon \in (0, 1]$ is the effective, robust level. This can be defined as follows:

$$\theta^*(\varepsilon) = min\{\theta : P(Q > \theta) \leq \varepsilon\} \tag{5.9}$$

where P is the probability of a malicious node to mine the block before the fair node. The $\theta^*(\varepsilon)$ can be calculated as

$$\theta^* = -\log(\varepsilon)/\log(\lambda/\Lambda) \tag{5.10}$$

The high value of θ^* has the efficiency to counterattack the intentional forks. The computational power of the fair node needs to improve to strengthen the robustness of the blockchain network. The fair node's effective computational power is the minimum to complete the mining in the due time interval. In the case of SmartCoin, the computation power of the block proposer to derive a new block in the time interval is high from the malicious node. Hence, an adversary cannot create a fork in the SmartCoin. Bitcoin, Bitcoin-NG, ByzCoin, Algorand, and OmniLedger all have problems with fork generation, but in SmartCoin, nodes create blocks at a fixed interval. The next node in the network starts creating the blocks when the interval is over. SmartCoin has removed the issue of needless fork generation because there is no chance of two miner nodes coming up with the same block at the same time. There is no parallel block generation in SmartCoin. Thus, there is no possible way for multiple nodes to submit a block simultaneously, thus no fork creation. □

Lemma 13. *The SmartCoin protocol is robust and fault-tolerant.*

Proof. The SmartCoin is the blockchain-based protocol and employs the consensus algorithm 15 to propose the block for the new set of transactions. The robustness and fault tolerance are the system's properties that enable it to function correctly even when some of the nodes behave byzantine. The high fault-tolerant system's performance continues to be consistent even though some nodes of the system failed to perform their task. The SmartCoin is robust and fault-tolerant in terms of topology as well as node failure or node compromising. As illustrated in Figure 5.16, if any node is not functioning correctly or becomes a dead node, the network still not be affected. Thus, the failure of any node does not affect the SmartCoin's working. Hence, SmartCoin is fault-tolerant. The proposer miner node broadcasts the block in the network for verification. All the nodes in the network verify the blocks and set their status as valid. The leader node adds the blocks only after receiving 3N/4 valid responses, i.e., if there are N nodes, and some of the nodes are behaving byzantine, then the system will not fail. However, if the failed nodes are more than the $3N/4$, then it will degrade the system's performance. □

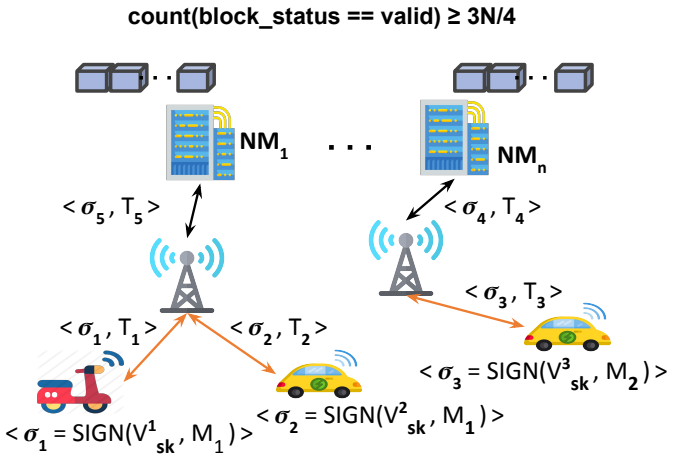


Figure 5.16: Fault tolerance in SmartCoin

A. Identification Theft

Identity theft is the intentional use of legitimate users' identity, usually as a way to achieve an advantage. In the LBSV user first register vehicle V_i to controller as, $A_1 = H(ID_u^i || ID_v^i)$ and $A_2 = A_1 \oplus n_u$, then controllers do $Get(A_1) = A_2 \oplus n_u$. Then it calculate $K1$,$K2$ and Kc as $A_1 \oplus RSU_{pub} \oplus n_u$, $A_1 \oplus Ctr_{pub} \oplus n_u$ and $n_u \oplus A_1$ respectively. At the time of registration, the value of Kc received from the vehicle and stored at the controller is matched by the controller; if it is found unmatched, then the malicious activity is reported to all the RSUs and controllers. Hence, communication with a fake identity or duplicate identity is impossible, i.e., only legitimate and identified vehicles can communicate with other vehicles, RSUs, and SDN controllers.

B. Message Integrity

This attack happens when someone can modify the message's content in transit, i.e., integrity violation. In the LBSV, the interaction between vehicles (TPD), RSUs and SDN controller is secured with the hash (SHA256) and ECC-based cryptographic functions. Vehicle V_i create hash $M_1 = H(Ps||ID_v^i||ID_u^i||ID_v^j)$, and do encryption of message $M_2 = E(V_{pri}^i(M_{S1}, Ps, ID_u^i))$. Then the vehicle V_i sends $< M_1, M_2, T_1, n_i, ID_v^i, ID_v^j >$ to another vehicle. Vehicle V_j checked the received hash with calculated hash as $M_1^* = M_1$ if found unmatched, then reported the malicious activity to the RSUs and controllers about V_i. The message is encrypted with the vehicle's private key, so attackers cannot modify the transit content.

C. Sybil Attack

In the cyber world, the Sybil attack is a new threat, especially in the IoT and IoV, where a fake entity tries to take control of the network by flooding fake nodes. In LBSV, each vehicle ID along with its public key is stored at controllers as $M_{info} = V_{pub}^i || ID_u^i || ID_v^i$, and $T_H = H(M_{info})$. Then, after it creates a *block*. Hence, the device can have only one identity, and each vehicle can have only one key pair at a given time. Before creating the block at the controller, we did one more check as $Kc^*(received) = Kc(stored)$, and if found unmatched, then a malicious vehicle was reported. The entity's key pair (i.e., vehicle, RSU, SDN controller) secures the entire communication. Moreover, the participating vehicles are first authenticated by the SDN controller; only then are they able to communicate with other components of the network. Thus, an attacker cannot use fake identities.

D. Message Replay Attack

When an attacker sends the same message the legitimate user already sent, this attack is called a replay attack. This attack is difficult to detect because the message is not modified; therefore, the receiver accepts it as a genuine message and takes actions desired by the attacker. This attack is impossible for two reasons: first, We have used the blockchain concept, which eliminates the double-spending problem, and second, we use the timestamp and a threshold value for the message. If the attacker sends duplicate messages, the receiver discards them as the condition $\triangle T >= T_i - T_j$ will not hold. Hence, a replay attack is not possible with our scheme. In LBSV, timestamp checking is done between the communication of every component and all the phases of the scheme.

Table 5.11: Comparative security analysis of LBSV and SmartCoin with state-of-the-art schemes based on different attacks

Attacks	[156]	[157]	[110]	[158]	[120]	LBSV	SmartCoin
Identification:	✓	✓	✓	✓	✓	✓	✓
Message integrity:	✓	✓	✓	✓	✓	✓	✓
Sybil attack:	✓	×	✓	✓	×	✓	✓
Message reply attack:	✓	✓	✓	✓	×	✓	✓
Botnet attack:	×	✓	✓	✓	×	✓	✓
Man-in-Middle attack:	✓	×	✓	×	✓	✓	✓
RF Interference/Jamming:	✓	✓	×	×	×	✓	✓
Fake Node injection:	✓	✓	✓	×	✓	✓	✓
Tampering:	✓	×	✓	✓	×	✓	✓

E. Botnet Attack

It is the most dangerous attack in IoV. A botnet works under the command and control of the bot master. It is a network of compromised machines (or bots) that run malicious software. Botnets have a broad spectrum of wicked purposes, including email spamming, DoS/DDoS attacks, password guessing, key-logging, and cryptocurrency mining. The LBSV is protected from the botnet attack because the vehicle becomes part of the IoV only after the proper authentication, i.e., the controller issued a TPD to the vehicle after the initialization phase, which consists of RSU_{pub}, Ctr_{pub}, ID_C and ID_{RSU}. In addition to this vehicle, V_i generates a key pair V^i_{pri}, V^i_{pub} and stores it into the TPD. Therefore, a botnet attack is not possible.

F. Man-in-The-Middle (MiTM) Attack

It is assumed that the ongoing communication between two or more entities is compromised and changed by unauthorized or unregistered entities, then we say that a man-in-the-middle attack has occurred. In the LBSV, V_i sends information to all nearby connected vehicles and RSUs via the open channel (insecure channel), which consists of $< M_1, M_2, T_1, n_i, ID^i_v, ID^j_v >$ during message transmission. If an unauthorized vehicle V_m wants to know S_M, then it should exactly know M_2, but it is encrypted using the private key of vehicle V_i, so V_m cannot see the content of the message.

5.3 BLOCKCHAIN-BASED SECURED SMART GRID SYSTEM

Distributed renewable energy resources, such as solar panels, batteries, electric vehicles, adaptable loads, and other modern technologies, are increasingly being incorporated into the smart grid as an energy distribution system. For instance, the current global market for rooftop photovoltaic (PV) panels is projected to be worth about $33 billion, with an expected growth of more than 3.7 GW by 2025 due to societal adoption [160]. Utilizing these additional energy resources is essential for controlling energy demand and significantly integrating renewable energy into the smart grid. The smart grid merges internet and communication technologies (ICTs) with power networks, enhancing energy generation, transmission, and distribution to be more efficient, cost-effective, and environmentally friendly [161]. The smart grid represents a modern approach to energy distribution, offering greater control and observability compared to traditional energy systems and promoting efficient energy use. By integrating internet technology, the smart grid enhances access to distributed and scalable energy resources on a vast scale [162]. The primary objective of these

innovations is to ensure consistent energy availability. However, the conventional smart grid, being centralized, is susceptible to single points of failure and potential data integrity issues at the central node. As a result, there's a growing shift towards decentralization in smart grid technology [163]. Looking ahead, the focus is on distributed energy sharing, which effectively manages energy generation and use among smart grid components [164]. But transitioning from a centralized to a decentralized smart grid introduces challenges, including transparency issues, high transaction costs, and delays in settlements, affecting throughput, scalability, and data security.

Blockchain technology addresses many of these challenges by enhancing the security of energy transactions through its immutable record-keeping, which ensures that transactions cannot be altered or deleted. It also secures communication between smart meters and the grid, allowing for transparent energy market operations and enabling peer-to-peer energy trading, thus reducing reliance on centralized energy providers and allowing consumers to participate directly in the energy market. However, blockchain-based smart grids face challenges related to transaction processing speed, particularly when dealing with high volumes of data and transactions, due to the latency introduced by existing blockchain consensus algorithms. These algorithms are resource-intensive, consuming significant energy during computation, which can be counterproductive in the context of smart grids focused on energy efficiency. To address these issues, advanced and secure ICTs like blockchain are pivotal in developing a sustainable distributed energy-sharing system. This has led us to introduce SmartGrid-NG, ETradeChain, and zk-SGB blockchain-driven smart grid systems designed for secure energy trading in next-generation smart grids. In this section, we have covered SmartGrid-NG and ETradeChain. The zk-SGB is covered in section 5.4.

5.3.1 Motivation and Contribution

Energy trading within smart grids is increasingly driven by the adoption of blockchain technology, which is motivated by its potential to enhance transparency, security, efficiency, and decentralization. With multiple parties involved in a decentralized energy market, ensuring every transaction is transparent is crucial. Blockchain's immutable ledger provides a clear record of all transactions, ensuring that every participant can verify and trust the recorded transactions without relying on intermediaries. As energy trading platforms become more digital, they also become targets for cyber-attacks. Blockchain's inherent cryptographic security measures and decentralized architecture make it resistant to a wide range of cyber threats. This shift towards a decentralized model promises to overcome the inefficiencies and vulnerabilities inherent in traditional centralized energy systems, paving the way for a more equitable and efficient distribution of energy resources. With this motivation, we have proposed two schemes for energy trading in the smart grid using blockchain technologies, which are SmartGrid-NG and ETradeChain. Specifically, our main contributions are as follows:

1. **Enhanced Security and Efficiency:** We introduce SmartGrid-NG, a system that seamlessly integrates smart grid technology with blockchain, aiming to strengthen the security and reliability of user transactions within the smart grid ecosystem. This integration is designed to eliminate vulnerabilities like single points of failure inherent in centralized systems. We have designed a consensus algorithm called GridChain to achieve efficiency and trustworthiness. This method employs a reputation-based leader selection scheme, ensuring that block generation and verification are both swift and entrusted to reliable nodes.

2. **Advanced Identity Authentication for Robust Security:** We have designed an efficient identity authentication scheme for the smart grid environment. This scheme prioritizes non-interactive key sharing between smart meters and consensus nodes, ensuring swift and secure communication. We have integrated the secure authentication MAC to strengthen data integrity further. For enhanced data confidentiality and a robust digital signature mechanism, we have incorporated elliptic curve cryptography (ECC) and ECDSA, respectively.

3. **New Cryptocurrency: Pcoin** In the proposed ETradeChain, we have introduced a novel energy cryptocurrency called Pcoin, which stands for Power Coin. Unlike existing cryptocurrencies such as Bitcoin, Ether, etc, Pcoins are associated with energy generated by any of the prosumers. Therefore, we have used Pcoins instead of an existing cryptocurrency. Further, the creation of Pcoins is completely dependent on the energy generated by the prosumers, unlike Bitcoin, Ether, and other cryptocurrencies, which are created based on high computing mining and reward. To create, transfer, and consume the Pcoins, there are three types of transactions, which are PcoinCreate, PcoinTransfer, and PcoinConsume.

4. **Fair Energy Trading and Prevention from Overpricing** Along with the new energy cryptocurrency in EtradeChain, we have addressed the issue of unfair mining and overpricing of energy using the modified double auction and Vickrey auction method. This combination helps the auction to increase its social welfare and also avoid overpaying. An optimal approach is proposed to allocate the consumer's bid to the prosumers. Before bid allocation, a novel sorting factor is added on the consumer end to avoid biased behaviour. During Bid allocation, thresholding is applied to sell the Pcoins so that starvation is avoided on the prosumer's end. The pricing calculation depends on Vickrey's auction and the mean bid prices. Also, the selection of temporary leaders is based on the prosumers who have won the auction.

5. **Comprehensive Transaction Security:** SmartGrid-NG and ETradeChain are both designed to be a beacon of transactional security, ensuring integrity, confidentiality, and authentication for every transaction. Beyond these foundational security measures, our proposed GridChain stands out by addressing and resolving prevalent blockchain challenges. These include ensuring fairness in transaction processing, safeguarding against double-spending attacks, and preventing forking in the chain, thereby maintaining the blockchain's integrity and trustworthiness.

6. **Performance and Security Analysis** We have measured the system's performance of SmartGrid-NG and ETradeChain based on parameters like consensus latency, throughput, computation overhead, energy consumption, and storage and communication overhead to show the effectiveness of this P2P energy trading system compared with state-of-the-art schemes. We compared the security analysis based on parameters like fork prevention, prevention of double-spending, and non-repudiation with state-of-the-art schemes.

5.3.2 Basic Terminologies in Smart Grid

There are some basic terminologies in smart grid that are important to understand when working with the proposed schemes. The basic terminologies are as follows.

- **Smart Meter:** The smart meter collects data on power consumption and real-time monitoring information of the consumer. They are installed at the consumer location. Every smart meter has a unique

ID, which is generated during the production of smart meters. When a smart meter is installed, this unique ID is registered in the blockchain via the consensus node. Any data recorded in the smart meter is considered a transaction, and these transactions are sent to the connected consensus node. These smart meters generate multidimensional data and send them to the connected consensus node. The connected consensus node may change if some fault occurs at the consensus node. If the connected consensus node goes down, then the smart meters connect to the nearest next consensus node. The smart meter is also connected to the smartphone to perform smart feedback to the grid operator. The Smart meters are also connected to the grid operator.

- **Consumer and Prosumer:** A consumer is typically defined as an individual or entity that purchases and consumes electricity. This includes residential households, commercial businesses, and industrial facilities. With the integration of blockchain technology and the evolution of smart grids, the role of consumers has expanded beyond traditional consumption. In a smart grid that utilizes blockchain, consumers often have the capability to generate their own electricity using renewable energy sources like solar panels or wind turbines. These consumers are then referred to as prosumers (producer-consumers). As prosumers, they not only consume the energy they need but can also sell any surplus energy back to the grid or directly to other consumers within the network. This is facilitated by blockchain technology, which provides a secure, transparent, and efficient platform for recording transactions and managing the flow of energy between different grid participants.

- **Grid Operator** The grid operator is the entity that is responsible for power distribution and sales of electricity in the connected region. The grid operator is connected to the region using the smart meter. Ensuring real-time monitoring and making a balance between demand and production are the main and important functions of the grid operator. The grid operator collects the consumer power consumption in real-time and maintains the demand for the region. This practice reduces the sudden pressure on the grid during peak hours, such as during the night hours, and enhances the reliability and robustness of the system. This practice also made the system fault-tolerant.

5.3.3 SmartGrid-NG: Blockchain Protocol for Secure Transaction Processing in Next Generation Smart Grid

SmartGrid-NG addresses the challenges highlighted above by leveraging blockchain's capability to store user information and transaction data securely. Traditional consensus algorithms such as PoS [6], PBFT [12], and PoW [1], fall short in the smart grid context. They are vulnerable to double-spending attacks, fork creation, and transaction integrity breaches and suffer from high consensus delays and low throughput, primarily due to their computational demands. Many existing methods also grapple with prolonged transaction processing times and significant computational, storage, and communication overheads. We introduced the GridChain consensus algorithm for SmartGrid-NG to overcome these limitations. The resultant SmartGrid-NG system represents a seamless fusion of smart grid technology and blockchain underpinned by the reputation-based GridChain consensus algorithm. We first explain the system architecture of SmartGrid-NG, then the methodology followed by the proposed novel consensus algorithm GridChain.

Smart Grid System Architecture

This section describes the general architecture of the SmartGrid-NG as shown in Figure 5.17. It has three components, i.e., Energy service provider (grid operator), permissioned blockchain network, and Consumer

Table 5.12: Mathematical symbol table for SmartGrid-NG and ETradeChain

Symbol	Meaning	Symbol	Meaning
TP	Transaction Pool	SD_i	i^{th} smart meter data
M_{ID}	Smart meter unique identity	T_i	Current Timestamp
Tr_i	i^{th} Transaction	δT	Threshold Time
SK_{pub}	Public Key of smart meter	SK_{pri}	Private key of smart meter
M_{pub}	Public Key of consensus node	M_{pri}	Private key of consensus node

(smart meter). All the components are connected, and relevant information is exchanged. The smart meters are installed at the place of the consumer and monitor the power consumption. The Grid operator is the service provider that distributes the power to the Consumer. The blockchain network is the permissioned blockchain network, where the miners are trusted and use the consensus algorithm for mining the blocks. In the proposed scheme SmartGrid-NG, we have proposed a consensus *Algorithm* 16 called GridChain. The blockchain network securely stores the user information and the transaction of the smart meter in the form of a blockchain.

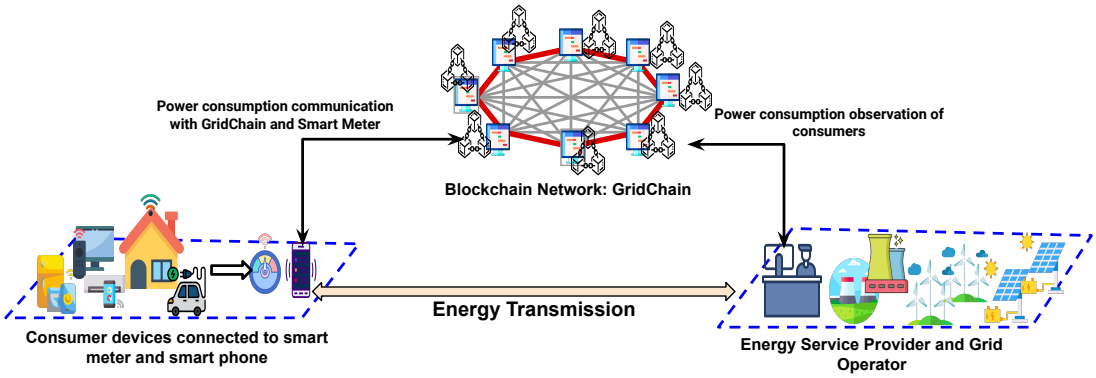


Figure 5.17: SmartGrid-NG system architecture

Methodology of SmartGrid-NG

This section presents the proposed consensus *Algorithm* GridChain and the framework SmartGrid-NG. As per the complete system execution, the process is divided into four phases: system initialization, secure transaction reporting, transaction extraction, and transaction verification and consensus. In the system initialization phase, all the entities of SmartGrid-NG generate secret parameters for secure communication of transactions. Further, all the consensus nodes are initialized with initial reputation values since the proposed Gridchain is a reputation value-based leader selection scheme to reach consensus for block generation and verification. The next two phases (transaction reporting and transaction extraction and verification) are the secure and authentic communication of transactions between the smart meter and the consensus nodes. In the fourth phase, the consensus is responsible for selecting the leader node and creating a blockchain ledger of authentic transactions. The symbol Table 5.12 defined various symbols that are used in the proposed schemes.

Phase 1: System Initialization

In this phase, the smart meter or energy consumer, grid operator, and consensus nodes generate the public and private Keypair-based elliptic curve cryptography method. Also, we have used the secure hash and authentication function to authenticate the smart meter and the consensus nodes.

1. Let $q > 3$ be a prime number. An equation of the form defines an elliptic curve E over F_q (finite field)

$$y^2 = x^3 + ax + b \tag{5.11}$$

where $a, b \in F_q$ and $4a^3 + 27b^2 \neq 0$ (mod q). The set $E(F_q)$ consists of all points $(x, y), x \in F_q, y \in F_q$, which satisfy the defining equation (5.11). In ECC keypair generation, each entity A does the following:

   (a) Select a random or pseudo random integer d in the interval $[1, n-1]$ (here n is an finite integer number).

   (b) Compute $Q = d * G$. (here, G is the generator polynomial point on the ECC curve defined by equation (5.11)).

   (c) A's public key is Q; A's private key is d.

2. Execute the following computation to assign the secret information of all the smart meters $SM = \{SM_1, SM_2, ...SM_N\}$ and consensus node $CN = \{CN_1, CN_2, ...CN_M\}$.

   (a) Randomly select $Pr_{SM_i} \in F_q$ and calculate $Pb_{SM_i} = Pr_{SM_i} * G$. Pr_{SM_i} and Pb_{SM_i} are the private and public keys of SM_i with identity ID_{SM_i}, respectively.

   (b) Randomly select $Pr_{CN_j} \in F_q$ and calculate $Pb_{CN_j} = Pr_{CN_j} * G$. Pr_{CN_j} and Pb_{CN_j} are the private and public keys of CN_j with identity ID_{CN_j}, respectively.

   (c) Select one-way secure hash function $H : 0, 1^*$, which is SHA256.

   (d) Initialize the secure authentication function, i.e., message authentication code (MAC) as $MAC_{sk} = HMAC(sk, Tr)$, where sk is the shared secret key, and Tr is the transaction data.

   (e) Publish the public system parameters $< E, G, q, H() >$, $< ID_{SM_i}, Pb_{SM_i} >$, $< ID_{CN_j}, Pb_{CN_j} >$, where $i = 1, 2, \ldots, N$, and $j = 1, 2, \ldots, M$ amongst the consensus nodes.

3. Based on the performance and status of each consensus node CN_j is initialized with a reputation value $Rep_{CN_j} \in [01, 1]$, $\forall j, j = 1, 2, ..., M$, and the reputation threshold is

$$Rep = \frac{\sum_{j=1}^{M} Rep_{CN_j}}{M} \tag{5.12}$$

Phase 2: Transaction Reporting

In this phase, the SM_i generates the transactions and reports them to the consensus node CN_j, $\forall j, j = 1, 2, ..M$. The SM_i performs the following computation.

1. Transactions generated at time $t_\tau < tr_1, tr_2, ...tr_k >, k = 1, 2, ..P$.

2. Encode the transactions $Tr_k = H(tr_k||ID_{SM_i}||ID_{CN_j}||t_\tau)$

3. Calculate the shared session key with CN_j
   $Sk_i^{CN_j} = Pr_{SM_i} * Pb_{CN_j}$.

4. To encrypt and send the transactions to CN_j,

   (a) SM_i select a random positive integer k.

   (b) SM_i then generate the ciphertext CT_{Tr_k}
   $CT_{Tr_k} = \{k * G, Tr_k + k * Sk_i^{CN_j}\}$. SM_i used the shared secret key for the encryption.

5. Calculate $MAC(Sk_i^{CN_j}) = HMAC(Sk_i^{CN_j}, < Tr_k, t_\tau >)$.

6. send $< CT_{Tr_k}, MAC(Sk_i^{CN_j}), t_\tau >$ to CN_j.

Phase 3: Transaction Extraction and Verification

In this phase, the CN_j first generates the shared session key with SM_i, then extracts the transaction from received ciphertext CT_{Tr_k}. It also verifies the authenticity of received information by comparing the received MAC and calculated MAC.

1. Calculate the shared session key with SM_i
   $Sk_{CN_j}^i = Pr_{CN_j} * Pb_{SM_i}$.

2. To decrypt and receive information from SM_i.

   (a) SM_i select a random positive integer k.

   (b) SM_i then generate the ciphertext CT_{Tr_k}

3. Calculate $MAC(Sk_{CN_j}^i) = HMAC(Sk_{CN_j}^i, < Tr_k, t_\tau >)$. if it equals to $MAC(Sk_i^{CN_j})$, then the transaction is accepted otherwise rejected. It is possible because both the shared secret keys are equal, i.e., $Sk_i^{CN_j}$ is equal to $Sk_{CN_j}^i$.

4. calculate $\sigma_{CN_j}^{(1)} = H(Tr_k)^{Pr_{CN_j}}$, where $\sigma_{CN_j}^{(1)}$ is the signature of Tr_k.

5. send $< Tr_k, \sigma_{CN_j}^{(1)} >$ to CN_x. CN_x is the leader node of the next round of consensus process. In the next section, we have defined how the leader node is selected.

GridChain: Reputation based Consensus

This phase is responsible for the blockchain-specific task, that is, block generation and verification, and adding this block into the blockchain ledger. To accomplish this task, we proposed a GridChain blockchain protocol, which is a leader-based consensus algorithm, and the leader node is selected based on the corresponding reputation value. Let, we have a set of consensus nodes $CN_j = \{CN_1, CN_2, ..., CN_M\}$, where $j = 1, 2, ..., M$. Among these nodes, the CN_x, $x \in j$ is the leader node, which receives the information from

the smart meter SM_i. The whole consensus process is divided into three phases, i.e., 1. leader node selection, 2. block generation and verification, and 3. node reputation updation.

1. Leader Node Selection In each round of the consensus process, a leader node is selected. To make the leader selection process more secure and fair, first, the consensus nodes (CN) whose reputation value is above or equal to the reputation threshold are selected to form the pre-selected group. Second, to make the process fair or prevent the leader node from being known in advance by the attacker, the nodes that satisfy the random number threshold are selected from the pre-selected group. Finally, from these pre-selected consensus nodes, a node with a high reputation value is selected as the leader node. Thus, the leader selection process is a three-step process, i.e., create a pre-selection group, random reputation threshold, and find the leader node.

1. **Create Pre-selection Group**: There are two possible cases.
   (a) **Case 1: If it is the first round of consensus** In this case, the consensus nodes CN_j with $Rep_{CN_j} \geq Rep$ are selected to form the pre-selection group. Here, the Rep_{CN_j} is the node's initial reputation value, and Rep is the reputation threshold value.
   (b) **Case 2: Otherwise** In this case the consensus nodes CN_j with $Rep^*_{CN_j} \geq Rep$ are selected to form the pre-selection group. Here, $Rep^*_{CN_j}$ is the updated reputation value of the node CN_j for $j = 1, 2, ..., M$.
   (c) Let, there are $S \in M$ number of consensus nodes in the pre-selection group, i.e., $CN_s = \{CN_1, CN_2, ..., CN_s\}$, $s = 1, 2, ..., S$.
2. **Random Number Threshold** Consensus nodes CN_s in the $S \in M$ pre-selection group perform the following computation.
   (a) Select the random number X_s.
   (b) Sign the X_s using ECDSA as $\sigma_{CN_s} = H(X_s)^{Pr_{CN_s}}$, where, Pr_{CN_s} is the private key of CN_s.
   (c) Broadcast this into the other nodes in the pre-selection group.
3. The current leader node CN_x performs the following steps to generate the random number threshold.
   (a) Select the random number X_{CN_x} as the threshold ω.
   (b) Sign the X_{CN_x} using ECDSA as $\sigma_{CN_x} = H(X_{CN_x})^{Pr_{CN_x}}$, where Pr_{CN_x} is the private key of the leader node CN_x.
   (c) Send $< \omega, \sigma_{CN_x} >$ to each node CN_s.
   (d) Consensus nodes CN_s with $X_s \geq \omega$ form the group $Q \in S$.
4. Among the consensus nodes group Q, the node with the highest reputation value $Rep^*_{CN_j}$ is selected as the leader node CN_x.
5. The information of CN_x is broadcasted to each consensus node group CN_j, $j = 1, 2, ..., M$.

2. Block Generation and Verification The block verification scheme is also simple yet very effective. The newly elected leader node sends the newly created block along with the nonce value. Every verification node again creates the block with the received nonce value and checks the received block's validity. Therefore, the verification process also does not take much time to block verification. These are the two novel

things that are introduced in the GridChain consensus algorithm to create, verify, and add the block to the distributed ledger. We have considered that every miner node has a transaction pool, where the smart meters of the consumers put all the transactions. The miner node sends the transaction to other miner nodes after the transactions are verified. The consensus *Algorithm* 16 works as follows:

Algorithm 16: Block Generation and Verification

Input: $Tr_i : Tr_1, Tr_2, \ldots, Tr_n$
Output: *Block*
$TP.append(Tr_i)$;
Function `blockGenerate(`Tr_{New}`):`
  if *node == leader* **then**
    $Mrt = createHashTree(TP.getAllTransactions)$;
    $PBH = H(PB)$;
    $BH = H(T, PBH, Mrt)$;
    $block = createBlock(BH, PBH, T, n, Mrt)$;
    $broadcast(sign(block, nonce))$;
Function `blockVerification(`*block, nonce*`):`
  if *node != leader* **then**
    $getBlock(block, nonce)$;
    $Mrt = createHashTree(TP.getAllTransactions)$;
    $block* = createBlock(Mrt, T, PBH, BH, n)$;
    if *block == block** **then**
      setBlockStatus(valid);
    else
      setBlockStatus(invalid);
      node.status = bad;
  if $(count(block_status == valid) \geq N/2+1)$ **then**
    blockchain.append(block);

1. The smart meter (SM_i) is considered an ordinary node that is responsible for generating the transactions. All the generated transactions are put into the transaction pool TP. The selected leader node collects the transactions from the TP for verification. After verification, it transfers those transactions to other miner nodes. Then, the leader node calls the function *createHashTree* to calculate the Mrt of these transactions. The hash PBH of the previous block is calculated with the function $H()$, and the current timestamps T and *nonce* are set to the appropriate value.

2. Using the values of $Mrt, PBH, T, nonce$, the leader node creates the block using $createBlock()$ function. The leader node sends the recently mined block to all the nodes for verification. The digital signature is used to prevent the block from modification and impersonation attacks.

3. When the rest of the nodes (other than the leader node) get the copy of the block, they first verify the signature of the block to check whether it was sent from the legitimate leader node or not. After validating, they also verified the correctness of the block. To validate the block, the non-leader nodes recreate the block from the same set of transactions, and for that, it calculates the $Mrt, PBH, T, nonce$.

The block is created using the same set of transactions and other required data, and then non-leader nodes match this with the received block. If a match is found, the node updates *block_status* to *Valid*; otherwise, it updates to *Invalid*. In addition to *block_status*, the node will mark the leader node's status (*node.status*) as bad for future leader selection.

4. Assume the leader node gets at least $\frac{N}{2}+1$ responses as *Valid*. In that case, it appends the block in the distributed ledger, i.e., a block is considered valid when half of the nodes verify the block. At least 51% nodes must verify the block to prevent any attack.

3. Node Reputation Updation Reputation is the continuous evaluation of the long-term behaviour of consensus nodes. Based on the subjective logic theory in [158], we derive our reputation model. Lets, assume the consensus node CN_i interact with CN_j of each round of consensus, where $i = 1,2,...,M$, $j = 1,2,...,M$, and $i \neq j$. Based on the interaction between both the nodes, CN_i calculates reputation $Rep^k_{i,j}$ of CN_j in the k^{th} round of the consensus process through the subjective logical reputation model. In the last step, the current leader node CN_x obtain the final reputation value $Rep^*_{CN_j,CN_i}$ of CN_j. To accomplish this, it integrate the $Rep^k_{i,j}$ and $Rep^{*(k-1)}_{CN_j}$, where $Rep^{*(k-1)}_{CN_j}$ is the reputation value of CN_j in the $(k-1)^{th}$ round.

The reputation value of $Rep^*_{CN_j,CN_i}$ is computed between the nodes CN_i and CN_j based on their interaction during the consensus process. In the consensus process, the block verifier node also responds to the node status as good or bad. This can be divided into two steps, i.e., opinion calculation and integration with past reputation values.

1. **Opinion Calculation:** Based on the interaction behavior between CN_i and CN_j, we calculate trust ($T_{i,j}$) and uncertainty ($U_{i,j}$), respectively, where $i = 1,2,...,M$, $j = 1,2,...,M$, and $i \neq j$. The opinion calculation of CN_i to CN_j is calculated based on the $T_{i,j}$ and $U_{i,j}$. The computation is as follows:

   (a) **Interaction Behavior:** As shown in the consensus *Algorithm* 16, the verifier node also responds to the status of the leader node. Therefore, interaction behaviour between CN_i and CN_j is of two types, i.e., positive interaction behaviour depicted as $\mathbb{P}$ and negative interaction behaviour depicted as $\mathbb{N}$. This can be quantified as depicted as $\mathbb{P}+\mathbb{N}=1$, and $\mathbb{P}>\mathbb{N}$.

      i. **Positive Interaction Behavior:** If the consensus node CN_i sends some information to CN_j, then this interaction is considered a positive interaction if 1) the signature of the received information is successfully verified and 2) it is a faster response time. Let's say the response timeline is T_τ, and the active timeline is T_0, and if $T_\tau < T_0$, then it is considered a faster response time. Assume δ and ξ are the fast and slow response times, respectively, then $\delta+\xi=1$, and $\delta>\xi$.

      ii. **Negative Interaction Behavior:** The negative interaction between any two consensus nodes CN_i and CN_j can be quantified based on the two cases: 1) wrong signature and 2) sending delayed signature. The quantification can be expressed as $\lambda+\eta=1$,$\lambda>\eta$, where λ is depicting the wrong signature and η is depicting as sending a delayed signature.

   (b) **Effectiveness of Recent Interaction:** It is important to note that the recent interaction between CN_i and CN_j has a greater effect on the reputation evaluation than the past interaction. The quantification of this interaction can be expressed as $\rho+\phi=1$, $\rho>\phi$, where ρ is the recent interaction and ϕ is the past interaction.

Based on the above evaluation parameters, the trust and uncertainty can be computed as follows:

$$T_{i,j} = \frac{S_{i,j}.\alpha_{i,j}}{\alpha_{i,j} + \beta_{i,j}} \tag{5.13}$$

$$\alpha_{i,j} = \mathbb{P}[\rho(\delta\alpha_{1,1} + \xi\alpha_{1,2}) + \phi(\delta\alpha_{2,1} + \xi\alpha_{2,2})] \tag{5.14}$$

$$\beta_{i,j} = \mathbb{N}[\rho(\lambda\beta_{1,1} + \eta\beta_{1,2}) + \phi(\lambda\beta_{2,1} + \eta\beta_{2,2})] \tag{5.15}$$

$$U_{i,j} = 1 - S_{i,j} \tag{5.16}$$

Here, $S_{i,j}$ is the probability of successful interaction between CN_i and CN_j, where $i \neq j$. $\alpha_{1,1}$ and $\alpha_{1,2}$ are the recent interactions with fast/slow response time in positive interaction. $\alpha_{2,1}$ and $\alpha_{2,2}$ are the past interactions with fast/slow response time in positive interaction. $\beta_{1,1}$ and $\beta_{1,2}$ are the recent/past interactions with the wrong signature in negative interaction. $\beta_{2,1}$ and $\beta_{2,2}$ are the recent/past interactions with delayed signatures in negative interaction. Based on the above computation, the opinion calculation can be expressed as

$$Rep^{*(k)}_{CN_i,CN_j} = T_{i,j} + Rep^{*(k-1)}_{CN_j}.U_{i,j} \tag{5.17}$$

Computing Reputation Value: When the above computation is completed, the current leader node CN_x computes the final reputation value of CN_j as follows.

1. The consensus node CN_i send the reputation value $Rep^{*(k)}_{CN_i,CN_j}$ which is the local reputation value of CN_i to CN_j based on the opinion behavior in the k^{th} round of the consensus process.
2. The current leader node prepares the 2X2 reputation matrix as follows:

$$\begin{bmatrix} Rep^{*(k)}_{CN_1,CN_j} & , & Rep^{*(k-1)}_{CN_1} \\ Rep^{*(k)}_{CN_2,CN_j} & , & Rep^{*(k-1)}_{CN_2} \\ & \vdots & \\ Rep^{*(k)}_{CN_M,CN_j} & , & Rep^{*(k-1)}_{CN_M} \end{bmatrix}$$

Here, $Rep^{*(k)}_{CN_1,CN_j}$ depicts the reputation value of CN_j on CN_i in the k_{th} round of consensus, and $Rep^{*(k-1)}_{CN_j}$ is the reputation value of CN_j in the $(k-1)^{th}$ consensus round.

The Current leader node CN_x updates the final reputation value of CN_j in the k_{th} round of consensus based on the above computation as follows.

$$Rep^{(final,k)}_{CN_j} = \frac{Rep^{*(k)}_{CN_1,CN_j} Rep^{*(k-1)}_{CN_1} + ... + Rep^{*(k)}_{CN_M,CN_j} Rep^{*(k-1)}_{CN_M}}{\sum_{j=1}^{M} Rep^{*(k-1)}_{CN_j}} \tag{5.18}$$

Reward Mechanism in SmartGrid-NG

leader node and the verifier consensus nodes get the reward for mining the block in GridChain. The remuneration or reward is provided based on the blocks mined by the leader node, i.e., each block mining

contains a fee, which is referred to as a transaction fee. Instead of using Bitcoin as a reward, GridChain uses a transaction fee as a recompense, which means that every transaction has a transaction cost. If every block contains N transactions, then the general equation for total transaction fee (TTF) equals to

$$TTF = \sum_{i=1}^{N} [TF_i] \tag{5.19}$$

where TF is a transaction fee, and N is the number of transactions in the j^{th} block. This transaction fee is divided into two parts, where 60% is sent to the leader node (CN_x) and the 40% is equally distributed to the honest transaction verifier consensus nodes (CN_j^{honest}, $j = 1, 2, ..., S$). Here, S is the number of consensus nodes in the pre-selection group. This can be expressed as

$$< CN_j^{honest}, TTF_{CN_j} > \leftarrow \frac{40\%(TTF)}{S} \tag{5.20}$$

$$< CN_x^{honest}, TTF_{CN_x} > \leftarrow 60\%(TTF) \tag{5.21}$$

For example, let the leader node mine block where the block contains 516 transactions, and the transaction fee is 0.1$ per transaction, then the TTF would be as $TTF = \sum_{i=1}^{516}[0.1]$, $TTF = 51.6$\$. This is how the remuneration of the miner node is done, and it gets benefits for the transaction processing and mining of the block. The equal distribution of 40% of block mining reward in the transaction verifier consensus nodes is very important To encourage and get equal and honest participation from them. Additionally, it is also important to avoid malicious block generation possibilities from the leader nodes. Therefore, due to these challenges, we have divided the block mining reward and equally distributed it amongst the honest verifier consensus nodes.

5.3.4 ETradeChain: Blockchain-Based Energy Trading in Local Energy Market (LEM) using Modified Double Auction Protocol

The proposed ETradeChain is a next-gen smart grid technology that resolves issues in current methods. It's a concurrent blockchain-based energy trading system ideal for decentralized LEM. This concurrent approach offers better efficiency and higher throughput than traditional blockchain for smart grids. In this permissionless network, energy is traded for smart homes using "Pcoins" (Power Coins), which represent energy generation. Prosumers can exchange generated energy for Pcoins, and others can buy these Pcoins to get energy. An auction mechanism is essential for fair energy trading and allocation. An auction mechanism is a great way to manage and allocate energy. Auctions have been widely researched for trading in areas like cloud virtual machines [134] [132], edge computing resources [134], [165], Smart Grids [166] [133] [167] [168]. Most auction systems rely on a central authority to oversee the process, and some are computationally intensive, making them unsuitable for devices like smart meters. Many are also unfair and suboptimal. In contrast, ETradeChain offers a lightweight, fair, and decentralized auction mechanism. This auction also selects temporary leaders for consensus in block creation and verification.

Goal of ETradeChain

ETradeChain is a decentralized P2P network with N mining nodes. Each node uses a key pair to sign and verify blocks and transactions, eliminating the need for a central PKI. Transactions are cryptographically hashed using function $H(x)$ before block inclusion. At the time t, some nodes, $M(t) \in N$, are Byzantine

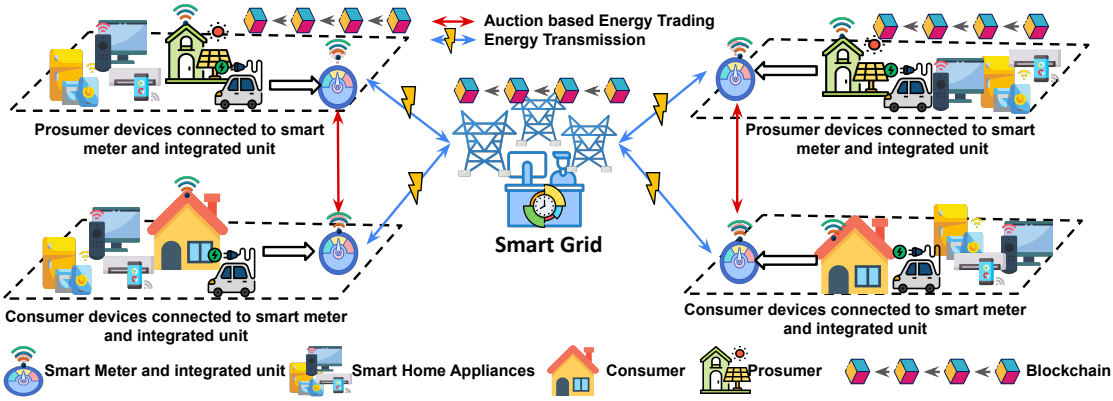


Figure 5.18: ETradeChain system model

and controlled by an adversary, while others are honest. Given a node's mining power as $Mining(K)$, the combined power of Byzantine nodes must be under 51% of the total N nodes' power. This is because ETradeChain included all such vulnerable nodes, which do selfish mining, under the control of the adversary.

$$\sum_{k \in M(t)} Mining(k) < 51\% \sum_{i \in N} Mining(i) : \forall K, i \tag{5.22}$$

System Model of ETradeChain

ETradeChain's architecture comprises three components: Smart Grid (Auctioneer), prosumer (Seller + Buyer), and consumer (Buyer), depicted in Figure 5.18. A LEM Testbed is set up using Raspberry Pi boards, which act as smart meters with integrated energy trading units. These meters, placed at consumer locations, track power consumption. The Smart Grid distributes power to consumers. Prosumers create a permissionless blockchain network, which securely logs user details and smart meter transactions in a ledger.

There's a consistent market-clearing price for each time slot $t \in T$. Customers with full bid orders pay this price for the specified power amount. Blockchain facilitates both market operations and payments. Smart meters automatically measure and predict each customer's demand d_{it} and generation g_{it}. Based on this, any excess demand or supply is computed and credited to the customer's blockchain account, ensuring data privacy while aligning with consumption patterns. Accounts hold Pcoin balances and energy data. By combining a customer's energy surplus and credit balance, orders are tailored to their needs. This virtual trading doesn't affect actual electricity transfers. We use a supply and demand matching online mechanism and price signals to indicate the local scarcity of energy. As a result, the large-scale power grid can be accessed. A forecast of consumption and production at the customer level is used to match the LEM's demand and supply restrictions. We only include electricity trading and ignore bundled orders (like electricity and heat) to keep things simple. The LEM's result is calculated as the total cost of electricity for all customers. In other words, this is the total amount of electricity x_{ijt} sold from one customer i to another customer j at a price c_{ijt} at time t. The minimum cost of the energy sold from prosumer i to prosumer/consumer j is represented in Equation (5.23). Equation (5.24) ensures that every prosumer satisfies their demand d_{it} and sells the surplus generation g_{it}. Equation (5.25) and (5.26) ensure that the

smart grid balances supply and demand within the LEM. The hard constraints for this are the equations from (5.23) to (5.26). The decision variable x_{ijt} has to be positive because only a positive amount of electricity can be traded.

$$min \sum_{i \in I, j \in J} c_{ijt} * x_{ijt} \tag{5.23}$$

$$g_{it} - d_{it} - \sum_{j \in I} (x_{ijt} - x_{jit}) = 0, \forall i \in I \tag{5.24}$$

$$\sum (g_{it} - d_{it} \leq \sum_{i \in I} X_{igt}) \tag{5.25}$$

$$-(\sum_{i \in I} (g_{it} - d_{it})) \leq \sum_{i \in I} x_{git} \tag{5.26}$$

Problem Definition of Energy Trading in ETradeChain

Let us define and formulate our problem. We consider that we have N prosumers and M consumers. Let P define the set of prosumers, and C define the set of consumers such that $P = \{1,2,3...,N\}$ and $C = \{1,2,3...,M\}$. Any prosumer, while bidding, will have two factors ($bidP_j$, s_j). $bidP_j$ denotes the selling price of one Pcoin as a bid raised by j^{th} prosumer and s_j denotes the total number of Pcoins j^{th} prosumer wants to sell. Similarly, consumers also have two factors ($bidC_i$, d_i). $bidC_i$ denotes the bidding price of one Pcoin raised by i^{th} consumer and d_i denotes the total Pcoins demand of i^{th} consumer. The utility of i^{th} consumer will be $u(C_i) = d_i \cdot bidC_i - d_{si} \cdot avg(C_i)$ where the $avg(C_i)$ denotes the average amount i^{th} consumer has to pay for one Pcoin to the seller and d_{si} denotes the total Pcoins received by i^{th} consumer. Similarly, the utility of j^{th} prosumer is equal to $u(P_j) = s_{sj} \cdot act(P_j) - s_j \cdot bidP_j$ where $act(P_j)$ denotes the actual amount received by the prosumer for the Pcoins and s_{sj} denotes the total Pcoins received by the j^{th} prosumer. Therefore, the total Utility of the auction can be defined as follows

$$U = \sum_{j=1}^{N} u(P_j) + \sum_{i=1}^{M} u(C_i) \tag{5.27}$$

We are considering the auction process to be balanced, i.e., the total amount received by the prosumer should be equal to the total amount given by the seller at the end of the auction process. Therefore $\sum_{j=1}^{N} s_{sj} \cdot act(P_j) = \sum_{i=1}^{M} d_{si} \cdot avg(C_i)$

$$U = \sum_{j=1}^{N} s_j \cdot bidP_j - \sum_{i=1}^{M} d_i \cdot bidC_i \tag{5.28}$$

Therefore, we can define our problem as follows.

Maximise $U = \sum_{j=1}^{N} s_j \cdot bidP_j - \sum_{i=1}^{M} d_i \cdot bidC_i$

Subject to:

$$0 \leq s_{sj} \leq s_j, \forall j \in N \tag{5.29}$$

$$0 \leq d_{si} \leq d_i, \forall i \in M \tag{5.30}$$

$$\sum_i d_{si} = \sum_j s_{sj}, \forall i \in M, \forall j \in N \tag{5.31}$$

$$d_i, s_j > 0, \forall i \in M, \forall j \in N \tag{5.32}$$

According to equation (5.29), the total number of Pcoins sold by the prosumer should be less than or equal to the total number of Pcoins available with the prosumer to sell. Equation (5.30) tells that the total number of Pcoins received by the consumer should be less than or equal to the Pcoins requested. Equation (5.31) defines the total Pcoins sold by the prosumer should be equal to the total Pcoins received by the consumer. Further, Equation (5.32) explains the total Pcoins bid by both prosumer and consumer should be greater than 0.

Methodology of ETradeChain

This section explains the protocol in detail. ETradeChain is a blockchain-based Smart Grid system for processing transactions. Here, we have considered three types of transactions: PcoinCreate, PcoinTransfer, and PcoinConsume. The detail of these transactions is given in the subsequent section. All the energy trading happens with the use of a modified double auction scheme. The winners of the auction are selected as temporary leaders. These leaders create the blockchain network for the consensus on block creation. In this network, one super leader is elected. The super leader selection is based on the stake deposited by the leader nodes. The Pcoins are used as a stake. The leader node that deposited the maximum stake will have the highest probability of being selected as the super leader node. The super leader node is responsible for block creation and transaction verification. However, transactions are verified by the other leader nodes.

A. Type of Transactions

The energy is traded between the nodes via a smart grid energy distributor in the LEM. Three types of transactions are generated for trading energy, which are described below.

1. **PcoinCreate**: Pcoins are created by prosumers based on the generated electricity. The excess electricity is given to the power grid, which generates the equivalent coins. With respect to this coin generation, the transaction is generated, which is called the PcoinCreate transaction.

2. **PcoinTransfer**: The consumer who wants to get electricity from the smart grid first needs to buy some Pcoins. This happens via the auction process. After the Bid is allocated and the pricing of Pcoins is calculated between the prosumer and consumer, the Pcoins and money are transferred between them. This type of transaction is called "PcoinTransfer."

3. **PcoinConsume**: The consumer, or prosumer, can get electricity from the smart grid in exchange for Pcoins, and this transaction is called a "PcoinConsume" transaction. The Pcoins that are consumed will never be used again to get electricity.

B. Auction Mechanism

Below is the process of a modified double auction mechanism where both consumers and prosumers can bid to buy Pcoins and ask to sell Pcoins. Further, we are going to use the Vickrey auction or second-price auction to allow for truthful action. Whenever any prosumer wants to participate in the trading process, they must first register at the smart grid (certificate authority) for generating key pairs. After getting the key pair, the public key acts as an identity for the node. The auction is done so that prosumers can sell their Pcoins and consumers can buy those Pcoins. There are three important steps in the complete auction process.

- *Bid*: It is the first step in the auction process. It has a dedicated time phase during which prosumer and consumer share their bid with the smart grid. After the completion of this phase, the smart grid broadcasts all the collected bids to the peers in the network.
- *Allocate*: After receiving the smart grid's bid details, all the blockchain nodes calculate the bid allocation and pricing scheme and share the final output with the peers. Further, all the nodes verify the schemes and reach on consensus using the voting mechanism.
- *Energy Exchange*: Once the above steps are completed, the consumers can get their energy in exchange for Pcoins from the smart grid by invoking the PcoinConsume transaction.

C. Bidding Process

The bidding process initiates the auction mechanism. This bidding process is executed in a dedicated time phase or a slot. During this phase, all the consumers and prosumers who want to take part in the auction will share their bids with the smart grid. The consumer also shares the demand for the Pcoins that it requires, and the prosumers will share the number of Pcoins that it wants to sell. Once this phase of the bidding process is over, the Smart grid will then broadcast the collected bids, consumer demand, and prosumer supply details with the peers in the network. The peers then use these details for the next steps.

D. Sorting Mechanism

In our case, the sorting mechanism is done off-chain, all the nodes conduct the auction mechanism, and the final allocation and pricing scheme is broadcasted to other nodes, unlike the central authority or auctioneer performing the Auction mechanism as per in [166] [167]. The allocation and pricing scheme, which has been taken by 50% of the nodes in the network, is chosen. Further, [167] tries to sort the buyers (consumers) in descending order on the basis of their bid price. But on the consumer end, there is a possibility that a single buyer bids the highest for a large amount of Pcoins, so to avoid that, we created a sorting factor for the consumer as follows

$$sf(C_i) = \frac{bidC_i}{\sqrt{d_i}} \tag{5.33}$$

From the above equation, it can be seen that the sorting factor doesn't only depend upon the bid price but is inversely proportional to the square root of the Pcoins demanded. This keeps a single buyer from buying all the Pcoins by paying a high price. After this, we sort this Sorting Factor of consumers in descending order. In the case of the prosumers, we are trying to sort them in ascending order of their bid prices.

E. Bid Allocation Mechanism

We have implemented an optimal approach to allocating the consumer's bid to the prosumers. To avoid starvation, we are creating a threshold value on the number of Pcoins a prosumer could sell to a particular consumer. If consumer demand is not satisfied by the first prosumer, his demand will be satisfied by the next prosumer in the sorted order, and so on. The first prosumer, after satisfying one consumer's demands, will satisfy the next consumer in reverse sorted order of the sorting factor. Below is the formulation of the threshold value.

$$Th_s = \frac{\sum_{j=1}^{N} s_j}{M \cdot N} \tag{5.34}$$

In the *Algorithm* 17, we are considering that the consumers and prosumers are already sorted as per the sorting mechanism discussed above, and C_1 and C_M are the consumers having the highest and lowest sorting factors, respectively, and also P_1 and P_N are the prosumers having the lowest and highest bid price, respectively. The algorithm for bid allocation works as follows:

1. After prosumers and consumers are sorted as per the sorting mechanism, a threshold is calculated using (5.34) and mapping is done between the consumers and the prosumers. The allocation matrix (AM) stores the number of Pcoins that need to be transferred from j^{th} prosumer to the i^{th} consumer.
2. The following steps are executed only if the bid amount of consumers is greater than the bid amount of prosumers. Step-1: If the d_i of the consumer is satisfied, it goes for the i+1 consumer. Step 2: If the j^{th} prosumer has no more Pcoins, then go for the j+1 prosumer
3. If the d_i is pending and greater than or equal to the threshold value and also if the j^{th} prosumer has greater than or equal to the th_s value of Pcoin then Increment the allocation matrix of i^{th} consumer and j^{th} prosumer by threshold and decrement the s_j and d_i by threshold.
4. If the d_i is pending and less than the threshold value and also if the j^{th} prosumer has greater than or equal to the requested value of Pcoin by i^{th} consumer, then Increment the allocation matrix of i^{th} consumer and j^{th} prosumer by d_i and decrement the s_j by d_i and make d_i as 0.
5. If the j^{th} prosumer has less Pcoins than the d_i or th_s then Increment the allocation matrix of i^{th} consumer and j^{th} prosumer by s_j and decrement the d_i by the s_j and make s_j as 0.

F. Pricing Mechanism

After the bid allocation is done, the next step is to create the pricing mechanism. We make use of the mean of consumer and prosumer bid prices and the Vickrey Auction to calculate the pricing mechanism. A Vickrey or second-price auction prevents the consumer from overpaying, improves the competition, and optimizes future bids.

Suppose the i^{th} consumer receives x Pcoin from the j^{th} prosumer. Below is the amount the i^{th} consumer needs to pay to the j^{th} consumer.

$$pay_{ij} = x \cdot (\frac{bidP_j + bidC_{i+1}}{2}) \tag{5.35}$$

For the last consumer, the bid price paid will be the amount they bid on only. As per the algorithm 18, a pay matrix (PM) is created, which contains the payment mapping of i^{th} consumer to the j^{th} prosumer. The

Algorithm 17: Bid Allocation Mechanism

Input: M Consumers, N Prosumers, d_i is the number of Pcoin i^{th} consumer (C_i) requires, s_j is the number of Pcoin j^{th} prosumer (P_j) wants to sell

Output: Allocation Matrix (AM)

Function `generateAM`$(M, N, bidC_i, bidP_j, s_j, d_i)$**:**

$Th_s = \frac{\sum_{j=1}^{N} s_j}{M \cdot N}$;
for $i \leftarrow 1$ **to** M **do**
  for $j \leftarrow 1$ **to** N **do**
    if $bidC_i \geq bidP_j$ **then**
      if $d_i == 0$ **then**
        break;
      else if $s_j == 0$ **then**
        continue;
      else if $d_i \geq Th_s$ ***and*** $s_j \geq Th_s$ **then**
        $AM_{ij} += Th_s$;
        $d_i -= Th_s$;
        $s_j -= Th_s$;
      else if $d_i < Th_s$ ***and*** $s_j \geq d_i$ **then**
        $AM_{ij} += d_i$;
        $s_j -= d_i$; $d_i = 0$;
      else
        $AM_{ij} += s_j$;
        $d_i -= s_j$;
        $s_j = 0$;

pay matrix is updated with the help of (5.35) except for the last consumer, which pays the price bided by it. From the algorithm, it can be seen that the total payment received by the j^{th} prosumer is equal to the sum of the j^{th} column in the pay matrix. Also, the total payment given by the i^{th} consumer is equal to the sum of the i^{th} row in the pay matrix.

G. Temporary Leader Selection From Auction

Miner nodes are temporary leaders chosen from among all prosumers. One of these miner nodes is chosen as the super leader, who creates the block, and the other mining node is chosen as the verifier. The winners of the auction system are used to determine the temporary leaders. It is based on the concept of the original double auction method. The goal of selecting the auction scheme leaders is to involve more prosumers and consumers in trade. The procedure for selecting a temporary leader through an auction is described below.

1. Sort the Consumers in decreasing order of their bid: $bidC_1 \geq bidC_2 \geq ... \geq bidC_m$.
2. Sort the Prosumers in increasing order of their bid: $bidP_1 \leq bidP_2 \leq ... \leq bidP_n$.
3. Let k be the largest "breakeven index" index such that $bidC_k \geq bidP_k$

Algorithm 18: Pricing Mechanism

Input: M Consumers, N Prosumers, AM Allocation Matrix, $bidC_i$ is the bid price of one Pcoin of i^{th} consumer (C_i), $bidP_j$ is the ask price of one Pcoin of j^{th} prosumer (P_j)

Output: Pay Matrix (PM)

Function `payMatrix`$(M,N,AM,bidC_i,bidP_j)$**:**

```
for i ← 1 to M do
    for j ← 1 to N do
        if i == M then
            PM_ij = AM_ij · (bidP_j + bidC_i)/2;
        else
            PM_ij = AM_ij · (bidP_j + bidC_{i+1})/2;
```

4. These first k prosumers are the winners of the double auction, and the group of these prosumers will act as our temporary leader for that time phase.

H. Super Leader Selection and Consensus Process

The "super leader" is the leader of all the temporary leaders. We have employed the PoS consensus algorithm for block creation among the temporary leaders. Temporary leaders can increase their chances of being selected as super leaders by putting in a large number of Pcoins. The temporary leaders put in a certain amount of Pcoins to indicate their interest in participating in the consensus process. Along with the amount of Pcoins, the concept of randomization is used to select the super leader. The probability of selection is influenced by the product of these two factors, which are the maximum Pcoins and the randomization. This means that temporary leaders with a large amount of Pcoins have a high chance of being selected as the super leader. By giving temporary leaders with large Pcoins a higher chance of being selected as super leaders, ETradeChain aims to encourage the nodes (prosumers) to generate the maximum possible green energy and avoid fossil fuel-based energy.

The Pcoins created in the last period of δT will be put at stake. The super leader gets remuneration for processing the transactions and creating the block. *Algorithm* 19 describes the PoS consensus for block creation and super leader selection. The algorithm for super leader selection and block creation works as follows:

1. All the nodes put their transactions in the transaction pool, which is accessible to all the other nodes. The temporary leaders verify the transactions in the pool.
2. The status of transactions in the pool is set to "pending" to indicate that they are waiting to be included in the next block creation. Once these transactions are included in the block, the status is updated to "processed."
3. All the temporary leaders put their recently generated Pcoins during δT time as a stake to participate in the super leader selection process. As per the algorithm, the super leader is selected randomly out of all the temporary leaders; however, those who put in the maximum Pcoins will have the highest chance of being selected as the super leader. The selected super leader is called the validator.

4. We have given priority to nodes who put maximum Pcoins at stake because it will motivate the nodes (prosumers) to generate the maximum possible green energy and avoid fossil fuel-based energy.

5. The validator starts forging the block. First, it collects the verified transactions from the transaction pool, creates the block by following the intermediate processes like Merkle roots (Mrt), previous block hash (PBH), and block hash (BH), and broadcasts the block in the network for verification.

6. After the verification, the newly created block is added to the existing blockchain, and the network is ready for the next round of blocks to be created, followed by the selection of the super leaders.

7. The other node in the network does the verification. If the other node finds any illegal activity by the leader, the first one to report it will receive the stake of the bad leader.

Algorithm 19: Super Leader Selection and Block Creation

```
Input: Pcoins ;                          /* Pcoins are taken as stake for consensus */
Output: Leader Node, Block (BL)
Function selectValidator(nodes, Pcoin, node.status):
    while node ≠ validator do
        stake ← putStake(Pcoins);
        selectValidator(stake+randomization and node.status);
Function blockGenerate(Tr_New):
    if validator.isForging then
        stake.Locked(node.stake);
        while TP ≠ empty or no_of_transaction = 516 do
            if TP.status == pending then
                Mrt ← createHashTree(TP.getAllTransactions);
                PBH ← H(PB) & BH = H(Mrt, T, PBH);
                block ← createBlock(Mrt, T, PBH, BH);
                broadcast_block(block);
                /* return to the initial step again to create a new block.     */
            else
                Waits for new transactions.;
Function blockVerification(block):
    while node.verifying do
        receive_block(block) & verify_block(block);
        if block == verified then
            stake.Unlocked(node.stake);
            node.status ← good;
            reward ← transaction_fees;
            TP.status ← processed;
        else
            stake.deduct(node.stake);
            node.status ← bad;
```

Remuneration Model

By compensating validators or super leaders for their efforts, block creation has been made more appealing to ETradeChain. The individual who contributes blocks to the chain receives a payment. Each block provided has a cost, which is referred to as transaction fees; hence, remuneration is decided by the blocks added by leaders. Therefore, every transaction is subject to a transaction fee. If every block contains N transactions, then the general equation for the total transaction fee (TTF) can be depicted as follows:

$$TTF = [\sum_{i=1}^{N} TF_i] \tag{5.36}$$

where TF is a transaction fee, and N is the number of transactions in some j^{th} block.

Avoid Malicious Block Generation

In the previous section, we described the remuneration model to reward the leader node for creating the block and appending it to the blockchain. However, applying this approach increases the direct motivation of the leader node to bunch the verified transactions and put them into the block. However, it reduces the motivation of other verifier consensus nodes. So, to solve this difficulty, ETradeChain employs a concept to encourage the verifier consensus nodes to participate honestly in the consensus process. Suppose any validator node has found that the newly created block is malicious or contains some malicious transactions (such as double-spent transactions). In that case, that node is awarded the total transaction fee TTF instead of the leader node, and the total Pcoins deposited by the leader node is also credited to that node. Additionally, the leader node is removed from the network as a punishment and marked as a malicious node. By applying this concept, all the miner nodes will actively participate in the consensus process, which will eliminate the possibility of various attacks.

5.3.5 Implementation Detail and Performance Analysis

In this section, we first give a detailed description of both schemes' implementation environments. Then, we show the performance analysis of both schemes. In SmartGrid-NG, we analyze the GridChain and the whole SmartGrid-NG framework separately. First, we describe the performance analysis of GridChain and explain the performance analysis of SmartGrid-NG. We evaluate and compare the performance of the GridChain based on the consensus algorithm and throughput and SmartGrid-NG with state-of-the-art schemes based on the two parameters, i.e., computation overhead and energy consumption. The performance of ETradeChain is measured based on the five parameters: consensus delay, throughout, computational cost, energy consumption, storage, and communication cost.

Implementation Detail of SmartGrid-NG

We have set up a testbed comprising a Computer System functioning as a Grid Operator and Raspberry Pi 4 model B (RP4) boards serving as consumers. The RP4 board embodies a device akin to a smart meter in its capabilities. We selected the RP4 to represent the smart meter and the integrated unit due to its ability to handle and store the security protocol. It comes equipped with a microcontroller for data processing and a storage unit for data retention—features that are essential for a smart meter. Additionally, this device can

establish communication with other devices via WiFi or Bluetooth. We utilized eight RP4 boards for our consumer setup, each with a 4GB main memory. The computer system operates on a Windows 11 platform and boasts hardware specifications like a 4.4 GHz Intel 64-bit Core i5 processor and 8 GB DDR4 RAM. On the software front, it's equipped with Netbeans 8.2 and Java 8. As for the RP4s, they run on the Raspberry Pi OS (formerly known as Raspbian OS), a Debian-based OS tailored for the RP4. Each RP4 is powered by a 1.5 GHz Broadcom BCM2711 quad-core cortex-A72 processor. We employed the Java programming language to create programs for the testbed, encompassing one grid operator and eight consumers. Figure 5.19 illustrate the testbed setup of SmartGrid-NG.

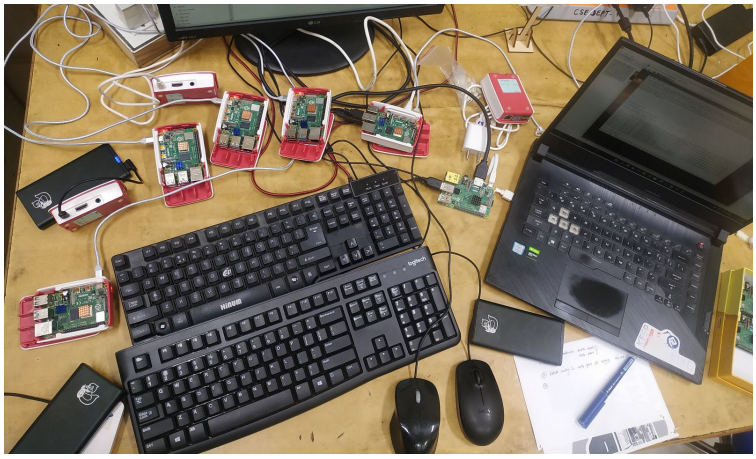


Figure 5.19: Testbed setup for SmartGrid-NG using one computer system and eight Raspberry Pi boards

Implementation Detail of ETradechain

We have developed a testbed that consists of 32 Raspberry Pi 4 model B (RP4) boards. Out of 32, 1 is represented as a smart grid (shown in the system model). 8 out of 32 RP4 boards act as prosumer nodes, and the rest of the RP4 boards act as consumers, where each prosumer is connected to 3 RP4 board consumers. The consumer RP4 board represents a device with the capability of a smart meter; the prosumer RP4 board acts as a smart meter and integrated unit. We chose the RP4 as the smart meter and integrated unit because it has the capacity to process and store the security protocol. It has a microcontroller for data processing and a memory unit for data storage. The smart meter should also have these properties. The device can also communicate with other devices over a WiFi or Bluetooth wireless connection. We have used two types of RP4 boards with different configurations. The RP4 for consumers has 2 GB of memory and 4GB for prosumers. Both boards have the same software and hardware configurations except for the memory. The operating system installed in each RP4 is Raspberry Pi OS, also called Raspbian OS, a Debian-based operating system for RP4. Each RP4 is equipped with a 1.5 GHz Broadcom BCM2711 quad-core cortex-A72 processor. We have used the Java programming language to develop the programs for the testbed. Figure 5.20 shows the implementation environment of ETradeChain. This figure shows seven prosumer RPs, eight consumer RPs, and one smart grid RP4 unit. We show only a few RP4s in our testbed image for clear visibility. Figure 5.21 shows the internal working process of the proposed scheme in the RP4 module. Figure 5.21a shows that the prosumer generates the block. Figure 5.21b shows the consensus process is taking place, i.e., the selection of a super leader.

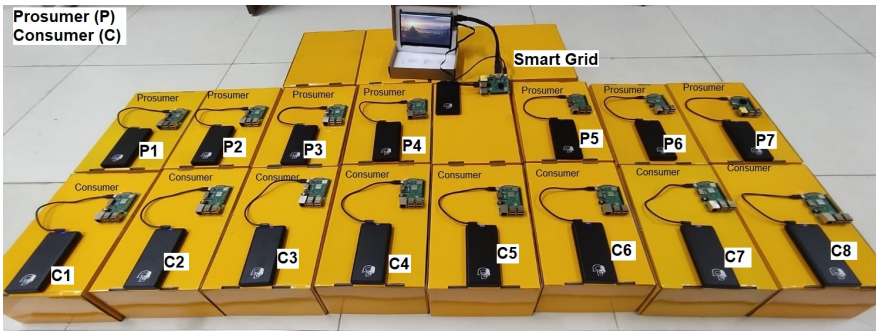


Figure 5.20: Implementation environment for ETradeChain testbed

Evaluation of GridChain

The GridChain consensus algorithm is evaluated based on consensus delay and throughput parameters. **Consensus Delay:** The consensus delay of GridChain is better than state-of-the-art schemes for two reasons.

- Leader-Based Consensus: GridChain's leader-based consensus mechanism ensures that only one node (the leader) is responsible for block generation at any given time. This eliminates the need for multiple nodes to compete for block generation, as seen in state-of-the-art schemes like PoW. This focused approach can significantly reduce the time taken to reach a consensus.
- Reputation-Based Leader Selection: Another reason is that, by selecting leader nodes based on reputation, GridChain ensures that only the most trustworthy and efficient nodes take the lead. This can reduce the chances of selecting a malicious or slow node as a leader, which could otherwise introduce delays in the consensus process. However, some of the other schemes employ consensus mechanisms that require nodes to solve cryptographic puzzles or engage in multiple rounds of communication. Such processes can be time-consuming, leading to higher consensus delays.

Throughput: Like the consensus delay, the throughput of the GridChain is also better compared with state-of-the-art schemes. Below are the three reasons to support the better performance of the GridChain.

- Efficient Block Verification: In GridChain's block verification process, the leader node sends the new block with a nonce value, and verification nodes recreate the block to check its validity. This direct approach can lead to faster block verifications, allowing the system to process more blocks in a given timeframe, thereby increasing throughput.
- Reputation System: By continuously evaluating the behaviour of consensus nodes, GridChain ensures that only efficient and trustworthy nodes participate actively. This can lead to faster transaction verifications and block additions, enhancing the system's overall throughput.
- Intricate Verification Processes: Some of the other schemes have more intricate block verification processes, which could involve multiple rounds of communication or complex cryptographic verifications. Such processes can be time-consuming, limiting the number of transactions and blocks that can be processed in a given time frame and reducing throughput.

```
pi@raspberrypt :~ /Desktop/EnTrade/Block/Client

a876e4628f50f7e721dffac18e1abb2a5f06bef02cac4\",\n \"TransID\": 500,1
n \"CoinID\": 500,\n \"CoinStatus\": \"U\", \n \"BlockID\": 500, \n
\"BlockTransactionNo\": 1,\n \"Sender\": \"USER11\", \n \"Receiver\":
\"SMARTGRID\",\n \"TransactionType\": \"C\", \n \"TransactionDate\":
\"2021-11-12 00:00:00\", \n \"Broadcasted\": \"Y\", \n \"previousHash
\": \"39a9ed60769f055904eba8fb9fe86a239b091af6bbb2c6116754a4bae34ee73a
\",\n \"timeStamp\": 0, \n \"nonce\": 0\n}", "ReceiverPort": "9004", "Re
ceiverUsername":"SMARTGRID", "Sender IPAddress":"192. 168.137. 21",
"hash":
"f5a4c2055670ffb617ea876e4628f50f7e721dffac18e1abb2a5f06bef02cac4","Re
ceiverIPAddress":"192.168.137.10"}ENTRADEM20CS017MIGIAkIAzajsnsocpj4Nw
idQtUrNbCgIkglgY0TckmbogUiv9y04y7DZAZVCXHCHLGYOHRxjQOWYHtaeE5VqgV1RRmF
OV2WCQgHk6410ykHJwy4EeGBepJogIh5Ri4E4V5AamYGoBjZUR+lYwBjNF6ptORh7VqN8s
AB8I8NtDnZ6bjYlQfuSeb5Zog ==
i am finally done
PBFT Time2631
POS Time896
Block Creation Time: 384670
pi@raspberrypi :- /Desktop/EnTrade/Block/client $
```

(a) Prosumer Generating Block

```
pi@raspberrypi: ~/Desktop/EnTrade/Block/Server

Valid user, processing ahead
POSFLAG
inside stringtosecretkey:e40yBMNTfMSgMgwhOa301EY56FVfLi/7ewtgs5GPt48=
Valid user, processing ahead
Received Super Leader
inside stringtoSecretkey:e40yBMNTfMSgMgwhOa30LEY56FVfLi/7eWtgs5GPt48=
Valid user, processing ahead
POSFLAG
inside stringtosecretkey:e40yBMNTfMSgMgwhOa301EY56FVfLi/7ewtgs5GPt48=
valid user, processing ahead
POSFLAG
inside stringtoSecretKey:e40yBMNTfMSgMgwh0a301EY56FVfLi/7eWtgs5GPt48=
Valid user, processing ahead
Received Super Leader
inside stringtoSecretKey:e40yBMNTfMSgMgwhOa301EY56FVfLi/7ewtgs5GPt48=
Valid user, processing ahead
Received Super Leader
```

(b) Consensus Algorithm Processing

Figure 5.21: ETradeChain working process on the RP4 devices. These are the command line interface of RP4 doing all the required processing

The evaluation results show that GridChain performed very well compared with state-of-the-art schemes while maintaining a high level of security. Most of the previous cryptocurrencies have fork generation issues that are addressed by the proposed GridChain protocol. To reduce the propagation delay, we have fixed the upper bound of the block size to 1 MB. The maximum number of transactions in a single block is 516 in order to keep the block size under 1 MB. Figure 5.22a shows the improved performance of GridChain for consensus delay compared with state-of-the-art schemes. In GridChain, the consensus delay is reduced by approximately 80% compared with the other schemes. Figure 5.22b depicts the GridChain's throughput. The results show that GridChain outperformed other blockchain protocols in terms of performance. GridChain-NG's throughput has also been tested with varying time intervals between the leader nodes, as presented below.

Evaluation of SmartGrid-NG

We evaluate and compare the performance of the SmartGrid-NG with state-of-the-art schemes based on the mentioned two parameters, i.e., computation overhead and energy consumption. We have compared

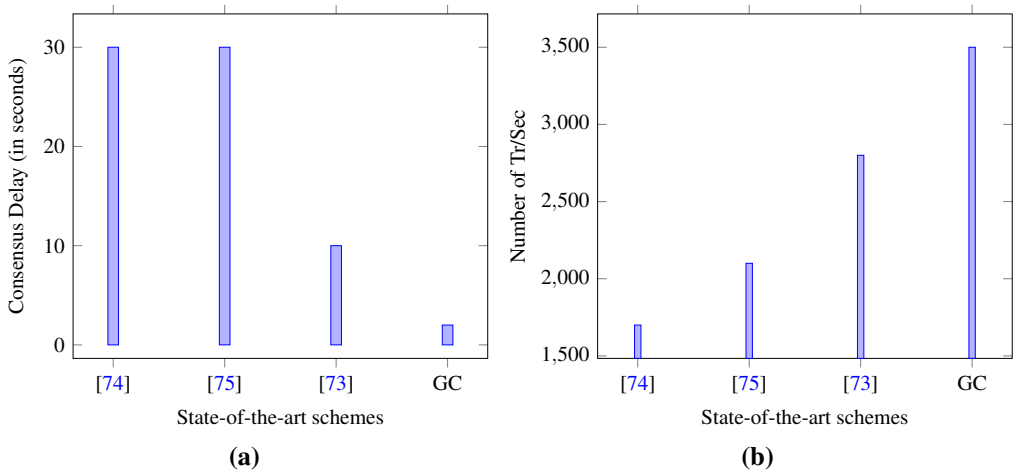


Figure 5.22: Comparative analysis of GridChain based on (a) consensus delay and (b) throughput (number of transactions executed per second) with state-of-the-art schemes

the performance of SmartGrid-NG with state-of-the-art schemes [122], [169], [119], [170], and [118]. We selected these schemes from the others mentioned in the related work section for the following reasons: The selected papers align more closely with the proposed SmartGrid-NG. Additionally, these selected papers provide data and results that aided our understanding of the schemes and facilitated comparison analysis. Computational cost is determined by the time taken to execute various cryptographic tasks within the framework. These tasks include authentication, verification, communication, and the creation and verification of blocks. The average execution time (after performing the execution 1000 times) for the different cryptographic functions is shown in Table 5.13. This table also illustrates the time required to execute these cryptographic functions on the specified platforms, namely the computer system and Raspberry Pi 4 model. A comparison of the computational cost of SmartGrid-NG with state-of-the-art is presented in Table 5.14. Once the computational cost is determined, it's possible to compute the energy usage using the formula, $E_{rg} = T_{ct} * C_p$. In our testing environment, the desktop system consumes 10.88 W of power for computing one task, while the Raspberry Pi 4 consumes 2.4W. As shown in Table 5.14, it is evident that the computation cost of SmartGrid-NG is significantly lower compared to state-of-the-art schemes. This is attributed to the number of cryptographic functions used throughout the process, from generating transactions to their verification and addition to the blockchain ledger. The state-of-the-art schemes employed intensive cryptographic functions, which increased the computational cost. Consequently, the energy consumption at the device level for executing operations is also high. The last two columns of Table 5.14 clearly highlight the lower energy consumption of SmartGrid-NG compared to state-of-the-art schemes.

Table 5.13: Individual computation time of different cryptographic functions for smart grid testbed

Functions	**On Desktop**	**On Raspberry Pi 4GB**	**On Raspberry Pi 2GB**
SHA256 Hash function: $T_{H()}$	0.0008	0.005	0.045
ECC Crypto-system: T_{ECC}	0.053	1.6	3.9
ECDSA signature: T_{sign}	0.071	2.94	6.13
ECDSA verification: T_{veri}	0.065	2.86	5.63
point multiplication: $T_{mul(.)}$	0.117	5.0	11.0
AES Crypto-system 192-bit: T_{AES}	0.125	5.96	12.71
Modulo operation: $T_{mod(.)}$	0.067	2.0	5.0

Table 5.14: Performance analysis of SmartGrid-NG based on the comparison of computation cost (CC) in milliseconds and energy consumption (EC) in Joule on Desktop and Raspberry Pi 4 2GB (RP) with state-of-the-art schemes

Scheme	Computation Cost Equation	CC on Desktop	CC on RP4	EC on Desktop	EC on RP4
[122]	$5T_{H(.)}+3T_{sign}+3T_{veri}+3T_{ECC}$	0.592	22.23	178.704	85.068
[119]	$6T_{H(.)}+4T_{sign}+4T_{veri}$	0.549	23.79	126.53	57.096
[169]	$6T_{H(.)}+3T_{sign}+3T_{veri}$	0.413	17.43	189.639	85.32
[170]	$6T_{AES}+2T_{sign}+2T_{veri}$	1.02	47.36	515.277	167.472
[118]	$5T_{sign}+5T_{veri}+2T_{ECC}$	0.786	32.20	350.336	159.84
S-NG	$3T_{H(.)}+3T_{sign}+3T_{veri}$	0.410	17.41	63.267	28.548

Evaluation of ETradeChain

We have evaluated ETradeChain based on six parameters, which are consensus delay, throughput, computation cost, energy consumption, storage cost, and communication cost.

1. Consensus Delay and Throughput: When evaluating ETradeChain, we look at the number of transactions per second, the time it takes to reach an agreement, and how fair it is when selecting new leaders. The assessment results demonstrate that, without compromising the security level, reducing consensus latency and increasing throughput is feasible by altering the configuration of the miner nodes, i.e., two-level leader selection, first auction-based temporary leader selection, and a second super leader. Nearly every statistic across all of the tested characteristics shows that ETradeChain is superior to the competition. The state-of-the-art blockchain methods have a common weakness: fork creation and high consensus delay. ETradeChain addressed these weaknesses. To lower the propagation delay in the blockchain network, we fix the block size at less than or equal to 1 MB. Each block contains a maximum of 516 transactions, making the block size less than 1 MB. Each block contains three components: the block header, transactions, and transaction count. The block header size is 80 bytes, the transaction count is 4 bytes, and the remaining space, i.e., 10,48,492 bytes, is occupied by the transactions. Each transaction has a maximum of 2000 bytes. Therefore, each block can contain a maximum of 516 transactions. Figure 5.23a shows the consensus delay of ETradeChain compared with state-of-the-art schemes. The result shows that the consensus delay of ETradeChain is approximately 90% low as compared to other blockchain protocols.

Figure 5.23b depicts ETradeChain's throughput. When compared to other schemes, ETradeChain performed exceptionally well. Changing the time gap between leader nodes changed the throughput of the ETradeChain. As the number of forgers grows, the likelihood that various forgers will have an equal probability of generating a block increases. The ETradeChain is fairer than other platforms. The state-of-the-art schemes show significant congestion in the network and unnecessary forks due to the competition in solving computational puzzles first to win the reward. In ETradeChain, the architecture first chooses the validator as a temporary leader and then as a super leader. This reduces congestion and increases throughput.

2. Computational Cost and Energy Consumption: Computational time is the time taken by the Testbed for the execution. The measuring unit is milliseconds and is represented as T_{ct}. We illustrate the computational cost (also known as execution time) based on the operational costs for authentication, verification, communication, and block creation and verification. Table 5.13 shows different cryptographic functions and their computation time on both types of RP4 boards. The time taken by all these cryptographic operations is based on the implementation environment we defined earlier. Table 5.15 shows the comparative computation cost of ETradeChain versus state-of-the-art schemes. The energy consumption of a system is the amount of energy consumed by the system in order to carry out its operations of the system. During

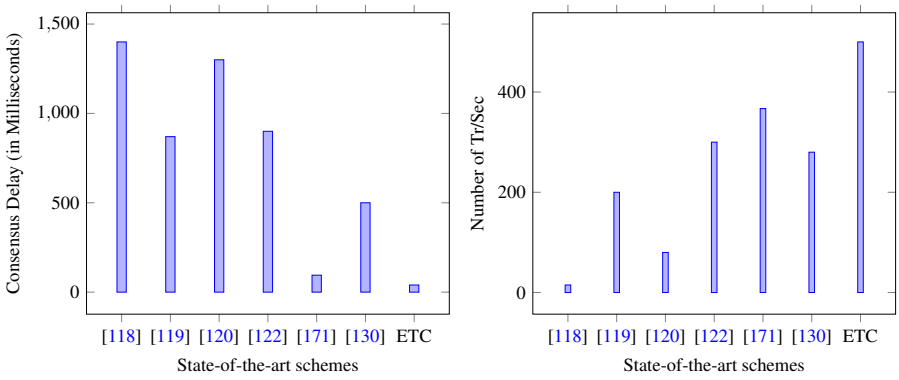


Figure 5.23: Comparative analysis of ETradeChain based on (a) consensus delay and (b) throughput with state-of-the-art schemes

the execution process, the system requires a certain amount of energy in order to complete all of the phases of the scheme. The equation $E_{rg} = T_{ct} * C_P$ is used to calculate energy consumption, where C_P is the CPU power (5.6 W for RP4 4GB and 2.4 W for RP4 2 GB) and T_{ct} is the computational cost. The computation cost was already measured in the previous section. Like computation cost, energy consumption is also measured on two platforms, i.e., RP4 with 4 GB and RP4 with 2 GB. Table 5.15 shows the comparison of ETradeChain with state-of-the-art schemes.

Table 5.15: Performance analysis of ETradeChain based on the comparison of computation cost (CC) in milliseconds and energy consumption (EC) in Joule of ETradeChain on Raspberry Pi 4 (RP) 2GB and 4GB with state-of-the-art schemes

Scheme	Computation Cost Equation	CC on RP4 4GB	CC on RP4 2GB	EC on RP4 4GB	EC on RP4 2GB
[170]	$6T_{AES} + 2T_{sign} + 2T_{veri}$	47.36	99.78	265.22	239.48
[118]	$5T_{sign} + 5T_{veri}$	29	59.95	162.4	143.88
[119]	$6T_{H(.)} + 2T_{sign} + 2T_{veri}$	11.63	24.25	65.128	58.20
[120]	$11T_{H(.)} + 8T_{mul(.)}$	40.05	88.50	224.28	212.38
[122]	$5T_{H(.)} + 3T_{sign} + 3T_{veri}$	17.43	35.97	97.61	86.33
ETradeChain	$3T_{H(.)} + 1T_{sign} + 1T_{veri}$	5.82	12.13	32.54	29.11

3. Storage and Communication Cost: There are various computations performed in the processing of transactions and blockchain operations, such as computing hash code, secret keys, digital signatures and many others. These computations consume some memory space on the node side and in communications. Therefore, the storage cost refers to the storage space required for storing the output value of the computing operations. The measuring unit for storage cost is bytes and represented as S_{sc}. Generally, a Key S_k requires 24B (B=Bytes), ECC encryption/decryption S_{ECC} takes 48B, AES encryption/decryption S_{AES} takes 24B, a hash function (SHA-256 bit) $S_{H(.)}$ takes 32B, and a signature S_{sign} and verification S_{veri} requires 10B. Table 5.16 shows the comparative cost of ETradeChain along with the state-of-the-art schemes. As shown in Table, 5.16, the number of operations in ETradeChain is much less compared with state-of-the-art schemes. Therefore, the storage cost is also very low in ETradeChain.

Communication cost refers to the storage required for storing the communicating variables such as hash code, ciphertext, and other parameters. The measuring unit is a byte and is represented as C_{cc}. The operations and variables that have been used in communication between the smart meter and the miner

node are the hash function and the ECDSA digital signature. Table 5.16 shows the comparative analysis of different mechanisms against ETradeChain.

Table 5.16: Comparison of storage and communication cost of ETradeChain with state-of-the-art schemes

Scheme	Storage Cost	Cost in MB	Communication Cost	Cost in MB
[170]	$2S_{AES}+2S_{sign}$	52	$4S_{AES}+2S_{sign}$	84
[118]	$8S_{H(.)}+8S_{sign}$	592	$8S_{H(.)}+8S_{sign}$	592
[119]	$5S_{H(.)}+2S_{sign}$	340	$3S_{H(.)}+1S_{sign}$	202
[120]	$10S_{H(.)}+3S_{mul(.)}$	700	$ElementData(230Bytes)$	230
[122]	$3S_{H(.)}+1S_{sign}$	202	$3S_{H(.)}+1S_{sign}$	202
ETradeChain	$2S_{H(.)}+1S_{sign}$	74	$2S_{H(.)}+3S_{sign}$	94

5.3.6 Security Analysis for SmartGrid-NG and ETradeChain

In this section, we have shown the security analysis of the proposed GridChain and SmartGrid-NG. We have shown that the GridChain is secure from double-spending attacks and fork attacks. Also, we have shown that the SmartGrid-NG achieved all the security requirements mentioned in our contribution, which are transaction integrity, confidentiality, authentication, and fairness. In this section, we have also explained the auction property analysis and security analysis of ETradeChain. The security analysis of the ETradeChain justifies the prevention of double-spending attacks, the prevention of forking, fault tolerance, and non-repudiation. Below, we have proven with the help of a lemma and proof that the system is protected against the attacks mentioned above. Table 5.17 shows the security comparison of ETradeChain with the state-of-the-art approach where $\surd$ represents "Secure from Attack" and $\otimes$ illustrates "Not Secure from Attack."

Table 5.17: Security comparisons of SmartGrid-NG and ETradeChain with state-of-the-art schemes

Attacks	[118]	[119]	[120]	[122]	[171]	[130]	SmartGrid-NG	*ETradeChain*
Double-Spending	$\surd$	$\surd$	$\otimes$	$\surd$	$\otimes$	$\otimes$	$\surd$	$\surd$
Forking	$\otimes$	$\otimes$	$\surd$	$\surd$	$\otimes$	$\surd$	$\surd$	$\surd$
Non-Repudiation	$\otimes$	$\otimes$	$\surd$	$\otimes$	$\surd$	$\otimes$	$\surd$	$\surd$

Lemma 14. *The risk of a double spending attack is minimized in GridChain based on the attacker's computational power (q) in comparison with honest node (p). Thus, the GridChain protocol prevents the double-spending attack.*

Proof. Following the binomial random walk [172], a race condition can be modelled. A race condition R is the difference in the computational capacity of mining blocks between honest (p) and malicious $(q = 1-p)$ nodes, i.e., the attack can be derived as

$$R_{i+1} = \begin{cases} R_i + 1, for\ p \\ R_i - 1, for\ q \end{cases} \tag{5.37}$$

where i is the individual block generation race. It is clear from (5.37), that if $q > p$, then the attacker continues to generate the blocks until $R < 0$, meaning it can replace all the blocks generated by the honest

node. Therefore, the probability of an attacker overtaking the honest node using Gambler's Ruin Problem [173] can be modelled as

$$Pr_i = qPr_{i+1} + pPr_{i-1} \tag{5.38}$$

since, p+q=1, the equation (5.38) can be rewritten as

$$Pr_{i+1} - Pr_i = \frac{p}{q}(Pr_i - Pr_{i-1}) \tag{5.39}$$

The equation (5.39) can written in generalized form as

$$Pr_{i+1} = Pr_1 \sum_{j=0}^{i} (\frac{p}{q})^j = \begin{cases} Pr_1 \frac{(1-\frac{p}{q})^{i+1}}{1-\frac{p}{q}}, if\ p \neq q \\ Pr_1(i+1), if\ p = q = 0.5 \end{cases} \tag{5.40}$$

Let, $i = N-1, Pr_N = 1$ By solving Pr_1, we will get,

$$Pr_i = \begin{cases} \frac{1-(\frac{p}{q})^i}{1-(\frac{p}{q})^N}, if\ p \neq q \\ \frac{i}{N}, if\ p = q = 0.5 \end{cases} \tag{5.41}$$

If we consider that the attacker has unlimited computational power, i.e., $i \longrightarrow \infty$, and $q > p$, then,

$$\lim_{i \to \infty} \frac{1-(\frac{p}{q})^i}{1-(\frac{p}{q})^N} = 1 \tag{5.42}$$

By normalizing the (5.42), we will get the probability of the attacker overtaking the honest node, which can be defined as

$$Qr_z = \begin{cases} (\frac{q}{p})^{z+1}, if\ q < p, z \geq 0 \\ 1, if\ q > p, z < 0 \end{cases} \tag{5.43}$$

It is clear from the above mathematical derivation that, in the proposed permissioned blockchain protocol GridChain, if the attacker's computational power (q) of block generation is higher than the computational power of the honest node (p), then there is a possibility of a double-spending attack. However, GridChain is a leader-based consensus algorithm where the leader is responsible for generating blocks, and the rest of the network is responsible for block verification. Therefore, to launch a successful attack, an attacker needs to increase the computational power greater than the whole network, which is nearly impossible. Thus, GridChain is protected from double-spending attacks. □

Lemma 15. *Let h_b be the height of honest block b and $h_{b'}$ is the height of malicious block, where $h_b = h_{b'}$, then a fork will occur if $b, b' \in \mathbb{B}$ and $b \neq b'$. The GridChain prevents the fork because if $b \neq b' \in \mathbb{B}$, then $h_b \neq h_{b'}$.*

Proof. As per the statement in the lemma, a fork condition can also be illustrated in the following form:

$$\exists b, b' \in \mathbb{B}\ and\ b \neq b' | h_b = h_{b'} \tag{5.44}$$

Consider a node in $\mathbb{B}$ of height h_b gets a block b' with a height of $h_{b'}$ such that $h_{b'} > h_b$. According to the longest chain rule, the new height of blockchain $\mathbb{B}$ changed to $h_{b'}$. Also, block b is the ancestor of block b'. However, if both the blocks are added in different branches, the block b will be orphaned, i.e., removed

from the main branch. Further, the local state of the $\mathbb{B}$ maintained by the node should be consistent with the rest of the network, considering this as the longest chain. GridChain follows the leader-based consensus algorithm, where only one node is responsible for generating the block at a time. Suppose two different blocks arrive at the same time. In that case, it is clear that one of the blocks is malicious, which will be identified by the network and later discarded. Therefore, two different blocks b and b' can not belong to the same blockchain $\mathbb{B}$ at the same height, i.e., $h_b \neq h_{b'}$. □

Lemma 16. *Let Tr be the transaction generated by smart meter SM_i, $i = 1,2,...,N$, and Λ is the adversary who might attack and alter its integrity of it. The SmartGrid-NG provides the authentication and integrity of data shared using MAC and a secure hash function.*

Proof. The proposed SmartGrid-NG is secured from transaction alteration and authentication, which can be illustrated as follows.

1. To secure the integrity of transactions, the proposed SmartGrid-NG does the, the smart meter SM_i, $i = 1,2,...,N$ and consensus node CN_j, $j = 1,2,...,M$ compute the $MAC_{Sk_i^{CH_j}}$ and $MAC_{CH_j{}^{Sk_i}}$, where MAC_{sk} is message authentication technique utilizing the one-way secure hash function $SHA256$ with the secret key sk.
2. It is worth noting that, the security of MAC_{sk} depneds on the security of sk and the hash function used which is $SHA256$ in our case. Due to the discrete logarithm problem (DLP) on the group G (generator of ECC), the adversary Λ can not obtain the shared secret session key $sk = Sk_i^{CN_j} = Sk_{CN_j}^i$. Therefore, Λ can not forge and tamper transactions or unique identities of SM_i and the consensus node CN_j.
3. Besides, the transaction and data sharing between SM_i and CN_j incorporate the timestamp t_τ in computing the shared transactions and MAC_{sk}. Because only the MAC_{sk} with current timestamp t_τ can go through the successful verification process. In this way, the SmartGrid-NG is also prevented from message replay attacks.
4. Thus, it is ensured that, in SmartGrid-NG, the data is authenticated at the receiving end, and its integrity is ensured. Let's consider that an adversary is able to get transactions while in transit and tries to do some modification. To resend these transactions, the signature of a legitimate sender is required, which is not possible, as we have assumed that the adversary can not get the private key of any node. Thus, the scheme ensures the integrity of the transactions.

□

Lemma 17. *Let Tr be the transaction generated by smart meter SM_i, $i = 1,2,...,N$, and Λ is the adversary who might attack and expose the confidentiality of the communication. The SmartGrid-NG provides confidentiality using the Elliptic curve Diffie–Hellman (ECDH) and secure hash function.*

Proof. The SmartGrid-NG is secured from any type of confidentiality attack due to the following computation in the communication between SM_i and CN_j and also between CN_i and CN_j.

1. According to the adversary model defined above, it is ensured that the Λ can not calculate the $Sk_i^{CN_j} = Pr_{SM_i} * Pb_{CN_j}$ and $Sk_{CN_j}^i = Pr_{CN_j} * Pb_{SM_i}$ due to the computation diffie-hellman problem.

2. Due to the shared session key being calculated at the SM_i end and CN_j end, it is impossible for Λ to obtain the shared secret session key. Therefore, Λ can not decrypt the CT_{Tr_k} without knowing the secret key. Thus, confidentiality is ensured.

□

Lemma 18. *The GridChain employed the reputation value for fair leader selection for the consensus mechanism. Thus it provides fairness in leader selection and is free from unbiased consensus.*

Proof. The leader node is responsible for every new block generation and verification of it. Therefore, a fair leader node selection algorithm is crucial for the unbiased behaviour of the method. The fairness of leader selection in GridChain can be expressed as follows to show the unbiased nature of it.

1. We have computed the reputation value $Rep_{CN_j}^{final,k}$ using the reputation updation model based on opinion calculation and interaction behavior.
2. Node reputation updation model is based on the many parameters and recent and past behavior of the node, such as positive and negative behavior, and effectiveness of the recent interactions. Through the reputation computation model, we calculate the trust ($T_{i,j}$) and uncertainty ($U_{i,j}$) as mentioned in subsection 4.4.3. Also, the final reputation value is computed by the current leader node CN_x, which is $Rep_{CN_j}^{(final,k)}$ in the k^{th} round of the consensus.
3. It is ensured that the node with the highest reputation value will not always be selected because, in the leader selection section, we have defined that in every round, a new group is formed based on the threshold value, and out of them, one node is selected. Thus, GridChain ensured fairness in the leader selection and consensus mechanism.

□

Lemma 19. *ETradeChain's auction is budget balanced.*

Proof. From the Algorithm 18, we can see that the total payment received by the j^{th} prosumer is the sum of the j^{th} column in the pay matrix. Every column contains i rows. Therefore, the total payment received by j^{th} prosumer is $Pay_j = \sum_{i=1}^{M} PM_{ij}$. Similarly, total payment given by i^{th} consumer will be $Pay_i = \sum_{j=1}^{N} PM_{ij}$. The total payment received by all the prosumers will be

$$TotalPay_j = \sum_{j=1}^{N}\sum_{i=1}^{M} PM_{ji} \tag{5.45}$$

Similarly, the total payment given by all the consumers can be represented as,

$$TotalPay_i = \sum_{i=1}^{M}\sum_{j=1}^{N} PM_{ij} \tag{5.46}$$

From equation (5.45) and (5.46), we can see that the Total payment received by prosumers is equal to the total payment given by the consumers. Therefore, we proved that our system is budget-balanced. □

Lemma 20. *ETradeChain's auction satisfies individual rationality.*

Proof. Individual Rationality means no peer should lose by joining the auction. In our case, the utility of consumers will never be negative. As per the condition in Algorithm 17, we are ensuring that in the allocation scheme, the bid price of the consumer should be greater than or equal to that of the prosumer. Also, considering the equation (5.35), we can say that the buyer will never pay more than the bid value, and the seller will never receive less than the asking value. Therefore, the utility of the buyer and seller will never be negative. It will be zero if they lose. Therefore, individual rationality is satisfied. □

Lemma 21. *The ETradeChain protocol ensures no fork generation in the blockchain or that there is no possibility of parallel block generation with the same set of transactions.*

Proof. The fork generation can be easily understood. The minimum difference between the number of blocks generated by a malicious node and a fair node to confirm a transaction and lower the blockchain network's vulnerability probability than the threshold $\varepsilon \in (0,1]$ is defined as follows:

$$\alpha^*(\varepsilon) = min\{\alpha : P(Q > \alpha) \leq \varepsilon\} \tag{5.47}$$

$$\alpha^* = -\log(\varepsilon)/\log(\lambda/\Lambda) \tag{5.48}$$

where P is the probability of a malicious node generating the block before the fair node. The high value of α^* has the efficiency to counterattack the intentional forks. To ensure blockchain network stability, fair node computing power must be increased. The computing power required to finish the forging in the allotted time frame is equal to the fair node's effective computational power. ETC's fair leader node has a lot more computing power than the malicious node in order to generate a block. As a result, an attacker is unable to fork the ETradeChain. ETradeChain also does not enable many nodes to forge the same block simultaneously, meaning that only the fair super leader node can forge the block at any one time. Thus, fork formation in the ETradeChain is less likely to occur.

However, ETradeChain has a time period in which the leader creates the blocks. This is in contrast to other blockchain protocols like Bitcoin-NG [70], and OmniLedger [74] that have fork generation problems. After the interval has passed, the next node in the network generates the block after selecting it as the super leader node. Due to this, ETradeChain does not have the problem of fork formation because two forger nodes cannot generate the block simultaneously. As a result, ETradeChain has no issues with fork creation. □

5.4 PRIVACY-PRESERVING ENERGY TRADING IN SMART GRID

With the inclusion of blockchain technology in smart energy, consumers can also harvest solar energy and sell the surplus energy to neighbours or the main grid. Such consumers are called prosumers. This advancement solved much of the problem of long-distance energy transmission. Additionally, this is also cost-effective for consumers [174]. In transaction processing, sensitive information is included, which needs

to be protected. Therefore, user data privacy is a big concern [49]. Additionally, the existing blockchain solutions are not suitable for applications like a smart grid because they do not offer any privacy preservation of sensitive information [175]. However, privacy is achieved in the blockchain-based cryptocurrency Zcash using the zk-SNARK concept in [176]. Similarly, privacy in electronic devices in consortium blockchain is achieved in [177].

5.4.1 Techniques Used to Provide Privacy in Blockchain

Following are the widely used techniques for privacy preservation.

1. **Zero-Knowledge Proofs (ZKPs):** A cryptographic method where one party can prove to another that a statement is true without revealing any specific details about the statement itself. Cryptocurrencies like Zcash [31] employ ZKPs (specifically zk-SNARKs [32]) to ensure transactions are recorded, but details like sender, receiver, and transaction amount remain concealed [33].

2. **Traceable Ring Signatures:** This technique, integral to the Monero cryptocurrency, ensures sender anonymity by allowing a transaction to be signed and verified by a group. Within this group, the actual signer remains anonymous, ensuring transactional privacy [178].

3. **Homomorphic Encryption:** A groundbreaking technique, homomorphic encryption allows computations on encrypted data (ciphertexts), producing an encrypted result. When decrypted, this result matches as if operations were performed on the original data, ensuring data privacy even during computations [179].

4. **Private Sidechains:** These are auxiliary blockchains running parallel to the primary chain, facilitating private transactions. Once a transaction concludes, only its final state might be recorded on the main chain, keeping transactional nuances private [97].

5.4.2 *zk-SNARK* Background

A *zk-SNARK* is a widely used scheme for zero-knowledge proof, which enables the provers to prove the validity of a given assertion to any verifier without revealing any information [180], [181]. It can be represented as the tuple of probabilistic polynomial algorithms $\zeta_{zk} = (Preprocessing, Prove, Verify)$ such that:

1. *Preprocessing* ($S(\mathbb{C})$): Given a arithmetic circuit $\mathbb{C}$, the preprocessing $S(\mathbb{C}$ produces two public parameters, the S_P and S_V for the prover and verifier respectively. It is important to note that the further process is dependent on these parameters.

$$S(\mathbb{C}) \longrightarrow (S_P, S_V)$$

2. *Prove* $P()$: It is the algorithm that is responsible for generating the proof π provided the public statement X and secret statement ω in $\mathbb{C}$.

$$P(S_P, x, \omega) \longrightarrow \pi$$

3. *Verifier* $V()$: The verification algorithm uses the public statement x and the proof π to verifiy the secret witness ω.

$$V(S_V, x, \pi) \longrightarrow \begin{cases} 0; Accept \\ 1; Reject \end{cases}$$

It is important to note that the verification process reveals nothing about the witness or secret value ω.

Definition 6. *Let $\mathbb{C}$ be an arithmetic circuit, P is the prover, V is the verifier, π is the proof, and ω is the witness, then $(\mathbb{C}, P, V)$ is zero knowledge if for every statement $x \in \mathbb{F}^n$, the proof π generated by prover reveals nothing about the ω. The Proof π reveals nothing about ω if the verifier can generate π on its own and learnt nothing from the π.*

5.4.3 Public Traceability

A traceable ring signature scheme requires that the following condition must hold: For every $k \in N$, every $n \in N$, every i,$i \in N = 1, ..., n$, every $issue \in 0, 1^*$, and every m, $m \in 0, 1^*$, if $(pk_N, sk_N) \leftarrow Gen(1^k)$, $\sigma \leftarrow Sig_{ski}(L, m)$, where $L = (issue, pk_N)$, and $\sigma \leftarrow Sig_{ski}(L, m)$, it holds with an overwhelming probability $(\in k)$ that

$$T(Tg, Msg, \sigma_{Msg}, Msg', \sigma'_{Msg}) = \begin{cases} indep & if\ N_i \neq N'_i, N_i \in N \\ linked & else\ if\ Msg = Msg' \\ Pubk_{N_i} & otherwise \end{cases} \tag{5.49}$$

The equation 5.49 can be interpreted as follows. The function $\mathbb{T}()$ never outputs "linked" if $Msg \neq MSg'$. If a traceable signature scheme has tag-linkability (along with public traceability), $\mathbb{T}(Tg, Msg, \sigma_{Msg}, Msg', \sigma'_{Msg})$ = "indep" implies that these two signatures are generated by different signers.

Third things is that $\mathbb{T}(Tg, Msg, \sigma_{Msg}, Msg', \sigma'_{Msg}) = Pubk_{N_i}$ implies that they are signed by the same signer N_i. In addition, if $Msg \neq MSg'$, Trace never outputs "linked." Public traceability is a correctness condition; that is, it does not assure that the opposite holds. However, if a traceable signature scheme has tag-linkability (along with public traceability), $Trace(Tg, Msg, \sigma_{Msg}, Msg', \sigma'_{Msg})$ = "indep" implies that these two signatures are generated by different signers. If it has exculpability,

$$Trace(Tg, Msg, \sigma_{Msg}, Msg', \sigma'_{Msg}) = pk_{N_i}$$

implies that they are signed by the same signer i. Note that $Trace(Tg, Msg, \sigma_{Msg}, Msg', \sigma'_{Msg})$ = "linked" doesn't mean that they are always generated by the same signer (because anyone can make a "dead" copy of any signature).

5.4.4 *zk-SGB*: Privacy-Preserved Blockchain Framework for Energy Trading in Smart Grid Using Zero Knowledge Proof

We design and develop a privacy-preserved energy trading system called *zk-SGB* using blockchain technology, which is more reliable, scalable, robust, and resistant to cyber-attacks. For the best result possible in *zk-SGB* scheme, we have also proposed a novel consensus algorithm called Weighted Node Selection Byzantine Fault Tolerance (WNS-BFT), which is suitable for permissioned blockchain Smart Grid network.

It provides a fair and robust consensus scheme utilizing active, honest past participation and computational power as the core element to select the honest group of consensus nodes. Additionally, *zk-SGB* integrated with the Zero-Knowledge Succinct Non-interactive Argument of Knowledge (zk-SNARK) to preserve privacy. It has three components, i.e., Smart Grid (Energy Provider), Prosumer (Energy Seller + Buyer), and Consumer (Buyer). All the components are connected, and relevant information is exchanged. The smart meters are installed at the consumer's place and monitor the consumer's power consumption. The Smart Grid is the service provider that distributes power to the consumer. The prosumers formed the Blockchain network, which is a permissioned blockchain network. The blockchain network securely stores the user information and the transactions of the smart meter in the form of blockchain. Our primary contributions are as follows:

1. **Novel Consensus Algorithm:** The proposed novel WNS-BFT consensus algorithm is a fair and robust leader-based consensus algorithm. The WNS-BFT creates the group of honest consensus nodes using two parameters: 1) honest participation score and 2) computational power of the consensus nodes.
2. **Real World Testbed Implementation:** We have developed a testbed using Raspberry Pi devices for conducting experiments. This practical implementation showcases the viability of the proposed protocol in a real-world setting.
3. **Quantitative Performance Metrics:** The *zk-SGB* provides specific performance metrics, such as up to 90% reduction in consensus delay, up to 60% increase in throughput, and up to 80% reduction in computational overhead. These quantified improvements resulting from the use of *zk-SGB* demonstrate the practical benefits of the proposed solution compared with state-of-the-art schemes.
4. **Targeted Security Analysis:** The *zk-SGB* ensures privacy preservation and protection against double-spending, forking, modification attacks, and confidentiality breaches. This targeted security analysis illustrates the challenges specific to energy trading in the smart grid, highlighting the framework's ability to provide robust security in this context.

5.4.5 Consumer and Prosumer Communication Based on zk-SNARK

This section describes the communication between the smart meter acting as the prover and the consensus node acting as the verifier. It is important to note that we have used the zero-knowledge proof cryptographic scheme to ensure the privacy of the user information. The background description of zk-SNARK is described in the Preliminary section. As we have explained in the previous section, the smart meter is connected to the grid operator as well as to the consensus node. It collects various data such as daily power consumption, peak hours of usage, etc. This information is very valuable for a consumer to optimize their power usage and manage their appliance accordingly. Figure 5.24 illustrates the communication process using the zero-knowledge proof, which is defined as follows.

1. Every smart meter is identified by a unique identifier M_{ID}. The smart meter collects all the data called SD_i, i.e., i^{th} collected data. It also records the current timestamp denoted as Tr_1 in the diagram.
2. The hash of all the collected data is calculated as $H(SD_i)$, where $H()$ is the SHA256 hash function. After calculating the hash, it is xor-ed with meter id M_{ID} as depicted in the figure as ($A_1 = H(SD_1) \oplus M_{ID}$). After that, all the transaction is created and signed with the private key of the meter. Finally,

before sending the information to the miner (verifier), the smart meter (prover) creates the proof π where $Tr_1, Tr_2, ...Tr_n$ are treated as information, and A is as a witness and sent to the consensus node with a timestamp as $< \pi, T_1 >$. The transactions are signed with the private key of the meter. Hence these transactions can not be forged in the transmission. Additionally, zero knowledge provides privacy.

3. The consensus node, which receives the transactions from the smart meter, first verifies the proof π and, after that, verifies the signature of signed transactions. It checks whether the received transactions are new or old by checking $\delta T > T_2 - T_1$ where T_2 is the miners' current timestamp. If the condition is true, it further verifies the transaction and creates a block. Here we have considered the transaction-receiving node as the leader node.

4. The leader node verified all the transactions using the public key of the smart meter and forwarded them to other consensus nodes. Then it calculates the Mrt using the function $merkle_tree()$, and PBG and set other parameters as DB_{ID}, Tr_2, and n. After this, the node creates the block, as depicted in the figure, and broadcasts it into the network for verification.

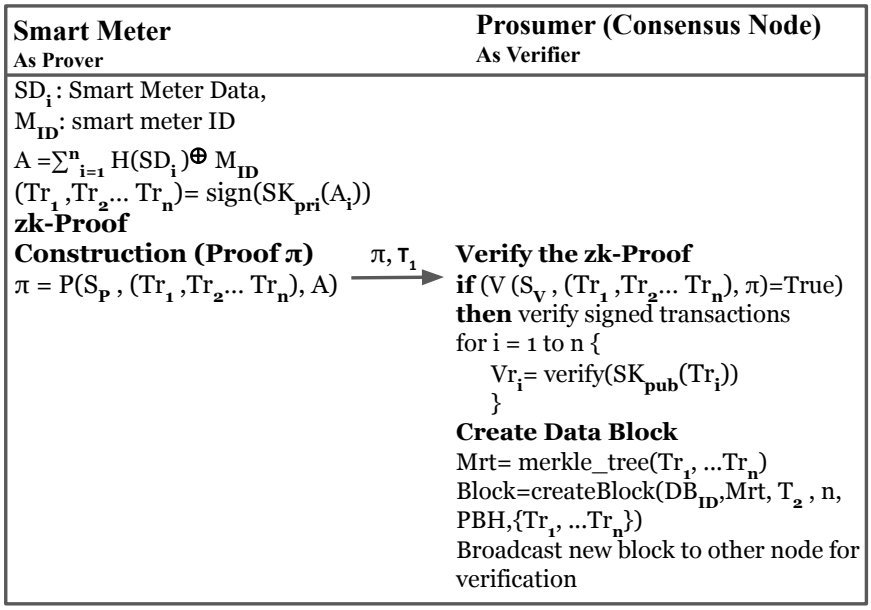


Figure 5.24: Sequence diagram for communication between the consumer (smart meter) and prosumer (consensus node) in *zk-SGB*

5.4.6 WNS-BFT: Consensus Algorithm

The proposed novel Weighted Node Selection Byzantine Fault Tolerance (WNS-BFT) algorithm is the consensus algorithm for permissioned blockchain networks like smart grid energy trading. The WNS-BFT creates the subgroup of consensus nodes out of all the consensus nodes based on the reputation computation. It uses the two important parameters to calculate the Reputation Score (*RS*) of each consensus node, which are the Participation Score (*PS*) and the Computation Power Score (*CPS*). After calculating the *RS* of each consensus node, a subgroup of consensus nodes is created, which is actually the 51% nodes whose *RS* is highest in the whole blockchain network, i.e., amongst all the consensus nodes. This selected subgroup

is responsible for the transaction verification, block generation, block verification, and block commit and append it into the longest chain of the valid blockchain ledger. After appending the newly generated block in the ledger, the *RS* of each node is revised, and the consensus process repeats for the new set of transactions. Let's understand the process in detail.

Let, we have a set of consensus nodes $CN_j = \{CN_1, CN_2, ..., CN_M\}$, where $j = 1, 2, ..., M$. Among these nodes, the CN_x, $x \in j$ is the leader node, which receives the information from the smart meter SM_i. The whole consensus process is divided into three phases, i.e., reputation score computation, fair leader selection, block generation and verification, and node reputation updation of each node.

Step-1: Reputation Score (*RS*) Calculation
To calculate the *RS*, first, we need to calculate the two parameters, i.e., PS_{CN_i} and CPS_{CN_i} of consensus node CN_i, where, $i = 1, 2, ..., M$.

- **Participation Score (PS_{CN_i})**:
  The participation Score (*PS*) represents the involvement of a node in network activities. It can be calculated based on the number of valid blocks the node submits and the frequency of block validations. The PS_{CN_i} of node CN_i expressed as follows:

$$PS_{CN_i} = \frac{Number\ of\ Valid\ Blocks\ from\ CN_i}{Total\ number\ of\ blocks\ in\ the\ chain}$$

- **Computation Power Score (CPS_{CN_i})**:
  The computational Power Score (CPS) represents the computing resources contributed by a node to the network. Similar to the PoW systems, this can be directly related to the node's hashing power. The CPS_{CN_i} of node CN_i expressed as follows:

$$CPS_{CN_i} = \frac{Hashing\ Power\ of\ CN_i}{Total\ hashing\ power\ in\ the\ network}$$

- **Normalization and Scaling**:
  Normalize both participation and computational power scores to be within the range $[0, 1]$.

- **Weighting Factors:**
  Assign weighting factors α and β to *PS* and *CPS*, respectively, to reflect their importance in determining reputation. These weighting factors should be such that $\alpha + \beta = 1$, i.e., these weighting factors control the balance between the two parameters so that only one parameter does not overly inflate the *RS* if the other factor is lower.

- **Weighted Sum Calculation:**
  Calculate the weighted sum (*WS*) of PS_{CN_i} and CPS_{CN_i} as

$$WS = \alpha \cdot PS_{CN_i} + \beta \cdot CPS_{CN_i}$$

- **Limiting Factor (*LF*) Determination:**
  Identify the parameter with the lower value between PS_{CN_i} and CPS_{CN_i} as

$$LF_{CN_i} = min(PS_{CN_i}, CPS_{CN_i})$$

- **Final Reputation Score:**
  Calculate the reputation score (RS_i) using a weighted geometric mean approach (it balances the influence of both parameters) for each node CN_i using the finalized reputation calculation method:

$$RS_{CN_i} = \sqrt[\alpha+\beta]{[WS_{CN_i}]^{\alpha} \cdot [LF_{CN_i}]^{\beta}}$$

Step-2: Fair Leader Selection
In each round of the consensus process, a leader node is selected. First, the consensus nodes (CN) whose reputation score is above or equal to the reputation threshold are selected to form the pre-selected group to make the leader selection process more secure and fair. Second, to make the process fair or prevent the leader node from being known in advance by the attacker, the nodes that satisfy the random number threshold are selected from the pre-selected group. Finally, from these pre-selected consensus nodes, a node with a high reputation score is selected as the leader node. Thus, the leader selection process is a three-step process, i.e., create a pre-selection group, random reputation threshold, and find the leader node.

1. **Create Pre-selection Group**: There are two possible cases.
   (a) **Case 1: If it is the first round of consensus** In this case, the consensus nodes CN_j with $RS_{CN_j} \geq RS$ are selected to form the pre-selection group. Here, the RS_{CN_j} is the node's initial reputation score, and RS is the reputation threshold value.
   (b) **Case 2: Otherwise** In this case, the consensus nodes CN_j with $RS_{CN_j} \geq RS$ are selected to form the pre-selection group. Here, RS_{CN_j} is the updated reputation score of the node CN_j for $j = 1, 2, ..., M$.
   (c) Let, there are $S \in M$ number of consensus nodes in the pre-selection group, i.e., $CN_s = \{CN_1, CN_2, ..., CN_s\}$, $s = 1, 2, ..., S$.
2. **Random Number Threshold**
   Consensus nodes CN_s in the $S \in M$ pre-selection group perform the following computation.
   (a) Select the random number X_s.
   (b) Sign the X_s using ECDSA as $\sigma_{CN_s} = H(X_s)^{Pr_{CN_s}}$, where, Pr_{CN_s} is the private key of CN_s.
   (c) Broadcast this into the other nodes in the pre-selection group.
3. The current leader node CN_x performs the following steps to generate the random number threshold.
   (a) Select the random number X_{CN_x} as the threshold ω.
   (b) Sign the X_{CN_x} using ECDSA as $\sigma_{CN_x} = H(X_{CN_x})^{Pr_{CN_x}}$, where Pr_{CN_x} is the private key of the leader node CN_x.
   (c) Send $< \omega, \sigma_{CN_x} >$ to each node CN_s.
4. Among the consensus nodes group S, the node with the highest reputation score RS_{CN_j} is selected as the leader node CN_x.
5. The information of CN_x is broadcasted to each consensus node group CN_j, $j = 1, 2, ..., M$.

Step-3: Block Generation and Verification:
The block verification scheme is also simple yet very effective. The newly elected leader node sends the newly created block along with the nonce value. Every verifier node again creates the block with the received nonce value and checks the received block's validity. Therefore, the verification process also does not take much time to block verification. These are the two novel things that are introduced in the WNS-BFT. The WNS-BFT uses the following consensus algorithm to create, verify, and add the block to the distributed ledger. We have considered that every consensus node has a transaction pool (TP), where the smart meters of the consumers put all the transactions. The consensus node sends the transaction to other consensus nodes after the transactions are verified. The block generation and validation Algorithm 20 works as follows:

1. The consumer (smart meter) is considered an ordinary node that is responsible for generating the transactions. The consumer is connected to the smart grid, which acts as the consensus node in the *zk-SGB*. All the generated transactions are put into the transaction pool TP. The selected leader node in the blockchain network collects the transactions from the TP and verifies them. After verification, transfer those transactions to other consensus nodes. Then, the leader node calls the function $createHashTree$ to calculate the Merkle root (Mrt) of these transactions. The previous block's hash PBH is calculated using the function $H()$, the current timestamp T, and a nonce (n) are initialized with the appropriate value.

2. Using the values of $Mrt, PBH, T, nonce$, the leader node create the block using $createBlock()$ function. The leader node sends the newly created block in the subgroup (CN_s) for verification. The digital signature is used to prevent the block from modification and impersonation attacks.

3. When the rest of the nodes (other than the leader node) get the copy of the block, first, they verify the signature of the block to check whether it was sent from the legitimate leader node or not. After validating, they also verified the correctness of the block as follows. To validate the block, the verifier nodes recreate the block from the same set of transactions, and for that, it calculates the Mrt, PBG, T, n. After recreating the block, nodes check whether the recreated block matches the received block. If the match is found, the node will set the status of $block_status$ as $Valid$, otherwise set as $Invalid$. In addition to $block_status$, the node will also set the status of leader node $node.status$ as good or bad for the future leader selection process.

4. Suppose the leader node gets at least $2N/3$ responses as $Valid$. In that case, it appends the block in the distributed ledger, i.e., a block is considered valid when half of the nodes verify the block. At least 66% nodes must verify the block to prevent any worst attack.

Step-4: Node Reputation Updation
In *Algorithm* 20, the verifier node responds to the leader node with the status of the block's validity, either 'valid' or 'invalid,' depending on the verification process of the newly proposed block. Alongside the block status, the verifier node also provides the reputation status of the leader node as 'good' or 'bad.' Specifically, if the proposed block from the leader node is deemed valid, the leader node's reputation status is set as 'good'; otherwise, it is set as 'bad.' Accordingly, the participation score (PS) is updated by incrementing the count of valid blocks whenever a newly proposed block is validated. Apart from PS, the computational power score (CPS) is also updated when a consensus node upgrades its computing resources. However, the frequency of updating CPS is significantly lower compared to PS.

Algorithm 20: Block Generation and Verification

Input: Transactions: $Tr_i, i = 1, 2, \ldots, n$
Output: Block
$TP.append(Tr)$;
if *node* = *leader* **then**
    $Mrt =$ createHashTree(TP.getAllTransactions);
    $PBH = H(PB)$;
    $BH = H(Mrt, T, PBH)$;
    $block =$ createBlock(Mrt, T, PBH, BH, n);
    sign($block$);
else
    getBlock($block, nonce$);
    $Mrt =$ createHashTree(TP.getAllTransactions);
    $block^* =$ createBlock(Mrt, T, PBH, BH, n);
    if $block = block^*$ **then**
        setBlockStatus(valid);
        $node$.status = good;
    else
        setBlockStatus(invalid);
        $node$.status = bad;
if $count(block_status = valid) \geq \frac{2N}{3}$ **then**
    $blockchain$.append($block$);

5.4.7 Performance Analysis

The evaluation of *zk-SGB*'s performance revolves around five key parameters: consensus delay, throughput, computational cost, and energy consumption. However, preceding the performance assessment, we provide a comprehensive overview of the implementation environment.

Implementation Environment

In order to bring the proposed scheme into a practical implementation, a carefully designed testbed was constructed. This testbed consisted of a total of 9 Raspberry Pi (RP) boards, with a specific distribution to emulate the roles of various entities within the system. Among these 9 boards, 2 were designated as prosumer nodes, and the remaining 6 RP boards were configured as consumer nodes. Each prosumer was associated with a group of consumers, symbolizing the interaction between residential units in the energy market. The consumer RP boards were designed to replicate the capabilities of smart meters, effectively simulating the data collection and communication functionalities required within the smart grid ecosystem. On the other hand, the prosumer RP boards were implemented as integrated units that combined smart meter capabilities as well as additional features. The choice of Raspberry Pi boards for this implementation stemmed from their computational prowess and their ability to handle the security protocols inherent to the proposed scheme. These RP boards were equipped with microcontrollers capable of processing data and memory units for storing crucial information. By closely mirroring the characteristics of a genuine smart meter, these RP boards became integral components of the testbed. Crucially, these devices were

equipped with Wi-Fi communication capabilities. Two variations of Raspberry Pi boards were employed in this testbed: one with 2GB of memory, which represented the consumer nodes, and the other with 4GB of memory, designated for the prosumer nodes. While the memory capacities differed, both types of boards shared identical software and hardware configurations, which ensured a consistent testing environment. The operating system chosen for all RP boards was Raspberry Pi OS, also known as Raspbian OS, a Debian-based distribution optimized for Raspberry Pi devices. Each RP board was equipped with a robust 1.5GHz Broadcom BCM2711 Quad-core Cortex-A72 processor, enabling efficient data processing and communication. The development of programs for this testbed was carried out using the Java programming language, leveraging its versatility and suitability for implementing the proposed scheme's functionalities.

Consensus Delay and Throughput

During the comprehensive evaluation of *zk-SGB*, various assessment was conducted, encompassing critical performance metrics such as transaction throughput, consensus latency, and the equitable selection of leader nodes. This thorough examination aimed to ascertain the scheme's operational efficiency and resilience while upholding robust security standards. Notably, the evaluation sought to optimize consensus latency and enhance overall throughput without compromising the system's inherent security provisions. The achieved results emphasize the practical viability of these optimizations within the context of the *zk-SGB*.

A particularly noteworthy achievement is the successful reduction of consensus latency and the concurrent elevation of throughput through the novel WNS-BFT consensus algorithm. This optimization was achieved while maintaining the scheme's security measures, reinforcing the notion that performance enhancements need not come at the expense of robust security. The comprehensive analysis of various parameters revealed the superior performance of *zk-SGB* compared with state-of-the-art techniques. It's important to underline that prevailing weaknesses in state-of-the-art techniques are often tied to their susceptibility to the associated high consensus delay and low throughput. These issues can undermine the operational efficiency and reliability of any blockchain-based system. In contrast, the proposed *zk-SGB* stands out as a solution that addresses these issues by implementing an efficient, fair, and robust consensus algorithm WNS-BFT that effectively mitigates the risk of forks and significantly reduces consensus delay. This strategic fortification of the system contributes to its exceptional performance in real-world scenarios. A key architectural decision to reduce propagation delays within the blockchain network involved the deliberate limitation of block size to 1 MB or less. By imposing this restriction, each block can accommodate a maximum of 516 transactions, thereby preventing excessive block sizes that could hinder efficient network propagation. Figure 5.25a clearly demonstrates the notable reduction in consensus delay achieved by *zk-SGB* compared with state-of-the-art schemes. Another significant aspect of the scheme's performance is its throughput, which is aptly showcased in Figure 5.25b. The comparative analysis shows that *zk-SGB* outperforms other blockchain systems in terms of throughput. Unlike traditional systems, where multiple nodes vie for block rewards and simultaneous block creations lead to congestion and forks, *zk-SGB* strategically selects the consensus node as the leader. This approach reduces congestion, increases throughput, and ensures a fair distribution of responsibilities among consensus nodes. In summation, the evaluation of *zk-SGB* highlights its ability to address critical issues in state-of-the-art blockchain methodologies. Through thoughtful optimizations, strategic consensus mechanisms, and innovative leader node selection, *zk-SGB* achieves remarkable gains in consensus latency, throughput, and fairness—all without compromising the high standards of security and reliability demanded by contemporary smart grid applications.

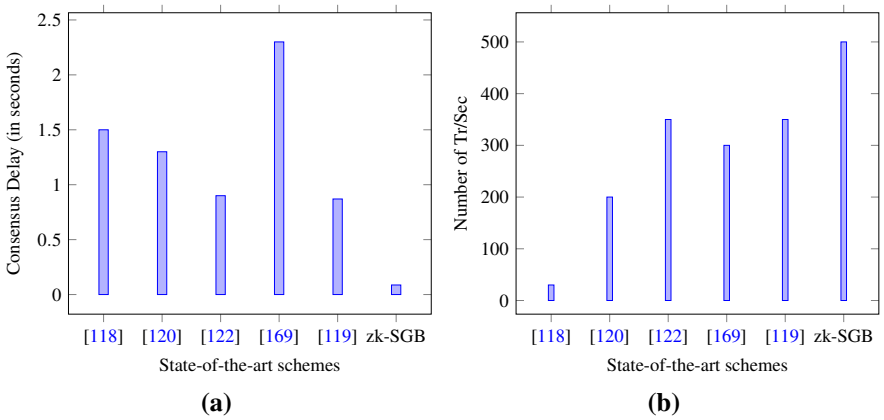


Figure 5.25: Comparative analysis of *zk-SGB* based on (a) consensus delay and (b) throughput with state-of-the-art schemes

Computational Cost and Energy Consumption

The computational cost, denoted as T_{ct} and measured in milliseconds, serves as a fundamental metric to measure the efficiency of the proposed scheme's execution. It encapsulates the time required for various essential operations within the system. These operations encompass critical functions such as the verification of transactions, communication, the creation of new blocks, and the subsequent verification of these blocks. By measuring the computation of these operations, we obtain insights into the overall efficiency of the system's execution. Table 5.13 provides time requirements associated with various cryptographic functions. These functions, including hashing, digital signature generation and verification, encryption, and decryption, are pivotal components of the proposed scheme. Understanding the time allocation for each of these functions offers a detailed perspective on the computational workload involved in the system's operations. Moreover, we evaluate the computational cost of the proposed *zk-SGB* scheme compared with existing schemes. Table 5.18 shows the comparison of the computational cost associated with executing state-of-the-art schemes. This quantitative analysis shows the relative computational efficiency and overhead of the proposed scheme.

Energy consumption is a crucial element in evaluating the practicality and sustainability of a system, especially when it comprises lightweight devices such as smart meters. The energy consumption, denoted as E_{rg}, is derived from the product of computational time (T_{ct}) and the CPU power (C_P), expressed as $E_{rg} = T_{ct} * C_P$. This equation quantifies the total energy consumed by the system during its operation. By considering the CPU power and computational cost, we compute the energy consumption of the system during the whole operation. To visually compare the proposed scheme's computational efficiency against state-of-the-art schemes. This thorough evaluation of computational cost and energy consumption shows the comprehensive perspective on the practical feasibility and efficiency of the proposed *zk-SGB* scheme.

5.4.8 Security Analysis of zk-SGB

This section delves into the security analysis of the *zk-SGB*. The assessment of its security highlights its effectiveness in countering double-spending attacks, preventing forks, ensuring privacy preservation, confidentiality, and prevention from modification. We confirm these assertions through a detailed lemma and

Table 5.18: Comparison of computation cost (CC) in milliseconds and energy consumption (EC) in Joule of *zk-SGB* on Raspberry Pi 4 (RP) 2GB and 4GB with state-of-the-art schemes

Scheme	Computation Cost Equation	CC on RP4 4GB	CC on RP4 2GB	EC on RP4 4GB	EC on RP4 2GB
[122]	$5T_{H(.)} + 3T_{sign} + 3T_{veri}$	17.43	35.97	97.61	86.33
[119]	$6T_{H(.)} + 2T_{sign} + 2T_{veri}$	11.63	24.25	65.128	58.20
[169]	$6T_{H(.)} + 3T_{sign} + 3T_{veri}$	17.44	35.55	97.60	85.32
[170]	$6T_{AES} + 2T_{sign} + 2T_{veri}$	47.36	99.78	265.22	239.48
[118]	$5T_{sign} + 5T_{veri}$	29.00	59.95	162.4	143.88
[120]	$11T_{H(.)} + 8T_{mul(.)}$	40.05	55.49	224.30	133.18
zk-SGB	$3T_{H(.)} + 1T_{sign} + 1T_{veri}$	5.82	11.89	32.56	28.54

proof that clarifies the mechanisms that safeguard the system from the threats mentioned above. The smart grid's security capabilities have been strengthened against cyberattacks through a blockchain-based security framework, as proposed by Liang et al. [118]. Their approach employed a PoW-based consensus mechanism devoid of any reward system. While mining was a competitive process, there was no reward incentive for the miner who solved the puzzle first. Although the scheme appeared resilient against double-spending attacks, it shows vulnerabilities related to forking and privacy attacks. Similarly, the Guardian scheme proposed in [119] also adopted the PoW consensus algorithm. Although it was adept at protecting against double-spending attacks, it fell short in protecting against forking and privacy attacks. Additionally, fault tolerance was not a characteristic of this scheme. A smart contract-based smart grid system was put forth in [120]. This scheme facilitated anonymous authentication; however, it remained susceptible to double-spending and lacked robust fault tolerance. Addressing the issues of double spending and forking, the scheme proposed in [122]. However, when it came to a substantial number of nodes (prosumers and consumers), it revealed its inability to maintain fault tolerance [171]. In contrast, the *zk-SGB* proposed a novel consensus algorithm called WNS-BFT. This effectively thwarts all four attacks and performs better than the state-of-the-art techniques. With the help of 3 lemmas below, we have described how the *zk-SGB* is secured against the double-spending attack and fork attack and ensured the preservation.

Lemma 22. *Let Tr_1 and Tr_2 be two conflicting transactions aiming to spend the same energy token within the zk-SGB framework. The objective is that the probability of both Tr_1 and Tr_2 being successfully validated and added to the blockchain is negligible. The zk-SGB, incorporating the WNS-BFT consensus algorithm, prevents such double-spending attacks.*

Proof. Assumption: Both transactions, Tr_1 and Tr_2, are simultaneously submitted to the *zk-SGB* network in an attempt to execute a double-spending attack. Each transaction T_i $(i = 1, 2)$ undergoes a consensus process involving N consensus nodes. To be deemed valid, a transaction requires approval from at least $2N/3$ nodes. Let p_v be the probability that a single consensus node validates transaction Tr_i. Given the requirement of $2N/3$ approvals for validity, the probability of Tr_i being successfully validated is

$$p_{\text{valid}} = \binom{N}{2N/3} \cdot p_v^{2N/3} \cdot (1 - p_v)^{1N/3}.$$

Conflicting transactions, Tr_1 and Tr_2 compete to be added to the blockchain, but only one can prevail due to their conflicting nature. The probability of a double-spending attack succeeding is the probability that both Tr_1 and Tr_2 are successfully validated and added to the blockchain. This probability can be bounded by the probability of individual validation p_{valid} and the assumption that only one transaction will prevail in the blockchain.

Let p_m be the probability that a consensus node behaves maliciously and approves both Tr_1 and Tr_2. Considering the competitive and reputation-based consensus node selection process, it is reasonable to assume that the majority of nodes will not engage in such malicious behavior. Thus, p_m is expected to be negligible. The probability of the double-spending attack succeeding is bounded by $p_{\text{attack}} = p_{\text{valid}} \cdot p_m$, which is negligible due to the negligible nature of p_m. Thus, by combining the negligible probabilities of both successful validation of conflicting transactions and malicious behavior, it can be concluded that the probability of a double-spending attack succeeding in the *zk-SGB* is negligible. However, it's important to emphasize that while this proof offers a probabilistic perspective on the prevention of double-spending attacks, the practical implications may vary due to factors beyond the scope of this simplified analysis. The actual behavior of consensus nodes, network conditions, and other considerations can influence the true probabilities in real-world scenarios. □

Lemma 23. *Let P_{fork} be the fork probability in the WNS-BFT consensus algorithm. The P_{fork} is almost negligible in the zk-SGB under the consensus algorithm WNS-BFT.*

Proof. An attacker attempts to create a fork by manipulating the leader selection process and controlling consensus nodes. The fork probability P_{fork} can be defined as:

$$P_{\text{fork}} = \frac{\textit{Number of Successful Forks}}{\textit{Total Number of Consensus Rounds}}$$

Each consensus node CN_i is assigned a reputation score R_i using a weighted geometric mean approach:

$$R_i = \sqrt[\alpha+\beta]{[WS_{CN_i}]^{\alpha} \cdot [LF_{CN_i}]^{\beta}}$$

where $WS_{CN_i} = \alpha \cdot PS_{CN_i} + \beta \cdot CPS_{CN_i}$ and $\alpha + \beta = 1$. The attacker must maintain a consistently high reputation R_i based on both PS_{CN_i} and CPS_{CN_i}. Now Let X_s be the random number chosen by consensus node CN_s and σ_{CN_s} its corresponding signature. The attacker would need to predict a threshold ω generated by the leader CN_x and broadcasted as $< \omega, \sigma_{CN_x} >$. This cryptographic unpredictability makes prediction unfeasible. When a leader CN_x is selected, its reputation status is broadcasted. The attacker's reputation status as "bad" would be known if they deviate from consensus rules, undermining their credibility. Verifier nodes collectively validate the proposed block. The attacker's malicious block proposal would be rejected by the majority, preventing a fork. Additionally, the *CPS* update frequency is low compared to *PS*, preventing rapid manipulation of CPS_{CN_i} by an attacker.

Considering reputation-based selection, random threshold, reputation broadcast, collective verification, and limited *CPS* update frequency, we conclude that the fork probability P_{fork} is almost negligible. The combination of reputation, cryptographic unpredictability, transparency, collective verification, and limited influence of *CPS* fortifies the consensus mechanism against fork attempts. □

Lemma 24. *Let $\mathbb{M}$ denote the set of smart meters and $\mathbb{V}$ denote the set of verifier nodes (consensus nodes). zk-SGB maintains privacy and security through zk-SNARK, a combination of reputation-based leader selection, cryptographic techniques, and communication protocols. Thus, zk-SGB ensures privacy preservation and protects against modification and confidentiality attacks.*

Proof. The *zk-SGB* scheme incorporates zero-knowledge proof (zk-SNARK) cryptographic techniques in the communication between smart meters and verifier nodes. These techniques ensure that sensitive user information remains private during the communication process. The steps are as follows:

For each smart meter $M_i \in \mathbb{M}$, SD_i represents the collected data, and Tr_1 is the timestamp of communication. The smart meter generates a proof π_i using zk-SNARK after converting the smart meter data into transactions Tr_i, $i = 1,2,3,...,n$: $\pi_i = P(S_P, Tr_i, T_1)$. For each verifier node $V_j \in \mathbb{V}$, the verifier receives the proof π_i, transactions, and timestamp Tr_1 from the smart meter. The verifier validates the proof using zk-SNARK verification: $verify(S_v, Tr_i, T_1, ..., \pi_i)$. The zk-SNARK cryptographic scheme guarantees that the verifier only learns the validity of transactions without gaining access to actual transaction details. This ensures user privacy while allowing the network to verify the integrity of transactions. Additionally, the *zk-SGB* is also protection against modification and confidentiality attacks. This can be expressed as follows. Transactions created by smart meters are signed using their private keys Pr_{M_i}. Let Tr_i denote the transaction signed by smart meter M_i. This prevents unauthorized modification of transactions during transmission: $Tr_i = \text{Sign}(SD_i, Pr_{M_i})$. For each verifier node $V_j \in \mathbb{V}$, the verifier checks the timestamp of received transactions to prevent replay attacks:$T_1 \leq T_{\text{current}}$. Reputation-based leader selection ensures that only nodes with high reputation scores become leaders. Let R_{M_i} denote the reputation score of smart meter M_i. The leader selection process involves selecting nodes from $\mathbb{V}$ that satisfy reputation and random number thresholds. This prevents malicious nodes from taking control of the consensus process. The combination of privacy-preserving zk-SNARK techniques, data integrity through digital signatures, timestamp verification, reputation-based leader selection, and collective verification ensures that the *zk-SGB* maintains privacy, guards against modification and confidentiality attacks and provides a robust consensus mechanism for the smart grid. □

Concluding Remark

In this Chapter, we explored the application of blockchain technology within the context of smart cities, focusing on three major domains: IoT, Intelligent Transportation Systems (ITS), and Smart Grids. For IoT, we introduced BLISS and SCAB-IOTA, which enhance data integrity and scalability in IoT networks. In the domain of ITS, LBSV and SmartCoin were developed to secure communication and enable efficient micropayments within intelligent transportation networks. For the Smart Grid, SmartGrid-NG, ETradeChain, and zk-SGB were proposed to facilitate secure energy trading and efficient grid management. Each technique was designed to address specific challenges within its respective domain, demonstrating the versatility and potential of blockchain in improving various aspects of smart city infrastructure. This chapter has illustrated the practical benefits of applying blockchain technology to real-world problems, particularly in enhancing security, efficiency, transparency, and privacy. In the final chapter of this thesis, we will summarize the key contributions made across all chapters and discuss their broader implications for the future of blockchain technology, along with the future directions.