A Vehicular Trust Blockchain Framework with Scalable Byzantine Consensus

A Vehicular Trust Blockchain Framework with Scalable Byzantine Consensus

Xiao Chen, Guoliang Xue, *Fellow, IEEE*, Ruozhou Yu, *Senior Member, IEEE*, Haiqin Wu, and Dawei Wang

Abstract—The maturing blockchain technology has gradually promoted decentralized data storage from cryptocurrencies to other applications, such as trust management, resulting in new challenges based on specific scenarios. Taking the mobile trust blockchain within a vehicular network as an example, many users require the system to process massive traffic information for accurate trust assessment, preserve data reliably, and respond quickly. While existing vehicular blockchain systems ensure immutability, transparency, and traceability, they are limited in terms of scalability, performance, and security. To address these issues, this paper proposes a novel decentralized vehicle trust management solution and a well-matched blockchain framework that provides both security and performance. The paper primarily addresses two issues: i) To provide accurate trust evaluation, the trust model adopts a decentralized and peer-review-based trust computation method secured by trusted execution environments (TEEs). ii) To ensure reliable trust management, a multi-shard blockchain framework is developed with a novel hierarchical Byzantine consensus protocol, improving efficiency and security while providing high scalability and performance. The proposed scheme combines the decentralized trust model with a multi-shard blockchain, preserving trust information through a hierarchical consensus protocol. Finally, real-world experiments are conducted by developing a testbed deployed on both local and cloud servers for performance measurements.

Index Terms—Trust management, blockchain, multi-shard consensus, Byzantine fault-tolerance, TEE.

1 Introduction

Trust management, introduced by Matt Blaze [1], aids in the automated verification of actions against security policies. It symbolically automates social decisions regarding trust, preventing services from unfairness and equivocation caused by dishonest users. In recent years, numerous trust management models have been developed for various domains, including wireless sensor networks, social networks, peer-to-peer systems, and the social Internet of Things/Vehicles [2]–[6]. Trust management plays a crucial role in vehicular networks by enabling the evaluation of message reliability from vehicles and providing a means to detect malicious vehicles. For instance, vehicles with trust levels below a predefined threshold are considered malicious and should be removed from the system. Although these schemes contribute to trust evaluation with the shared objective of finding a practical and reliable solution, many of them still rely on gathering trust parameters from a large number of nodes to compute the trustworthiness of service providers. However, some of these nodes may produce inaccurate rating results due to being unaware of the actions being evaluated. The involvement of numerous nodes in the trust evaluation process consumes substantial communication and computation resources, resulting in significant competition with other applications and performance issues, particularly in mobile scenarios such as the Internet of Vehicles (IoVs) or other Internet of Things (IoTs) applications [7], [8].

On the other hand, these schemes employ a classic trust management design where trust computation relies on centralized computing clusters. This design offers efficient calculation and fast response within the Infrastructure as a Service (IaaS) paradigm [9]. Although IaaS provides cost-effective computational resources, the lack of trust in the opaque evaluation process hampers user adoption of the results. Consequently, classic trust management continually faces a significant challenge: *C1: how to ensure an efficient and trustworthy trust evaluation process, as well as accurate and fair trust evaluation results, in a mobile scenario, such as IoVs*?

To address the aforementioned issue, researchers have recently turned their attention to blockchain technologies, which offer several key advantages, such as immutability, transparency, and traceability. BlockTDM [10] is a trust data management scheme that leverages blockchain and is built on the Hyperledger Fabric [11]. It specifically addresses fraud, dishonest data, and unauthorized use of data on IoT devices or edge terminals. Li et al. [12] utilize blockchain-based trust management to optimize the distributed k-anonymity algorithm, which prevents the leakage of real information in communication within VANETs. Kouicem et al. [13] explore trust management by considering the mobility feature of IoT devices. Their proposed BC-Trust solution, based on Tendermint [14], enables mobile nodes to accurately assess and share trust recommendations about other nodes. These recent schemes focus on preserving trust data through mutual blockchain platforms, but they overlook the key challenge *C1*. Additionally, the blockchain systems employed in these schemes still encounter scalability issues, particularly when employing Byzantine consensus in large-

- *X. Chen is with the School of Informatics, University of Edinburgh, Edinburgh, UK, EH8 9AB. Email: xiao.chen@ed.ac.uk*
- *G. Xue is with the School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ 85287-8809, USA. Email: xue@asu.edu*
- *R. Yu is with the Computer Science Department, North Carolina State University, Raleigh, NC 27606. Email: ryu5@ncsu.edu*
- *H. Wu is with the Department of Software Engineering, East China Normal University, Shanghai, 200062, China. Email: hqwu@sei.ecnu.edu.cn*
- *D. Wang is with School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, 212000, China.*

scale networks.

To enhance scalability, *sharding* has emerged as a practical solution for achieving a scale-out blockchain system, enabling simultaneous computation, storage, and processing. Building upon pioneering proposals such as RSCoin and Elastico [16], [17], sharding technology divides the blockchain network into multiple subnetworks, or *shards*. These shards are processed concurrently in parallel, enabling the simultaneous handling of numerous transactions. The capacity and throughput of the system can scale linearly with the number of shards or participating nodes. Various blockchains, such as Elastico, Chainspace, and Omniledger [17]–[19], adopt multiple shards to execute Byzantine consensus concurrently, resulting in high scalability and throughput. However, a challenge arises in sharding blockchains: the implementation of classic Byzantin fault tolerance (BFT) protocols (e.g., PBFT) on independent shards leads to quadratic message complexity, causing performance bottlenecks within each shard. Therefore, another challenge is presented: *C2: how to improve the efficiency of blockchain consensus while maintaining scalability*?

Based on the previous discussions, we put forth a novel vehicular trust blockchain named *VT-chain*. This system aims to establish a dependable trust evaluation model on a scalable blockchain.

To address the challenge *C1*, we present a novel vehicular trust evaluation (*VT-Eva*) model. This model incorporates a peer-review-based trust evaluation method and a secure computation design using Trusted Execution Environments (TEEs). In contrast to previous schemes such as those proposed in [10], [12], [13], the *VT-Eva* assigns trust-rating requests to groups of raters with similar attributes to the requested object. For instance, a real-time traffic information request is assigned to qualified raters in the same area, enabling them to assess the actual road conditions and provide accurate evaluations. As a result, this peer-review-based approach enhances trust evaluation accuracy. Moreover, since this method only involves congeneric nodes (i.e. qualified raters), it significantly reduces the message traffic and processing overhead in vehicular networks, thereby improving communication efficiency. Additionally, the trust computation process leverages built-in TEEs to ensure data confidentiality and computation integrity, mitigating the risk of malicious entities compromising the system [15]. These design elements collectively enhance the evaluation accuracy, communication efficiency, and computation security of the *VT-Eva* model, making it particularly suitable for vehicular scenarios.

To tackle challenge *C2*, the proposed scalable blockchain scheme is designed to support a trust management system in vehicular networks. This blockchain scheme ensures a trustworthy and equitable trust management process while incorporating a scalable and efficient consensus mechanism specifically tailored for vehicular networks. We introduce an innovative *HierBFT*, which is a hierarchical BFT consensus framework. In this framework, the traditional three-phase PBFT (refer to [21]) is evolved into a hierarchical structure to achieve scalable consensus. In comparison to PBFT and its variants such as HotStuff [27] and MinBFT [28], HierBFT is capable of seamlessly integrating into a multi-shard blockchain system and executing consensus services in parallel. As a result, it achieves higher scalability and increased throughput in consensus. Unlike the conventional sharding schemes described in [17], [18], HierBFT employs an aggregated signature scheme, which reduces the message complexity from the quadratic to a linear level. This reduction in message complexity contributes to enhanced consensus efficiency by requiring fewer messages. Consequently, HierBFT combines the benefits of a hierarchical multi-shard consensus framework, ensuring high scalability, with the linear message complexity, resulting in improved efficiency. The main contributions are as follows:

- *Decentralized and hardware-secured trust evaluation*: The proposed *VT-Eva* model employs a peer-review-based method to ensure dependable trust evaluation, and it utilizes TEEs for secure computation, thereby ensuring the integrity and confidentiality of the evaluation process.
- *Scalable trust blockchain with hierarchical consensus*: The proposed *VT-chain* incorporates a multi-shard blockchain scheme to guarantee trustworthy and fair trust management. Additionally, it integrates a hierarchical BFT consensus framework, which enhances scalability and efficiency in the system.
- *Practical experiments*: To validate the effectiveness of the proposed solutions, practical experiments are conducted by deploying testbeds on distributed cloud servers. The real-world testbed experiments demonstrate that VT-chain achieves significantly improved throughput as the network size, i.e., the number of consensus nodes, increases.

To the best of our knowledge, this study represents the initial exploration of decentralized mobile trust management in conjunction with TEEs and multi-shard blockchain technology, aiming to achieve trustworthy evaluation of trust and scalable consensus on the blockchain. Our findings demonstrate that the proposed VT-chain exhibits exceptional performance in TEE-based trust evaluation and multi-shard Byzantine consensus. This scheme holds significant potential for practical implementation, particularly within mobile trust management systems that prioritize both security and performance.

2 System Overview

Vehicular Trust Management Architecture. Fig. 1 depicts the architecture of the VT-chain system, which is built on a two-layer (i.e., *road-side unit (RSU) layer* and *vehicle layer*) vehicular service system. The key notations used in the design are defined in Table 1.

In the RSU layer, each RSU, first of all, plays a leading role in its dominated area. As a local leader, it receives/sends information from/to the local vehicles and other RSUs, which guarantees the message transmission when implementing trust evaluation and management models. The communication can be secured by employing related security protocols. The second role of RSUs is to execute trust computations based on raters' feedback. In addition, RSUs operate the hierarchical Byzantine consensus for updating the trust results to the blockchain. In the vehicle layer, vehicle nodes behave as either service requesters

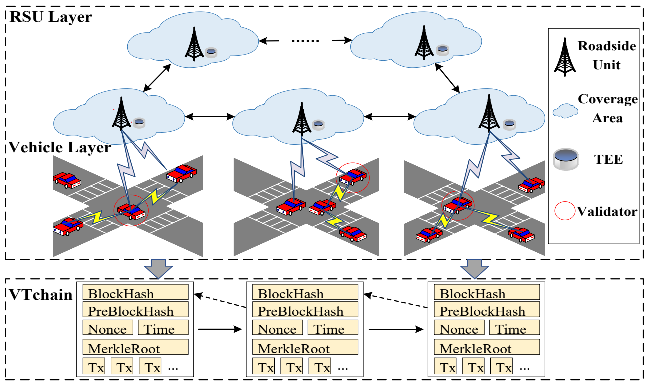

Fig. 1. The Architecture of VT-chain System.

or providers. To run the trust model, vehicle nodes can be selected to perform a rating task, i.e., as *raters*; and to run the consensus protocol, these nodes also behave as *validators*.

In vehicular trust management, trust computation involves evaluating vehicle trustworthiness and transmitted information within a network. Various sources contribute to trust computation: 1) *Direct vehicle interactions*: Trust stems from collaborations, behavior, traffic adherence, and reliable communication. 2) *Message verification and authentication*: Techniques like digital signatures, certificates, and cryptography ensure data integrity and authenticity. 3) *Trust networks and recommendations*: Vehicles establish trust networks through interactions and receive endorsements from trusted peers. 4) *Sensor data and environmental context*: Reliability and credibility of a vehicle's perception rely on accurate sensor data, including GPS, speed, collision detection, and environmental sensors. 5) *Trust authorities and history*: Trusted third-party entities, such as traffic control centers, government agencies, or service providers, offer trust evaluation services. Historical data on performance, reliability, and rule adherence are considered. Combining and weighting these sources depend on the network's objectives and trust management mechanism. Adaptation is crucial to accommodate dynamic trust information, employing algorithms that adjust values based on changing circumstances and evolving sources.

Key Distribution and Management. It is worth noting that, traditionally, vehicle registration, key distribution (or certificate issuance), update, and revocation have relied on a trusted authority (TA), such as the department of motor vehicles [34]–[36]. However, in order to address concerns regarding trust and centralization, recent years have seen the emergence of decentralized key or certificate management mechanisms based on blockchain technology [31]–[33]. For instance, Ma et al. [33] proposed a system that employs smart contracts to enable automatic public key management. In this system, the blockchain records information pertaining to user's public key registration, update, and revocation, which is submitted by an honest-but-curious authority instead of a trusted TA. Similarly, Sang et al. [32] introduced an on-chain certificate management mechanism for IoVs. Our proposed VT-chain framework is is orthogonal to public key management, and any of the existing mechanisms can be inserted into our solution. Consequently, in this paper, we exclude the TA from the system architecture depicted in Fig. 1, as well as the associated system setup, authentication, and key management procedures.

Hierarchical Consensus Framework. Based on the architecture shown in Fig. 1, the node sets (i.e., the RSU-node-set $r_k \in R$ and vehicle-node set $v_i \in V$) are divided into a number of node-shards (i.e., *shards*) including a root shard (denoted $\tilde{S}$) formed by all RSU nodes and several consensus shards (denoted S^k) in which each S^k contains an adequate number of nodes from one or more vehicle-node sets. $\tilde{S}$ connected by all S^ks forms a *hierarchical consensus* framework in which $\tilde{S}$ undertakes the signature verification and block generation. In each S^k, a Byzantine consensus protocol is executed to ensure the *local consistency* on a block (named a *micro block*). Moreover, in $\tilde{S}$, all micro blocks will be combined to a *final block* ensuring the total order, which must be agreed upon by running another round of Byzantine consensus among all members of $\tilde{S}$. Finally, all nodes in the system must verify the final block proposed by $\tilde{S}$ to reach the *global consensus* before appending it to their local chains. Overall, in the local consensus, each S^k runs concurrently to ensure the local consistency, while $\tilde{S}$ confirms the global consistency (i.e., total order) of the final block through running another consensus round. The global consensus is eventually reached among the correct nodes of all shards after they successfully commit to the final block.

Network Model. Nodes in the VT-chain system need to know the state of their neighbours, whose network layer enables nodes to exchange messages in a peer-to-peer (P2P) communication setting. The VT-chain is assumed to run on an asynchronous network that cannot provide any guarantee of message delay. In other words, messages can be arbitrarily delayed and no upper bound in time can be estimated on these deliveries due to the lack of a clock synchronisation service or the adversary over the communication channels (e.g., intended message delay or duplication). However, the VT-chain is assumed a *partially synchronous* system, in which an upper bound (i.e., a timeout Δ) on message delivery time exists. The timeout can provide termination for the consensus by triggering a view change when the timeout expires. Therefore, we consider the Byzantine fault model in which the total number of nodes is $N \geq 3f+1$, where f is the maximum number of nodes that may have faulty behaviours in the system.

Threat Model. In this study, we present a comprehensive threat model that outlines the various types of threats the proposed protocol can withstand. Our analysis encompasses both internal and external threats. Internal threats primarily arise from either faulty or malicious nodes within the system, which can be classified into two main categories: fail-stop faults and Byzantine faults. Fail-stop faults occur when nodes become unresponsive and fail to restart autonomously, resulting in a crash failure. Such failures can be attributed to factors such as power shutdowns, software or hardware errors, or denial-of-service (DoS) attacks. In contrast, nodes experiencing Byzantine faults exhibit arbitrary behavior but appear normal when compared to other nodes. Byzantine nodes may deliberately withhold messages they should have sent (i.e., omission fault caused by a routing attack) or send conflicting messages to disrupt consensus (i.e., commission fault caused by an eclipse attack, potentially leading to double spending attacks). Furthermore, multiple Byzantine nodes may collude under the control

TABLE 1
Commonly Used Notations in This Paper

Notations	**Description**
$v_i, V_{k,t}$	A vehicle node with a unique ID i in the system, and a vehicular node set at RSU k with in the time-cycle t.
$V_{rat}^{k,t}, V_{eva}^{k,t}$	A set of vehicular nodes behaving as raters, and a set of vehicular nodes behaving as evaluators.
r_k, R	An RSU with ID k, and an RSU set.
a_s, A_{v_i}	A message attribute with type s; an attribute set of vehicle node v_i.
$m_{j,i,k,t}^{s}$	The j^{th} message in attribute type s sent by vehicle i within the coverage of RSU k and time-cycle t.
$M_{k,t}$	A message set including all messages sent by vehicles within the coverage of RSU k and time-cycle t.
$e_{j,i,k,t}^{v_i}, E_{j,i,k,t}$	The v_i's rating result of the message $m_{j,i,k,t}$ in the set $E_{j,i,k,t}$; a rating result set for the message $m_{j,i,k,t}$.
$H_{i,t}$	A rating result set for all messages sent by vehicle v_i within the time-cycle t.
$l(v_i, t)$	An individual trust value of sending node v_i based on a specific rating result of it.
$L_{i,t}$	A trust value set for each evaluator, which includes a set of trust values generated on all of its rating results.
$\tau_{snd}(v_i, t), \tau_{rat}(v_i, t)$	The trust values for message senders and raters within the time-cycle t, respectively.
$\tau(v_i, t), \tau(v_i, t')$	The current and history global trust values of node v_i, respectively, within the time-cycle t.
$msgRtLt_{v_i,k,t}$	A list of rating results given by the vehicle node v_i within the coverage of RSU k and time-cycle t.
$mTrsLt_t$	A list of global rating values of all messages within the time-cycle t.
$sndTrsLt_t$	A list of final trust degrees of all message senders within the time-cycle t.
$ratTrsLt_t$	A list of final trust degrees of all message raters within the time-cycle t.
$ndGTrsLt_t$	A list of global trust degree of all nodes within the time-cycle t.
$vadLt_{k,t}$	A list of validators selected at RSU k and within time-cycle t.
$u,\ \mathcal{U},\ m,\ m^k,\ \mathcal{M}$	Client; client set; message; message accepted by shard k; universal message set.
$seq,\ d,\ k,\ K$	Sequence number; message digest; consensus shard number, total numbers of shards.
$S^k,\ \tilde{S},\ f^k$	k^{th} consensus shard ($k \in [1, K-1]$); root shard ($k = 0$); number of faulty replicas in the k^{th} shard.
$p_i^k; b_i^k; \mathcal{R}; \lvert\mathcal{R}\rvert = N$	Primary node of S^k ($i = 0$); the i^{th} backup node of S^k ($i \in [1, N^k - 1]$); replica set in system; size of replica set.
$\mathcal{R}^k;\ \lvert\mathcal{R}^k\rvert = N^k$	Replica set in the shard k; number of replicas in the shard k.
$\mathcal{R}_b^k = \mathcal{R}^k \setminus \{p_k\};\ \lvert\mathcal{R}_b^k\rvert$	Set of replicas in the shard k excluding p^k; Number of replicas in shard k excluding p^k.
$o,\ \mathcal{O},\ x,\ \mathcal{O}'$	An operation; the operations set; an operation result; the operations' results set.
$pk,\ sk;\ l,\ h$	Public key; private key; lower/upper bound of a message sequence number, respectively.
$PK,\ \sigma,\ \sigma_u,\ \sigma_p,\ \sigma_b$	Public key set; multi-signature; signature of a client; signature a primary; signature of a backup.

of one or more adversaries, thereby launching collusion attacks. Unlike the Bitcoin system that relies on proof of work, our proposed VT-chain system necessitates an access management mechanism to prevent malicious nodes from assuming multiple identities simultaneously (Sybil attack). Specifically, when a vehicle submits a message rating, other vehicle and RSU nodes may seek to ascertain the rating value and the true identity of the vehicle. Moreover, RSU nodes may maliciously tamper with the global rating result if the calculation process lacks security measures. In this research, we assume that each RSU server is equipped with TEE-enabled hardware (e.g., Intel CPUs with SGX [22]), providing secure memory and execution that are isolated from the external operating system and other applications and ensuring the correct configuration of TEEs ascertained by a *remote attestation* [23]. TEEs build secure networking through their authentication and encryption.

External threats originate from adversaries who intercept communication channels within vehicular networks with the intent to infer the true identities of legitimate vehicles and compromise rating confidentiality and integrity. These external attackers may also impersonate legal vehicles to gain access to the system and send unauthorized messages. To safeguard identity privacy, existing techniques such as pseudonym-based approaches [37], [38] or group/blind signature-based techniques [39], [40] can be employed to achieve anonymity with appropriate adjustments. Concerning illegitimate participation, we assume that vehicles undergo authentication prior to joining and interacting with other vehicles, an area of active research in the field of IoVs [37]–[39], [41]. Additionally, the system can authenticate Byzantine nodes by verifying the correct signing of transmitted messages by the message sender.

3 THE VEHICULAR TRUST EVALUATION MODEL

VT-chain contains a core component named the *vehicular trust evaluation* (VT-Eva) model using a decentralized and peer-review-based trust computation design. In this way, events/messages awaiting trust assessment will be processed by a group of raters/nodes (in the *Vehicle Layer* in Fig. 1) with similar attributes to the events so that the raters can make the best decisions on events, which improves the accuracy of trust evaluation.

On the other hand, the VT-Eva model uses trusted execution environments (TEEs) to improve the security of trust computation based on RSUs. Since RSUs may be malicious and break the data confidentiality and computation integrity (i.e., modify the trust computation results), our designed scheme uses built-in TEEs to secure the trust computation at RSUs and provides VT-Eva with enhanced security, such as $TEE.sort\_rating()$ and $TEE.cal\_rating()$ in Algorithm 2 as well as the four functions in Algorithm 3. This section presents the VT-Eva model design based on Algorithms 1-3.

According to the workflow of the VT-Eva model, the trust evaluation process starts with the message classification and evaluator selection. Next, the message quality rating on each vehicle node side (see Algorithm 1), and the global rating calculation for messages is secured by RSUs' TEEs and then executed based on Algorithm 2. The last step is to compute the trust values of nodes, including message senders and raters and eventually generate a global trust degree for each participating node based on Algorithm 3.

To maintain the global node trust, we present the VT-chain as a solution, which enhances trust management through blockchain technology. The VT-chain incorporates additional features such as immutability, transparency, and traceability, as discussed in Section 4.

Algorithm 1: Message Quality Rating (Node-side)

```
Input: All messages at each $r_k$
Output: $msgRtLt_{v_i,k,t}$
1  foreach $v_i \in V_{rat}^{k,t}$ do
2      foreach $m_{j,i,k,t}^s \in M_{k,t}$ do
3          if $a_s \in A_{v_i}$ and $v_i$ accepts rating mission then
4              Rate each $m_{j,i,k,t}^s$ with a value $e_{j,i,k,t}^{v_i} \in [0,1]$;
5              Add $\langle m_{j,i,k,t}^s, e_{j,i,k,t}^{v_i} \rangle$ in a list $msgRtLt_{v_i,k,t}$;
6      Sign $msgRtLt_{v_i,k,t}$ with $v_i$'s secret key and derive
         $Sig(msgRtLt_{v_i,k,t})$;
7      Encrypt $Sig(msgRtLt_{v_i,k,t})$ with $r_k$'s public key,
         $Encrypted(Sig(msgRtLt_{v_i,k,t}))$;
8      Return $Encrypted(Sig(msgRtLt_{v_i,k,t}))$ to $r_k$;
```

3.1 Message Classification and Rater Selection

To improve trust evaluation accuracy, the peer-review-based VT-Eva model starts by classifying messages awaiting evaluation into different groups in terms of their attributes and then continues to assign each group to a set of raters who have the same attribute. More specifically, each message (denoted $m_{j,i,k,t}^s$) indicates the j^{th} message sent by vehicle i within the coverage of RSU k and time-cycle t, in which s is the message attribute indicating a type of information. In vehicular services, s could be a type of "information" such as accidents, traffic, parking, road conditions, and cooperative awareness.

Message classification: Once a new round of trust evaluation starts, each base station collects messages during the past time segment in its coverage, so all the messages will eventually be collected by the system. Next, the base station (e.g., r_k) groups these messages $m_{j,i,k}$s into different lists ($atbLt_{n,k}$s) based on their corresponding attributes. Then, for each $atbLt_{n,k}$, the RSU r_k generates an index list that contains an attribute ID and the number of messages in $atbLt_{n,k}$. Finally, r_k broadcasts the index list to other base stations for message evaluator selection.

Message rater selection: Based on each message group's index list, a number of evaluation candidates can be selected by calculating the similarity of the attributes between the message group and the vehicle node through Jaccard similarity. The appropriate nodes are selected raters for this message group. Then, evaluation requests are sent to these raters for their acceptance of the message evaluation until a sufficient number of raters are confirmed. Once the raters are confirmed for each message group, message quality rating will be operated as defined in Algorithm 1.

3.2 Message Rating at Vehicle Nodes

Algorithm 1 defines the operations of message quality rating over the network. The message quality rating aims to assess the credibility of this information via a set of rating nodes[1] $V_{rat}^{k,t}$ grouped by each RSU r_k. Every node in the coverage of r_k can join the set to be a rater. The rating details are as follows:

Step 1: Each rater initially confirms whether it has appropriate attribute similarity with a targeted message; if yes, it can choose to accept the rating mission. Otherwise, the rater may not accept the mission, lest an inaccurate rating result is generated, which leads to trust degradation.

Step 2: Once the rater accepts the mission, it is required to provide a rating value $e_{j,i,k,t}^{v_i}$ for a given message $m_{j,i,k,t}^s$, and save the rating result pair $\langle m_{j,i,k,t}^s, e_{j,i,k,t}^{v_i} \rangle$ in a list $msgRtLt_{v_i,k,t}$.

Step 3: After completing all missions, the rater submits the list $msgRtLt_{v_i,k,t}$ to r_k at which it starts the rating. r_k then broadcasts all received lists (i.e., all $msgRtLt_{v_i,k,t}$s returned by the raters in the coverage of r_k) to other RSUs. Note that, during the submission and broadcasting process, the message rating lists are first signed by the rater for data integrity, and are then encrypted for data confidentiality.

The VT-Eva model executes trust evaluation based on time cycles, which periodically set a timestamp (denoted t) indicating the start point of a time cycle. Once a time cycle starts, the RSU will broadcast some rating tasks to the vehicle nodes in the coverage area and will continue to be in charge of these tasks until the end of the cycle. The vehicle node may move to another area with a different RSU when completing the accepted tasks. However, the vehicle node needs to return the results to the RSU from which it received the tasks. Once the current time cycle expires, vehicle nodes can accept new tasks from the nearest RSU. If nodes behave as message senders, they can perform message-sending at any time and the messages can be buffered in the RSU while waiting to be proposed as new rating tasks.

1. Assume that certain incentive mechanisms are applied to encourage vehicles to provide sufficient ratings.

Algorithm 2: Message Global Rating Calculation and Node Trust Evaluation (RSU's TEE)

```
Input: $msgRtLt_{v_i,k,t}$.
Output: $mTrsLt_{k,t}$, Transactions of node trust value: $Tx_{v_i,t}$.
1  After Algorithm 1, $r_k$ decrypts
     $Encrypted(Sig(msgRtLt_{v_i,k,t}))$;
2  $r_k$ then re-encrypts and broadcasts every $Sig(msgRtLt_{v_i,k,t})$
     with other RSUs' public keys, respectively;
3  Step1: RSU-side calculation for global rating values.
4  /*Sort every message rating list and group rating values by
   each message.*/
5  Function $TEE.sort\_rating()$ :
6      foreach $msgRtLt_{v_i,k,t}$ do
7          foreach $< m_{j,i,k,t}^s, e_{j,i,k,t}^{v_i} > \in msgRtLt_{v_i,k,t}$ do
8              if $m_{j,i,k,t}^s \in M_{k,t}$ then
9                  Add $e_{j,i,k,t}^{v_i}$ to its associated $E_{j,i,k,t}$;
10     Return all $E_{j,i,k,t}$s;
11 /*Calculate the global rating value of each message based on a
   set of associated rating values.*/
12 Function $TEE.cal\_rating()$ :
13     foreach $E_{j,i,k,t}$ do
14         Calculate the global rating value, $\phi(m_{j,i,k,t})$, of
             message $m_{j,i,k,t}$ through the Baysian inference ;
15         Add $< m_{j,i,k,t}^s, \phi(m_{j,i,k,t}) >$ in $mTrsLt_{k,t}$ ;
16     Save $mTrsLt_{k,t}$ in the trusted storage at $r_k$;
17 Step2: Node trust evaluation and transaction generation.
18 Observe all $msgRtLt_{v_i,k,t}$s and $mTrsLt_{k,t}$s;
19 while an unevaluated node trust $\tau(v_i,t)$ still exists based on the
     observation do
20     /* RSU needs to calculate the global trust value of each
       node by Algorithm 3 and broadcasts each value as a
       transaction over the network.*/
21     Calculate $\tau(v_i,t)$ by invoking Node Trust Evaluation
         (Algorithm 3);
22     Broadcast $\tau(v_i,t)$ as a transaction $Tx_{v_i,t}$ over network;
```

3.3 Global Rating Calculation at RSUs

Algorithm 2 calculates the global rating value of messages based on the outputs of Algorithm 1, and then computes the global trust value for each participating node, which form a *transaction* for the trust blockchain. The operations of Algorithm 2 are executed by built-in TEEs of RSUs that include the following two core steps:

Step 1 calculates the global rating value at a TEE-based RSU. The main operations are as follows: 1) All messages need to be aggregated by each RSU r_k at which these messages are initially proposed for rating. 2) The r_k verifies $Sig(msgRtLt_{v_i,k,t})$ with v_i's public key to ensure data integrity. Then, it classifies all rating results $e^{v_i}_{j,i,k,t}$s for each message $m^s_{j,i,k,t}$ and adds the rating results into a corresponding set $E_{j,i,k,t}$. Thereafter, r_k obtains all aggregated rating results stored in several $E_{j,i,k,t}$s. 3) Based on each $E_{j,i,k,t}$, r_k calculates the global rating value, $\phi(m_{j,i,k,t})$, for message $m_{j,i,k,t}$ using the Baysian inference [24], and then adds the computed $\langle m^s_{j,i,k,t}, \phi(m_{j,i,k,t})\rangle$ into $mTrsLt_{k,t}$.

Step 2 evaluates the global trust for each node by invoking Algorithm 3 and then forms a transaction containing the latest trust value of a node.

This step eventually returns a global rating result for each rating request and updates a computed global node trust by generating a transaction that will be accepted by a consensus shard for the blockchain update (see Section 4).

3.4 Node Trust Evaluation at RSUs

Algorithm 3, namely the *node trust evaluation algorithm*, specifies a collection of operations to quantify the trust degree of each node based on their behaviors. According to the design paradigm, the trust values of message senders and raters are calculated respectively, and then the global trust value of a node is obtained by taking the weighted average of its trust value behaving as a sender, trust value as a rater, and historical trust value. The detailed operations are as follows:

Step 1: *Trust evaluation for message senders* is conducted according to all message rating results of each sender. The algorithm first sorts all messages as well as their rating values (i.e., $\langle m_{j,i,k,t}, \phi(m_{j,i,k,t})\rangle$) in terms of every sender v_i, and then groups each sender's messages and their corresponding rating values in a set $H_{i,t}$. Thereafter, the overall trust value (i.e., $\tau_{snd}(v_i,t)$) of each sender is calculated on the basis of message rating values stored in the associated set $H_{i,t}$ through a Bayesian inference, and each $\tau_{snd}(v_i,t)$ can be saved in a list $sndTrsLt_t$.

Step 2: The *trust evaluation for message raters* is implemented firstly by calculating the relative difference and absolute difference between each message rating result $e_{j,i,k,t}$ generated from an evaluator and the overall rating value $\phi(m_{j,i,k,t})$ of the calculated message. Then, the algorithm determines the trust degree of a rater by comparing it to a set of trust-degree thresholds (i.e., δ and δ') and yields a degree-value $l(v_i,t)$ for the sender with respect to a given message. Finally, the overall trust value $\tau_{rat}(v_i,t)$ of a sending node v_i is obtained by the Bayesian inference based on all $l(v_i,t)$s (saved in $L_{i,t}$) of the sending node.

Step 3: *Global trust evaluation* is conducted by calculating the weighted average of the sender trust value $\tau_{snd}(v_i,t)$, evaluator trust value $\tau_{rat}(v_i,t)$, and history trust value $\tau'(v_i,t')$, since a node in the system can act as the message sender, evaluator, or both. All obtained global trust values i.e., $\tau(v_i,t)$s are temporally preserved in the list $ndGTrsLt_t$.

In this algorithm, the trust degree of the node is bounded in $[0,1]$. Such a degree-value is initialized with zero for each node, which means that nodes must behave as evaluators to gain enough trust credits to participate in blockchain

Algorithm 3: Node Trust Evaluation (RSUs' TEE)

```
Input: $msgRtLt_{v_i,k,t}$s; $mTrsLt_{k,t}$s.
Output: $ndGTrsLt_k$.
Step 1: Trust evaluation for message senders.
Function TEE.sort_sendMsg()
    Generate a set $H_{i,t}$ for each sending node.
    foreach $mTrsLt_{k,t}$ do
        foreach $\langle m_{j,i,k,t}, \phi(m_{j,i,k,t})\rangle$ do
            if $m_{j,i,k,t}$ is sent by $v_i$ then
                Add $\langle m_{j,i,k,t}, \phi(m_{j,i,k,t})\rangle$ to $H_{i,t}$;
    Return all $H_{i,t}$s;
Function TEE.cal_sendTrus()
    /*Calculate overall trust value of each sender with
      $H_{i,t}$s.*/
    foreach $H_{i,t}$ do
        Calculate the overall trust value of the corresponding
          sender through Bayesian inference, i.e.,
          $\tau_{snd}(v_i,t) = p(v_i|H_{i,t})$;
        Add current $v_i$ and its $\tau_{snd}(v_i,t)$ in $sndTrsLt_t$;
    Save $sndTrsLt_t$ in the trusted storage at $r_k$;
Step 2: Trust evaluation for message raters.
Function TEE.sort_rateMsg()
    Generate a set $L_{i,t}$ for each rating node;
    foreach $msgRtLt_{v_i,k,t}$ do
        foreach $e^{v_i}_{j,i,k,t}$ do
            if $\frac{|e_{j,i,k,t} - \phi(m_{j,i,k,t})|}{\phi(m_{j,i,k,t})} > \delta_0$ and
              $|e_{j,i,k,t} - \phi(m_{j,i,k,t})| > \delta_0'$ then
                $l(v_i,t) \in [0, 0.3]$
            else if $\delta_0 \geq \frac{|e_{j,i,k,t} - \phi(m_{j,i,k,t})|}{\phi(m_{j,i,k,t})} > \delta_1$ and
              $\delta_0' \geq |e_{j,i,k} - \phi(m_{j,i,k,t})| > \delta_1'$ then
                $l(v_i,t) \in (0.3, 0.7]$
            else
                $l(v_i,t) \in (0.7, 1.0]$
            Add the $l(v_i,t)$ to its corresponding $L_{i,t}$ ;
    Return all $L_{i,t}$s;
Function TEE.cal_rateTrus()
    foreach $L_{i,t}$ do
        Calculate the overall trust value of the corresponding
          rater through Bayesian inference, i.e.,
          $\tau_{rat}(v_i,t) = p(v_i|L_{i,t})$ ;
        Add current $v_i$ and its $\tau_{rat}(v_i,t)$ in $ratTrsLt_t$ ;
    Save $ratTrsLt_t$ in the trusted storage at $r_k$;
Step 3: Global trust evaluation.
Function TEE.cal_GlobTrus()
    foreach $v_i$ do
        Sort $sndTrsLt_k$ for the $v_i$'s trust value $\tau_{snd}(v_i,t)$ as a
          sender; if not exist, set $\tau_{snd}(v_i,t) = 0$ ;
        Sort $ratTrsLt_k$ for the $v_i$'s trust value $\tau_{rat}(v_i,t)$ as a
          rater; if not exist, set $\tau_{rat}(v_i,t) = 0$ ;
        Sort $ndGTrsLt_k$ for the $v_i$'s history global trust value
          $\tau'(v_i,t')$, if not exist, set $\tau'(v_i,t') = 0$ ;
        Calculate the current global trust value:
          $\tau(v_i,t) = \omega_{snd} \cdot \tau_{snd}(v_i,t) + \omega_{rat} \cdot \tau_{rat}(v_i,t) +$
          $\omega_{his} \cdot \tau'(v_i,t'), \quad \omega_{snd} + \omega_{rat} + \omega_{his} = 1$;
        Update $ndGTrsLt_t$ with the current $\tau(v_i,t)$ ;
    Save $ndGTrsLt_t$ in the trusted storage at $r_k$;
```

Algorithm 4: Verifier Selection (RSU-side)

```
Input: $ndGTrsLt_t$;
Output: $vadLt_{k,t}$;
1 /*Each RSU randomly selects validators from $v_i$ candidates
  who have appropriate $\tau'(v_i) \geq \delta_\tau$. /
2 foreach $v_i$ do
3     if $v_i \in V_k$ and $\tau'(v_i) \geq \delta_\tau$ and $v_i$ agrees to be a validator
      then
4         Add $v_i$ in validator list $vadLt_{k,t}$;
5         Notify $v_i$ the permission of validating;
6 Save $vadLt_{k,t}$ in the trusted storage at $r_k$;
```

consensus behaving as validators. The rating results and the computed global trust values will be temporally stored in the TEEs by RSUs until the blockchain is updated after executing the consensus.

In conclusion, the VT-Eva model has a decentralized structure that uses a peer-review-based and TEE-secured trust evaluation scheme, which provides increased accuracy and reliability compared to the traditional trust computation methods. On the other hand, in case of data corruption caused by possible device failure, hacker intrusion, or even malicious data holders, the VT-Eva model is integrated into blockchain-based trust storage in which all trust records are maintained by all participants rather than a centralized TTP (i.e., a trusted third party). To further overcome the low scalability issue in existing blockchains, in Section 4, we will introduce a decentralized vehicular trust management blockchain built with blockchain-sharding technologies and a novel hierarchical BFT consensus protocol.

Complexity. In this paper, we focus on studying the computation/time complexity at an RSU during a time cycle. Recall that in Algorithm 1, each rater v_i evaluates each message $m^s_{j,i,k,t}$ in the message set $M_{k,t}$. Therefore, the time complexity on each rater side depends on the number of messages $|M_{k,t}|$. Let the number of raters be $|V^{k,t}_{rat}|$, the time complexity of Algorithm 1 is $O(|V^{k,t}_{rat}| \cdot |M_{k,t}|)$. In Step 1 of Algorithm 2, there are $|V^{k,t}_{rat}| \cdot |M_{k,t}|$ rating results classified into $E_{j,i,k,t}$, and then for each $E_{j,i,k,t}$, a global rating value is calculated through Bayesian inference. Therefore, the time complexity is $O(|V^{k,t}_{rat}| \cdot |M_{k,t}|)$. In Algorithm 3, the time complexity in Step 1 depends on the number of messages $|M_{k,t}|$ and the total number of vehicles, $|V_{k,t}|$, in the coverage area of RSU (i.e., $O(|M_{k,t}|) + |V_{k,t}|)$). The time complexity in Steps 2 and 3 is $O(|V^{k,t}_{rat}| \cdot |M_{k,t}|)$ and $O(|V_{k,t}|)$, respectively. Therefore, the overall time complexity of the algorithm is $O(|V^{k,t}_{rat}| \cdot |M_{k,t}| + |V_{k,t}|)$. In Algorithm 4, the computation complexity is both $O(|V_{k,t}|)$. As analyses, the overall time complexity of our scheme is $O(|V^{k,t}_{rat}| \cdot |M_{k,t}| + |V_{k,t}|)$.

4 Vehicular Trust Blockchain

This section presents a vehicular trust blockchain (i.e., VT-chain) to preserve nodes' trust degree values computed in Algorithm 3. When an RSU node completes the global trust for a node, it generates a transaction, denoted $m^k = \langle v_i, \tau \rangle_{r_k}$ containing the node information v_i and the computed trust result τ, and then broadcasts all transactions in the current time cycle to a consensus shard (i.e., S^k)[2]. At the end of the time cycle (also known as the *block interval* in blockchains), the primary node will propose all transactions and start the consensus in the shard. In our design, the VT-chain adopts a multi-shard blockchain system and a novel hierarchical BFT (i.e., HierBFT) consensus framework for scalable consensus and efficient trust management.

2. Assume that all transactions generated by the same RUS in a time cycle must be sent to the same consensus shard.

4.1 Multi-shard Architecture

The multi-shard blockchain framework has been introduced in Section 2. With this framework, the VT-chain can concurrently run Byzantine consensus based on several local shards, i.e., consensus shards denoted S^ks. According to Algorithm 4, the RSU decides a set of candidate validators (i.e., $vadLt_{k,t}$) from all connected vehicle nodes on the condition that the trust degrees of these nodes are greater than a lower bound (i.e., δ_τ) and the nodes agree to be validators. Each S^k contains adequate vehicle nodes (i.e., $N^k \geq 3f^k + 1$, where f^k is the number of faulty nodes in S^k) that are randomly selected based on a δ_τ. The selected nodes in each S^k are known as replicas, including a *primary* p^k selected with a policy (i.e., $p^k = v_i \mod n_i$ similar to PBFT [21]), and the rest nodes named for *backups* are denoted as b^ks and connect to p^k. The replicas communicate with each other through a peer-to-peer network (see Section 2). In addition, all RSU nodes join together and form the root shard denoted $\tilde{S}$ connected by all S^ks, which performs signature verification and block generation across the system.

$\tilde{S}$ and all S^ks build the multi-shard framework that provides the blockchain scalability and also supports the hierarchical consensus operations. With the framework, the primary of a S^k periodically combines all transactions received from the same RSU and proposes a micro block for the following consensus. All replicas of the S^k execute a hierarchical BFT protocol to reach a consensus on the micro block among $2f^k$+1 local replicas. After that, all micro blocks proposed by different S^ks are sent to $\tilde{S}$, in which all micro blocks are combined into a final block with a specified total order. All replicas of $\tilde{S}$ then run another round of consensus and ensure that $2f^k + 1$ replicas of $\tilde{S}$ agree on the same final block. Lastly, the final block must be sent to all S^ks, where each local replica validates the final block before appending it to the local chain. Section 4.2 presents the detailed hierarchical BFT protocol.

4.2 The Hierarchical BFT Consensus Framework

With the global node trust degrees saved in the list $ndGTrsLt_t$, this section elaborates on how to efficiently record these degree values on the blockchain by running the proposed HierBFT consensus protocol. The protocol extends the classic PBFT protocol to a hierarchical BFT paradigm for scalable consensus and high throughput. Moreover, the protocol improves consensus efficiency by using aggregated signatures to minimise message complexity from $O(n^2)$ to $O(n)$ [25].

Normal Case Operations. Similar to the classic three-phase BFT protocol (e.g., PBFT [21]), the normal case operations include three classic phases: *pre-prepare*, *prepare* and *commit* phases. Based on our multi-shard framework, these three phases can be concurrently executed in different shards to ensure consensus scalability and throughput. Furthermore, the normal case operations also ensure that at least $2f+1$ replicas eventually commit to the final block. For the sake of simplicity, we will describe the detailed normal case operations based on $\tilde{S}$ and any S^k, as others are implemented in the same way. The protocol details are as follows:

Local Pre-prepare. When the node trust transactions (denoted Txs) are broadcast to a S^k, the primary p^k combines these latest Txs into a micro block (denoted Bk). If all Txs in the Bk have not been processed, p^k proposes a consensus request m for the micro block in the from: $m = \langle REQ, Bk\rangle_{\sigma_p}$ where REQ indicates the current state of the request and σ_p is p^k's signature. Then p^k orders m by assigning a unique sequence number in the system, i.e., $seq = (k, v, c)$, where k is the consensus shard id, and v and c are the current view number and counter value at the kth shard, respectively. Finally, p^k generates a *pre-prepare* message $\langle PREP, seq, m, d\rangle_{\sigma_p}$ (where $d = D(m)$) for the ordered m and multicasts this message to all backups of S^k.

Local Prepare. Upon the receipt of $\langle PREP, seq, m, d\rangle_{\sigma_p}$, each backup b^k accepts the pre-prepare message if the message includes a valid signature σ_p and a correct digest d, and the b^k's log does not contain another pre-prepare message with $seq' = seq$ but $d' \neq d$, which indicates an incorrect order caused by a faulty p^k. Thereafter, b^k preserves Bk retrieved from m after verifying m's signature σ_p, and then validates all Txs included in Bk via signature verification [3]. If all Txs are valid, b^k responds to p^k by sending a *prepare* message: $\langle PRE, seq, d, pk_b\rangle_{\sigma_b}$ where pk_b is b^k's public key.

The p^k waits and receives prepare messages from b^ks and builds a record bitmap (denoted ϕ_{pre}) for each valid receipt. The ϕ_{pre} proves that at least $2f^k+1$ replicas have voted their prepare messages. After that, p^k aggregates all public keys contained in the received prepare messages into a single fixed-length signature σ_{pre} and generates an aggregated prepare message: $\langle PRE, seq, d, \phi_{pre}, \sigma_{pre}, t\rangle_{\sigma_p}$, where t is a timestamp.

Local Commit. In this phase, all replicas of S^k are required to validate the aggregated prepare message by verifying ϕ_{pre} and σ_{pre}. If the prepare message is valid, all correct replicas send p^k their commit messages to generate an aggregated *commit* message: $\langle CMT, seq, m, d, \phi_{cmt}, \sigma_{cmt}\rangle_{\sigma_p}$. Then p^k sends the aggregated commit message to $\tilde{S}$ for the final block generation. In this process, all correct replicas of S^k also complete their local consensus and reach an agreement on their local micro block Bk.

Total Ordering. In the root shard $\tilde{S}$, the primary of $\tilde{S}$, first of all, uses the received commit messages to build a final block (denoted BK) in which all micro blocks Bks can be totally ordered based on their sequence numbers and timestamps. Second, the primary starts another round of consensus (the same to that in a local consensus shard) to reach an agreement on a valid BK. During the root-shard consensus, all correct replicas need to 1) verify the aggregated signature contained in each commit message using ϕ_{cmt} and σ_{cmt}, which checks whether at least $2f_k+1$ replicas have voted for the corresponding Bk in the local commit phase; 2) validate every transaction in each Bk to ensure that no two different transactions (i.e., $seq' \neq seq$) have the same content $d' = d$ (i.e., double spending); and 3) validate the total order of all Bks. If the above validations are successful, all correct replicas of $\tilde{S}$ eventually agree on the same valid BK. Finally, BK will be sent to all S^ks so that it can be validated by replicas of each S^k before appending it to the blockchain.

Global Consensus. In the consensus process, the root shard $\tilde{S}$ supports the hierarchical consensus by allowing several local S^ks to execute concurrently. This design enables a scalable consensus to ensure global consistency among multiple shards. Each S^k submits a group of locally ordered transactions to the $\tilde{S}$, in which all local transaction groups (i.e., micro blocks) are combined into a block BK containing all transactions grouped by different micro blocks and specified with a total order. The *global consensus* is a round of Byzantine consensus executed in the $\tilde{S}$ to validate and agree on the totally ordered block BK, which ensures the total order consistency of BK.

The HierBFT protocol incorporates a view change protocol to handle situations where primary peers encounter failures, drawing inspiration from PBFT's view change mechanism [21]. This view change protocol is designed to be executed concurrently and independently across all shards in parallel. For the specific view-change operations involved, please refer to Appendix A.

4.3 Cross-shard Transactions and Consensus

The hierarchical BFT protocol achieves global consensus over several parallel shards by providing a consistent total order of all valid transactions, including value transfers within a local shard and across different shards. The cross-shard transaction can be formed from multi-inputs to multi-outputs based on nodes located in different shards. When a transaction is generated from a consensus shard S^k, the proposed consensus request m must include the latest state of input in the transaction operation. If two conflicting transactions, including the same input but for different outputs (e.g., a double-spending attack), are based on the same shard, they can be detected by correct replicas of this shard in the local prepare phase. However, according to the threat model in Section 2, if a malicious client sends two conflicting transactions referring to different shards, these conflicting transactions cannot be directly validated within a local shard. In this case, the hierarchical BFT protocol needs to work with the widely used two-phase commit (2PC) protocol [26] to handle cross-shard transactions.

In the 2PC protocol, there is a coordinator who is responsible for collecting the *availability certificate* of inputs and transmitting them to the related participating shard. The availability certificate is used to prove the currently available inputs based on all outputs that have been proposed. Taking the client-driven 2PC protocol as an example [19], when the client holds two cross-shard transactions (i.e., Tx_1 and Tx_2) from Input Shard 1 to Output Shards 2 and 3, it first sends a *prepare* request to Shard 1 to check whether the input is available. In this *prepare* step, replicas of Shard 1 must locally run Byzantine consensus to generate an availability certificate for each of two transactions. After that, in the commit step, the client sends two transactions and their availability certificates to Output Shards 2 and 3,

3. Since each Tx is generated and signed by the TEE component of the RSU, the valid signature indicates the integrity of the Tx.

respectively, as well as Input Shard 1. Then, Tx_1 and Tx_2 will be proposed in Shards 2 and 3, respectively. Following the hierarchical BFT protocol, Tx_1 and Tx_2 will be totally ordered in a final block and committed by all correct replicas in the system. In this process, correct replicas can detect the conflicting transaction based on the availability certificate and identify a Byzantine fault. The validation for cross-shard transactions can be executed in $\tilde{S}$ before generating a total order and executed in S^k before committing the final block (see Section 4.2). Therefore, a global consensus can be reached by the correct replicas of all shards if there is a successful validation for the final block. Compared to the output shards, replicas of the Input Shard 1 only commit to the final block including these two transactions if they are validated, and then update their local chains.

4.4 Message Complexity

The hierarchical BFT protocol improves message-exchange by using multisignatures [25], which reduces the message complexity from $O(N^2)$ to $O(1)$ in the local shard and $O(N)$ in the system compared to the classic BFT algorithm (e.g., $O(N^2)$ at PBFT [21]) and even some state-of-the-art algorithms (e.g., $O(N)$ at HotStuff [27]). To compute the message complexity, we use N_k to represent the average number of replicas in the consensus shard S^k. According to the consensus protocol, the *local pre-prepare* phase generates $N_k - 1$ message multicasts, and the following *local prepare* phase also includes $N_k - 1$ voting message. Moreover, the *local commit* phase comprises a message round trip, which is equal to $2(N_k - 1)$ passing messages. Thus, each local S^k requires $4(N_k - 1)$ exchanged messages in total for each consensus round. In addition, since the hierarchical BFT protocol implements cross-shard consensus, all local S^ks reply on the root shard to achieve global consensus in which there is another round of normal case operations for the final block. Therefore, the total message steps in the root shard equals $5(N_k - 1)$. Therefore, the local shard complexity is $O(4N_k - 4) \approx O(N_k)$, and the root shard complexity is $O(5N_k - 5) \approx O(N_k)$. Since every shard usually has a fixed number of nodes (i.e., N_k is a constant denoted c), the shard-level complexity can be considered $O(c) \approx O(1)$.

The total number of message exchanges based on all shards in the system can be calculated by $N_m = 5(N_k - 1) + \sum_{i=1}^{i=K-1} 4(N_k - 1)$ where $K - 1$ is the number of consensus shards. Assuming that each shard has roughly the same number of nodes, the overall system-level complexity approximates to $O(4N_k \cdot K) \approx O(4N)$, where N is the total number of nodes in the system. Hence, the message complexity in the system is confined to a linear level (i.e., $O(4n)$), and the complexity in every shard can be minimized to a constant level (i.e., $O(1)$).

5 Security Analysis

5.1 VT-Eva Security

The proposed VT-Eva model operates with several RSUs and a large number of vehicular nodes, which might be challenged by various adversaries, such as compromised vehicle/RSU nodes and malicious attackers. For this reason, we consider the security design for the VT-Eva model based on the following aspects: 1) the security of temporary data (e.g., message rating lists), 2) malicious nodes and their behaviours (e.g., tampered/biased ratings), and 3) compromised RSUs, which have been analysed in detail (see Theorems 1-3 in Appendix B).

Theorem 1. *The message rating lists are well protected from external attackers and other raters. Attackers cannot tamper with the lists without being detected.*

Theorem 2. *Malicious nodes cannot change the truth of an event by sending fake messages. Moreover, the message global rating and node trust calculations are considered correct even if some malicious vehicles report unfair ratings. Nobody can modify the on-chain node trust .*

Theorem 3. *The node trust evaluation is secure even if there are compromised/malicious RSUs. The security of VT-chain can be guaranteed even if all RSUs are malicious .*

5.2 HierBFT Correctness

The hierarchical BFT consensus protocol is designed for *state machine replication* that tolerates Byzantine faults, and it can be applied to any deterministic replicated *service* with a *state* and a set of *operations*. The correctness of the protocol is provided by specifying and proving *safety* and *liveness*.

Safety means that the replicated service satisfies linearisability [21]. More specifically, an algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that they commit locally. The proposed protocol is proved to be safe when it satisfies Theorem 4, referring to Appendix C.

Theorem 4 (Safety). *Any correct replica holds a sequence of requests in the total order* (m, m')*, then all other correct replicas in the system hold them in the same total order.*

Furthermore, the protocol can provide consensus implementation in an asynchronous system when the *liveness* is guaranteed, i.e., clients eventually receive replies to their requests, provided that at most $\lfloor \frac{|\mathcal{R}|-1}{3} \rfloor$ replicas are faulty and the delay of reply to the message request at the time point does not go beyond an indefinite delay-threshold (see Theorem 5) [21].

Theorem 5. ***Liveness****. All correct replicas will eventually reach a decision for any ordered request, assuming that at most* f^k *are faulty in* S^k *replicas and there is partial synchrony.*

6 Performance Evaluation

We implemented testbeds for the VT-chain system, including the *vehicular trust evaluation model* and *hierarchical BFT consensus mechanism*, which have been deployed on both local servers and cloud virtual machines for performance measurements. The vehicular trust evaluation model is developed in C language, which implements the core trust computation and message relay operations based on servers equipped with TEEs. We use Intel Software Guard Extensions (SGX) for TEEs. The Intel SGX helps protect data in use via application isolation technology. By protecting selected code and data from modification, developers can partition their application into hardened enclaves or trusted execution modules to increase application security [22], [23].

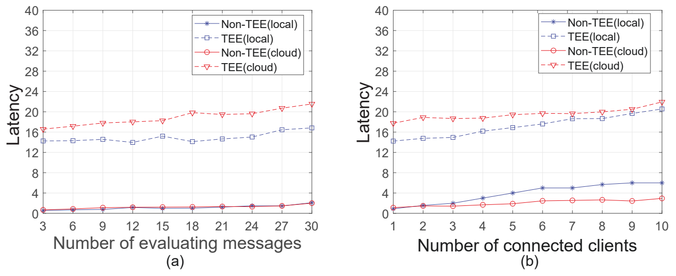

Fig. 2. Time of Message Rating Calculation in the Single-server Setting.

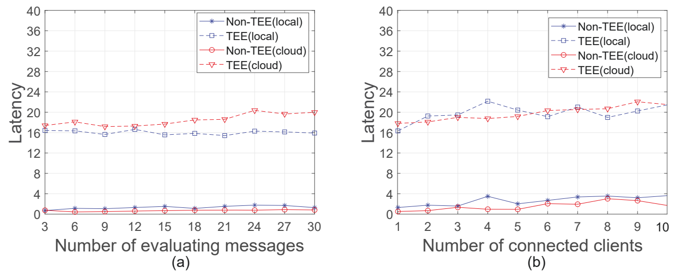

Fig. 3. Time of Message Rating Calculation in the Multi-server Setting.

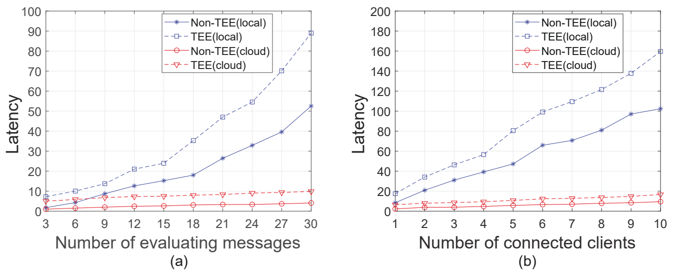

Fig. 4. Time of Node Trust Calculation in the Single-server Setting.

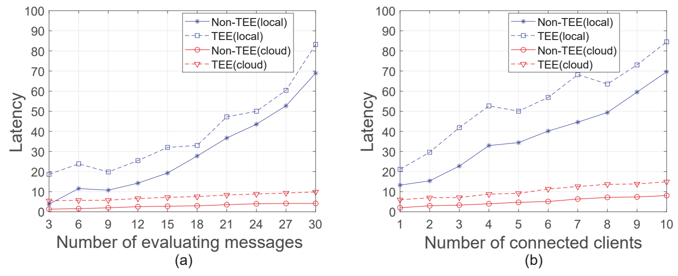

Fig. 5. Time of Node Trust Calculation in the Multi-server Setting.

6.1 Analysis of TEE-secured Calculation

To explore the performance of TEE-secured execution, the experiments observe the latency and throughput by varying the numbers of messages and clients on both local and cloud servers. In the experiments, we use a local server (with Intel 4-core i7 CPU, 16G RAM, and Ubuntu OS) and a remote cloud server (Google Cloud VM with 4-core vCPU, 16G RAM, and Ubuntu OS) to deploy several RSU and vehicle nodes, which simulates a vehicular network. In the experiments, we set the average payload of trust messages to 1 KB, respectively.

Latency of Message Rating Operations. This part analyses the average latency of message rating operations based on a single-server case (i.e., a local or remote server) and a multi-server case (i.e., joint local and remote servers); see Fig. 2 and Fig. 3, respectively. Fig. 2 shows that the TEE-enabled model needs increased time to complete the same message rating operations when using a single server. However, the time increment remains less than 20 μs whenever there is an increase in the number of request messages (from 3 to 30) or clients (from 1 to 10); see Fig. 2(a) and Fig. 2(b), respectively. Moreover, Fig. 3 depicts a similar trend, which indicates that the time increment of using TEEs is no more than 20 μs when jointly using multi-servers. Figs. 2 and 3 prove the fact that TEE consumes more time on encryption and decryption operations to provide secured hardware isolation. Fortunately, according to the results, such a minor and stable increment will not have a serious impact on the overall system performance.

Latency of Node-trust Computation. This part analyses the average latency of node-trust computations based on a single-server case (i.e., a local or remote server) and a multi-server case (i.e., joint local and remote servers); see Fig. 4 and Fig. 5, respectively. As shown in Fig. 4, the latency of node-trust computation in the local server has a much faster increase, while the latency in the remote server remains stable with a growing number of messages or clients. This is because the remote server obtains the optimised context switch from the cloud platform in contrast to local servers. Fig.5 presents the latency measured from the execution under multiple local and remote servers and shows a trend similar to that in Fig. 4. In summary, the node-trust computation can be executed more stably in remote cloud servers.

6.2 Performance of VT-chain System

To evaluate the VT-chain system, we built a blockchain testbed using C language, which implements all consensus operations since clients (i.e., RSU nodes) broadcast requests (i.e., transactions) to consensus shards. Nodes in the testbed are connected by TCP/IP connections and deployed on multiple remote servers. The testbed implements the VT-Eva model and our proposed HierBFT consensus framework in which each node behaves as one of four roles (i.e., the consensus primary node, consensus backup node, verification primary node, or verification backup node). To achieve a maximally realistic experimental environment, we deploy the VT-chain system based on 20 Google cloud virtual machines (VMs), each of which has two cores and 8G RAM. Our experiments use 4 VM servers to deploy RSU nodes (i.e., *RSU Servers*) and 16 VM servers to deploy vehicle nodes at random (i.e., *Vehicle Servers*). Each RSU server connects to four client servers and communicates via TCP/IP connections.

Performance of Trust Evaluation Model. To evaluate the performance of the VT-Eva model, we measured the system throughput, i.e., the number of messages (including rating-request messages and rating-result messages) per second. Fig. 6 presents the throughput of VT-Eva execution based on a varying number of clients and raters, seeing Fig. 6(a) and Fig. 6(b), respectively. Fig. 6(a) shows the rating-request throughput (i.e., the number of rating requests processed per second) when using 100 raters and a group of 100 to 200 clients. With the increased number of clients, there will be a growing number of rating requests sent to the system, which means that the rating-request throughput will also increase until the system reaches full capacity. As shown in Fig. 6(a), the maximum throughput approaches 800 messages per second (mps) even with 200 clients.

Fig. 6(b) presents the rating-result throughput (i.e., the number of rating-result messages processed per second)

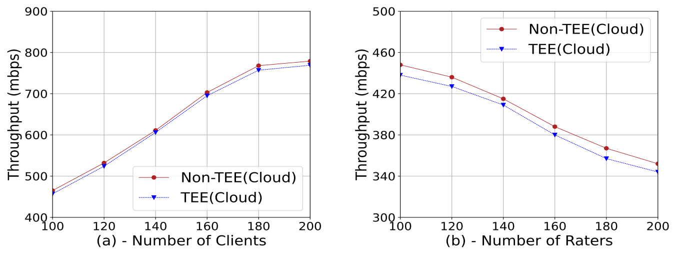

Fig. 6. Throughput of Vehicular Trust Evaluation Operations.

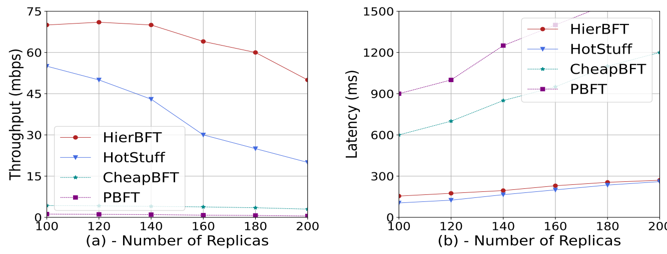

Fig. 7. Throughput and Latency of Blockchain Consensus Protocols.

when using 100 clients and a group of 100 to 200 raters. In this figure, the throughput declines with the increased number of raters. This is because more raters result in a larger number of rating messages processed in the system to compute a final rating result.

Performance of VT-chain. To evaluate the performance of blockchain consensus, we use several protocols (i.e., HotStuff [27], PBFT [21], and CheapBFT [29]) as baselines. Moreover, our experiments adopt the same parameters for each protocol, which mainly include a transaction message size of 250 bytes, a micro block size of 1 MB, and a replica set varying from 100 to 200. The selected sizes of the transaction and block follow the general parameters of the Bitcoin system, which is used by most researchers in their experiments [27], [30]. The experiments collect statistical results of throughput and latency based on micro blocks - i.e., micro blocks per second ($mbps$) and milliseconds (ms). Based on the size of the message and operation, each micro block contains around 4200 transactions.

Fig. 7(a) shows that the proposed hierarchical BFT consensus (i.e., HierBFT) protocol has much higher throughput (around $50 \sim 75$ $mbps$, with an increased replica set from 100 to 200) compared to PBFT and CheapBFT protocols, which are all below five $mbps$ with the same varying replica set. HotStuff, a BFT-like protocol, still has less throughput of around $20 \sim 60$ $mbps$. The difference between the HierBFT and HotStuff is more than doubles when the number of replicas is nearly 200. This indicates that HierBFT has higher scalability due to the hierarchical consensus design. On the other hand, as HierBFT applies a two-layer framework in consensus, it takes more time to achieve multi-layer consensus and cross-shard communication. Therefore, as shown in Fig. 7(b), the HierBFT protocol has a slightly longer average latency than HotStuff. Nevertheless, such a difference is diminishing with the growing replica set due to the limited scalability of HotStuff. In summary, the recent HierBFT and HotStuff protocols show their superior performance compared to other classic protocols, but more than that, the hierarchical design gives HierBFT an advantage in scalability, particularly when serving a large-scale system.

Local Storage Utilisation. In the VT-chain system, the usage of local hard drive storage can be limited by the execution of garbage collection (see Appendix A). However, each node instance implements a set of concurrent operations during the trust evaluation and blockchain consensus processes, resulting in heavier memory footprints. It is worth analysing the usage of local storage (RAM) on each VM server. However, Table 2 presents the average usage of RAM in terms of two types of servers (i.e., RSU and client servers). With the growing number of clients from 100 to 200, the average usage of RAM on RSU servers increases by 15% while the average RAM usage increment on vehicle servers is greater than 30%. This is because when the increased number of vehicle clients results in more concurrent operations based on each client instance. However, the number of RSU instances remains the same compared to the increased number of clients. Since the increased clients generate more service requests which need additional local storage on RSU servers, there is still a RAM usage increment (i.e., 15%) on RSU servers but less than that of vehicle servers.

TABLE 2
Average Local RAM Usage (%)

# Clients	100	120	140	160	180	200
RSU Server	61.1	66.2	68.5	71.4	72.9	75.8
Vehicle Server	53.9	60.8	67.0	75.3	79.2	88.4

7 VT-chain Deployment Discussion

The proposed VT-chain introduces a blockchain framework tailored for managing trust in vehicular networks, compatible with existing open-source blockchain platforms like IBM Hyperledger Fabric [11]. Specifically, our proposed VT-Eva model, which defines decentralized trust evaluation, can be implemented as smart contracts within the blockchain system. According to our design, certain operations of the VT-Eva model necessitate execution and security measures provided by TEEs on participating servers. In the vehicular network context, RSU servers and vehicle nodes collaborate to implement the VT-Eva model by facilitating the exchange of messages based on Algorithms 1-4.

Concurrently, RSU servers and vehicle nodes employ a hierarchical consensus mechanism, as outlined in Section 4.2, to achieve consensus on the trust evaluation results prior to updating the blockchain. Within this consensus process, vehicle nodes within each consensus shard independently validate the results and concurrently cast their votes. Meanwhile, the RSU servers forming the root shard ensure global consistency of all valid trust evaluation results and subsequently create blocks for each group of results. The predefined vehicular trust management network architecture (Fig. 1 in Section 2) supports the message communication required for executing the VT-Eva model and hierarchical consensus.

In general, the VT-chain system can be extended to other use cases, such as the Internet of Things or smart homes. However, for implementing the VT-chain in a specific use case scenario, the participating nodes need to establish a network architecture akin to the one depicted in Fig. 1, with the nodes forming the root shard being mandated to enable TEEs based on the design principles.

8 Conclusions

This paper proposed a vehicular trust blockchain (VT-chain) system, firstly by including a novel trust evaluation model that uses peer-review-based trust assessment and TEE-enabled trust computation to improve accuracy and security, respectively, and presented a multi-shard blockchain framework for trust data management. Secondly, to build the multi-shard blockchain, we proposed a hierarchical BFT consensus protocol that allows several node groups to concurrently run BFT consensus in the lower layer and leverages a global node group to ensure the total order. Such a design aims to improve blockchain scalability to adapt to the requirements of a large-scale system (e.g., the vehicular trust management system).

Our future work will explore another key challenge which is the mobility of vehicles, which would raise difficulties in trust evaluation and blockchain consensus due to frequently changing network topology caused by the switch across different RSUs. Furthermore, sharding configuration optimisation would be required to adapt to such a dynamic vehicular network environment, aiming for improved performance while ensuring security.

References

[1] M. Blaze, J. Feigenbaum and J. Lacy, "Decentralized trust management," in Proc. 1996 IEEE Symposium on Security and Privacy, pp. 164-173, 1996.

[2] F. Bao, R. Chen, M. Chang, and J.-H. Cho, "Hierarchical trust management for wireless sensor networks and its applications to trust-based routing and intrusion detection," IEEE Trans. Netw. Service Manag., vol. 9, no. 2, pp. 169-183, 2012.

[3] L. Guo, C. Zhang, and Y. Fang, "A trust-based privacy-preserving friend recommendation scheme for online social networks," IEEE Trans. Dependable Secure Comput., vol. 12, no. 4, pp. 413-427, 2015.

[4] W. Jiang, J. Wu, F. Li, G. Wang and H. Zheng, "Trust Evaluation in Online Social Networks Using Generalized Network Flow," IEEE Trans. Comput., vol. 65, no. 3, pp. 952-963, 2016.

[5] X. Meng and D. Liu, "Getrust: A guarantee-based trust model in chord-based p2p networks," IEEE Trans. Dependable Secure Comput., vol. 15, no. 1, pp. 54-68, 2018.

[6] C. Marche and M. Nitti, "Trust-Related Attacks and Their Detection: A Trust Management Model for the Social IoT," IEEE Trans. Netw. Service Manag., vol. 18, no. 3, pp. 3297-3308, 2021.

[7] R. Hussain, J. Lee and S. Zeadally, "Trust in VANET: A Survey of Current Solutions and Future Research Opportunities," IEEE Trans. Intell. Transp. Syst., vol. 22, no. 5, pp. 2553-2571, May 2021.

[8] A. Al-Fuqaha et al., "Internet of Things: A survey on enabling technologies protocols and applications," IEEE Commun. Surveys Tuts., vol. 17, no. 4, pp. 2347-2376, 2015.

[9] S. Thakur and J. G. Breslin, "A Robust Reputation Management Mechanism in the Federated Cloud," IEEE Trans. Cloud Comput., vol. 7, no. 3, pp. 625-637, 2019.

[10] M. Zhaofeng, W. Xiaochang, D. K. Jain, H. Khan, G. Hongmin and W. Zhen, "A Blockchain-Based Trusted Data Management Scheme in Edge Computing," IEEE Trans. Ind. Informat., vol. 16, no. 3, pp. 2013-2021, March 2020.

[11] E. Androulaki, et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in Proc. of EuroSys'18, New York, NY, USA, Article 30, pp. 1-15, 2018.

[12] B. Li et al., "Blockchain-Based Trust Management Model for Location Privacy Preserving in VANET," IEEE Trans. Ind. Informat., vol. 22, no. 6, pp. 3765-3775, June 2021.

[13] D. E. Kouicem, Y. Imine, A. Bouabdallah and H. Lakhlef, "A Decentralized Blockchain-Based Trust Management Protocol for the Internet of Things," IEEE Trans. Dependable Secure Comput., vol. 19, no. 2, pp. 1292-1306, 2022.

[14] E. Buchman, "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains," Doctoral dissertation, University of Guelph, 2016.

[15] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," in Proc. of 2015 IEEE Trustcom/BigDataSE/ISPA, pp. 57-64, 2015.

[16] G. Danezis and S. Meiklejohn, "Centrally Banked Cryptocurrencies," 2016. [Online]. Available: 10.48550/arXiv.1505.06895

[17] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A Secure Sharding Protocol For Open Blockchains," in Proc. of CCS'16, pp.17-30, Vienna, Austria, 2016.

[18] M. Al-Bassam et al., "Chainspace: A Sharded Smart Contracts Platform," in Proc. of NDSS'18, San Diego, CA, 2018.

[19] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in Proc. of S&P'18, San Francisco, US, 2018.

[20] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *Proc. CCS*, New York, NY, USA, 2018, pp. 931–948.

[21] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in Proc. of USENIX OSDI'99, New Orleans, USA, Feb. 1999.

[22] V. Costan and S. Devadas, "Intel SGX Explained," IACR Cryptol. ePrint Arch. vol. 2016, no. 86, 2016.

[23] Intel, "Software Guard Extensions Developer Guide," 2018.

[24] X. Yang, Y. Guo and Y. Liu, *Bayesian-Inference-Based Recommendation in Online Social Networks*, IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 4, pp. 642-651, April 2013.

[25] C. P. Schnorr, "Efficient Signature Generation by Smart Cards," J. Cryptol., vol.4, no.3, 1991.

[26] B. C. Desai, B. S. Boutros, "Performance of a Two-phase Commit Protocol," *Inf. Softw. Technol.*, vol. 38, no. 9, pp. 581–599, 1996.

[27] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT Consensus with Linearity and Responsiveness," in Proc. of PODC'19, pp. 347-356, Toronto ON, Canada, 2019.

[28] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient Byzantine Fault-tolerance," IEEE Trans. Comput., vol.62, no.1, pp.16-30, Jan. 2013.

[29] R. Kapitza, J. Behl, C. Cachin, T. Distler, and S. Kuhnle, "CheapBFT: Resource-efficient Byzantine Fault Tolerance," in Proc. of EuroSys'12, pp.295, Bern, Switzerland, 2012.

[30] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing," IEEE Trans. Comput., vol. 68, no. 1, pp. 139–151, Jan. 2019.

[31] Q. Wang, D. Gao, C. H. Foh, H. Zhang, and V. C. Leung, "Decentralized CRL Management for Vehicular Networks with Permissioned Blockchain," IEEE Trans. Veh. Technol., vol. 71, no. 11, pp. 408–1420, 2022.

[32] G. Sang, J. Chen, Y. Liu, H. Wu, Y. Zhou, and S. Jiang, "PACM: Privacy-preserving Authentication Scheme with On-chain Certificate Management for VANETs," IEEE Trans. Netw. Service Manag., 2022, doi:10.1109/TNSM.2022.3201551.

[33] Z. Ma et al., "An Efficient Decentralized Key Management Mechanism for VANET with Blockchain," IEEE Trans. Veh. Technol., vol. 69, no. 6, pp. 5836–5849, 2020.

[34] M. Azees, P. Vijayakumar, and L. J. Deboarh, "EAAP: Efficient Anonymous Authentication with Conditional Privacy-preserving Scheme for Vehicular Ad-hoc Networks," *IEEE Trans Intell. Transp. Syst.*, vol. 18, no. 9, pp. 2467–2476, 2017.

[35] R. Lu et al., "A Dynamic Privacy-preserving Key Management Scheme for Location-based Services in VANETs," IEEE Trans Intell. Transp. Syst., vol. 13, no. 1, pp.127–139, 2011.

[36] A. Mansour, K. Malik, A. Alkaff, and H. Kanaan, "ALMS: Asymmetric Lightweight Centralized Group Key Management Protocol for VANETs," IEEE Trans Intell. Transp. Syst., vol. 22, no. 3, pp. 1663–1678, 2020.

[37] R. Sharma and S. Chakraborty, "BlockAPP: Using Blockchain for Authentication and Privacy Preservation in IoV," in Proc. of IEEE Globecom Workshops (GC Wkshps), pp. 1–6, 2018.

[38] J. Qi and T. Gao, "An Anonymous Authentication Scheme Based on Self-generated Pseudonym for VANETs," in Proc. of IMIS'22, pp. 75–84, Kitakyushu, Japan, 2022.

[39] X. Zhu, S. Jiang, L. Wang, and H. Li, "Efficient Privacy-preserving Authentication for Vehicular Ad-hoc Networks," IEEE Trans. Veh. Technol., vol. 63, no. 2, pp. 907–919, 2014.

[40] P. Wang and Y. Liu, "SEMA: Secure and efficient message authentication protocol for VANETs," IEEE Syst. J., vol. 15, no. 1, pp. 846–855, 2021.

[41] V. T. Kilari, R. Yu, S. Misra and G. Xue, "Robust Revocable Anonymous Authentication for Vehicle to Grid Communications," IEEE Trans Intell. Transp. Syst., vol. 21, no. 11, pp. 4845-4857, 2020.

Xiao Chen received the M.Sc. and Ph.D. degrees in computing science from Newcastle University in 2009 and 2013, respectively. He is currently a research fellow (Marie Sklodowska-Curie) at School of Informatics in the University of Edinburgh (UK). He was a postdoctoral research fellow in the School of Computing, Informatics and Decision Systems Engineering at Arizona State University (USA). His research interests include the formal method, stochastic modelling, and AI-driven optimization for large-scale systems; and a focus on the theory of consensus mechanisms, privacy-preserving, trust management, and real-world blockchain application development.

Guoliang Xue is a full professor of computer science and engineering in the School of Computing, Informatics and Decision Systems Engineering at Arizona State University (ASU). He received his ph.D. degree in computer science from the University of Minnesota in 1991. Before joining ASU as a tenured associate professor in August 2001, he worked at the University of Vermont as an assistant/associate professor of computer science. His research interests are in the areas of computer networks and optimization. He has published over 180 refereed papers in these areas. He is on the editorial boards of IEEE/ACM Transactions on Networking, IEEE Transactions on Wireless Communications, IEEE Network, and Computer Networks. He served as TPC Co-Chair of IEEE INFOCOM 2010, and is a Distinguished Lecturer of the IEEE Communications Society for 2010 and 2011.

Ruozhou Yu (Student Member 2013, Member 2019, Senior Member 2021) is an Assistant Professor of Computer Science at North Carolina State University, USA. He received his PhD degree (2019) in Computer Science from Arizona State University, USA. His research interests include internet-of-things, cloud/edge computing, smart networking, algorithms and optimization, distributed machine learning, security and privacy, blockchain, etc. He has served on the organizing committees of IEEE INFOCOM 2022-2023 and IEEE IPCCC 2020-2022, and as members of the technical commitee of IEEE INFOCOM 2020-2022. He was a recipient of the NSF CAREER Award in 2021.

Haiqin Wu received her B.E. degree in Computer Science and Ph.D degree in Computer Application Technology from Jiangsu University in 2014 and 2019, respectively. She is an Associate Professor in the Department of Software Engineering, East China Normal University, China. Before that, she was a postdoctoral researcher in the Department of Computer Science, University of Copenhagen, Denmark. She was also a visiting student in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University, US. Her research interests include data security and privacy protection, mobile crowdsensing, and blockchain-based applications.

Dawei Wang received his B.Sc. and M.Sc. degrees in computer science and technology at Jiangsu University in 2018 and 2021, respectively. He is currently a senior software engineer of fintech information system. His research interests include blockchains, consensus protocols and hardware security.