

Article

GKDBV-EF: A Lightweight and Provably Secure Group Key Distribution with Update and Batch Verification Protocol for Cloud–Fog–Edge Computing Networks

Narendra Kumar Upadhyay ¹, Sudhakar Periyasamy ¹ and Vinod Kumar ^{2,*} 

¹ School of Computer Science and Engineering, Galgotias University, Greater Noida 203201, India; narendra16may@gmail.com (N.K.U.); p.sudhakar@galgotiasuniversity.edu.in (S.P.)

² Department of Electronics and Communication, University of Allahabad, Prayagraj 211002, India

* Correspondence: vk@allduniv.ac.in

Abstract

The advent of cloud–fog–edge computing has transformed distributed data processing by performing computation closer to end devices. Due to resource constraints at edge nodes and the dynamic nature of fog-assisted communication, secure and efficient group key distribution and batch verification in such decentralized systems remain a major challenge. Many existing protocols based on Chinese remainder theorem (CRT) use a straightforward scalar product to mask the group key and hence fail in multifactor security. Others suffer from architectural overhead since they require distinct and independent sets of moduli equations with multiple mathematical structures for different network layers, which increases computing overhead, limits scalability and delays synchronization during frequent node leave/join. To mitigate these challenges, this paper proposes a unified distributed CRT-based protocol for cloud–fog–edge environments. Our protocol introduces a two-factor modular key masking mechanism by incorporating a unique secret parameter for every edge node to strengthen group key protection and enhance the overall robustness of the key distribution mechanism. Additionally, our protocol uses a single set of moduli equations across cloud–fog–edge networks, which drastically reduces computation and storage costs at the fog layer. Our protocol achieves $\mathcal{O}(1)$ efficiency for rekeying. Formal security analysis using ProVerif and the ROR model demonstrates that our protocol has considerable security advantages. To prove its practicality, an ESP32-based simulation on Wokwi is used to verify the correctness of group key distribution, retrieval, and batch message verification. The performance analysis findings show that our protocol outperforms others in computation cost, communication cost, security and applicability for resource-constrained cloud–fog–edge computing networks.

Keywords: cloud–fog–edge computing; group key distribution; group key update; lightweight; provably secure; batch verification; group rekeying; Chinese remainder theorem; Euler’s function



Received: 20 May 2026

Revised: 2 July 2026

Accepted: 5 July 2026

Published: 11 July 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Over the past few decades, there has been significant growth in fog computing. Fog computing systems include cloud servers, nodes, and edge devices. The Internet of Things (IoT) has grown at an exponential rate, resulting in large amounts of data generated by heterogeneous devices deployed in a diversity of locations. The inherent drawbacks of traditional cloud-centric architectures, like high latency, bandwidth congestion, and centralized

dependency, make it unable to manage this increase. By putting computational and storage resources closer to data sources, the cloud–fog–edge computing system has emerged as a potential solution to these problems. The cloud–fog–edge computing environment can be used to accelerate processing for delay-sensitive cloud applications by delivering services at the network’s edge. In this hierarchical architecture, edge nodes process IoT data directly, while fog nodes collaborate and aggregate data before sending selective data to the cloud. Designing a lightweight cryptographic protocol that can work efficiently on resource-constrained devices without degrading performance is a major challenge.

The majority of existing common key distribution techniques are susceptible to various security attacks and require complex mathematical operations that lead to excessive computational and communication overhead. In addition, frequent changes in group membership, including adding or removing edge nodes, need continuous key updates and rekeying operations, which, if not handled properly, can cause greater vulnerability to attacks and synchronization delays. To overcome these challenges, a mathematically sound, lightweight, and secure key distribution framework is required that supports efficient key updates and batch verification of multiple edge nodes in cloud–fog–edge networks. For secure communication in many domains such as IoT, WSNs, VANETs, IoDs, etc., many key distribution and mutual authentication protocols have been introduced using Chinese remainder theorem (CRT). The majority of these schemes compute the inverse element using the extended Euclidean method. The inverse element calculation is known to be very time-consuming. Moreover, in some cases, the inverse element may not be present, rendering the computation ineffective. Chen et al. [1] developed a new way to solve CRT using Euler’s totient function. The congruent equation system can be solved using CRT with Euler’s totient function instead of the extended Euclidean method.

To achieve computational efficiency with robust security, we present a new group key distribution and batch verification protocol for secure communication in cloud–fog–edge computing networks based on Euler’s totient function and CRT. Euler’s function makes it possible to generate cryptographic parameters to avoid the computation of the inverse element with reduced complexity, whereas CRT facilitates parallel modular computations for fast group key distribution and rekeying. These number-theoretic primitives, when combined with bilinear pairing, provides a strong mathematical foundation for fast secure key distribution and effective batch verification without the need for complex computations. The proposed protocol presents a hierarchical key management structure that is compatible with the cloud–fog–edge architecture. The fog nodes act as semi-trusted entities responsible for coordinating with edge nodes and IoT devices in group key distribution and batch verification. In the proposed protocol, key distribution and retrieval functionality is achieved with a drastic reduction in computation and communication overhead. Moreover, our protocol provides the functionality of batch verification, allowing multiple edge nodes to be verified simultaneously, thereby minimizing communication overhead and latency. Furthermore, the protocol maintains forward and backward secrecy, guaranteeing that a newly joined edge node is unable to access past transmissions and a departed edge node is unable to access future ones. Comprehensive security analysis confirms that our protocol is resistant to known cryptographic attacks. Formal verification through the ProVerif tool demonstrates that our protocol fulfils key security properties and confidentiality. Performance evaluations further show that our protocol significantly reduces computation time and communication cost compared to related protocols.

1.1. Motivation

The existing protocols proposed for security in edge–fog computing networks are either insecure against various known attacks or computationally expensive, making them

unsuitable for practical deployment. Wu, S. et al.'s [2] protocol has a major architectural overhead in which a cloud server and individual fog nodes maintain different sets of moduli equations for key distribution. Managing distinct sets of moduli equations and computing secret parameters for key distribution at both layers are extremely difficult. A critical security gap exists in the protocols of Zhang J. et al. [3], Vijaykumar, P. et al. [4], Wu, S. et al. [2] and Vijaykumar, P. et al. [5] due to their reliance on simple multiplicative relationships for key distribution. In these protocols, the group key k_g is masked by a parameter μ using basic scalar multiplication, resulting in linear ciphertext ($C = k_g \times \mu$). This structure is mathematically vulnerable, because in order to generate the ciphertext C , it multiplies k_g with μ , and the leakage of the μ allows an attacker to obtain the group key. Therefore, these protocols fail to provide better security. To address these gaps, we propose a unified distributed architecture that uses a single set of moduli equations across all layers of the fog-edge computing network and shifts expensive computations to an offline pre-computation phase, which greatly decreases overhead. By incorporating a modular group key encryption logic and providing a two-factor security derivation based on both the private key and the secret parameter, rather than just the private key, the system remains resistant to collusion, modulus leakage, and other known cryptographic attacks while maintaining superior real-time performance.

1.2. Major Contributions

In this paper, we propose GKDBV-EF, a new lightweight and provably secure group key distribution for cloud-fog-edge computing networks that includes key updates and batch verification.

The contributions of this paper are summarized as follows:

We present a new lightweight and robust group key distribution and updating protocol that combines the Euler function and CRT, eliminating the need for modular inverse element calculation. This design considerably simplifies essential update processes and avoids feasibility concerns caused by missing inverses, making it ideal for dynamic and large-scale networks.

The proposed protocol allows for efficient group key updates when edge nodes join or depart the group. By reconstructing broadcast parameters with Euler's function attributes, the updated group key can be securely provided to valid edge nodes while preventing revoked or newly joined nodes from obtaining group key. As a result, the method provides robust forward and backward security.

Based on the updated group key, we design a message authentication mechanism that enables both single and batch signature verification. The batch verification process enables the fog node to verify several messages simultaneously, significantly reducing verification time and computational overhead compared to individual verification.

The proposed protocol's robustness against known cryptographic attacks is proved by formal security verification using the random oracle model (ROM) and the ProVerif tool, as well as by conducting informal security analysis.

Extensive performance analysis demonstrates that our protocol outperforms the related protocol in terms of security features and computational and communication costs, which proves that it is suitable for deployments in the real world.

1.3. Paper Organization

The rest of this paper is organized as follows. In Section 2, we discuss some related work and highlight the limitations of existing protocols. Section 3 outlines the proposed system model and cryptographic preliminaries. Section 4 describes the proposed protocol's design. Section 5 examines security of the proposed protocol. The performance evaluation is shown in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

In recent years, various studies have presented a variety of authentication and key distribution mechanisms to allow system entities to transmit information securely in an untrustworthy environment. In [6], the authors proposed a scheme to efficiently distribute a group key using the Generalized CRT, with low communication cost and good scalability, while ensuring legitimate user authentication. However, it has security flaws such as vulnerability to collusion and algebraic attacks, a lack of strong forward/backward secrecy, and reliance on a trustworthy KGC. Zhang et al. [3] introduced an efficient authentication technique for VANETs with conditional privacy preservation that avoids master key pre-loading and allows trusted authorities to distribute dynamic group keys using CRT. It resists side-channel leakage and provides robust security with minimum computation and communication cost. However, its performance can degrade in highly dynamic vehicular environments. Kumar et al. [7] introduced an effective and robust authentication and key distribution technique with low rekeying cost. It uses a star topology for key distribution and dramatically reduces computation and communication costs. The proposed protocol is also integrated into a distributed architecture to improve its scalability.

Kumar et al. [8] presented a cloud-based TMIS protocol that provides efficient authentication and patient privacy. However, it is vulnerable to attacks like man-in-the-middle and replay. Furthermore, it depends on a trusted cloud/server for some specific operations, which may result in a single point of failure. Amanlou et al. [9] presents an ECDHE-PSK-based authentication protocol for IoT-fog environments that strikes a balance between security and efficiency. It uses lightweight operations while providing increased security features such as perfect forward secrecy. However, its scalability in large and heterogeneous IoT networks is still a question. Kumar et al. [10] proposed a scheme that reduces key server computing and storage costs by integrating lightweight join and leave procedures. It also uses a scalable clustered-tree CGKD design to efficiently manage dynamic membership, resulting in minimal cost for both the server and members. Vijayakumar et al. [4] present a group key management and double authentication technique for safe data transmission in VANETs with increased efficiency. However, subsequent investigation demonstrates that the technique is susceptible to replay and masquerade attacks, as previously collected messages can be exploited to compromise the system. This emphasizes the need for additional enhancements to maintain strong security.

Wazid et al. [11] presented a key management technique for secure communication in fog-based IoV with formal security validation. However, the approach is mainly tested through simulations and has not been deployed in the real world, leaving open questions about its scalability and performance in large-scale vehicle networks. Awais et al. [12] presented a new fog-based key agreement and authentication technique for VANETs that ensures mutual authentication and quick key establishment without the use of bilinear pairings. The scheme requires low computing and communication costs, and its security is verified using ROM. Ali et al. [13] introduced a lightweight key agreement and anonymous authentication mechanism for fog-based IoT systems that utilizes XOR and hash operations. It facilitates secure mutual authentication while minimizing computational overhead. However, it requires enhancement in mobility support and large-scale deployment. Gupta et al. [14] introduced, a novel key management technique for securing bidirectional communication in AMI of smart grids. It ensures efficient key updates at $O(1)$ cost during dynamic membership changes. It provides strong security assurances, such as forward and reverse secrecy, which are confirmed by formal analysis.

Gupta et al. [15] developed a mutual authentication approach based on ECC for TMIS that provides patient privacy, anonymity, and secure session key formation. It uses ROM and ProVerif for formal security validation. Moreover, it reduces computational and

communication overhead. Hassan et al. [16] provide an edge–fog-based system for real-time traffic management in smart cities, with considerable savings in latency, network usage, and execution costs. Cui et al. [17] presented a hash-based authentication technique that uses a CRT-based group key mechanism to ensure safe key distribution and updates. However, the technique has linkability issues and requires additional overhead during variable substitution. Shao et al. [18] introduced a decentralized VANET authentication scheme that utilizes a unique group signature to facilitate reliable and anonymous communication via bilinear pairing. It enables threshold verification, revocation, traceability, and fast batch authentication. Zhong et al. [19] introduced an authentication technique that supports secure V2I communication. It uses aggregate signatures and pseudonyms to provide batch verification and conditional privacy. Moreover, pre-computation and fixed-size aggregated signatures decrease computing, communication, and storage expenses. Ali et al. [20] introduced a hybrid signcryption technique for secure vehicular communication in heterogeneous VANETs that ensures authentication, non-repudiation, and secrecy in a single phase. It increases efficiency by batch unsigncryption and requires lower computing costs. However, using bilinear pairings may result in overhead.

In recent years, several protocols for key management based on CRT have been presented. Chen et al. [21] presented a CRT-based group signature technique that may revoke a group member while not affecting other members' keys. Zheng et al. [22] presented a secure group key updating technique for centralized environments. Members used CRT to retrieve the updated group key. Agrawal et al. [23] presented a mutual-healing mechanism for group key broadcasting. The scheme included both key confirmation and mutual authentication mechanisms. However, the scheme's scalability is limited because it is not applicable in distributed environments. The majority of earlier works have focused on updating the key group key in centralized environments and are not suitable in decentralized environments such as cloud–fog–edge networks.

3. Preliminaries

3.1. System Model

The suggested protocol has three layers, (1) cloud, (2) fog, and (3) edge, which allows for easy scalability. At the first cloud layer, just one entity cloud server is available, which is responsible for generating the requisite keys and secret parameters as well as performing various system initialization tasks. The second fog layer comprises fog nodes that manage the associated group of edge nodes. Fog nodes are also responsible for generating the subgroup key encryption key and distributing the private keys of edge nodes for their respective groups. The last edge layer consists of edge nodes, which are organized into distinct subgroups known as clusters. This layer has several subgroups/clusters of edge nodes, each of which is managed by the fog node to which it is linked. These clusters communicate with the intermediate-layer fog nodes, which then communicate with the root-layer cloud server. Subgroups/clusters are formed based on the number of edge nodes and cluster size, which is determined by the cloud server. Figure 1 shows the architecture of the proposed edge–fog computing network, as well as the interaction and data exchange between entities.

3.2. Security Model

The security of the proposed scheme is analyzed under a standard adversarial model for decentralized IoT–fog–cloud systems. The adversary is supposed to be a probabilistic polynomial-time entity that has complete control over public transmission channels, including the capacity to intercept, edit, inject, and replay communications. It also assumed that fog nodes operate as legitimate network entities and are not fully compromised. The

adversary may not be able to compromise the edge nodes and unable to recover stored information, such as the private key, secret parameters and group key. The scheme's major security goal is to establish existential unforgeability against chosen-message attacks, which ensures that no adversary can generate a valid signature on a new message that can be accepted by the fog nodes or cloud server. Hash functions are modelled as random oracles, and the security of the protocol relies on the hardness of the Computational co-Diffie–Hellman Problem (co-CDHP). The incorporation of per-message randomness and secure batch aggregation guarantees resistance against replay attacks, signature linking, and batch forgery, such that acceptance of a batch verification implies the validity of all individual signatures except with negligible probability.

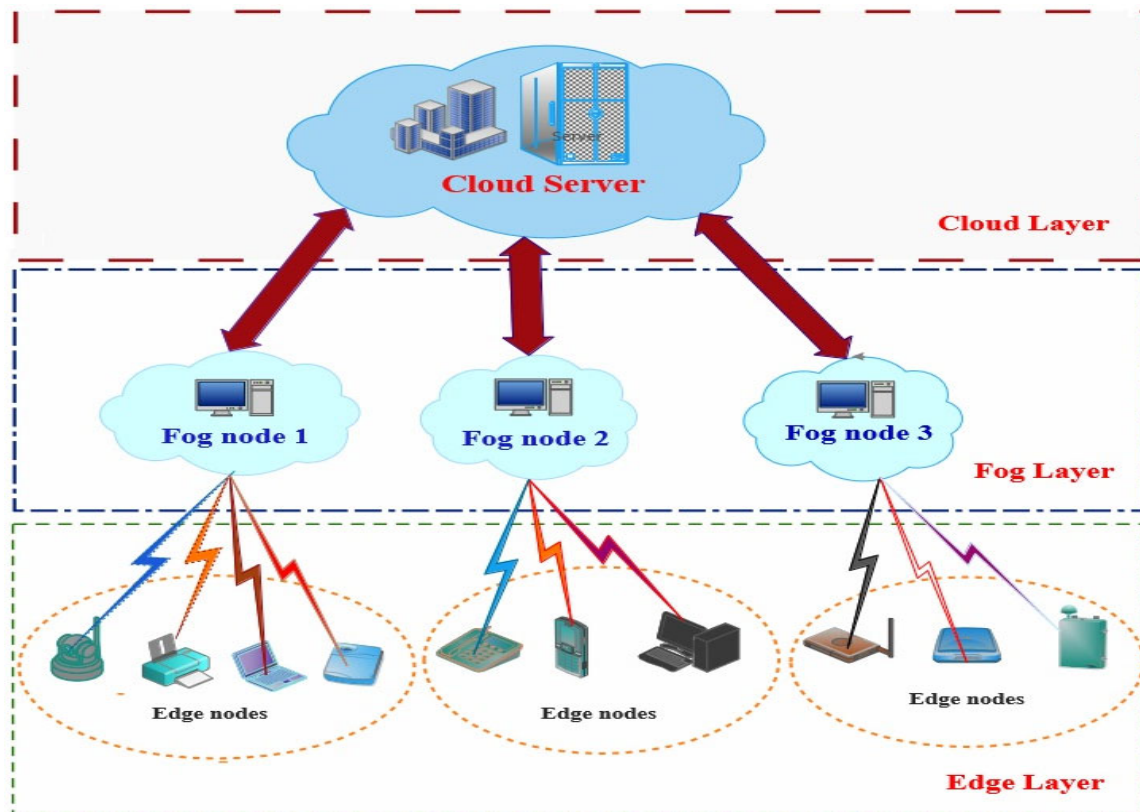


Figure 1. System model of proposed edge–fog computing protocol.

3.3. Notations

Table 1 describes the notations used throughout the paper.

Table 1. Abbreviations and Notations.

Notation	Descriptions
$\mathcal{FN}_i, \mathcal{EN}_{ij}$	Fog node and edge node
$P_C, P_F, \text{ and } P_E$	Set of private keys for cloud server, fog node and edge node
$p_c, p_{f_i}, p_{e_{ij}}$	Private keys for cloud server, fog node and edge node
S_C, S_F, S_E	Set of secret parameters for cloud server, fog node and edge node
$s_c, s_{f_i}, s_{e_{ij}}$	Secret parameters for cloud server, fog node and edge node
$ID_{f_i} \text{ and } ID_{e_{ij}}$	The real identity for fog node and edge node
h_0, h_1, h_2	One-way hash function
δ_i	The group key encryption key
$TID_{f_i} \text{ and } TID_{e_{ij}}$	Temporary identity for fog node and edge node
T_{ij}	Timestamps
$\mathcal{ADV}$	Adversary

Table 1. Cont.

Notation	Descriptions
$\mathbb{G}$ and $\mathbb{G}_T$	Additive and multiplicative cyclic groups
P	A generator of cyclic group $\mathbb{G}$
$\phi(\cdot)$	Euler's function
M	Product of all private keys
n	No. of fog nodes
m	No. of edge nodes
$\mathcal{J}_{e_{ij}}^{-1}$	Modular inverse of secret parameter $\mathcal{J}_{e_{ij}}$ of edge node EN_{ij}
$\mathcal{G}K_i, \mathcal{G}K'_i, \mathcal{G}K''_i$	Group k
$r_{e_{ij}}$	A secret random number of edge node EN_{ij}
P_{pub_i}	Public key element generated by $\mathcal{FN}_i$

4. Proposed Protocol

4.1. System Initialization

In our protocol, we assume that there is a cloud server and n fog nodes, each of which manages m edge nodes. As a result, the protocol consists of $n \times m$ edge nodes. Each fog node connects to m edge nodes, forming an m -node cluster. The cloud server generates secret parameters and private keys for itself and fog nodes and edge nodes from Z_p^* . Initially, the system generates one secret parameter and one private key for the cloud server and n secret parameters and n private keys for the fog nodes. Similarly, it generates $n \times m$ private keys and $n \times m$ secret parameters for maximum number of edge nodes. The private keys are generated by the cloud server's Private Key Generator (PKG) component. The system initialization process is explained in the steps below.

The cloud server chooses two cyclic groups: a multiplicative group $\mathbb{G}_T$ and an additive group $\mathbb{G}$ of order λ , where λ is a large prime integer. It also chooses a residue class ring $\mathbb{Z}_\lambda$.

The cloud server chooses two cryptographic one-way hash functions, namely $h_1 : \{0, 1\}^* \rightarrow \mathbb{G}$ and $h_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_\lambda^*$. It also defines an efficient bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$.

The cloud server selects three set of private keys $P_E = \{\{p_{e_{1j}}\}, \{p_{e_{2j}}\}, \{p_{e_{3j}}\}, \dots, \{p_{e_{nj}}\}\}$, for $j = 1, 2, 3, \dots, m$, $P_F = \{p_{f_1}, p_{f_2}, p_{f_3} \dots p_{f_n}\}$ and $P_C = \{p_c\}$ from Z_λ^* such that the private keys of all sets must be co-prime to each other, i.e., $\gcd(\{p_{e_{1j}}\}, \{p_{e_{2j}}\}, \{p_{e_{3j}}\}, \dots, \{p_{e_{nj}}\}, \{p_{f_1}, p_{f_2}, p_{f_3} \dots p_{f_n}\}, \{p_c\}) = 1$. The set P_E consist of n subsets, and each subset $p_{e_{1j}}$ contains m private keys.

Similarly, the cloud server generates three sets of secret parameters $\mathcal{S}_E = \{\{\mathcal{J}_{e_{1j}}\}, \{\mathcal{J}_{e_{2j}}\}, \{\mathcal{J}_{e_{3j}}\}, \dots, \{\mathcal{J}_{e_{nj}}\}\}$, for $j = 1, 2, 3, \dots, m$, $\mathcal{S}_F = \{\mathcal{J}_{f_1}, \mathcal{J}_{f_2}, \mathcal{J}_{f_3} \dots \mathcal{J}_{f_n}\}$ and $\mathcal{S}_C = \{\mathcal{J}_c\}$ from Z_λ^* .

The cloud server computes the product M of all entities' private keys as $M = \prod_{i=1}^n (p_{f_i}) \times \prod_{i=1}^n \prod_{j=1}^m (p_{e_{ij}}) \times p_c$. Also, it computes $M_{f_i} = \frac{M}{p_{f_i}}, \forall i = 1, 2, 3, \dots, n$, $M_{e_{ij}} = \frac{M}{p_{e_{ij}}}, \forall i = 1, 2, 3, \dots, n$, and $j = 1, 2, 3, \dots, m$, and $M_c = \frac{M}{p_c}$.

Next, the cloud server computes the following:

$$\mathcal{C}_u = \mathcal{J}_c \times (M_c)^{\phi(p_c)}$$

$$\mathcal{F}_{u_i} = \mathcal{J}_{f_i} \times (M_{f_i})^{\phi(p_{f_i})}, \forall i = 1, 2, 3, \dots, n$$

$$\text{and } E_{u_{ij}} = \mathcal{J}_{e_{ij}} \times (M_{e_{ij}})^{\phi(p_{e_{ij}})}, \forall i = 1, 2, 3, \dots, n \text{ and } j = 1, 2, 3, \dots, m.$$

The set of private keys, secret parameters, $\mathcal{C}_u$, $\mathcal{F}_{u_i}$ and $E_{u_{ij}}$ are stored in its memory.

Initially, the system generates $n \times (m + 1) + 1$ private keys and $n \times (m + 1) + 1$ secret parameters. When the system reaches a limit of $n \times m$ edge nodes, it is re-initialized for $t \times (n \times m)$ edge nodes. The factor t represents the number of times the edge nodes may join the system in the future.

4.2. Fog Node Registration Phase

When a fog node joins a network, it should register with the cloud server. The fog node $\mathcal{FN}_i$ picks a unique identity ID_{f_i} and a secret random number r_{f_i} and computes $A_{f_i} = h_2(ID_{f_i} \parallel r_{f_i})$. A_{f_i} is securely transmitted to the cloud server, which then generates a temporary identity TID_{f_i} , and securely transmits it to the fog node for storage. The cloud server also keeps secure A_{f_i} and TID_{f_i} in its database. The cloud server sends secret parameter s_{f_i} and private key p_{f_i} to the associated fog nodes $\mathcal{FN}_i$ from sets $\mathcal{S}_F$ and P_F respectively. Additionally, it transmits sets $\{s_{e_{ij}}\}$ and $\{p_{e_{ij}}\}$ of secret parameters and private keys from sets $\mathcal{S}_E$ and P_E respectively to the fog node $\mathcal{FN}_i$. $\mathcal{FN}_i$ keeps sets $\{s_{e_{ij}}\}$ and $\{p_{e_{ij}}\}$ secret in its database and delivers individual $s_{e_{ij}}$ and $p_{e_{ij}}$ to edge node EN_{ij} when they join the corresponding $\mathcal{FN}_i$. Additionally, the cloud server also sends $\mathcal{F}_{u_i}$ and $E_{u_{ij}}$ to the corresponding $\mathcal{FN}_i$ and keeps it secret in its database for further usage.

4.3. Edge Node Registration Phase

When an edge node EN_{ij} joins a network, it should register with the corresponding fog node $\mathcal{FN}_i$. The edge node EN_{ij} picks a unique identity $ID_{e_{ij}}$ and a secret random number $r_{e_{ij}}$ and computes $A_{e_{ij}} = h_2(ID_{e_{ij}} \parallel r_{e_{ij}})$. The edge node securely transmits $A_{e_{ij}}$ and $r_{e_{ij}}$ to the fog node $\mathcal{FN}_i$, which then generates a temporary identity $TID_{e_{ij}}$ and generates $B_{e_{ij}} = h_2(A_{e_{ij}} \parallel A_{f_i} \parallel s_{e_{ij}})$. $\mathcal{FN}_i$ selects private key $p_{e_{ij}}$ and $s_{e_{ij}}$ from the sets already sent by the cloud server to the fog node for edge nodes. $\mathcal{FN}_i$ then transmits $(TID_{e_{ij}}, TID_{f_i}, A_{e_{ij}}, B_{e_{ij}}, p_{e_{ij}}, s_{e_{ij}})$ to the edge node EN_{ij} through a secure channel. The edge node EN_{ij} stores the received values in its memory.

4.4. Group Key Distribution and Extraction Phase

$\mathcal{FN}_i$ generates a group key $\mathcal{GK}_i$ as a random prime integer form Z_λ^* as $\mathcal{GK}_i \xleftarrow{\$} Z_\lambda^*$. Additionally, the generated $\mathcal{GK}_i$ must be smaller than the $p_{e_{ij}}$ value of every edge node EN_{ij} attached with $\mathcal{FN}_i$, such that $\mathcal{GK}_i < \min\{p_{e_{i1}}, p_{e_{i2}}, p_{e_{i3}} \dots p_{e_{im}}\}$. Additionally, $\mathcal{FN}_i$ generates public key element P_{pub_i} as $P_{pub_i} = \mathcal{GK}_i \cdot P$.

The fog node $\mathcal{FN}_i$ computes the group key encryption key δ_i as $\delta_i = E_{u_{i1}} + E_{u_{i2}} + E_{u_{i3}} \dots + E_{u_{im}} = \sum_{j=1}^m E_{u_{ij}}$. $\mathcal{FN}_i$ also computes $M_i^m = \prod_{j=1}^m (p_{e_{ij}})$.

Next, $\mathcal{FN}_i$ encrypts $\mathcal{GK}_i$ and generates the ciphertext $CT_{\mathcal{GK}_i}$ as $CT_{\mathcal{GK}_i} = (\mathcal{GK}_i \times \delta_i) \bmod M_i^m$ and computes $HC = h_2(CT_{\mathcal{GK}_i} \parallel \mathcal{GK}_i \parallel T_{ij})$, where T_{ij} is the current timestamp. $\mathcal{FN}_i$ then broadcasts $CT_{\mathcal{GK}_i}, HC, T_{ij}$ to all edge nodes $EN_{i1}, EN_{i2}, EN_{i3}, \dots, EN_{im}$ in the group that are connected to it.

After receiving $CT_{\mathcal{GK}_i}$, each EN_{ij} can extract $\mathcal{GK}_i$ as $\mathcal{GK}_i = (CT_{\mathcal{GK}_i} \times s_{e_{ij}}^{-1}) \bmod p_{e_{ij}}$ and compute $HC^* = h_2(CT_{\mathcal{GK}_i} \parallel \mathcal{GK}_i \parallel T_{ij})$, and checks whether $HC^* = HC$ follows or not. If it follows, the extracted group key is accepted as a valid group key.

As a result, the total m edge nodes $EN_{ij}, \forall j = 1, 2, 3, \dots, m$ can extract $\mathcal{GK}_i$ from $CT_{\mathcal{GK}_i}$ by their private key $p_{e_{ij}}$ and secret parameter $s_{e_{ij}}$.

4.5. Batch Verification Based on Group Key Phase

When a node EN_{ij} collects a message m_{ij} , it signs it before forwarding it. For signing the message, the edge node EN_{ij} completes the following:

First, it calculates the blinded message hash h_{ij} as $h_{ij} = h_1(m_{ij} \parallel TID_{e_{ij}} \parallel T_{ij})$, where T_{ij} is a timestamp.

Next, it computes the blinding scalar $\alpha_{ij} = h_2(m_{ij} \parallel T_{ij}) \in Z_q^*$.

Further, it computes the signature component $\sigma_{ij} = (\mathcal{G}K_i + \alpha_{ij}) \cdot r_{e_{ij}} \cdot h_{ij} \in \mathbb{G}$ and generates the signature $sig_{ij} = (\sigma_{ij}, TID_{e_{ij}}, T_{ij})$.

EN_{ij} transmits the message $\{m_{ij}, \sigma_{ij}, TID_{e_{ij}}, T_{ij}\}$ to $\mathcal{FN}_i$. After receiving the message, $\mathcal{FN}_i$ initiates the message verification process given as follows.

Case 1. The fog node $\mathcal{FN}_i$ requires verifying only a single message.

The fog node $\mathcal{FN}_i$ verifies a single signature using the following pairing Equation (1):

$$e(\sigma_{ij}, P) = e(h_{ij}, P_{pub} + \alpha_{ij} \cdot P)^{r_{e_{ij}}} \quad (1)$$

Correctness

$$\begin{aligned} L.H.S &= e(\sigma_{ij}, P) \\ &= e((\mathcal{G}K_i + \alpha_{ij}) \cdot r_{e_{ij}} \cdot h_{ij}, P) \\ &= e(h_{ij}, P)^{(\mathcal{G}K_i + \alpha_{ij}) \cdot r_{e_{ij}}} \quad (\text{bilinearity}) \\ &= (e(h_{ij}, P)^{(\mathcal{G}K_i + \alpha_{ij})})^{r_{e_{ij}}} \\ &= (e(h_{ij}, \mathcal{G}K_i \cdot P + \alpha_{ij} \cdot P))^{r_{e_{ij}}} \quad \text{since, } e(h_{ij}, P)^x = e(h_{ij}, x \cdot P) \\ &= (e(h_{ij}, P_{pub} + \alpha_{ij} \cdot P))^{r_{e_{ij}}} \quad \text{since } P_{pub} = \mathcal{G}K_i \cdot P \\ &= e(h_{ij}, P_{pub} + \alpha_{ij} \cdot P)^{r_{e_{ij}}} \\ &= R.H.S. \end{aligned}$$

Case 2. The fog node $\mathcal{FN}_i$ requires verifying multiple messages.

When the fog node $\mathcal{FN}_i$ requires verifying multiple messages, it needs to execute the aggregated batch verification process. When $\mathcal{FN}_i$ receives m signatures $\{(\sigma_{ij}, m_{ij}, TID_{e_{ij}}, T_{ij})\}_{i=1}^m$, it verifies that whether pairing Equation (2) holds or not.

$$e(\sum_{j=1}^m \sigma_{ij}, P) = \prod_{j=1}^m e(h_{ij}, P_{pub} + \alpha_{ij} \cdot P)^{r_{e_{ij}}} \quad (2)$$

Correctness

$$\begin{aligned} L.H.S &= e(\sum_{j=1}^m \sigma_{ij}, P) \\ &= e(\sum_{j=1}^m (\mathcal{G}K_i + \alpha_{ij}) \cdot r_{e_{ij}} \cdot h_{ij}, P) \\ &= \prod_{j=1}^m e((\mathcal{G}K_i + \alpha_{ij}) \cdot r_{e_{ij}} \cdot h_{ij}, P) \quad \text{By bilinearity} \\ &= \prod_{j=1}^m e(h_{ij}, P)^{(\mathcal{G}K_i + \alpha_{ij}) \cdot r_{e_{ij}}} \\ &= \prod_{j=1}^m e(h_{ij}, (\mathcal{G}K_i + \alpha_{ij}) \cdot P)^{r_{e_{ij}}} \quad \text{since } e(h_{ij}, P)^x = e(h_{ij}, x \cdot P) \\ &= \prod_{j=1}^m e(h_{ij}, (\mathcal{G}K_i \cdot P + \alpha_{ij} \cdot P))^{r_{e_{ij}}} \\ &= \prod_{j=1}^m e(h_{ij}, P_{pub} + \alpha_{ij} \cdot P)^{r_{e_{ij}}} \quad \text{since } P_{pub} = \mathcal{G}K_i \cdot P \\ &= R.H.S. \end{aligned}$$

4.6. Group Key Update Phase

Whenever the group membership changes, the group key must be updated. During this phase, our protocol mainly focuses on nodes joining/leaving the group encompassing four distinct scenarios. Case (1): Edge node(s) joining the group. Case (2): Edge node(s) exiting the group. Case (3): Fog node(s) joining the group. Case (4): Fog node(s) leaving the group.

4.6.1. Single or Multiple Edge Nodes Join

It is assumed that the group has ℓ edge nodes. When k edge nodes join, it becomes $\ell + k$ edge nodes in the group. After that, the fog node $\mathcal{FN}_i$ updates the group key encryption key δ_i^m . The fog node $\mathcal{FN}_i$ updates and encrypts the group key $\mathcal{GK}_i'$. It generates the ciphertext $CT'_{\mathcal{GK}_i'}$ and sends it to all edge nodes available in the group. The steps of the procedure are outlined below.

Step 1. First, $\mathcal{FN}_i$ computes the modified group key encryption key $\delta_i^{\ell+k}$ as $\delta_i^{\ell+k} = \delta_i^\ell + \sum_{j=\ell+1}^{\ell+k} E_{u_{ij}}$. Here, δ_i^ℓ represents the old group key encryption key before joining the new edge nodes and $\delta_i^{\ell+k}$ represents the modified group key encryption key after joining the new k edge nodes.

Step 2. Next, the fog node $\mathcal{FN}_i$ updates the group key $\mathcal{GK}_i'$, where $\mathcal{GK}_i' < \min\left(\left\{p_{f_i}\right\} \cup \left\{p_{e_{ij}}\right\}_{i=1}^{\ell+k}\right)$.

Step 3. $\mathcal{FN}_i$ updates M_i^ℓ as $M_i^{\ell+k} = M_i^\ell \times \prod_{j=\ell+1}^{\ell+k} (p_{e_{ij}})$.

Step 4. $\mathcal{FN}_i$ recomputes a new $CT'_{\mathcal{GK}_i'}$ as $CT'_{\mathcal{GK}_i'} = \mathcal{GK}_i' \times \delta_i^{\ell+k} \bmod M_i^{\ell+k}$. Then, $\mathcal{FN}_i$ transmits $CT'_{\mathcal{GK}_i'}$ to all $\ell + k$ edge nodes available in the group.

Step 5. Upon receiving $CT'_{\mathcal{GK}_i'}$ from $\mathcal{FN}_i$, the new k nodes and old edge nodes can retrieve the updated $\mathcal{GK}_i'$ from $CT'_{\mathcal{GK}_i'}$ as $\mathcal{GK}_i' = CT'_{\mathcal{GK}_i'} \times \delta_i^{-1} \bmod p_{e_{ij}}, \forall i = 1, 2, 3, \dots, \ell + k$. In this case, both newly joined edge nodes and existing edge nodes derive the same updated group key $\mathcal{GK}_i'$.

4.6.2. Single or Multiple Edge Nodes Leave

It is assumed that the group initially has ℓ edge nodes. It is also assumed that k edge nodes ($1 \leq k < \ell$) depart the group. When k edge nodes leave, the number of edge nodes in the group decreases to $\ell - k$. After that, $\mathcal{FN}_i$ updates the group key $\mathcal{GK}_i''$ and computes the modified group key encryption key $\delta_i^{\ell-k}$. Furthermore, $\mathcal{FN}_i$ computes the updated $CT''_{\mathcal{GK}_i''}$ for the updated $\mathcal{GK}_i''$. The procedure is outlined in the steps below.

Step 1. $\mathcal{FN}_i$ reconstructs the group key encryption key $\delta_i^{\ell-k}$ as $\delta_i^{\ell-k} = \delta_i^{\ell+k} - \sum_{i=\ell+1}^{\ell+k} E_{u_{ij}}, k < m$.

Step 2. $\mathcal{FN}_i$ updates the value of M_i^ℓ as $M_i^{\ell-k} = M_i^\ell / \prod_{i=\ell+1}^{\ell+k} p_{e_{ij}}$.

Step 3. Next, the fog node $\mathcal{FN}_i$ updates the group key $\mathcal{GK}_i''$, where $\mathcal{GK}_i'' < \min\left(\left\{p_{f_i}\right\} \cup \left\{p_{e_{ij}}\right\}_{i=1}^{\ell-k}\right)$.

Step 4. $\mathcal{FN}_i$ recomputes a new ciphertext message $CT''_{\mathcal{GK}_i''}$ as $CT''_{\mathcal{GK}_i''} = \mathcal{GK}_i'' \times \delta_i^{\ell-k} \bmod M_i^{\ell-k}$.

Step 5. Upon receiving $CT''_{\mathcal{GK}_i''}$ from $\mathcal{FN}_i$, the remaining edge nodes in the group can obtain updated $\mathcal{GK}_i''$ as $\mathcal{GK}_i'' = CT''_{\mathcal{GK}_i''} \times \delta_i^{-1} \bmod p_{e_{ij}}, \forall i = 1, 2, \dots, \ell - k$. The k edge nodes that left the group, on the other hand, cannot retrieve it, even if they have received a ciphertext $CT''_{\mathcal{GK}_i''}$. It can be expressed as follows:

$$\mathcal{GK}_i'' \neq CT''_{\mathcal{GK}_i''} \times \delta_i^{-1} \bmod p_{e_{ij}}, \forall i = \ell - k + 1, \ell - k + 2, \ell - k + 3, \dots, \ell.$$

Therefore, the edge nodes that have left the group cannot derive an updated $\mathcal{G}K_i''$, whereas those that have persisted in the group can obtain it.

5. Security Analysis

We perform a thorough informal security verification to evaluate the security strength of our proposed work. Formal security verification is performed using the ProVerif tool and the random oracle model.

5.1. Informal Security Analysis

Informal security analysis examines the robustness of our suggested protocol. The analysis demonstrates that the suggested work guarantees both backward and forward secrecy. The suggested method also offers protection against a number of well-known attacks and meets all the security needs desired for decentralized edge–fog–cloud computing networks.

5.1.1. Forward Secrecy

In the proposed protocol, forward secrecy is preserved, as an edge node EN_{ij} that has left the group is unable to obtain the future group keys. In our protocol, when an edge node EN_{ij} has left the group, the fog node $\mathcal{FN}_i$ updates both the group key $\mathcal{G}K_i''$ and group key encryption key $\delta_i^{\ell-j} = \delta_i^\ell - E_{u_{ij}}$. Further, it computes new ciphertext $CT_{\mathcal{G}K_i''}'' = \mathcal{G}K_i'' \times \delta_i^{\ell-j} \bmod M_i^{\ell-j}$, where $M_i^{\ell-j} = \frac{M_i^\ell}{p_{e_{ij}}}$. The departed EN_{ij} cannot decrypt the updated group key even if it gets a ciphertext $CT_{\mathcal{G}K_i''}''$ since its secret parameter $E_{u_{ij}} = \delta_{e_{ij}} \times (M_{e_{ij}})^{\phi(p_{e_{ij}})}$ and $p_{e_{ik}}$ are not utilized to encrypt the updated group key $\mathcal{G}K_i''$. Therefore, our proposed work ensures forward secrecy.

5.1.2. Backward Secrecy

Backward secrecy guarantees that a new edge node EN_{ik} that has joined the group recently cannot decrypt the old group keys. To access old communications, newly joined EN_{ik} may attempt to extract an old group key from the ciphertext $CT_{\mathcal{G}K_i} = (\mathcal{G}K_i \times \delta_i) \bmod M_i^m$. In our work, secret parameters $E_{u_{ik}} = \delta_{e_{ik}} \times (M_{e_{ik}})^{\phi(p_{e_{ik}})}$ and $p_{e_{ik}}$ of newly joined edge nodes are not used in the encryption of the old group key. Thus, our proposed work provides backward secrecy.

5.1.3. Message Integrity

In our proposed work, an $\mathcal{ADV}$ cannot obtain the group key; thus, he/she cannot forge the valid signature component $\sigma_{ij} = (\mathcal{G}K_i + \alpha_{ij}).r_{ij}$. h_{ij} and signature $sig_{ij} = (\sigma_{ij}, TID_{e_{ij}}, T_{ij})$. Thus, if the message and its signature satisfy $e(\sigma_{ij}, P) = e(h_{ij}, P_{pub} + \alpha_{ij}.P)^{r_{e_{ij}}}$, the suggested work guarantees message integrity.

5.1.4. Protection Against Modification Attacks

The $\mathcal{ADV}$ cannot modify the message $\{m_{ij}, \sigma_{ij}, TID_{e_{ij}}, T_{ij}\}$ because the message must fulfil the condition $e(\sigma_{ij}, P) = e(h_{ij}, P_{pub} + \alpha_{ij}.P)^{r_{e_{ij}}}$, where $\sigma_{ij} = (\mathcal{G}K_i + \alpha_{ij}).r_{e_{ij}}.h_{ij}$ and $\alpha_{ij} = h_2(m_{ij} \parallel T_{ij})$. If the message is changed, the fog node detects and rejects it. Thus, the suggested scheme provides protection against modification attacks.

5.1.5. Protection Against Replay Attacks

The proposed protocol uses a timestamp in the signature component $\sigma_{ij} = (\mathcal{G}K_i + \alpha_{ij}).r_{e_{ij}}.h_{ij}$ and signature $sig_{ij} = (\sigma_{ij}, TID_{e_{ij}}, T_{ij})$, where $h_{ij} = h_1(m_{ij} \parallel TID_{e_{ij}} \parallel T_{ij})$ and $\alpha_{ij} =$

$h_2(m_{ij} \parallel T_{ij})$ to resist replay attacks. Since T_{ij} is provided in the signature, it becomes impossible for an ADV to bypass the authentication by replaying the prior signature. Therefore, the suggested protocol is secure against replay attacks.

5.1.6. Protection Against Impersonation Attacks

In order to impersonate a valid edge node, the attacker must send a valid message $\{m_{ij}, \sigma_{ij}, TID_{e_{ij}}, T_{ij}\}$ to the fog node. However, our proposed protocol ensures that no adversary has the ability to construct the signature of the edge node with a non-negligible probability. Therefore, the proposed protocol ensures protection against impersonation attacks.

5.1.7. Data Confidentiality

In our scheme, only the authorized edge node EN_{ij} can extract the GK_i from the ciphertext $CT_{GK_i} = (GK_i \times \delta_i) \bmod M_i^m$ and decrypt the sensitive information. Thus, our proposed scheme ensures data confidentiality.

5.2. Formal Security Verification Using ProVerif Tool

An automated tool called ProVerif is used to verify the security of cryptographic schemes. It supports various cryptographic primitives that are expressed by equations or rewriting rules [24]. In the proposed protocol, the ProVerif analysis focuses on ensuring secrecy features at the symbolic level. In our work, we utilize the ProVerif tool to ensure the security of the group key and other secret parameters. Figure 2 shows the various functions and parameters in the ProVerif tool for the group key distribution and extraction phase of our protocol. Figure 3 shows the working of an event fog and edge. Figure 4 shows the various queries. We performed seven queries to ensure the security of the group key, group key encryption key, private key, and secret parameters utilized by the fog and edge nodes. Figure 5 depicts the result obtained from ProVerif after verification. The verification results clearly show that our protocol does not disclose the group key, group key encryption key, private key, and secret parameters to unauthorized entities. Because ProVerif does not natively handle modular arithmetic and CRT operations, these computations are represented by abstract functions and bitstring equations. Therefore, the ProVerif results provide symbolic (partial) validation of the secrecy features of the proposed key distribution and key extraction procedures.

```
(* Public Channel Definition*)
free c : channel.

(* Private items*)
const pe: bitstring [private].
const se: bitstring [private].
const GK:bitstring [private].
const delta: bitstring [private].
const Mi: bitstring [private].
free CTGK: bitstring.

(* functions *)
fun mod(bitstring, bitstring) : bitstring.
fun mult(bitstring, bitstring) : bitstring.
fun inv(bitstring): bitstring.

equation forall GKey: bitstring ,p_e: bitstring;
mod(mult(mod(mult(GKey, delta),Mi),inv(se)),p_e)=GKey.
```

Figure 2. Terms and functions declaration in ProVerif tool.

```

(*Event*)
event beginFog.
event endFog.
event beginEdge.
event endEdge.

(*Fog*)
let pFog=
event beginFog;
new GK : bitstring;
new delta: bitstring;
new Mi: bitstring;
let CTGK = mod(mult(GK, delta),Mi)in
out ( c , (CTGK));
event endFog.

(*Edge*)
let pEdge=
in(c, (CTGK : bitstring));
event beginEdge;
let GK= mod(mult(CTGK, inv(se)),pe) in
event endEdge.

```

Figure 3. Events in ProVerif tool.

```

query attacker (GK) .
query attacker(delta) .
query attacker(Mi) .
query attacker (pe) .
query attacker (se) .
query inj-event(endFog)==>inj-event(beginFog) .
query inj-event(endEdge)==>inj-event(beginEdge) .

process
((!pFog) | (!pEdge))

```

Figure 4. Queries in ProVerif.

```

Verification summary:

Query not attacker(GK) is true.

Query not attacker(delta) is true.

Query not attacker(Mi) is true.

Query not attacker(pe) is true.

Query not attacker(se) is true.

Query inj-event(endFog) ==> inj-event(beginFog) is true.

Query inj-event(endEdge) ==> inj-event(beginEdge) is true.

```

Figure 5. Verification summary in ProVerif.

5.3. Security Verification Using Random Oracle Model

In this section, we employ the random oracle model, a theoretical black-box model, for formal security verification of our suggested method. When a new query is inputted, the model generates a random output from its output domain. If the same query is provided as an input, the model will return the same result as it did previously.

Reveal works as a random oracle that, without knowing the method used to calculate the hash value y , creates a hash input $(x_1 x_2 \dots x_l)$ of length l from their related hash map $y = (x_1 x_2 \dots x_l)$.

Theorem 1. Suppose an adversary is able to obtain the message $(CT_{\mathcal{G}_{Ki}}, HC, T_{ij})$ from the insecure channel and the one-way hash function for the computation of HC acts as a random oracle. The polynomial time adversary can make q_H queries. The advantage of a polynomial time adversary that runs in time t against our suggested protocol is $adv_{\mathcal{ADV}}^{\text{Suggested}} \leq \frac{q_H+1}{2^l} + adv_{\mathcal{G}_{Ki}}$, which is negligible.

Proof. The following queries are initiated by the adversary $\mathcal{ADV}$:

$Exe(\Pi_{Fog}^i, \Pi_{Edge}^j)$: By executing this query, $\mathcal{ADV}$ can intercept the messages transmitted between the participant instances Π_{Fog}^i and Π_{Edge}^j across the public channel. Passive attacks are implemented by this query.

$Hash(\Pi_{XX}^k, \text{bitstring})$: The hash value of the input bitstring can be obtained by $\mathcal{ADV}$ in this query.

$Test(\Pi_{XX}^k)$: The query is executed by $\mathcal{ADV}$ by tossing a coin $\mathcal{C}$. The correct group key will be obtained by the $\mathcal{ADV}$ if $\mathcal{C}$ returns 1 and an arbitrary string if it does not.

$Reveal(\Pi_{Fog}^k)$: The adversary employs this query to reveal the group key provided by Π_{Fog}^k to its companion in the present session.

A series of games $\langle G_0, G_1, G_2, G_3, G_3 \rangle$ are used to explain the actual attack experiment. In the game G_i , an event in which $\mathcal{ADV}$ accurately predicts the bit $\mathcal{C}$ is denoted by SUC_i . $Pro[SUC_i]$ represents the corresponding probability. The following are the descriptions of these games:

G_0 : In this game, the $\mathcal{ADV}$ does not initiate any queries. This implies that the real procedure and the original game are identical. Then, we have:

$$adv_{\mathcal{A}}^{\text{Suggested}} = \left| Pro[SUC_0] - \frac{1}{2} \right|$$

G_1 : The $Exe(\Pi_{Fog}^i, \Pi_{Edge}^j)$ query is employed in this game to simulate an $\mathcal{ADV}$ eavesdropping attack. The adversary uses $Reveal(\Pi_X^k)$ to disclose the group key provided by Π_{Fog}^k to its companion in the present session. The $Test$ query is executed by the $\mathcal{ADV}$ at the end of the game. The query output must be verified by the $\mathcal{ADV}$ to ascertain whether it is a random number or an actual $\mathcal{G}_{Ki}$. Fog mode chooses $\mathcal{G}_{Ki}$; $Edge$ can get $\langle CT_{\mathcal{G}_{Ki}} \rangle$. In order to compute $CT_{\mathcal{G}_{Ki}}$, the $\mathcal{ADV}$ needs the private key $p_{e_{ij}}$ and secret parameter $s_{e_{ij}}^{-1}$ of the edge node. Eavesdropping on the message does not enhance the $\mathcal{ADV}$'s chances of obtaining the group key. Hence, we have:

$$Pro[SUC_0] = Pro[SUC_1]$$

G_2 : The challenger discards all replayed timestamps T_{ij}^* . When T_{ij}^* is outside the maximum transmission delay, the challenger aborts. The difference from Game 1 is only the implementation of replay protection. Thus, we have:

$$Pro[SUC_2] = Pr[SUC_1]$$

G_3 : Assume that the $\mathcal{ADV}$ makes at most q_H hash queries. Each query is independent. Consequently, the adversary can only guess the correct l -bit hash input value at random. The probability of correctly guessing $\mathcal{G}_{K_i}$ is:

$$Pro[SUC_3] - Pr[SUC_2] \leq \frac{q_H + 1}{2^l}$$

G_4 : Consider that an adversary can successfully perform a hash query. To generate this query, the adversary must already know the correct group key $\mathcal{G}_{K_i}$. However, recovering $\mathcal{G}_{K_i}$ from the captured message requires successful execution of the suggested group key extraction process without possessing any valid private key and secret parameter. By assumption, this problem is computationally infeasible. Let adv_{GK} denote the maximum success probability of solving the suggested group key extraction problem. Therefore,

$$|Pro[SUC_4] - Pr[SUC_3]| \leq adv_{GK}$$

By the triangle inequality, we get:

$$\begin{aligned} adv_{ADV}^{Suggested} &\leq |Pro[SUC_0] - Pro[SUC_4]| \\ &\leq |Pro[SUC_1] - Pro[SUC_0]| + |Pro[SUC_2] - Pro[SUC_1]| \\ &\quad + |Pro[SUC_3] - Pro[SUC_2]| + |Pro[SUC_4] - Pro[SUC_3]| \\ &\leq \frac{q_H + 1}{2^l} + adv_{GK} \end{aligned}$$

Since q_H is polynomial, 2^l is exponential and adv_{GK} is negligible, the adversary advantage to get the group key is negligible. Thus, our protocol provides resistance against group key disclosure. $\square$

Theorem 2. Suppose an $\mathcal{ADV}$ can generate a valid signature of an edge node in polynomial time $\mathcal{T}$ with a non-negligible advantage ϵ ; then, there is a challenger $\mathcal{C}$ capable of solving the co-CDHP with a non-negligible probability ϵ in polynomial time $\mathcal{T}$.

Proof. The simulator $\mathcal{C}$ executes the initialization process and obtains the public parameter $Q = r.P$. The simulator creates the three lists: $\mathcal{LH}_1$, $\mathcal{LH}_2$ and $\mathcal{LS}$. $\mathcal{LH}_1$ and $\mathcal{LH}_2$ store the values returned by random oracle $\mathcal{H}_1$ and $\mathcal{H}_2$. List $\mathcal{LS}$ stores the values return by sign queries.

$\mathcal{H}_1$ queries: When the $\mathcal{ADV}$ executes a hash query with the input m_{ij} , $TID_{e_{ij}}$, T_{ij} , $\mathcal{C}$ examines whether the input appears in list $\mathcal{LH}_1$. If there exists a tuple $(m_{ij}, TID_{e_{ij}}, T_{ij}, h_{ij})$ in list $\mathcal{LH}_1$, $\mathcal{ADV}$ gets h_{ij} from $\mathcal{C}$. If not, $\mathcal{C}$ chooses a random number $h_{ij} \in \mathbb{Z}_n^*$ and includes the tuple $(m_{ij}, TID_{e_{ij}}, T_{ij}, h_{ij})$ into list $\mathcal{LH}_1$. Further, the $\mathcal{ADV}$ obtains h_{ij} from $\mathcal{C}$.

$\mathcal{H}_2$ queries: When the $\mathcal{ADV}$ executes a hash query with the input m_{ij} , T_{ij} , $\mathcal{C}$ examines whether the input is present in list $\mathcal{LH}_2$. If there is a tuple m_{ij} , T_{ij} in list $\mathcal{LH}_2$, the $\mathcal{ADV}$ gets α_{ij} from $\mathcal{C}$. If not, $\mathcal{C}$ chooses a random number $\alpha_{ij} \in \mathbb{Z}_n^*$ and adds the tuple $(m_{ij}, T_{ij}, \alpha_{ij})$ into list $\mathcal{LH}_2$. Further, $\mathcal{C}$ returns α_{ij} to the $\mathcal{ADV}$.

Sig queries : During the sign query process, the $\mathcal{ADV}$ and $\mathcal{C}$ communicate at most Q_s times. An $\mathcal{ADV}$ makes a sign query on m_{ij} . Then, $\mathcal{C}$ checks whether $TID_{e_{ij}}$ and $TID_{e_{ij}}^*$ are equal or not. If $TID_{e_{ij}} \neq TID_{e_{ij}}^*$, $\mathcal{C}$ checks whether $(m_{ij}, TID_{e_{ij}}, \sigma_{ij})$ is present in $\mathcal{LS}$ or not. If $(m_{ij}, TID_{e_{ij}}, \sigma_{ij})$ is already in $\mathcal{LS}$, $\mathcal{C}$ returns σ_{ij} to the $\mathcal{ADV}$. Otherwise, $\mathcal{C}$ recomputes $\mathcal{H}_1$, $\mathcal{H}_2$ and σ_{ij} . $\mathcal{C}$ adds $(m_{ij}, TID_{e_{ij}}, \sigma_{ij})$ to $\mathcal{LS}$ and returns σ_{ij} to the $\mathcal{ADV}$. If $TID_{e_{ij}} \neq TID_{e_{ij}}^*$, $\mathcal{C}$ discards the query and ends the game.

Reveal : If the $\mathcal{ADV}$ is able to win the game, it indicates that the $\mathcal{ADV}$ is capable of forging a legitimate signature σ_{ij} with the identity $TID_{e_{ij}}$. when $\mathcal{C}$ responds to the signing query, the $\mathcal{ADV}$ can generate a valid signature in a polynomial time. Now, we get:

$$\sigma_{ij}^* = \mathbb{S} \cdot r_{e_{ij}}^* \cdot h_{ij}^*, \text{ where } \mathbb{S} = (\mathcal{G}K_i + \alpha_{ij})$$

$$\sigma_{ij}^* \cdot r_{e_{ij}}^{*-1} = \mathbb{S} \cdot h_{ij}^*$$

Because the game is played in polynomial time and interactions are limited, $\mathcal{C}$ can solve the co-CDHP in polynomial time with a non-negligible probability. This contradicts the cryptographic hard problem. Therefore, the suggested protocol is secure. $\square$

6. Performance Analysis

This section presents the security features and performance analysis of our protocol and a comparison with existing state-of-the-art protocols [25–30] in terms of the computation and communication costs for message signing, single message verification, and batch verification. The performance analysis is conducted based on the approach proposed by Chen et al. [30]. The computation time of edge and fog nodes is evaluated using the Python programming language.

For experimental evaluation of our protocol, we used a personal computer with an eighth-generation Intel Core i5 processor, 8 GB RAM, and Windows 11 OS using PyCharm Community Edition 2024.2.0.1. Table 2 presents the cryptographic primitives used in the protocol along with the execution time, and Table 3 summarizes the corresponding size of the parameters used.

Table 2. Execution time of cryptographic primitives.

Cryptographic Operations	Time Taken (ms)
t_h : hash operation	0.005
t_{gp} : Bilinear pairing operation	14.293
t_{em} : Scalar multiplication	4.141
t_e : Exponentiation operation	1.388
t_{pa} : Point addition	0.33
t_{mtp} : Map-to-point hash operation	11.875
t_m : Multiplication	0.015
t_{sm} : Small scalar multiplication	0.387

Table 3. Parameter sizes.

Parameter	Size (Byte)
$\mathbb{G}$	128
$\mathbb{Z}_n^*$	20
$\mathbb{G}_T$	128
T_{ij}	4

6.1. Security Feature Comparison

This section compares the security and functionality features of the proposed protocol to existing schemes [25–30]. Table 4 demonstrates that the related protocols [25–30] provide limited security features as compared to the proposed protocol. Bayat et al. [25] suffers from non-repudiation of batch messages and data confidentiality. Also, it does not provide the ability for secure group communication nor protection against modification attacks. Similarly, Wang et al. [26] and Feng et al. [28] lack similar security features but are resistant against modification attacks. Mei et al. [27] satisfies all the security and functionality

features as mentioned in Table 4 except signature forgery and secure group communication. Qi et al. [29] and Chen et al. [30] achieve most of the security and functionality features; however, they lack secure group communication. The proposed protocol supports non-repudiation and data confidentiality and is resistant against replay and impersonation attacks. Also, it supports message integrity and group communication while preventing signature forgery. Therefore, the suggested protocol outperforms the relevant protocols in terms of security and functionality.

Table 4. Comparison of security and functionality features.

Protocol(s)	Security Features							
	SFF_1	SFF_2	SFF_3	SFF_4	SFF_5	SFF_6	SFF_7	SFF_8
Bayat et al. [25]	×	×	×	✓	✓	✓	✓	×
Wang et al. [26]	✓	×	×	✓	✓	✓	✓	×
Mei et al. [27]	✓	✓	✓	✓	✓	×	✓	×
Feng et al. [28]	✓	×	×	✓	✓	✓	✓	×
Qi et al. [29]	✓	✓	✓	✓	✓	✓	✓	×
Chen et al. [30]	✓	✓	✓	✓	✓	✓	✓	×
Our	✓	✓	✓	✓	✓	✓	✓	✓

Note: SFF_1 : protection against modification attacks, SFF_2 : data confidentiality, SFF_3 : non-repudiation of batch messages, SFF_4 : message integrity, SFF_5 : protection against impersonation attacks, SFF_6 : protection against signature forgery, SFF_7 : protection against replay attacks, SFF_8 : secure group communication.

6.2. Computation Cost

In this section, a comparative analysis of the computational cost of our protocol and the state-of-the-art protocols [25–30] is presented and summarized in Table 5. In our protocol, the edge node performs one point addition and one scalar multiplication operations for signing the message. Therefore, the total time taken for message signing is $t_{em} + t_{pa} + 2t_h + t_{sm} \approx 4.868$ ms. The single verification process at the fog node involves one scalar multiplication, two bilinear pairing, one point addition, and two hash operations. Thus, the time taken in this phase is $2t_{\beta p} + t_{em} + t_{pa} + 2t_h \approx 33.067$ ms. The processing time of the batch verification of n messages is $2t_{\beta p} + t_{em} + 2nt_{pa} + 2nt_h \approx 32.727 + 0.67n$ ms, where $n \leq 5$. The computation cost of the related protocols [25–30] is computed in a similar manner. Figure 6, Figure 7, and Figure 8 illustrate the cost for signing the message, verification of the single message, and verification of the batch messages, respectively. This clearly highlights the superiority of our protocol over the related protocols [25–30].

Table 5. Comparison of computation cost.

Protocol(s)	Message Sign	Single Verification	Batch Verification
Bayat et al. [25]	$5t_{em} + t_{mtp} + t_{pa} + 2t_h$ ≈ 32.95	$3t_{\beta p} + t_{mtp} + t_{em} + t_h$ ≈ 58.9	$3t_{\beta p} + nt_{mtp} + nt_{em} + (3n - 3)t_{pa} + nt_h$ $\approx 41.889 + 17.011n$
Wang et al. [26]	$5t_{em} + 2t_{mtp} + t_{pa}$ ≈ 44.785	$3t_{\beta p} + t_{mtp} + t_{em} + t_m$ ≈ 58.91	$3t_{\beta p} + nt_{mtp} + nt_{em} + (3n - 3)t_{pa} + t_m$ $\approx 41.889 + 17.021n$
Mei et al. [27]	$4t_{em} + 2t_{mtp}$ ≈ 40.31	$4t_{\beta p} + 2t_{em}$ ≈ 65.454	$4t_{\beta p} + 2nt_{em}$ $\approx 57.172 + 8.282n$
Feng et al. [28]	$2t_{\beta p} + 11t_{em} + 12t_e$ ≈ 90.993	$4t_{\beta p} + 10t_{em} + 10t_e$ ≈ 112.462	$4t_{\beta p} + (6n - 1)t_{em}$ $\approx 50.031 + 24.846n$
Qi et al. [29]	$t_{\beta p} + 4t_{em}$ ≈ 30.857	$2t_{\beta p} + 4t_{em}$ ≈ 45.15	$(n + 1)t_{\beta p} + 4nt_{em}$ $\approx 14.293 + 30.857n$
Chen et al. [30]	$2t_{em} + t_{pa}$ ≈ 8.612	$2t_{\beta p} + t_{em} + t_{pa} + t_e$ ≈ 34.445	$2t_{\beta p} + nt_{em} + 2nt_{sm} + (3n - 2)t_{pa} + nt_m + nt_e$ $\approx 27.926 + 7.308n$
Our	$t_{em} + t_{pa} + 2t_h + t_{sm}$ ≈ 4.868	$2t_{\beta p} + t_{em} + t_{pa} + 2t_h$ ≈ 33.067	$2t_{\beta p} + t_{em} + 2nt_{pa} + 2nt_h$ $\approx 32.727 + 0.67n$

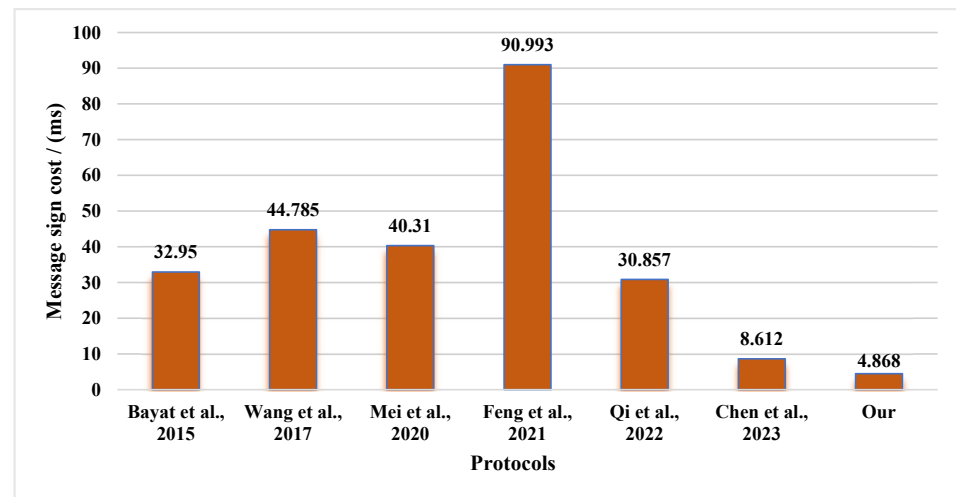


Figure 6. Comparisons of computation cost of related protocols Bayat et al. [25], Wang et al. [26], Mei et al. [27], Feng et al. [28], Qi et al. [29], Chen et al. [30] and our protocol for message signing.

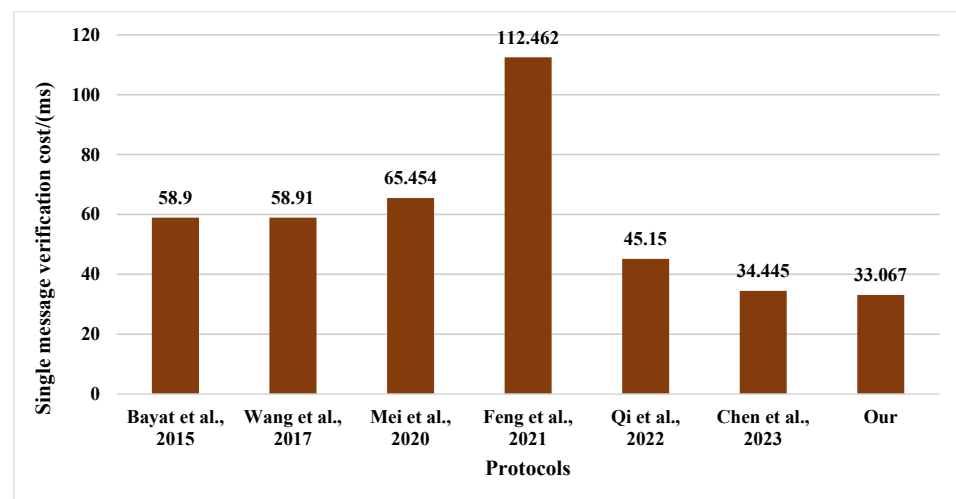


Figure 7. Comparisons of computation cost of related protocols Bayat et al. [25], Wang et al. [26], Mei et al. [27], Feng et al. [28], Qi et al. [29], Chen et al. [30] and our protocol for single verification.

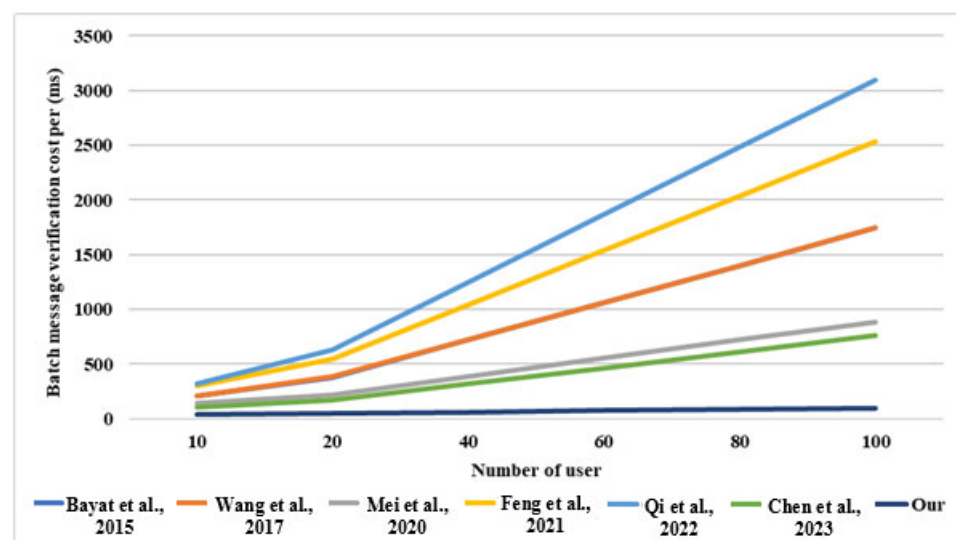


Figure 8. Comparisons of computation cost of related protocols Bayat et al. [25], Wang et al. [26], Mei et al. [27], Feng et al. [28], Qi et al. [29], Chen et al. [30] and our protocol for batch verification.

6.3. Communication Cost

We compare the communication cost of the proposed protocol with related protocols, as listed in Table 6. We assume that the size of the signed message is identical for all protocols; therefore, the communication cost of the signed message is not considered. Bayat et al. [25] illustrate that each vehicle transmits a message $\langle (ID_{i,1}, ID_{i,2}), \sigma_i, T_i \rangle$; therefore, the resulting communication cost is $3|G| + 4 = 388$ bytes. In Wang et al. [26], each vehicle broadcasts the message $\langle (PID_{i,1}, PID_{i,2}), M_i, \sigma_i, PK_{Ri}, T_i \rangle$; therefore, the communication cost is $4|G| + 4 = 516$ bytes. In Mei et al. [27], each vehicle transmits a message $\langle PSUID_i, IDR_s, M_i, \sigma_i, vepk_i, T_i \rangle$; hence, the communication cost is $4|G| + 4 = 516$ bytes. In Feng et al. [28], each vehicle broadcasts a message $\langle Cert_i, \sigma_i, M_i, T_i \rangle$. The communication cost of Feng et al. [28] is $15|G| + 4 = 1924$ bytes. In Qi et al. [29], each vehicle broadcasts a message $\langle P_{s_i}, PS_s, M_i, PK_i, T_i \rangle$. Hence, the communication cost of Qi et al. [29] is $3|G| + 4 = 388$ bytes. In the Chen et al. [30], each vehicle broadcasts a message $\langle PID_{ik}, R_{ik}, W_i, l_i, M_i, \sigma_i, T_i \rangle$. Therefore, the total communication cost of Chen et al. [30] is $4|G| + 4 = 516$ bytes. In our proposed protocol each edge node broadcasts a message $\langle m_{i,j}, \sigma_{i,j}, TID_{if}, T_{i,j} \rangle$, where $\sigma_{i,j}, TID_{if} \in \mathbb{G}$ and T_i is a timestamp. The total communication cost of our protocol is $2|G| + 4 = 260$ bytes. Table 6 clearly highlights that our proposed protocol is better as compared to the related protocols [25–30], as demonstrated in Figure 9.

Table 6. Comparison of communication cost.

Protocol(s)	Transmitted Message	Total Cost (Byte)
Bayat et al. [25]	$\langle (ID_{i,1}, ID_{i,2}), \sigma_i, T_i \rangle$	$3 G + 4 = 388$
Wang et al. [26]	$\langle (PID_{i,1}, PID_{i,2}), M_i, \sigma_i, PK_{Ri}, T_i \rangle$	$4 G + 4 = 516$
Mei et al. [27]	$\langle PSUID_i, IDR_s, M_i, \sigma_i, vepk_i, T_i \rangle$	$4 G + 4 = 516$
Feng et al. [28]	$\langle Cert_i, \sigma_i, M_i, T_i \rangle$	$15 G + 4 = 1924$
Qi et al. [29]	$\langle P_{s_i}, PS_s, M_i, PK_i, T_i \rangle$	$3 G + 4 = 388$
Chen et al. [30]	$\langle PID_{ik}, R_{ik}, W_i, l_i, M_i, \sigma_i, T_i \rangle$	$4 G + 4 = 516$
Our	$\langle m_{i,j}, \sigma_{i,j}, TID_{if}, T_{i,j} \rangle$	$2 G + 4 = 260$

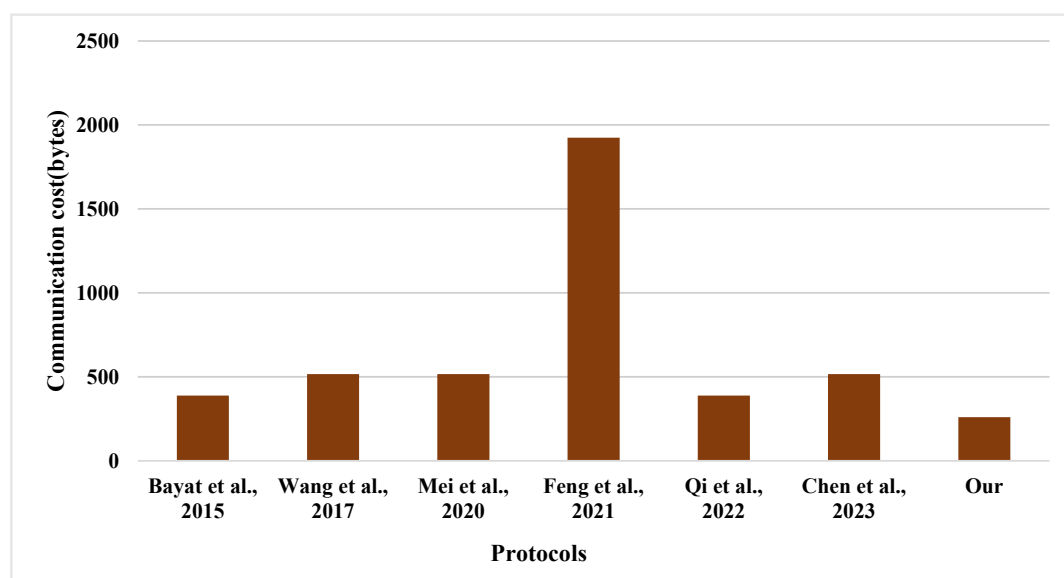


Figure 9. Comparisons of communication cost of related protocols Bayat et al. [25], Wang et al. [26], Mei et al. [27], Feng et al. [28], Qi et al. [29], Chen et al. [30] and our protocol.

6.4. Performance Validation Using Wokwi

In this section, we use the Wokwi platform with an ESP32 virtual environment to simulate and implement the proposed protocol in order to validate it. Wokwi is a browser-

based simulator for embedded system development without physical hardware. ESP32 Devkitc (v4) was chosen due to its simulated capability for cryptographic operations and compatibility with fog-edge network hardware requirements. Single message verification, batch message verification, group key distribution and retrieval were implemented in MicroPython (v1.28.0). Figures 10–12 demonstrate the correct execution of group key distribution, group key retrieval, single message verification and batch verification.



Figure 10. Wokwi ESP32 simulation results of group key distribution and retrieval.

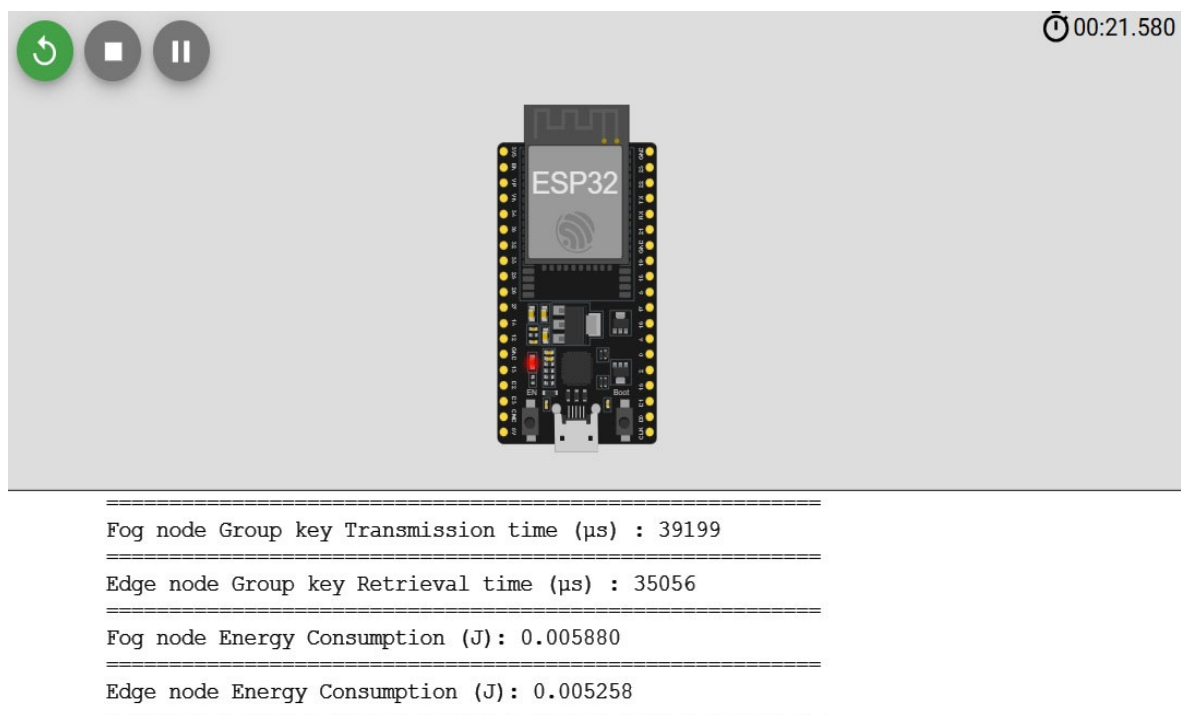


Figure 11. Simulation result showing latency time and energy consumption for group key distribution and retrieval.



Figure 12. Simulation result showing latency time and energy consumption for single message verification and batch verification.

The simulation was executed for 50 iterations in order to compute the performance. The average latency and energy consumption during group key distribution, group key retrieval, single message verification and batch verification are shown in Table 7. The average time for group key distribution and retrieval was 27,375 μ s, and 7384 μ s respectively. The average time for single message verification by the fog node was 6384 μ s, while the average time for batch verification by the fog node was 12,886 μ s. Based on ESP32's active current consumption of $I = 30$ mA ($=0.03$ A) at $V = 5.0$ V, the average energy consumption for group key distribution and retrieval was 0.004125 (J) and 0.001064 (J) respectively. The average energy usage for single message verification was 0.001203 (J), while the average energy consumption for batch verification was 0.001882 (J). The energy consumption is calculated using the standard relation: $\text{energy} = \text{voltage} \times \text{current} \times \text{execution time}$, where the execution time is obtained from the measured latency as: $\text{energy} = \text{voltage} \times \text{current} \times (\text{latency}/1,000,000)$. The correctness of group key distribution and retrieval, as well as single message verification and batch verification, was tested on 50 test runs (40 valid and 10 invalid credential sets). The protocol accepted 100% of valid credentials and rejected 100% of invalid attempts, with no false acceptances or rejections. The simulation results show that the suggested protocol is efficient, robust, and may be suited for use in edge–fog–cloud computing networks.

Table 7. The average latency and energy consumption using MicroPython on ESP32.

Phases	Average Latency (μ s)	Average Energy Consumption (J)
Group key distribution by fog node	27,375	0.004125
Group key retrieval by edge node	7384	0.001064
Single message verification	6384	0.001203
Batch message verification	12,886	0.001882

7. Conclusions

In this paper, we present a lightweight and secure key distribution protocol with update and batch verification for decentralized edge–fog–cloud computing networks that

eliminates the need to manage distinct and independent sets of moduli equations with multiple mathematical structures for different network layers. Our protocol uses CRT to distribute and update the group key without computing the inverse element, and by using a unique secret parameter for each edge node, it provides two-factor modular mechanism for masking the group key. By integrating elliptic curve cryptography, bilinear pairings, Euler's totient function, and per-signature randomness, our protocol achieves efficient single and batch signature verification while preserving robust security. The proposed protocol significantly reduces verification overhead at the fog layer by requiring a constant number of pairing operations for batch verification. Formal security analysis shows that our protocol is secure under the hardness assumptions of the co-Computational Diffie–Hellman (co-CDH) problem in the random oracle model. Formal security verification using the ProVerif tool confirms its robustness against various attacks. Furthermore, simulation on the Wokwi platform with ESP32 confirmed its correctness, efficiency, and suitability for cloud–fog–edge development. The performance analysis demonstrates that our protocol outperforms existing protocols in terms of computational and communicational efficiency, indicating that it might be suitable for large-scale and latency-sensitive edge–fog–cloud computing networks.

Author Contributions: Conceptualization, N.K.U. and V.K.; methodology, N.K.U. and V.K.; software, N.K.U.; validation, V.K. and S.P.; formal analysis, N.K.U. and V.K.; investigation, S.P. and V.K.; resources, V.K.; data curation, S.P.; writing—original draft preparation, N.K.U. and V.K.; visualization, S.P.; project administration, S.P.; funding acquisition, N.K.U. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

References

1. Chen, H.; Chang, C.C. A novel (T, N) secret sharing scheme based upon Euler's theorem. *Secur. Commun. Netw.* **2019**, *2019*, 2387358. [CrossRef]
2. Wu, S.; Zhang, A.; Luo, H. EF-CRT: Group key update and batch verification based on Euler function and Chinese remainder theorem for edge-fog computing networks. *IEEE Syst. J.* **2023**, *17*, 5987–5998.
3. Zhang, J.; Cui, J.; Zhong, H.; Chen, Z.; Liu, L. PA-CRT: Chinese remainder theorem based conditional privacy-preserving authentication scheme in vehicular ad-hoc networks. *IEEE Trans. Dependable Secur. Comput.* **2019**, *18*, 722–735.
4. Vijayakumar, P.; Azees, M.; Kannan, A.; Deborah, L.J. Dual authentication and key management techniques for secure data transmission in vehicular ad hoc networks. *IEEE Trans. Intell. Transp. Syst.* **2015**, *17*, 1015–1028. [CrossRef]
5. Vijayakumar, P.; Bose, S.; Kannan, A. Chinese remainder theorem based centralised group key management for secure multicast communication. *IET Inf. Secur.* **2014**, *8*, 179–187. [CrossRef]
6. Guo, C.; Chang, C.C. An authenticated group key distribution protocol based on the generalized Chinese remainder theorem. *Int. J. Commun. Syst.* **2014**, *27*, 126–134.
7. Kumar, V.; Kumar, R.; Pandey, S.K. A secure and robust group key distribution and authentication protocol with efficient rekey mechanism for dynamic access control in secure group communications. *Int. J. Commun. Syst.* **2020**, *33*, e4465. [CrossRef]
8. Kumar, V.; Ahmad, M.; Kumari, A. A secure elliptic curve cryptography based mutual authentication protocol for cloud-assisted TMIS. *Telemat. Inform.* **2019**, *38*, 100–117. [CrossRef]
9. Amanlou, S.; Hasan, M.K.; Bakar, K.A.A. Lightweight and secure authentication scheme for IoT network based on publish–subscribe fog computing model. *Comput. Netw.* **2021**, *199*, 108465. [CrossRef]
10. Kumar, V.; Kumar, R.; Pandey, S.K. A computationally efficient centralized group key distribution protocol for secure multicast communications based upon RSA public key cryptosystem. *J. King Saud Univ.-Comput. Inf. Sci.* **2020**, *32*, 1081–1094. [CrossRef]
11. Wazid, M.; Bagga, P.; Das, A.K.; Shetty, S.; Rodrigues, J.J.; Park, Y. AKM-IoV: Authenticated key management protocol in fog computing-based Internet of vehicles deployment. *IEEE Internet Things J.* **2019**, *6*, 8804–8817. [CrossRef]

12. Awais, S.M.; Yucheng, W.; Mahmood, K.; Badar, H.M.S.; Kharel, R.; Das, A.K. Provably secure fog-based authentication protocol for VANETs. *Comput. Netw.* **2024**, *246*, 110391. [[CrossRef](#)]
13. Ali, H.; Ahmed, I. LAAKA: Lightweight anonymous authentication and key agreement scheme for secure fog-driven IoT systems. *Comput. Secur.* **2024**, *140*, 103770. [[CrossRef](#)]
14. Gupta, K.; Kumar, V.; Prakash, R.; Pal, O.; Sharma, S. SAKM: A Scalable and Adaptive Key Management for Securing Demand-Response Bidirectional Communications in AMI of Next-Generation Smart Grids. *Secur. Priv.* **2025**, *8*, e70076. [[CrossRef](#)]
15. Gupta, K.; Kumar, V.; Prakash, R.; Pal, O. SLMAS: A secure and lightweight mutual authentication scheme for telecare medical information system based on ECC. *J. Supercomput.* **2025**, *81*, 639. [[CrossRef](#)]
16. Hassan, S.R.; Mehmood, A. A Multi-Tier Vehicular Edge-Fog Framework for Real-Time Traffic Management in Smart Cities. *Mathematics* **2025**, *13*, 3947. [[CrossRef](#)]
17. Cui, J.; Tao, X.; Zhang, J.; Xu, Y.; Zhong, H. HCPA-GKA: A hash function-based conditional privacy-preserving authentication and group-key agreement scheme for VANETs. *Veh. Commun.* **2018**, *14*, 15–25. [[CrossRef](#)]
18. Shao, J.; Lin, X.; Lu, R.; Zuo, C. A threshold anonymous authentication protocol for VANETs. *IEEE Trans. Veh. Technol.* **2015**, *65*, 1711–1720. [[CrossRef](#)]
19. Zhong, H.; Han, S.; Cui, J.; Zhang, J.; Xu, Y. Privacy-preserving authentication scheme with full aggregation in VANET. *Inf. Sci.* **2019**, *476*, 211–221. [[CrossRef](#)]
20. Ali, I.; Lawrence, T.; Omala, A.A.; Li, F. An efficient hybrid signcryption scheme with conditional privacy-preservation for heterogeneous vehicular communication in VANETs. *IEEE Trans. Veh. Technol.* **2020**, *69*, 11266–11280. [[CrossRef](#)]
21. Chen, Z.; Zhang, L.J.; Wang, Y.M.; Huang, J.W.; Huang, D.R. A group signature scheme based on Chinese residual theorem. *Acta Electron. Sin.* **2004**, *32*, 1062–1065.
22. Zheng, X.; Huang, C.-T.; Matthews, M. Chinese remainder theorem-based group key management. In *Proceedings of the 45th annual ACM Southeast Conference*; Association for Computing Machinery: New York, NY, USA, 2007; pp. 266–271.
23. Agrawal, S.; Das, M. Mutual healing enabled group-key distribution protocol in wireless sensor networks. *Comput. Commun.* **2017**, *112*, 131–140. [[CrossRef](#)]
24. Blanchet, B.; Smyth, B.; Cheval, V.; Sylvestre, M. ProVerif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial. *Version* **2018**, *16*, 05–16.
25. Bayat, M.; Barmshoory, M.; Rahimi, M.; Aref, M.R. A secure authentication scheme for VANETs with batch verification. *Wirel. Netw.* **2015**, *21*, 1733–1743.
26. Wang, S.; Yao, N. LIAP: A local identity-based anonymous message authentication protocol in VANETs. *Comput. Commun.* **2017**, *112*, 154–164. [[CrossRef](#)]
27. Mei, Q.; Xiong, H.; Chen, J.; Yang, M.; Kumari, S.; Khan, M.K. Efficient certificateless aggregate signature with conditional privacy preservation in IoV. *IEEE Syst. J.* **2020**, *15*, 245–256. [[CrossRef](#)]
28. Feng, X.; Shi, Q.; Xie, Q.; Wang, L. P2BA: A privacy-preserving protocol with batch authentication against semi-trusted RSUs in vehicular ad hoc networks. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 3888–3899. [[CrossRef](#)]
29. Qi, J.; Gao, T.; Deng, X.; Zhao, C. A pseudonym-based certificateless privacy-preserving authentication scheme for VANETs. *Veh. Commun.* **2022**, *38*, 100535. [[CrossRef](#)]
30. Chen, S.; Liu, Y.; Ning, J.; Zhu, X. BASRAC: An efficient batch authentication scheme with rule-based access control for VANETs. *Veh. Commun.* **2023**, *40*, 100575. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.