sensors

MDPI

Article

A Blockchain-Based Authentication and Dynamic Group Key Agreement Protocol

Zisang Xu [1,*], Feng Li [1], Han Deng [2], Minfu Tan [2], Jixin Zhang [3] and Jianbo Xu [4]

[1] Computer and Communication Engineer Institute, Changsha University of Science and Technology, Changsha 410114, China; lif@csust.edu.cn
[2] Big data development and Research Center, Guangzhou College of Technology and Business, Guangzhou 528138, China; aviva_daisy@163.com (H.D.); 13117329002@163.com (M.T.)
[3] School of Computer Science, Hubei University of Technology, Wuhan 430068, China; zhangjixin@hnu.edu.cn
[4] School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China; jbxu@hnust.edu.cn
* Correspondence: xzsszx111@sina.com

Received: 26 July 2020; Accepted: 24 August 2020; Published: 27 August 2020

Abstract: With the rapid development of mobile networks, there are more and more application scenarios that require group communication. For example, in mobile edge computing, group communication can be used to transmit messages to all group members with minimal resources. The group key directly affects the security of the group communication. Most existing group key agreement protocols are often flawed in performance, scalability, forward or backward secrecy, or single node failure. Therefore, this paper proposes a blockchain-based authentication and dynamic group key agreement protocol. With our protocol, each group member only needs to authenticate its left neighbor once to complete the authentication, which improved authentication efficiency. In addition, our protocol guarantees the forward secrecy of group members after joining the group and the backward secrecy of group members after leaving the group. Based on blockchain technology, we solve the problem of single node failure. Furthermore, we use mathematics to prove the correctness and security of our protocol, and the comparison to related protocols shows that our protocol reduces computation and communication costs.

Keywords: authentication; blockchain; cryptography; group key agreement

1. Introduction

With the rapid development of mobile networks, the secure transmission of data is no longer limited to both parties in communication, but is required in group communication. Group communication can transmit messages to all group members with minimal resources [1]. This is because the sending of the message only needs to be broadcast once within the group, instead of sending the same message to all group members one by one, which results in a significant increase in communication efficiency. In order to reduce network latency in cloud computing, Mobile Edge Computing (MEC) [2] is introduced. In MEC, the Small Cell Manager (SCM) needs to dynamically and elastically manage the computing and/or storage resources of multiple Small Cell Clouds (SCCs) [3]. Therefore, the use of group communication can greatly improve the efficiency of MEC.

In order to provide a reliable and scalable group communication service, the most basic and critical security issue is access control [4]. In most cases, access control can be achieved by encrypting or decrypting messages, because only legitimate group members can get the key and use this to decrypt the ciphertext to access the messages [5,6]. Therefore, in order to ensure efficient and secure communication in the group, all group members need to use the same session key, which is also called

the group key. This means that the group key agreement protocol directly affects the security and efficiency of group communication.

In recent years, many researchers have proposed many authentication and group key agreement protocols. However, according to the research of [7,8], we found that most of these protocols have the following defects: (1) performance: before the group key is negotiated, mutual authentication is usually required between group members, which may consume much of the computation and communication costs; (2) scalability: the protocol cannot efficiently handle the joining or leaving of group members, which results in the poor scalability of the protocol; (3) forward or backward secrecy: it is difficult to guarantee the forward or backward secrecy of group members after joining or leaving the group, such as [9]; (4) single node failure: since most existing protocols store the registration information of all group members in a single node, these protocols are vulnerable to the problem of single node failure. Therefore, these protocols are not suitable for MEC. We design a blockchain-based authentication and dynamic group key agreement protocol to solve the above problems in this paper. The protocol has the following characteristics:

- In our protocol, before negotiating the group key, each group member only needs to authenticate its left neighbor once and perform batch authentication once, instead of implementing mutual authentication between group members, which reduces much of the computation and communication costs.
- The blockchain can be regarded as a shared distributed ledger [10], which can effectively solve the problem of single node failure. Therefore, we use the blockchain to store the public parameters and registration information of all group members. This allows our protocol to solve the problem of single node failure, while also making all the parameters and information stored in the blockchain unmodifiable [11]. In addition, based on blockchain technology, group members can join any group in the entire system after completing registration on any server, which improves convenience.
- In our protocol, when group members join or leave a group, they only needs to update the parameters of an adjacent group member, which also improves the scalability of our protocol.
- Our protocol guarantees the forward secrecy of group members after joining the group and the backward secrecy of group members after leaving the group.

The rest of this article is arranged as follows. Section 2 describes the related works. Some preliminaries are introduced in Section 3. The proposed protocol is described completely in Section 4. Section 5 analyzes the security of our protocol. The performance analysis is shown in Section 6. Finally, we conclude the article in Section 7.

2. Related Works

Many researchers have proposed many solutions to ensure the security of group keys. These solutions are generally divided into the following three types [12,13].

(1) Centralized group key agreement protocol:

There is usually only one entity for controlling the entire group in this type of protocol, which is called a Key Distribution Center (KDC). The KDC is responsible for key generation, distribution, and management. It also needs to be responsible for tasks such as group communication. The protocol proposed by Wong et al. [14] is a typical group key agreement protocol based on a Logical Key Hierarchy (LKH). A binary tree is stored in the KDC of this type of protocol. The root node of the tree is a shared encryption key; the intermediate nodes of the tree hold different keys along the path from the leaves to themselves; and the leaf nodes of the tree hold keys related to group members. This type of protocol requires less space to store the keys, and when the keys need to be updated, the amount of communication is greatly reduced. The group key agreement protocol of Islam et al. [7] is proposed for the Internet of Vehicles, and the trusted authority in their protocol acts as the KDC. In the centralized group key agreement protocol, since the KDC needs to be responsible for storing

and distributing all keys and controlling group communication, the scalability and storage costs of this type of protocol are very large. In addition, once the KDC fails, the entire system cannot continue to operate normally.

(2) Decentralized group key agreement protocol:

This type of protocol usually divides the entire group into several subgroups, and each subgroup has a group controller to manage all group members in the subgroup. In this way, the burden of KDC is greatly reduced, and single node failure is also solved. Mittra [15] proposed a scalable multicast framework, which divides large groups into multiple subgroups, and each subgroup has a controller called a group security intermediate node or a group security agent. In the protocol of Setia et al. [16], the group key is updated at regular intervals, rather than when group members join or leave. Naresh et al. [8] proposed a cluster-based hybrid group key agreement protocol, which divides large groups into a certain number of clusters and specifies the last member of the cluster as the cluster head and group controller. In 2018, Gupta et al. [17] proposed a group key agreement protocol based on self-certified public keys. In their protocol, there is a group controller for each group. Moreover, before generating the group key, each group member needs to mutually authenticate with other group members, which leads to high computation and communication costs. Zheng et al. [18] proposed a multi-domain group key agreement protocol. Their protocol uses authentication between group members and a group controller instead of authentication between each group member to reduce computation and communication costs. However, this makes their protocol vulnerable to single node failure. The main problems faced by decentralized group key agreement protocols are key distribution efficiency, how to establish a trusted relationship with a third party, and mutual authentication of sub-members across groups.

(3) Distributed group key agreement protocol:

All group members in this type of protocol are equal, and there is no group controller. In addition, the KDC usually does not participate in the generation of group keys. Without a base station, Wang et al. [19] proposed a device-to-device group key agreement protocol. The protocol guarantees the anonymity of each device and uses a signature scheme based on the gap Diffie–Hellman group [20]. In 2018, Zhang et al. [21] proposed a distributed group key agreement protocol. Their agreement achieves cross-domain authentication and key self-certification. Based on the hyper elliptic curve digital signature and ElGamal algorithm, Kavitha et al. [22] proposed a distributed group authentication protocol for the healthcare system in the Internet of Things. Since the KDC or group controller in this type of protocol usually does not participate in the process of group key generation, the mutual authentication between group members becomes the largest computational overhead, which means that the cost of generating a group key will increase as the number of group members increases. Therefore, reducing the number of mutual authentications between group members is the core of reducing the cost of generating a group key. There are many protocols dedicated to reducing the number of authentications between group members by reducing the number of authentication rounds. The group key agreement protocol proposed by Geng et al. [23] and Zheng et al. [9] divides the entire protocol into two rounds. The first round is mutual authentication between members, and the second round is group key generation. In the above two protocols, each group member only needs to authenticate the two adjacent group members. The protocol proposed by Zhang et al. [24] and Shi et al. [25] merges the two processes described above into one. The protocol proposed by Alphonse and Reddy [26] forms each group member into a structure similar to a binary tree, and all group members authenticate each other from the leaf node to the root node. Although their protocol reduces the computation and communication costs required to generate group keys, all group members must wait for the final root node to be authenticated before they can negotiate the group key. Therefore, the computing time is still not low.

3. Preliminaries

3.1. Network Model

There are two parts in our network model, namely KDC and General Node (*GN*). In MEC, the KDC can be regarded as the SCM, and the *GN* can be regarded as the SCC. All *GN*s are equal, and there is no hierarchy or subordinate relationship. In addition, all *GN*s usually have certain computing and storage resources, and they can join or leave a group at any time. All KDCs are wire connected, and each KDC can manage one or more *GN*s. The network model used in our protocol is shown in Figure 1.

In our protocol, multiple KDCs form a blockchain network. In order to improve the efficiency of new block generation, we consider using a more efficient Proof-of-Stake (PoS) [27] or Delegated Proof-of-Stake (DPoS) [28] consensus mechanism, such as ouroboros, a provably secure PoS protocol [27], instead of using a Proof-of-Work (PoW) mechanism [29]. According to this consensus mechanism, at regular intervals, all KDCs will regenerate new blocks including groups whose *GN*s have changed during this period. In each block, in addition to the hash value of the previous block, the timestamp, and Merkle tree root, it also contains the identifier of these groups, the identity list of all *GN*s in these group, and the related parameters of all *GN*s in these group. All *GN*s only have the permission to read information from the blockchain. In addition, there may be multiple different groups, so after the *GN* enters the network, it first needs to select a group to join.

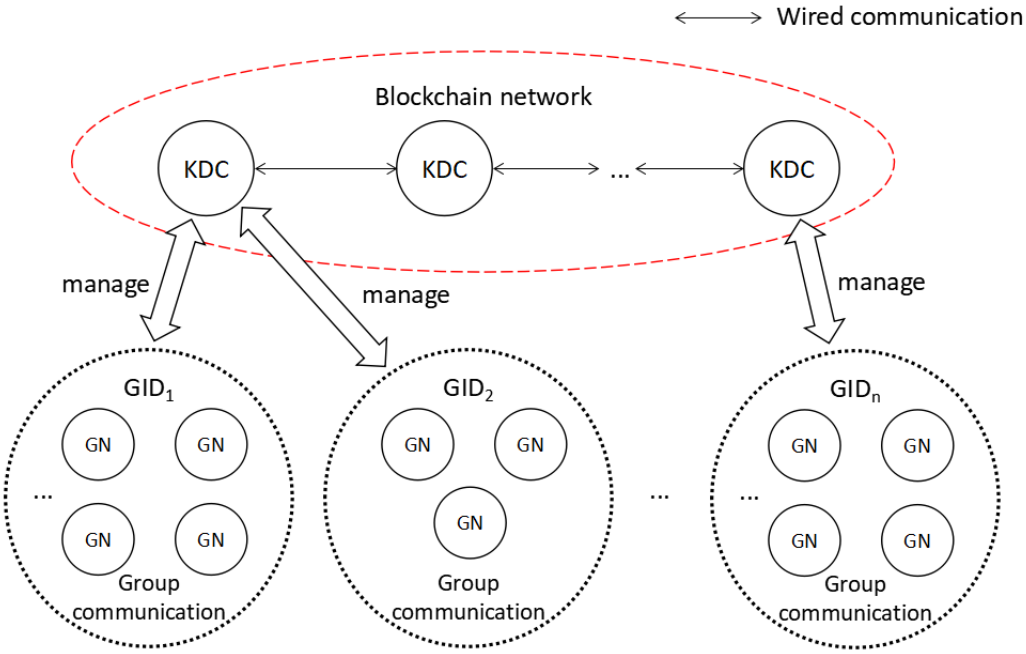

Figure 1. The network model used by our protocol.

Before the *GN* joins the network, it can submit its identity to a KDC closest to it. The KDC will calculate a pair of keys based on the identity and distribute it to the *GN*. After that, all KDCs will generate a new block containing the identity and related information of the newly added *GN* through the consensus mechanism. The detailed operation of KDCs is described in the next section. Note that not all KDCs participate in group key agreement.

3.2. Threat Model

We define the threat model as follows:

- The adversary has the ability to intercept all data transmitted over unsecured channels, and he/she can inject new data and replace or replay the previously sent data.
- All KDCs are semi-trusted parties, which means that they may misbehave themselves, but do not conspire with any other KDC [30].
- With the help of a Tamper-Proof Device (TPD), we assume that even if the adversary compromises the KDC, he/she cannot extract any secret keys from it.
- The adversary has the ability to capture any number of GNs and can access all the secret information stored in the GN's memory by capturing attack.

3.3. Bilinear Pairing

Let G_1 and G_2 be cyclic additive and multiplicative groups of prime order q, respectively. The generator of G_1 is g_1. Let $e : G_1 \times G_1 \rightarrow G_2$ be a bilinear pairing, which satisfies the following properties:

- Bilinearity: $\forall P, Q \in G_1$ and $\forall a, b \in Z_q^*$, $e(aP, bQ) = e(P, bQ)^a = e(aP, Q)^b = e(P, Q)^{ab}$ are satisfied.
- Non-degenerate: $\forall P, Q \in G_1$ such that $e(P, Q) \neq 1$.
- Computable: for all $P, Q \in G_1$, there is always an effective algorithm to compute $e(P, Q)$.

The security of our protocol is based on the following computationally infeasible problems.

- Elliptic Curve Discrete Logarithm problem (ECDL): Let $a \in Z_q^*$, given $P, aP \in G_1$, and compute a.
- Computational Diffie–Hellman problem (CDH): Let $a, b \in Z_q^*$, given g_1, ag_1, and bg_1, and find abg_1.
- Decisional Diffie–Hellman problem (DDH): Let $a, b, c \in Z_q^*$, given g_1, ag_1, bg_1, and cg_1, and decide if $e(ag_1, bg_1) = e(g_1, cg_1)$.

4. Proposed Protocol

In the distributed group key agreement protocol, each group member is equal, which means that before the group key is negotiated, each group member usually needs to consume many communication and computing resources to perform mutual authentication with all other group members. In order to reduce these costs, in our protocol, we arrange all GNs into a list according to their identities. According to the list, before the group key is negotiated, each GN only needs to send an authentication request to its right neighbor once and be authenticated by its right neighbor. In other words, each GN will receive an authentication request from its left neighbor and authenticate its left neighbor. Since each GN only needs to complete authentication once, this can greatly reduce the computation and communication costs caused by authentication between group members. In addition, when any GN needs to join or leave the group, only the left neighbor of the GN needs to update the parameters, which can also reduce the computation and communication overhead.

Our protocol has seven parts: the initialization phase, the registration phase, the mutual authentication phase, the group key generation phase, the GN join phase, the GN leave phase, and the internal attacker detection process. When the system runs for the first time, the initialization phase is performed by the System Administrator (SA). Each GN performs the registration phase before entering the network. When the group key needs to be negotiated, all GNs perform the mutual authentication phase and the group key generation phase. When a GN wants to join a group, it needs to perform the GN join phase. When a GN in the group wants to leave, the GN leave phase is performed. If the group key fails to be generated multiple times, the KDC will execute the internal attacker detection process to find the malicious GN and expel it from the group.

Suppose there are $GN_i (1 \leq i \leq n)$ that need to generate the group key, and their identities are $ID_i (1 \leq i \leq n)$, where n is the number of GN. Since there may be multiple groups, we named each

group GID_u, where u is the number of groups. Each group has a list L that stores the identity ID_i of all GNs in the group and is managed by the KDC. All ID_i in L are sorted in descending order, and L is a circular list, which means that the largest ID_i and the smallest ID_i are linked. Table 1 shows the description of the symbols. The details of the above seven parts are as follows.

Table 1. Symbols used in our protocol.

Symbol	Description
SA	System Administrator
KDC	Key Distribution Center
GN	General node
q	A large prime number
G_1	Cyclic additive groups of prime order q
G_2	Cyclic multiplicative groups of prime order q
Q	The generator of G_1
e	Bilinear pairing $e : G_1 \times G_1 \rightarrow G_2$
ID_i	The identity of GN_i
GID_u	The identity of the group
L	A circular list that stores all GN-related information in the group
s	The KDC's private key
P_{pub}	The KDC's public key
W_i, A_i	The GN's public key
S_i, a_i	The GN's private key
t_1, t_2	Timestamp
t_{new}	The timestamp when the latest information was received
Δt	Maximum communication transmission delay
KT_i	Symmetric key
E_k	Symmetric encryption algorithm
D_k	Symmetric decryption algorithm
$h(.)$	Hash operation
Ks	Group key
$\oplus$	Bitwise XOR operation
(a, b)	Concatenation of data a and data b

4.1. Initialization Phase

First, the SA picks $\{G_1, G_2, Q, e, p\}$, where G_1 is a cyclic additive group of order p, G_2 is a cyclic multiplicative group of order p, Q is a generator of G_1, and $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map. Second, the SA generates a random private key s and computes the corresponding public key $P_{pub} = sQ$. Finally, the SA publishes parameters $\{p, G_1, G_2, Q, e, P_{pub}, h(.), E_k, D_k\}$ and stores s in the memory of each KDC in a secure environment, where $h(.)$ is the hash function used by this protocol, E_k is the symmetric encryption algorithm, and D_k is the symmetric decryption algorithm.

4.2. Registration Phase

Before the GN joins the network, it needs to choose the nearest KDC to register and receive the corresponding key. The steps in this phase are as follows.

Step R1: The KDC generates a unique identity ID_i for the GN_i and computes its public key $W_i = h(ID_i)$ and the corresponding private key $S_i = sW_i$. Finally, the KDC sends S_i to GN_i through a secure channel.

Step R2: GN_i generates a random a_i, computes $A_i = a_iQ$, and sends A_i to the KDC through a secure channel.

Step R3: The KDC broadcasts ID_i, W_i, and A_i.

Step R4: According to the group's identity GID, the GN can choose to join a group or pick a new unique GID to create a new group. Based on the selection of the GN, there are three different situations: (1) Situation A: the GN joins a group that already has a group key; (2) Situation B: the GN joins a group that has not started negotiating group keys; (3) Situation C: the GN creates a new group. In Situation

A, the GN performs the GN join phase. In Situation B and Situation C, the KDC will package GID, the corresponding L, and multiple tuples (ID_i, W_i, A_i) related to all members of the group into a new block, which will be verified by all other KDCs. After successful verification, the new block will be linked to the blockchain. Figure 2 shows the three situations faced by the GN and the solution strategies.

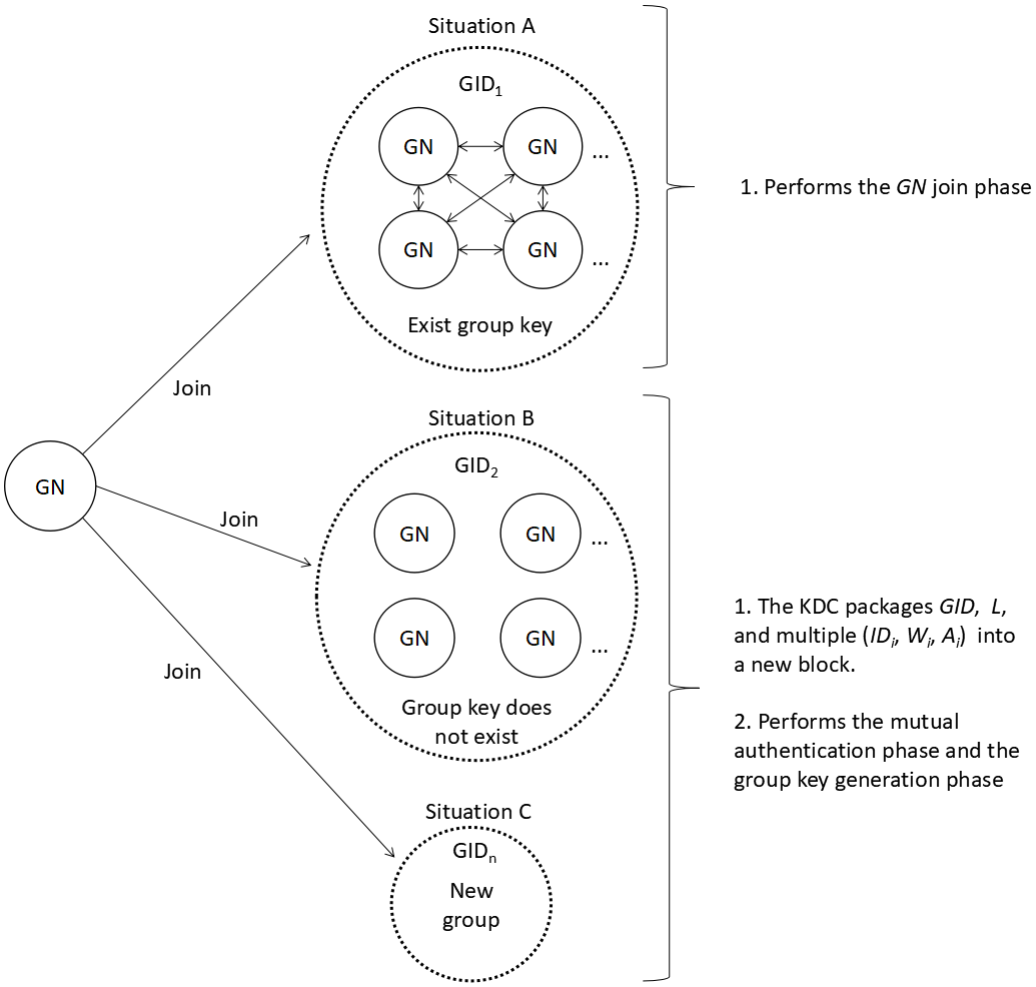

Figure 2. The three situations faced by the GN and the solution strategies.

4.3. Mutual Authentication Phase

In this phase, GN_i first sends a message to its right neighbor GN_{i+1}, and GN_{i+1} will authenticate GN_i. At the same time, GN_i will also receive a message from its left neighbor GN_{i-1}, and it needs to authenticate GN_{i-1}. Figure 3 shows the mutual authentication phase and the group key generation phase of our protocol. GN_i performs the following operations.

Step A1: Generates a random m_i and a timestamp t_{1_i} and gets A_{i+1} from the blockchain.

Step A2: Computes $M_i = m_iQ$, $KT_{i+1} = a_iA_{i+1}$, $SE_{i+1} = E_{KT_{i+1}}(M_i)$, $C_i = h(SE_{i+1}, KT_{i+1}, t_{1_i})S_i$.

Step A3: Sends message (SE_{i+1}, C_i, t_{1_i}) to GN_{i+1}.

Step A4: GN_i receives a message (SE_i, C_{i-1}, $t_{1_{i-1}}$) from GN_{i-1} and gets A_{i-1} from the blockchain.

Step A5: Checks that $t_{new} - t_{1_{i-1}} < \Delta t$ holds or not. If the check fails, it broadcasts an authentication failure message.

Step A6: Computes $KT_i = a_iA_{i-1}$.

Step A7: Checks whether the condition $e(Q, C_{i-1})? = e(P_{pub}, h(SE_i, KT_i, t_{1_{i-1}})W_{i-1})$ is satisfied. If the condition is not true, it broadcasts an authentication failure message.

Step A8: Uses KT_i to decrypt SE_i and get M_{i-1}.

Step A9: Generates a random b_i and a timestamp t_{2_i}.

Step A10: Computes $X_i = b_i W_i$, $Z_i = e(M_i - M_{i-1}, Q)$, $Y_i = (b_i + h(X_i, Z_i, t_{2_i}))S_i$.
Step A11: Broadcasts $R_i = (X_i, Y_i, Z_i, t_{2_i})$.

$\cdots GN_{i-2}$	GN_{i-1}	GN_i	$GN_{i+1}\cdots$
	Generates m_{i-1} and $t_{1_{i-1}}$	Generates m_i and t_{1_i}	
	Gets A_i from the blockchain	Gets A_{i+1} from the blockchain	
	Computes $M_{i-1} = m_{i-1}Q$,	Computes $M_i = m_i Q$,	
	$KT_i = a_{i-1}A_i$, $SE_i = E_{KT_i}(M_{i-1})$,	$KT_{i+1} = a_i A_{i+1}$, $SE_{i+1} = E_{KT_{i+1}}(M_i)$,	
	$C_{i-1} = h(SE_i, KT_i, t_{1_{i-1}})S_{i-1}$	$C_i = h(SE_{i+1}, KT_{i+1}, t_{1_i})S_i$	
	$\xrightarrow{(SE_i, C_{i-1}, t_{1_{i-1}})}$	$\xrightarrow{(SE_{i+1}, C_i, t_{1_i})}$	
	Gets A_{i-2} from the blockchain	Gets A_{i-1} from the blockchain	
	Checks validity of $t_{1_{i-2}}$	Checks validity of $t_{1_{i-1}}$	
	Computes $KT_{i-1} = a_{i-1}A_{i-2}$	Computes $KT_i = a_i A_{i-1}$	
	Checks	Checks	
	$e(Q, C_{i-2}) \stackrel{?}{=} e(P_{pub}, h(SE_{i-1}, KT_{i-1}, t_{1_{i-2}})W_{i-2})$	$e(Q, C_{i-1}) \stackrel{?}{=} e(P_{pub}, h(SE_i, KT_i, t_{1_{i-1}})W_{i-1})$	
	Uses KT_{i-1} to decrypt SE_{i-1} and gets M_{i-2}	Uses KT_i to decrypt SE_i and gets M_{i-1}	
	Generates b_{i-1} and $t_{2_{i-1}}$	Generates b_i and t_{2_i}	
	Computes $X_{i-1} = b_{i-1}W_{i-1}$,	Computes $X_i = b_i W_i$,	
	$Z_{i-1} = e(M_{i-1} - M_{i-2}, Q)$,	$Z_i = e(M_i - M_{i-1}, Q)$,	
	$Y_{i-1} = (b_{i-1} + h(X_{i-1}, Z_{i-1}, t_{2_{i-1}}))S_{i-1}$	$Y_i = (b_i + h(X_i, Z_i, t_{2_i}))S_i$	
	Broadcasts $R_{i-1} = (X_{i-1}, Y_{i-1}, Z_{i-1}, t_{2_{i-1}})$	Broadcasts $R_i = (X_i, Y_i, Z_i, t_{2_i})$	
	Checks validity of $t_{2_r}(r \in n, r \neq i-1)$	Checks validity of $t_{2_r}(r \in n, r \neq i)$	
	Checks	Checks	
	$e(\sum_{r\neq i-1} Y_r, Q) \stackrel{?}{=} e(\sum_{r\neq i-1}(X_r + h(Z_r, t_{2_r})W_r), P_{pub})$	$e(\sum_{r\neq i} Y_r, Q) \stackrel{?}{=} e(\sum_{r\neq i}(X_r + h(Z_r, t_{2_r})W_r), P_{pub})$	
	Computes $k = e(nM_{i-1}, Q)Z_i^{n-1}Z_{i+1}^{n-2}\cdots Z_{i-2}$,	Computes $k = e(nM_i, Q)Z_{i+1}^{n-1}Z_{i+2}^{n-2}\cdots Z_{i-1}$,	
	$Ks = h(k, R_1, R_2, \cdots, R_n)$	$Ks = h(k, R_1, R_2, \cdots, R_n)$	

Figure 3. The mutual authentication phase and the group key generation phase of our protocol.

4.4. Group Key Generation Phase

During this phase, each GN_i receives the message $R_r(r \in n, r \neq i)$ from all other GNs. At this point, each GN_i will perform a group authentication and then negotiate the group key. The execution steps of each GN_i are as follows.

Step K1: Checks that the timestamp $t_{new} - t_{2_r} < \Delta t, (r \in n, r \neq i)$ in each received message is valid. If the check fails, it broadcasts an authentication failure message.

Step K2: After receiving the message from all other GNs, it checks that:

$$e(\sum_{r\neq i} Y_r, Q) \stackrel{?}{=} e(\sum_{r\neq i}(X_r + h(Z_r, t_{2_r})W_r), P_{pub})$$

holds or not. If the check fails, it broadcasts an authentication failure message.

Step K3: Computes $k = e(nM_i, Q)Z_{i+1}^{n-1}Z_{i+2}^{n-2}\cdots Z_{i-1}$ and group key $Ks = h(k, R_1, R_2, \cdots, R_n)$.

4.5. GN Join Phase

When a new GN_j wants to join the group, it needs to be registered in the KDC first. This means that according to the registration phase, GN_j has selected a group to join. Figure 4 shows the GN join phase of our protocol. The detailed steps are as follows.

Step J1: GN_j obtains the corresponding list L of the group from the blockchain, inserts its identity ID_j into the appropriate position in L, and broadcasts the new L.

Step J2: GN_j generates a random number a_j and computes $A_j = a_j Q$. After that, GN_j broadcasts A_j.

Step J3: GN_j's left neighbor GN_{j-1} regenerates a new random number a'_{j-1} and computes $A'_{j-1} = a'_{j-1}Q$. After that, GN_{j-1} broadcasts A'_{j-1}.

Step J4: The KDC packages the updated L, corresponding GID, and multiple tuples (ID_i, W_i, A_i) related to all members of the group into a new block, which will be verified by all other KDCs. After successful verification, the new block will be linked to the blockchain.

Step J5: According to the steps in the mutual authentication phase, GN_j sends (SE_{j+1}, C_j, t_{1_j}) to its right neighbor GN_{j+1} and receives message (SE'_j, C'_{j_1}, $t'_{1_{j-1}}$) from GN_{j-1}.

Step J6: After the messages received by GN_j and GN_{j+1} are successfully authenticated, all GNs broadcast $R_i = (X_i, Y_i, Z_i, t_{2_i})$ and perform the group key generation phase to complete the key update.

4.6. GN Leave Phase

When a GN_j in the group wants to leave, the following steps need to be performed. Figure 5 shows the GN leave phase of our protocol.

Step L1: GN_j deletes its identity ID_j from list L and broadcasts the new L.

Step L2: GN_{j-1} regenerates a new random number a'_{j-1}, and computes $A'_{j-1} = a'_{j-1}Q$. After that, GN_{j-1} broadcasts A'_{j-1}.

Step L3: The KDC packages the updated L, corresponding GID, and multiple tuples (ID_i, W_i, A_i) related to all members of the group into a new block, which will be verified by all other KDCs. After successful verification, the new block will be linked to the blockchain.

Step L4: According to the steps in the mutual authentication phase, GN_{j-1} sends (SE'_j, D'_{j-1}, $t'_{1_{j-1}}$) to GN_{j+1}.

Step L5: After U_{j+1} authenticates U_{j-1}, all GNs broadcast $R_i = (X_i, Y_i, Z_i, t_{2_i})$ and perform the group key generation phase to complete the key update.

GN_{j-1}	GN_j	KDC
	Inserts ID_j into L Broadcasts the updated L	
Generates a'_{j-1} Computes $A'_{j-1} = a'_{j-1}Q$ Broadcasts A'_{j-1}	Generates a_j Computes $A_j = a_jQ$ Broadcasts A_j	
		Packages L, GID, and multiple (ID_i, W_i, A_i) into a new block
Performs the mutual authentication phase and the group key generation phase	Performs the mutual authentication phase and the group key generation phase	

Figure 4. The GN join phase of our protocol.

GN_{j-1}	GN_j	KDC
	Deletes ID_j from L Broadcasts the updated L	
Generates a'_{j-1} Computes $A'_{j-1} = a'_{j-1}Q$ Broadcasts A'_{j-1}		
		Packages L, GID, and multiple (ID_i, W_i, A_i) into a new block
Performs the mutual authentication phase and the group key generation phase		

Figure 5. The GN leave phase of our protocol.

4.7. Internal Attacker Detection Process

It can be found in our protocol that every GN is required to be honest during the group key generation phase. If a malicious GN intentionally broadcasts an error message, the entire group cannot generate a group key. However, since all GNs are equal, it is impossible to discover malicious GNs through these GNs. Therefore, our agreement regards KDC as the judging body. When the group fails to negotiate multiple times, the process will be executed and try to find the malicious GN. The steps of this process are as follows.

Step D1: The KDC first records all messages sent by each GN_i in the group GID.

Step D2: All GN_i generate a timestamp t_{D_i}, compute $H = h(W_i, S_i, ID_i, KT_{i+1}, b_i)$, and send the tuple ($ID_i$, W_i, H, KT_{i+1}, b_i) to the KDC.

Step D3: After receiving the tuple, the KDC first verifies the validity of the time stamp t_{D_i}. Then, the KDC computes $S_i' = sW_i$, $H' = h(W_i, S_i', ID_i, KT_{i+1}, b_i)$ and verifies $H'? = H$. If it does not hold, GN_i is regarded as a malicious GN.

Step D4: If the above formula holds, the KDC continues to compute $C_i' = h(SE_{i+1}, KT_{i+1}, t_{1_i})S_i$, $Y_i' = (b_i + h(X_i, Z_i, t_{2_i}))S_i$ and verify $C_i'? = C_i$, $Y_i'? = Y_i$. If it does not hold, the corresponding GN_i will be expelled from the group immediately.

5. Security and Performance Analysis

5.1. Correctness Analysis

Theorem 1. *GN_i and GN_{i+1} can calculate the same symmetric key KT_{i+1}, so that GN_{i+1} can get M_i.*

Proof. Since GN_i computes $KT_{i+1} = a_i A_{i+1}$ to get KT_{i+1} and GN_{i+1} computes $KT_{i+1} = a_{i+1} A_i$ to get KT_{i+1}, then:

$$\begin{aligned} KT_{i+1} &= a_i A_{i+1} \\ &= a_i a_{i+1} Q \\ &= a_{i+1} A_i. \end{aligned}$$

Since the same KT_{i+1} can be obtained by calculating $a_i A_{i+1}$ and $a_{i+1} A_i$, GN_i and GN_{i+1} can use the symmetric key KT_{i+1} to encrypt or decrypt transmitted messages. □

Theorem 2. *It is valid for GN_i to authenticate GN_{i-1}.*

Proof. The authentication of GN_i for GN_{i-1} is achieved by verifying whether the formula $e(Q, C_{i-1}) = e(P_{pub}, h(SE_i, KT_i, t_{1_{i-1}})W_{i-1})$ holds. The correctness of the formula is proven as follows.

$$\begin{aligned} e(Q, C_{i-1}) &= e(Q, h(SE_i, KT_i, t_{1_{i-1}})S_{i-1}) \\ &= e(Q, h(SE_i, KT_i, t_{1_{i-1}})sW_{i-1}) \\ &= e(P_{pub}, h(SE_i, KT_i, t_{1_{i-1}})W_{i-1}) \end{aligned}$$

Although the adversary can easily obtain $Q, P_{pub}, A_i \in G_1$, due to the ECDL and the CDH, he/she cannot calculate s, any a_i, or KT_i in polynomial time. Therefore, the adversary also cannot calculate $h(SE_i, KT_i, t_{1_{i-1}})$. If the adversary wants to forge a GN_i to pass the above verification, he/she needs to create a new valid $e(P_{pub}, h(SE_i, KT_i, t_{1_{i-1}})W_{i-1})$. However, since the adversary cannot obtain a_i and s, he/she cannot calculate a new $e(P_{pub}, h(SE_i, KT_i, t_{1_{i-1}})W_{i-1})$. In addition, due to the DDH, for a random $z \in G_1$, the adversary cannot decide if $e(Q, C_{i-1}) = e(P_{pub}, z)$ in polynomial time. □

Theorem 3. *During the group key generation phase, GN_i is valid for the batch authentication of other group members.*

Proof. In the group key generation phase, GN_i authenticates other group members in batches by verifying whether formula $e(\sum_{r\neq i} Y_r, Q) = e(\sum_{r\neq i}(X_r + h(X_r, Z_r, t_{2_r})W_r), P_{pub})$ holds. The correctness of the formula is proven as follows.

$$\begin{aligned} e(\sum_{r\neq i} Y_r, Q) &= e(\sum_{r\neq i}(b_r + h(X_r, Z_r, t_{2_r}))S_r, Q) \\ &= e(\sum_{r\neq i}(b_r + h(X_r, Z_r, t_{2_r}))sW_r, Q) \\ &= e(\sum_{r\neq i}(b_r W_r + h(X_r, Z_r, t_{2_r})W_r), sQ) \\ &= e(\sum_{r\neq i}(X_r + h(X_r, Z_r, t_{2_r})W_r), P_{pub}). \end{aligned}$$

If the adversary wants to forge a GN_i^* to pass the above batch authentication, he/she needs to create a valid X_i^* and Y_i^* to satisfy $e(Y_i^*, Q) = e((b_i + h(X_i^*, Z_i, t_{2_i}))sW_i, Q)$. First, the adversary cannot get b_i, so it is difficult for him/her to calculate $(b_i + h(X_i^*, Z_i, t_{2_i}))sW_i$. Second, even suppose that $(b_i + h(X_i^*, Z_i, t_{2_i}))$ is revealed by the adversary, but he/she still cannot calculate a valid X_i^* or Y_i^* because he/she cannot get s. In addition, due to the DDH, for a random $z \in G_1$, the adversary cannot decide if $e(\sum_{r\neq i} Y_r, Q) = e(z, P_{pub})$ in polynomial time. □

Theorem 4. *If all GN_is participating in the group key generation phase are honest, then all GN_is can negotiate the same group key.*

Proof. According to Theorem 1, as long as all GN_is participating in the group key generation phase are honest, each GN_i can obtain the parameter M_{i-1} sent by its left neighbor. Therefore,

$$\begin{aligned} k &= e(nM_i, Q)Z_{i+1}^{n-1}Z_{i+2}^{n-2}\cdots Z_{i-1} \\ &= e(m_iQ, Q)^n Z_{i+1}^{n-1}Z_{i+2}^{n-2}\cdots Z_{i-1} \\ &= e(Q, Q)^{nm_i+(n-1)(m_{i+1}-m_i)+(n-2)(m_{i+2}-m_{i+1})+\cdots+(m_{i-1}-m_{i-2})} \\ &= e(Q, Q)^{m_1+m_2+\cdots+m_i}. \end{aligned}$$

From the above, it can be found that all GN_is can calculate the same parameter k. Therefore, their group keys $Ks = h(k, R_1, R_2, \cdots, R_n)$ are also the same. □

5.2. Simulation Based on the ProVerif Tool

ProVerif is a widely known authentication protocol verification tool that can prove the security of multiple encryption schemes or authentication protocols, such as signature schemes and Diffie–Hellman key exchange algorithms [31,32]. Since each GN does not need to communicate with other nodes during the group key generation phase, we use ProVerif to verify the security of the mutual authentication phase of our protocol. In addition, since we do not need to verify the performance of our protocol in this section, we assume that there are only three GNs that need to negotiate a group key, which are GN_1, GN_2, and GN_3. Figure 6 shows the code for the mutual authentication phase of our protocol. Figure 7 shows the simulation results. The results show that in our protocol, the secret parameters M_1, M_2, and M_3 for group key generation, as well as the private keys s, a_1, a_2, and a_3 will not be obtained by the adversary.

```
free c:channel.
(*——constants——*)
type QP.
const q: QP.
free Q,W1,W2,W3: QP.
free a1,a2,a3,s:bitstring [private].
(*—functions,reductions and equations—*)
fun pm(QP,bitstring):QP.
equation forall x:bitstring,y:bitstring;
pm(pm(q, x), y) = pm(pm(q, y), x).
fun e(QP,QP):QP.
fun add(bitstring, bitstring): bitstring.
fun minus(QP, bitstring): QP.
fun senc(bitstring, bitstring): bitstring.
reduc forall m: bitstring, k: bitstring;
sdec(senc(m,k),k) = m.
fun con(bitstring,bitstring):bitstring.
fun h(bitstring):bitstring.
fun QPtoBS(QP):bitstring [typeConverter].
(*——queries——*)
query secret M1.
query secret M2.
query secret M3.
query attacker (s).
query attacker (a1).
query attacker (a2).
query attacker (a3).

let GN1(A1:QP,A2:QP,A3:QP,Q:QP,S1:QP,
W3:QP,W1:QP,a1:bitstring,Ppub:QP) =
new m1:bitstring;
new t11:bitstring;
let M1=pm(Q,m1) in
let KT2=pm(A2,a1) in
let SE2=senc(QPtoBS(M1),QPtoBS(KT2)) in
let C1=pm(S1,h(con(SE2,
con(QPtoBS(KT2),t11)))) in
out (c,(SE2,C1,t11));
in (c,(SE1:bitstring,C3:QP,t13:bitstring));
let KT1=pm(A3,a1) in
let M3=sdec(SE1,QPtoBS(KT1)) in
new b1:bitstring;
new t21:bitstring;
let X1=pm(W1,b1) in
let Z1=e(minus(M1,M3),Q) in
let Y1=pm(S1,add(b1,h(con(QPtoBS(X1),
con(QPtoBS(Z1),t21))))) in
out (c,(X1,Y1,Z1,t21)).

let GN2(A1:QP,A2:QP,A3:QP,Q:QP,S2:QP,
W1:QP,W2:QP,a2:bitstring,Ppub:QP) =
new m2:bitstring;
new t12:bitstring;
let M2=pm(Q,m2) in
let KT3=pm(A3,a2) in
let SE3=senc(QPtoBS(M2),QPtoBS(KT3)) in
let C2=pm(S2,h(con(SE3,
con(QPtoBS(KT3),t12)))) in
out (c,(SE3,C2,t12));
in (c,(SE2:bitstring,C1:QP,t11:bitstring));
let KT2=pm(A1,a2) in
let M1=sdec(SE2,QPtoBS(KT2)) in
new b2:bitstring;
new t22:bitstring;
let X2=pm(W2,b2) in
let Z2=e(minus(M2,M1),Q) in
let Y2=pm(S2,add(b2,h(con(QPtoBS(X2),
con(QPtoBS(Z2),t22))))) in
out (c,(X2,Y2,Z2,t22)).

let GN3(A1:QP,A2:QP,A3:QP,Q:QP,S3:QP,
W2:QP,W3:QP,a3:bitstring,Ppub:QP) =
new m3:bitstring;
new t13:bitstring;
let M3=pm(Q,m3) in
let KT1=pm(A1,a3) in
let SE1=senc(QPtoBS(M3),QPtoBS(KT1)) in
let C3=pm(S3,h(con(SE1,
con(QPtoBS(KT1),t13)))) in
out (c,(SE1,C3,t13));
in (c,(SE3:bitstring,C2:QP,t12:bitstring));
let KT3=pm(A2,a3) in
let M2=sdec(SE3,QPtoBS(KT3)) in
new b3:bitstring;
new t23:bitstring;
let X3=pm(W3,b3) in
let Z3=e(minus(M3,M2),Q) in
let Y3=pm(S3,add(b3,h(con(QPtoBS(X3),
con(QPtoBS(Z3),t23))))) in
out (c,(X3,Y3,Z3,t23)).

process
let A1=pm(Q,a1) in
let A2=pm(Q,a2) in
let A3=pm(Q,a3) in
let S1=pm(W1,s) in
let S2=pm(W2,s) in
let S3=pm(W3,s) in
let Ppub=pm(Q,s) in
(!GN1(A1,A2,A3,Q,S1,W3,W1,a1,Ppub)|
(!GN2(A1,A2,A3,Q,S2,W1,W2,a2,Ppub))|
(!GN3(A1,A2,A3,Q,S3,W2,W3,a3,Ppub))
```

Figure 6. The code for the mutual authentication phase of our protocol.

```
------------------------------------------------------------
Verification summary:

Query secret M1_1,M1 is true.

Query secret M2_1,M2 is true.

Query secret M3_1,M3 is true.

Query not attacker(s[]) is true.

Query not attacker(a1[]) is true.

Query not attacker(a2[]) is true.

Query not attacker(a3[]) is true.

------------------------------------------------------------
```

Figure 7. The PV simulation results of our protocol.

5.3. Informal Security Analysis

5.3.1. *GN* Impersonation Attack

The adversary needs to create valid messages (SE_{i+1}, C_i, t_{1_i}) or (X_i, Y_i, Z_i, t_{2_i}) to perform an impersonation attack on GN_i. However, according to Theorem 2 and Theorem 3, the adversary cannot create a valid message.

5.3.2. *GN* Capture Attack

After the adversary has captured several *GN*s, we need to ensure that the private key s cannot be obtained by the adversary. After the adversary captures a GN_i, he/she can obtain S_i containing information about s. However, according to the ECDL, although the adversary can obtain W_i and sW_i, he/she cannot calculate s in polynomial time.

5.3.3. Replay Attack

Our protocol uses timestamps t_1 and t_2 to defend against replay attacks. In addition, our protocol guarantees that the timestamp cannot be modified. For messages $(SE_{i+1},\ C_i,\ t_{1_i})$, the adversary must obtain KT_{i+1} and S_i before replacing a new valid timestamp t'_{1_i}. For messages (X_i, Y_i, Z_i, t_{2_i}), the adversary must obtain b_i and S_i before replacing a new valid timestamp t'_{2_i}. However, the adversary cannot get KT_{i+1}, b_i or S_i because they are never transmitted directly in the channel. In addition, due to the ECDL, it is difficult for the adversary to calculate these parameters.

5.3.4. Forward Secrecy after a New *GN* Joins

When a *GN* wants to join a group, it needs to ensure that GN_j cannot get the previous group key. In our protocol, we assume that GN_j has intercepted all historical communication messages of the group. This results in that when GN_j joins the group, as long as it obtains m_{j-1}, it can calculate the previous group key. However, when the GN_j joins the group, its left neighbor GN_{j-1} will regenerate a new m'_{j-1} and send it to GN_j, which means GN_j can only get m'_{j-1} instead of m_{j-1}. Therefore, our protocol ensures the forward secrecy after a new *GN* joins.

5.3.5. Backward Secrecy after a *GN* Leaves

When a GN_j leaves the group, it needs to ensure that it cannot get the group key generated in the future. We assume that after GN_j leaves the group, it still retains the secret parameter m_{j-1} sent by its left neighbor and is able to intercept all group communication messages. This means that as long as m_{j-1} is kept unchanged, GN_j can still calculate the group key after leaving the group. However, in our protocol, after GN_j leaves the group, its left neighbor U_{j-1} will regenerate a new m'_{j-1} and send it to GN_{j+1}. This results in that at Step A11, GN_{j-1} will broadcast $Z'_{j-1} = e(M'_{j-1} - M_{j-2}, Q)$ and U_{j+1} will broadcast $Z'_{j+1} = e(M_{j+1} - M'_{j-1}, Q)$. At this point, m_{j-1} has expired, and GN_j cannot calculate the group key through this parameter. Therefore, our protocol ensures the backward secrecy after a *GN* leaves.

5.3.6. Single Node Failure

Although all KDCs will not participate in the mutual authentication phase and the group key generation phase, each group still needs to be managed by a KDC. This is because each group may need the KDC to act as a judging body to expel malicious nodes. If the KDC fails, since all KDCs share the same blockchain that stores all *GN*s' authentication parameters, it only needs to switch to another KDC. This means that in our protocol, as long as there is a working KDC, the entire system can operate normally.

6. Performance Analysis and Comparison

6.1. Computation Cost

The symbols t_{sym}, t_h, t_{pm}, t_{pa}, and t_{bp} represent the computing time required to implement one symmetric encryption or decryption, one general hash function operation, one point multiplication operation on Elliptic Curve Cryptography (ECC), one point addition operation on ECC, and one bilinear pairing operation, respectively. In the mutual authentication phase and the group key generation phase of our protocol, each *GN* needs to perform $n + 7$ point multiplication operations on ECC, 4 bilinear pairing operations, $n + 3$ hash operations, 2 symmetric encryption or decryption operations, and $n + 1$ point addition operations on ECC, where n is the number of *GN*. In other words, the total computation cost of each *GN* is $(n + 7)t_{pm} + 4t_{bp} + (n + 3)t_h + 2t_{sym} + (n + 1)t_{pa}$.

6.2. Communication Cost

We assume C is 256 bits and timestamp T is 64 bits. In the mutual authentication phase of our protocol, each GN needs to send message (SE_{i+1}, C_i, t_{1_i}) to GN_{i+1} and broadcast (X_i, Y_i, Z_i, t_{2_i}). Therefore, each GN needs to send messages of length $5C + 2T$ and receive messages of length $(3n-1)C + nT$. The total communication cost in the the mutual authentication phase and the group key generation phase of our protocol is $(3n^2 + 4n)C + (n^2 + 2n)T$.

6.3. Comparison with Related Protocols

We compare our protocol with the protocol of Zheng et al. [9], the protocol of Zhang et al. [21], and the protocol of Gupta et al. [17] in terms of computation costs, energy consumption, communication costs, and security.

In the protocol of Zheng et al. [9], the total computation cost required to generate the group key is $(n+4)t_{pm} + 6t_{bp} + (n+4)t_h + (3n+1)t_{pa}$, and each group member needs to send messages of length $7C + 2T$ and receive messages of length $(7C + 2T)(n-1)$. In the protocol of Zhang et al. [21], the total computation cost required to generate the group key is $(3n+2)t_{pm} + 2nt_{bp}$, and each group member needs to send messages of length $4C$ and receive messages of length $4(n-1)C$. In the protocol of Gupta et al. [17], the total computation cost required to generate the group key is $4nt_{pm} + 5t_h + (2n+1)t_{pa}$. In their protocol, there is a group controller in each group, which undertakes most of the communication work and causes the communication cost of each group member to be very low. Therefore, we consider that the total length of the messages that need to be sent and received is $6nC + 1$, rather than the communication cost of each group member. Table 2 shows the comparison of computation cost and communication cost between our protocol and related protocols.

Table 2. The comparison of computation cost and communication cost between our protocol and related protocols.

	Zheng et al. [9]	**Zhang et al. [21]**	**Gupta et al. [17]**	**Our Protocol**
Point multiplication operations on ECC	$n+6$	$3n+2$	$4n$	$n+7$
Bilinear pairing	6	$2n$	0	4
Hash operation	$n+9$	0	5	$n+3$
Symmetric encryption or decryption	4	0	0	2
Point addition operations on ECC	$3n+1$	0	$2n+1$	$3n+1$
Message length sent by each group member	$7C+2T$	$4C$	-	$5C+2T$
Message length received by each group member	$(7C+2T)(n-1)$	$4(n-1)C$	-	$(3n-1)C+nT$
Total sent message length	$(7C+2T)n$	$4nC$	$6nC+1$	$(5C+2T)n$
Total received message length	$n(n-1)(7C+2T)$	$4n(n-1)C$	$6nC+1$	$(3n^2-n)C+n^2T$

Next, we will compare our protocol with related protocols in terms of energy consumption. According to the work of Carman et al. [33] and Zhang et al. [21], we obtained that a "Strong ARM" microprocessor running at 133MHz performing one symmetric encryption or decryption operation needs to consume 0.00217 mJ, one point multiplication operation on ECC requires 8.8 mJ, one general hash function operation requires 0.000108 mJ, and one bilinear pairing operation requires 47 mJ. According to the work of Makri and Konstantinou [34], we obtained that an IEEE 802.11 Spectrum24 WLAN card requires 0.00066 mJ for the transmission of 1 bit and 0.00031 mJ for the reception of 1 bit. According to [35], the computing time of one point addition operation on ECC is about half of one symmetric encryption or decryption operation. Therefore, we considered that the energy consumption of a point addition operation is about 0.001085 mJ. We summarize the above energy consumption in Table 3. Figure 8 shows the total communication energy consumption in different protocols. Figure 9

shows the communication energy consumption of each group member in different protocols. Figure 10 shows the total communication and computation energy consumption in different protocols.

It can be found from Table 2 that our protocol has the lowest computation costs, and the computation costs of our protocol and the protocol of Zheng et al. [9] are very similar. However, from Figure 8, it can be found that our protocol consumes less communication resources. In addition, because after group members join or leave a group, neither their left neighbor nor their right neighbor updated the corresponding temporary secret parameter, this leads to the newly joined group members being able to easily obtain the previous group key, and group members who leave the group can also easily obtain the subsequent group key. Therefore, the protocol of Zheng et al. [9] lacks forward or backward secrecy.

According to Figure 10, in the distributed group key agreement protocol, such as the protocol of Zheng et al. [9] and the protocol of Zhang et al. [21], our protocol has the least total communication and computation energy consumption. With the help of the group controller, the decentralized group key agreement protocol such as the protocol of Gupta et al. [17] will consume less energy when there is a large number of group members. However, in the protocol of Gupta et al. [17], each group member needs to perform mutual authentication with the group controller one by one, which requires much total computing time. Moreover, the existence of the group controller makes their protocol vulnerable to single node failure. Once the group controller fails, the group cannot continue to negotiate the group key. According to Figures 8 and 9, excluding the protocol of Gupta et al. [17], the total communication energy consumption of our protocol is the lowest, and the communication energy consumption of each group member in our protocol is also the lowest.

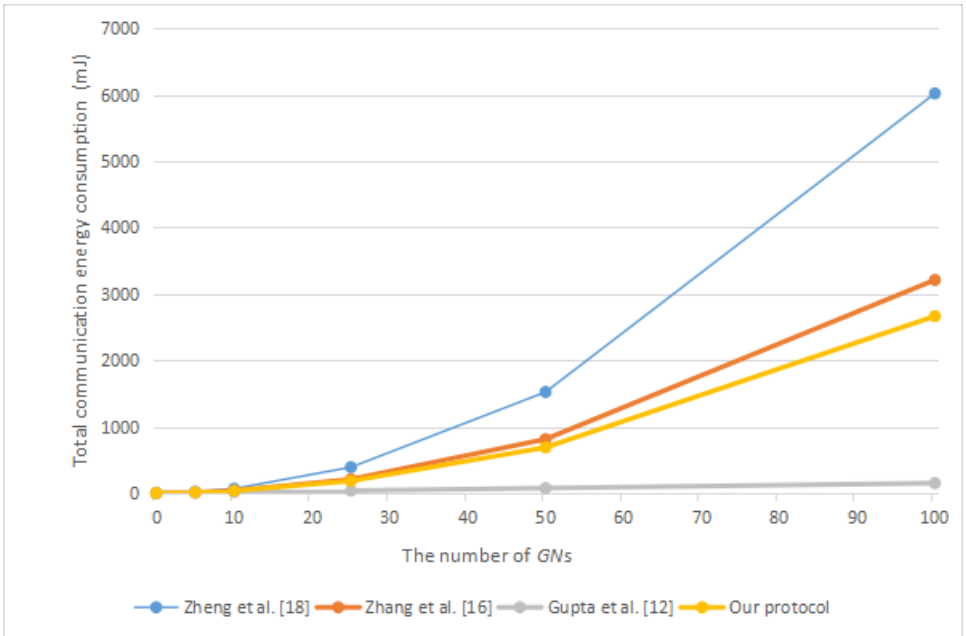

Figure 8. The total communication energy consumption in different protocols.

Table 3. Energy consumption for computing and communication.

Operations	Energy Consumption
t_{sym}	0.00217 mJ
t_{pm}	8.8 mJ
t_{pa}	0.001085 mJ
t_h	0.000108 mJ
t_{bp}	47 mJ
Transmitting a bit	0.00066 mJ
Receiving a bit	0.00031 mJ

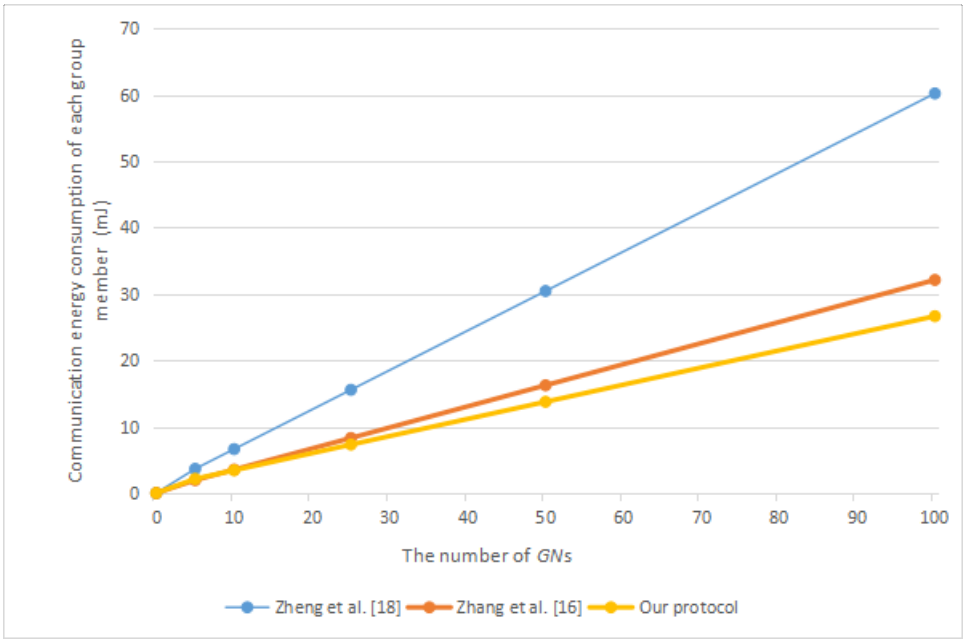

Figure 9. The communication energy consumption of each group member in different protocols.

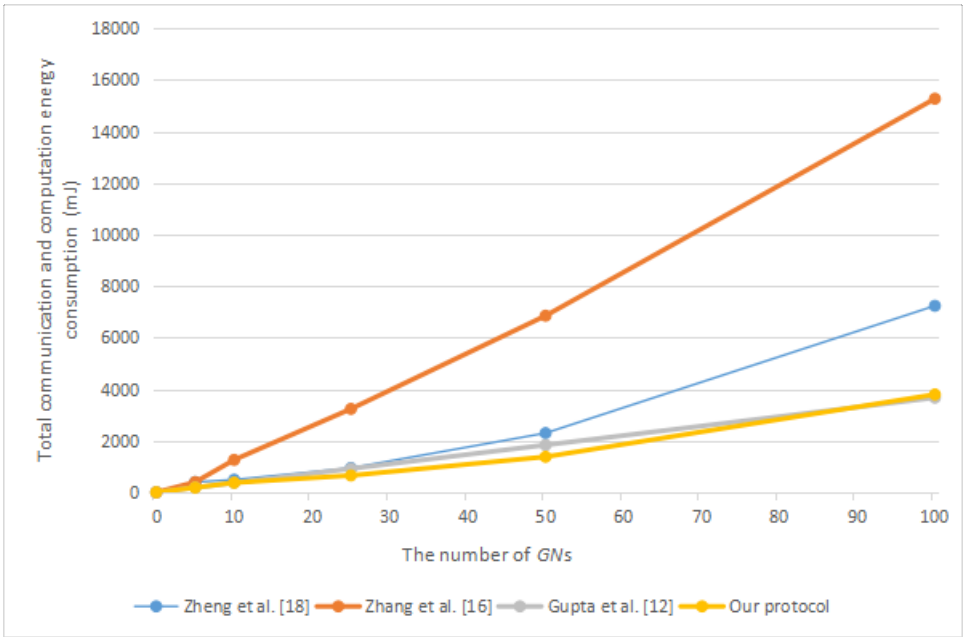

Figure 10. The total communication and computation energy consumption in different protocols.

7. Conclusions

This paper proposes a blockchain-based authentication and dynamic group key agreement protocol. Each group member in our protocol only needs to authenticate its left neighbor once to complete the authentication, which improves authentication efficiency. When a node joins or leaves a group, only the left neighbor of the node needs to update the data, which also improves the scalability of our protocol. Our protocol also guarantees forward or backward secrecy when group members join or leave the group. In addition, we use blockchain technology to store group identities, a list of group members, and some group member-related parameters, which can solve the problem of single node failure. Finally, we use mathematics and ProVerif to prove the correctness and security of our protocol. The comparison with related protocols shows that our protocol reduces computation and communication costs.

Author Contributions: Conceptualization, Z.X.; funding acquisition, F.L., H.D. and J.X.; project administration, M.T.; writing, original draft, Z.X.; writing, review and editing, F.L., Z.X. and J.Z. All authors read and agreed to the published version of the manuscript.

Funding: This work was supported in part by National Natural Science Foundation of China under Grant 61872138 and Grant 61772185.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MEC	Mobile Edge Computing
SCM	Small Cell Manager
SCC	Small Cell Cloud
GN	General Node
KDC	Key Distribution Center
LKH	Logical Key Hierarchy
TPD	Tamper-Proof Device
ECC	Elliptic Curve Cryptography
SA	System Administrator
ECDL	Elliptic Curve Discrete Logarithm problem
CDH	Computational Diffie–Hellman problem
DDH	Decisional Diffie–Hellman problem

References

1. Rafaeli, S.; Hutchison, D. A survey of key management for secure group communication. *ACM Comput. Surv. CSUR* **2003**, *35*, 309–329. [CrossRef]
2. Wang, T.; Qiu, L.; Sangaiah, A.K.; Liu, A.; Bhuiyan, M.Z.A.; Ma, Y. Edge-Computing-Based Trustworthy Data Collection Model in the Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4218–4227. [CrossRef]
3. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutorials* **2017**, *19*, 1628–1656. [CrossRef]
4. Gong, L.; Shacham, N. Multicast security and its extension to a mobile environment. *Wirel. Netw.* **1995**, *1*, 281–295. [CrossRef]
5. Liang, W.; Huang, W.; Long, J.; Zhang, K.; Li, K.C.; Zhang, D. Deep reinforcement learning for resource protection and real-time detection in IoT environment. *IEEE Internet Things J.* **2020**, *7*, 6392–6401. [CrossRef]
6. Wang, T.; Cao, Z.; Wang, S.; Wang, J.; Qi, L.; Liu, A.; Xie, M.; Li, X. Privacy-enhanced data collection based on deep learning for Internet of vehicles. *IEEE Trans. Ind. Inform.* **2019**, *16*, 6663–6672.
7. Islam, S.H.; Obaidat, M.S.; Vijayakumar, P.; Abdulhay, E.; Li, F.; Reddy, M.K.C. A robust and efficient password-based conditional privacy preserving authentication and group-key agreement protocol for VANETs. *Future Gener. Comput. Syst.* **2018**, *84*, 216–227. [CrossRef]
8. Naresh, V.S.; Reddi, S.; Murthy, N.V. A provably secure cluster-based hybrid hierarchical group key agreement for large wireless ad hoc networks. *Hum. Centric Comput. Inf. Sci.* **2019**, *9*, 26. [CrossRef]
9. Zheng, J.; Yang, C.; Xue, J.; Zhang, C. A Dynamic ID-based Authenticated Group Key Agreement Protocol. In Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering, Xi'an, China, 12–13 December 2015; Atlantis Press: Xi'an, China, 2015.
10. Hussien, H.M.; Yasin, S.M.; Udzir, S.; Zaidan, A.A.; Zaidan, B.B. A systematic review for enabling of develop a blockchain technology in healthcare application: Taxonomy, substantially analysis, motivations, challenges, recommendations and future direction. *J. Med. Syst.* **2019**, *43*, 320. [CrossRef]
11. Zubaydi, H.D.; Chong, Y.W.; Ko, K.; Hanshi, S.M.; Karuppayah, S. A review on the role of blockchain technology in the healthcare domain. *Electronics* **2019**, *8*, 679. [CrossRef]
12. Seetha, R.; Saravanan, R. A survey on group key management schemes. *Cybern. Inf. Technol.* **2015**, *15*, 3–25. [CrossRef]
13. Barskar, R.; Chawla, M. A survey on efficient group key management schemes in wireless networks. *Indian J. Sci. Technol.* **2016**, *9*, 14. [CrossRef]

14. Wong, C.K.; Gouda, M.; Lam, S.S. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.* **2000**, *8*, 16–30. [CrossRef]
15. Mittra, S. Iolus: A framework for scalable secure multicasting. In *ACM SIGCOMM Computer Communication Review*; ACM: New York, NY, USA, 1997; Volume 27, pp. 277–288.
16. Setia, S.; Koussih, S.; Jajodia, S.; Harder, E. Kronos: A scalable group re-keying approach for secure multicast. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, S&P 2000, Berkeley, CA, USA, 14–17 May 2000; IEEE: Piscataway, NY, USA, 2000; pp. 215–228.
17. Gupta, S.; Kumar, A.; Kumar, N. Design of ECC based authenticated group key agreement protocol using self-certified public keys. In Proceedings of the 2018 4th International Conference on Recent Advances in Information Technology (RAIT), Dhanbad, India, 15–17 March 2018; IEEE: Piscataway, NY, USA, 2018; pp. 1–5.
18. Zheng, J.; Zhang, X.; Zhang, Q.; Zhang, Q.; Zhang, C. Multi-domain lightweight asymmetric group key agreement. *Chin. J. Electron.* **2018**, *27*, 1085–1091. [CrossRef]
19. Wang, L.; Tian, Y.; Zhang, D.; Lu, Y. Constant-round authenticated and dynamic group key agreement protocol for D2D group communications. *Inf. Sci.* **2019**, *503*, 61–71. [CrossRef]
20. Boneh, D.; Lynn, B.; Shacham, H. Short signatures from the Weil pairing. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, 9–13 December 2001; Springer: Berlin/Heidelberg, Germany, 2001; pp. 514–532.
21. Zhang, Q.; Gan, Y.; Zhang, Q.; Wang, R.; Tan, Y.A. A dynamic and cross-domain authentication asymmetric group key agreement in telemedicine application. *IEEE Access* **2018**, *6*, 24064–24074.
22. Kavitha, S.; Alphonse, P.; Reddy, Y.V. An Improved Authentication and Security on Efficient Generalized Group Key Agreement Using Hyper Elliptic Curve Based Public Key Cryptography for IoT Health Care System. *J. Med. Syst.* **2019**, *43*, 260. [CrossRef]
23. Geng, M.; Zhang, F.; Gao, M. A secure certificateless authenticated group key agreement protocol. In Proceedings of the 2009 International Conference on Multimedia Information Networking and Security, Wuhan, China, 18–20 November 2009; IEEE: Piscataway, NY, USA, 2009; Volume 1, pp. 342–346.
24. Zhang, Q.; Wang, R.; Tan, Y. Identity-based authenticated asymmetric group key agreement. *J. Comput. Res. Dev.* **2014**, *51*, 1727–1738.
25. Shi, Y.; Chen, G.; Li, J. ID-based one round authenticated group key agreement protocol with bilinear pairings. In Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II, Las Vegas, NV, USA, 4–6 April 2005; IEEE: Piscataway, NY, USA, 2005; Volume 1, pp. 757–761.
26. Alphonse, P.; Reddy, Y.V. A method for obtaining authenticated scalable and efficient group key agreement for wireless ad-hoc networks. *Clust. Comput.* **2019**, *22*, 3145–3151. [CrossRef]
27. Kiayias, A.; Russell, A.; David, B.; Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 357–388.
28. Liang, W.; Fan, Y.; Li, K.C.; Zhang, D.; Gaudiot, J.L. Secure data storage and recovery in industrial blockchain network environments. *IEEE Trans. Ind. Inform.* **2020**, *16*, 6543–6552. [CrossRef]
29. Liang, W.; Zhang, D.; Lei, X.; Tang, M.; Li, K.C.; Zomaya, A. Circuit Copyright Blockchain: Blockchain-based Homomorphic Encryption for IP Circuit Protection. *IEEE Trans. Emerg. Top. Comput.* **2020**. [CrossRef]
30. Franklin, M.K.; Reiter, M.K. Fair exchange with a semi-trusted third party. In Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, 1–4 April 1997; pp. 1–5.
31. Xu, Z.; Xu, C.; Chen, H.; Yang, F. A lightweight anonymous mutual authentication and key agreement scheme for WBAN. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e5295. [CrossRef]
32. Blanchet, B.; Smyth, B. Automated reasoning for equivalences in the applied pi calculus with barriers. *J. Comput. Secur.* **2018**, *26*, 367–422. [CrossRef]
33. Carman, D.W.; Kruus, P.S.; Matt, B.J. Constraints and approaches for distributed sensor network security (final); Technical Report; The Security Research Division of Network Associates Inc: Glenwood, MD, USA, 1 September 2000.

34. Makri, E.; Konstantinou, E. Constant round group key agreement protocols: A comparative study. *Comput. Secur.* **2011**, *30*, 643–678. [CrossRef]
35. He, D.; Zeadally, S.; Kumar, N.; Wu, W. Efficient and anonymous mobile user authentication protocol using self-certified public key cryptography for multi-server architectures. *IEEE Trans. Inf. Forensics Secur.* **2016**, *119*, 2052–2064. [CrossRef]