A Hierarchical Multicast Key Distribution Protocol

Jie Li 1,2, Shaowen Yao 1,2, Jing Liu 1,2 and Yunyun Wu 3,*

Abstract: In secure group communication, group keys (GK) are used to ensure the confidentiality of communication. The group key distribution (GKD) is responsible for updating and distributing new group keys when the group membership changes. Most well-known GKD protocols are based on a logical key hierarchy (LKH), where only one group controller (GC) is used. These protocols have various issues, including a single point of failure, meaning that the GC often has a huge workload and can be easily overwhelmed. In this paper, we propose a hierarchical multicast key distribution protocol that supports multi-level controllers to manage a group. Each controller just needs to manage the next-level nodes, and if one fails, the superior controller can replace it with minimal work. The proposed protocol effectively balances the work of controllers, greatly improves the reliability of the group key distribution, and also allows group members to build dynamic conferences without controllers. We provide a security proof of the proposed protocol in a symbolic security model and compare it to other protocols in terms of efficiency, functionality, and security.

Keywords: group key distribution; multi-level server; reliability; dynamic conference; security

1. Introduction

Nowadays, multicast communication applications, such as stock quote streaming and multimedia conferencing, are widely used. These applications encrypt communications with a GK for confidentiality. When the group membership changes, how to securely update and distribute the new GK is a key issue. A new group member should not recover any information on communication before his/her joining. Similarly, a member leaving the group should recover no information on communication after his/her leaving. The first instance is called backward secrecy [1], and the latter is called forward secrecy [1]. GKD is an approach used to securely update and distribute the new GK.

When the membership of a group changes, the GKD usually needs a GC to generate, update, and distribute the new GK. Most GKD protocols are based on LKH [2], which is a centralized approach with only one GC to manage the group communication. In other words, only one GC should deal with the requests from group members. If the group is large and its membership changes frequently, the GC can easily fail; therefore, LKH-based protocols are limited by having a single point of failure or control. Currently, there are two kinds of decentralized protocols, although these methods were created to solve problems in other scenarios. The first kind of protocol [3,4] applies to the Internet of Things and mobile networks. It divides a group into a number of subgroups, and each subgroup is managed by a subgroup controller (SGC). If an SGC fails, then the subgroup cannot work. These protocols cannot address having a single point of failure or control. Another kind of protocol [5,6] applies to decentralized environments, such as blockchain. Group members manage communication themselves, which requires them to have sufficient storage and computational capabilities.

The above schemes focus on a continuous group. However, members may want to construct a dynamic and short-lived group, called a dynamic conference, such as a
multimedia conference. This group uses a conference key (CK) to encrypt communication. The group membership will not change frequently, but the group is often built or dissolved. When the conference members negotiate CK, due to the distrust among members, GC is responsible for managing the keys. This is also a problem of having a single point of failure or control.

To solve the single point of failure or control issue, we propose a hierarchical group key distribution protocol based on hierarchical identity-based encryption (HIBE), which supports multi-level controllers that manage keys for all group members. HIBE replaces the traditional encryption algorithm in the GDK. The private key of HIBE is divided into two parts: one is the encryption key, which takes the role of the key in the traditional encryption algorithm; the other is the generation key, which is used to generate the next level controller’s key, and is kept secret by its controller. The encryption key ensures that our protocol can run as usual, and the generation key ensures that only the superior controller has the ability to generate the subordinate nodes’ keys. A new concept of a temporary node is introduced to the proposed protocol. According to practical requirements, the group members are not evenly distributed in a subgroup. To keep the key tree efficient, we use temporary nodes to optimize the key tree.

**Contribution** Our main contributions are summarized as follows.

1. **A balanced GC works on multi-level servers.** Each controller manages only the next-level nodes. Thus, no matter the group size, controllers will not be overworked.
2. **SGC replacement mechanism.** The superior controller retains the ability to maintain the subordinate nodes’ keys (not only the next nodes). Hence, if an SGC breaks down, the upper controller can directly and easily take its place.
3. **Dynamic conference without controllers.** Each group member can easily find all sub-trees covering all group members, and use the IDs of their roots to encrypt the CK.
4. **Customized specific information to limit the group member.** We could add the time information into ID and it could be used to force the group member to decrypt the rekeying message at a certain time.

The remainder of this paper is organized as follows: Related work is provided in Section 2. Section 3 describes the system model and the security requirements. The main construction and security proof is given in Section 4. Section 5 compares the proposed protocol with other protocols on performance, functionality, and security. Finally, Section 6 summarizes this paper and discusses future research and application.

**2. Related Work**

GKD scenarios can be divided into three categories. The first is centralized, with a central controller, such as for banks or securities. The second is that multiple controllers manage multiple groups, such as the Internet of Things and mobile networks. The third has no controllers, such as blockchain.

The classic centralized GKD is the logical key hierarchy [2], which reduces the communication overload from $O(n)$ to $O(\log_2 n)$ by using a binary key tree. Many such schemes [7,8] for secure multicast communications adopt tree structures, which have a proper balance between communication, computation, and storage overload. Anand et al. [9] propose a scheme that integrates the principles of a routing tree mapped logical key tree for lightweight and secure group communication. Other schemes use different approaches. The scheme proposed in [10] and based on the Chinese remainder theorem (CRT) combines rekeying messages into a single message when the group membership changes. Manisha Y. Joshi et al. [11] propose a table-based scheme, which solves the secure problem of the collusion attack and uses a single message to update the group key when membership changes; however, it needs more storage space on the group-member side. Shaohua Tang et al. [12] propose an approach based on a hypersphere, where the GC needs more computational time. Tsoatoate et al. [13] propose a self-healing protocol that allows group members to decrypt rekeying messages when the membership changes from offline to online. The above schemes only have an entity GC, which maintains the whole group,
and is a potential bottleneck. Nabeel et al. [14] propose an attribute-based group key management. Their scheme uses the attributes as the key, which means that a member who satisfies the attributes can recover the GK. Hence, the GC only needs to generate the key for attributes, which is time efficient; however, this method cannot resist collusion attacks. Rajkumar et al. [15] propose an attribute-based data sharing scheme that can resist collusion attacks by using a LKH model.

In the second scenario, this kind of scheme applies the original group key distribution method to the subgroup. Salve et al. [3] directly use LKH in online social networks. Each subgroup has an SGC to maintain communication. The scheme proposed in [16] is applied to mobile networks with many base stations. Kung et al. [17] propose a scheme that maintains groups in IoT environments. Sun et al. [4] propose a lightweight noise-based group key generation method that utilizes noise signals imposed on raw acceleration signals to generate an M-bit key with high randomness and bit generation rate. In this scenario, multiple controllers work together, and if one of them fails, the corresponding subgroup cannot work.

There are no controllers in the third scenario. The communication group is maintained by the group members. There are a few protocols. Lei et al. [6] propose the use of the blockchain structure for group key distribution in a heterogeneous intelligent transportation system. Alphand et al. [18] propose an IoTChain to provide an E2E solution for secure authorized access to IoT resources. Li et al. [5] introduce a mutual-healing group key distribution scheme based on blockchain. Pal et al. [19] propose a scheme that enables network members to share the computational load of the server and scheme, and achieves forward and backward secrecy. Because the group members maintain it by themselves, each must do his/her share. There are requirements for the computing and storage capacity of members.

3. System Model and Security Requirements

This section provides the notation used in this paper, a description of the system model, and the definition of the security of the proposed scheme.

3.1. Notation

Table 1 lists the parameters used in the proposed construction.

Table 1. Summary of notation in the construction.

| Variables | Description |
|-----------|-------------|
| SGC       | Subgroup controller |
| GK<sub>i</sub> | Group key generated by GC or SGC whose identity is ID<sub>i</sub> |
| ID<sub>i</sub> | Identity of member U<sub>i</sub> |
| d<sub>i</sub> | Private key of member U<sub>i</sub> |
| KEK       | Key encryption key |
| TN        | Temporary node |
| CB        | Cipher block |
| CT        | Rekeying message that consists of cipher blocks |

3.2. System Model

The proposed hierarchical multicast key distribution system has three terminals: a GC, SGCs, and members. The GC is the key server that maintains the public parameters, the master key, and its lower-level SGCs’ keys. The SGC not only shares the responsibility of distributing keys to lower levels with the GC, but also maintains a subgroup. As shown in Figure 1, both GC and SGCs jointly manage the group. The proposed system effectively balances servers’ workloads. As the set of members covered by a server share a group key, members can construct a dynamic group by themselves, without the servers.
3.3. Security Requirements

In the proposed scheme, anyone who eavesdrops on communication has the ability to intercept its encrypted messages. There are three kinds of adversaries: the outsider who never joins the group, the leaver who leaves the group at some point, and the newcomer who joins the group at some point. To guarantee the security of communication between the GC, members, and SGCs, the scheme should satisfy four security requirements.

1. **Group confidentiality**: Anyone not in the group cannot obtain any information about group communication.
2. **Backward secrecy**: A new group member cannot decrypt group communication prior to his/her joining, even with new keys.
3. **Forward secrecy**: An old member cannot decrypt future group communication after leaving, even with the old keys.
4. **Collusion resistance**: Members who are not in the group at time $t$ cannot decrypt group communication of time $t$.

4. Our Construction

4.1. Hierarchical Identity-Based Encryption

Hierarchical identity-based encryption is an extension of identity-based encryption, which reduces the workload of a private key generator (PKG) by delegating the work of
private key generation to its lower levels. HIBE is constructed by four algorithms [20], as follows.

1. **Setup**: This algorithm takes a security parameter $1^k$ as input, and outputs the public parameters $PP$ and the master key $MK$. The description of the message space, ciphertext space, and identity space is given in $PP$. Let $l$ denote the maximal level of the HIBE. $PP$ and $l$ are public, and $MK$ is secret. This algorithm is run by the PKG.

2. **Key−Gen**: This algorithm generates a private key $d_i$ for an identity $ID_i$. It has two kinds of inputs, which apply to three situations. If the inputs are the identity $ID_i$ and master key $MK$, this algorithm is run by the PKG. If the inputs are the identity $ID_i$, private key $d_{i−k}$, and number of distinct layers $k$, this algorithm is run by the SGC.

3. **Encrypt**: This algorithm takes the public parameters $PP$, identity $ID_i$, and message as inputs, and outputs the ciphertext. It can be run by anyone.

4. **Decrypt**: This takes the private key $d_i$ and ciphertext as the inputs, and generates a plain message. This algorithm is run by member $U_i$.

   The private key [20] $d_i$ has two parts: (1) $d_{1i}$ is used to decrypt the ciphertext; (2) $d_{2i}$ is used to generate the next level’s private key. $d_{1}$ is given to lower nodes while $d_{2}$ is kept secure. According to the classic HIBE methods [20], it is easy to prove that no information can be obtained from $d_{2}$ with $d_{1}$.

4.2. **Construction**

The proposed scheme has five parts: System-Init, Member-Join, Member-Leave, Dynamic-Conference, and Sever-Replace.

4.2.1. **System-Init**

The GC firstly builds a key tree for the group members and SGCs, as shown in Figure 2. In the key tree, the root is associated with the GC, the internal nodes correspond to the SGCs, and the leaves are the group members. The structure of the key tree is determined by the distribution of the SGCs and group members. However, an SGC may maintain many group members in reality, and this subtree, whose root is the SGC, is not an efficient tree like SGC4. The size of the subtree’s rekeying message is $O(n)$, where $n$ is the number of subgroup members. To reduce the size, the subtree can be adjusted to a binary tree, and the size of the rekeying message is then $O(\log n)$. As shown in Figure 2, SGC4 changes the original tree to a binary tree by adding temporary nodes denoted as $TN$. At the same time, if a GC or SGC has too many next-level nodes, the tree can also be adjusted using $TN$. A temporary node is managed by the node that generates it. Each $ID$ is encoded by the location of the node in the key tree. The information of $ID$ contains the upper level’s $ID$. Each node in the key tree has two parts: (1) a version of public parameters; (2) an identifier and private key.

Once the key tree has been built, the GC first runs the Setup algorithm to generate the public parameters $PP$ and master key $MK$, and sets the version $v$ to 1. The master key is secret, and other parameters are public. Then, the GC runs the Keygen algorithm with the inputs of the SGC’s identifier $ID$ and $MK$ to output the private keys for the next level’s SGCs. Each SGC also runs Keygen with the inputs $ID$ and its private key $d$ to generate the private keys for its next level’s SGC or group members. The private key of $TN$ is generated by the GC or the SGC, which is responsible for managing the node. Let the associated nodes be those in the path from its leaf node to the root. Let $N_{ID, ID}$ be the collection of associated nodes, where $ID$ and $ID$ are the identities of the leaf and root, respectively. All the identities of the nodes in the tree are public, and the GC or the SGC sends $d_{1}$ of the nodes in $N_{ID, ID, GC}$ to group member $U_i$ by SSL. $d_{2}$ is secret. The GC generates the group key $GK_{GC}$ and sends the encrypted group key with $ID_{GC}$ to all the group members, like $\{GK_{GC}\}_{ID_{GC}}$. Similarly, each SGC distributes its group key to its subgroup members.
Figure 2. A key tree.

One thing to be noted is that the public key in the proposed protocol is $ID$, which means it could be customized. Some specific information can be added to $ID$. For example, adding time information limit the member to decrypt rekeying messages at a specific time.

We do not detail the process of member authentication, which can be accomplished by many methods [21].

4.2.2. Member-Join

When a new member joins a group, the GC allocates suitable SGCs by his/her location, or the new member chooses a specific SGC.

The last-level SGC allocates a new position in its key tree to the new member. A node with the smallest depth is chosen and divided into two nodes. One is associated with the original member, and the other is associated with the new member. The original member node becomes a temporary node, as seen in Figure 3. Due to limited space, the tree in Figure 3 is cropped from Figure 2.
Generating new public/private key pairs: When a new member joins the group, both GC and SGC only need to update the next-level node's key. Referring to Figures 2 and 3, the process is as follows. The GC updates version, generates a new group key $GK'_{GC}$, and updates its own public/private key pair $(1D'_{GC}, d'_{GC})$ and $d'_{GC} = (d1'_{GC}, d2'_{GC})$, and computes $(1D'_{SGC1}, d'_{SGC1})$ for SGC1 with $d2'_{GC}$. The GC sends $d2'_{SGC1}$ to SGC1 in a secure channel (IKE or SSL). The process continues until the last affected level SGC4 receives its own new $d2'_{SGC4}$. Finally, SGC4 generates new public/private key pairs for all affected nodes (e.g., the black nodes in Figure 3), and sends $d1_{U4}$ to $U5$ in a secure channel. All affected nodes' public keys are public. Only $d2'_{SGC4}$, $d2'_{SGC4}$, and $d1_{U4}$ are sent in a secure channel. The cost is not high, and this can be done in real-life situations.

A rekeying message $CT$ contains many cipher blocks $CB_j$, each consisting of an encryption ID, encrypted ID, and ciphertext, which is the message of key encryption, $KEK$. The generation of rekeying messages is a bottom-up process. For the affected leaf node $U_i$ (member’s node), the manager SGC runs Encrypt to encrypt a new $d1_{U_i}$ with its original $ID_{U_i}$ as $KEK$. The encryption ID is the original ID of $U_i$. The encrypted ID is the new ID of $U_i$. For temporary node $TN_j$, the manager SGC runs Encrypt to encrypt $d1_{TN_j}$ with its children’s original public key as $KEK$. The encryption ID is its children’s original public key. The encrypted ID is the new ID of $TN_j$. For SGC nodes, each node is just responsible for generating the next level’s CB. The generation of CB is similar to that of a temporary node. For a GC node, the GC should compute its next-level SGC’s CB. The method is the same as SGC’s. It also generates CB for the new group key. The encryption ID is the new ID of the encrypted ID is NULL, and $KEK$ is the ciphertext, which encrypts $GK'_{GC}$ with ID of.

Finally, the GC and the SGCs broadcast cipher blocks.

Referring to Figures 2 and 3, the cipher blocks are $CB_1 = (1D_{4}, 1D'_{4}, KEK = \{d1'_{4}\}_{ID_4})$, $CB_2 = (1D'_{4}, 1D'_{TN_3}, KEK = \{d1'_{TN_3}\}_{ID'_{3}})$, $CB_3 = (ID_3, 1D'_{TN_3}, KEK = \{d1'_{TN_3}\}_{ID_3})$, $CB_4 = (1D_3, 1D'_{TN_3}, KEK = \{d1'_{TN_3}\}_{ID_3})$, $CB_5 = (1D_{TN_3}, 1D'_{TN_3}, KEK = \{d1'_{TN_3}\}_{ID_{TN_3}})$, $CB_6 = (ID_{TN_3}, 1D'_{SGC1}, KEK = \{d1'_{SGC1}\}_{ID_{TN_3}})$, $CB_7 = (ID_{TN_3}, 1D'_{SGC4}, KEK = \{d1'_{SGC4}\}_{ID_{TN_3}})$, $CB_8 = (1D'_{SGC4}, 1D'_{SGC1}, KEK = \{d1'_{SGC1}\}_{ID_{SGC4}})$, $CB_9 = (ID_{SGC4}, 1D'_{SGC1}, KEK = \{d1'_{SGC1}\}_{ID_{SGC4}})$, $CB_10 = (1D'_{SGC1}, 1D'_{GC}, KEK = \{d1'_{GC}\}_{ID_{SGC1}})$, $CB_11 = (ID_{SGC2}, 1D'_{GC}, KEK = \{d1'_{GC}\}_{ID_{SGC2}})$, $CB_12 = (ID_{SGC3}, 1D'_{GC}, KEK = \{d1'_{GC}\}_{ID_{SGC3}})$, $CB_13 = (1D'_{GC}, NULL, KEK = \{GK'_{GC}\}_{ID_{GC}})$. 

Figure 3. New member joining.
Group members decrypt rekeying messages: This is done from bottom to top. Each group member compares the public key ID that they own with the first part of each CB. If one is matched, the group member decrypts the third part of the matched CB by running Decrypt, and updates the new ID’ and d1’. This process is repeated until the group member obtains the new group key.

Referring to Figure 3, group member U3 has (ID_{U3}, d1_{U3}), (ID_{TN2}, d1_{TN2}), (ID_{SGC4}, d1_{SGC4}), (ID_{SGC1}, d1_{SGC1}), and (ID_{GC}, d1_{GC}). Upon receiving the rekeying messages, he/she searches them, decrypts CB4, updates (ID_{TN2}, d1_{TN2}) to (ID’_{TN2}, d1’_{TN2}), and continues to search the rekeying messages until all keys are updated.

4.2.3. Member-Leave

When a member leaves the group, the manager SGC deletes the corresponding leaf node and adjusts the key tree to an optimal state, as seen in Figure 4.

![Figure 4. Group member leaving.](image)

The next steps are the same as Member-Join, and are not described.

4.2.4. SGC Replacement Mechanism

When an SGC fails, the SGC replacement mechanism is started so as to ensure the normal operation of the protocol. This is a temporary scheme, and the original key management strategy is maintained when the SGC returns to normal.

When an SGC does not respond for a specific time, its superior GC/SGC identifies it as a failure, and acts as its agent to update and distribute the key. As shown in Figure 5, when SGC4 does not work, the superior SGC1 takes the place of SGC4 to maintain the key tree.

The details are as follows. When SGC4 fails, SGC1 sends a notification of manager change to U1, U2, U3, U4. Since the identities of members and temporary nodes are public, SGC1 can build a tree that is the same as SGC4’s tree, and is not required to know the private keys. There are two possible scenarios of the membership changes at this time. If the change of membership occurred outside this group, then SGC1 works as usual. If it occurred in the group, then SGC1 inserts or deletes a member as in the above process, and updates public/private key pairs of the affected nodes, except for that of SGC4. The next steps are the same as the steps of Member-Join.
Figure 5. The failure of an SGC.

When SGC_4 returns to work, it judges whether the group key and its members have changed by the version and the identifies of its group members. If the group has not changed, then SGC_4 broadcasts its recovered message, and SGC_1 returns the manager work to SGC_4. If the membership changed outside this group, SGC_1 will use SGC_4’s public key to encrypt all the updated d1s and the group key that SGC_1 has, and send this to SGC_4. If the membership changed in this group, then SGC_4 builds a tree the same as SGC_1’s, as the identities of members and temporary nodes are public. SGC_4 is not required to know the private keys. Then, SGC_1 sends new private keys d'_{SGC_4} to SGC_4 in a secure channel. The next steps are the same as in the previous scenario.

4.3. Security Proof

There is no standard method to prove the security of multicast key distribution. In this paper, Pankwani’s symbolic security model [22] is used to prove the security of the proposed protocol by mathematical induction. This proof mainly focuses on group confidentiality, backward secrecy, forward secrecy, and collusion resistance.

Assume an n-member group, labeled 1, 2, ..., n. Each member shares \([\lfloor \log n \rfloor + 1] ID/d1\) key pairs. Any legitimate members can receive and send a broadcast message and decrypt right GK and ID/d1 key pairs. At any time \(t\), let \(S^{(t)} \subseteq \{1, 2, ..., n\}\) be a set of legitimate members, \(GK^{(t)}\) be the group key, and \(Tr^{(t)}\) be the key tree. Let \([n] = \{1, 2, ..., n\}\), and \(2^{[n]}\) is the power set of \([n]\). At time \(t\), the dynamic group can be represented by a sequence of sets \(S^{(t)} = (S^{(0)}, S^{(1)}, ..., S^{(t)}) \in (2^{[n]})^t\). The sequence is called simple if for all \(t > 1\), the group set changes from \(S^{(t-1)}\) to \(S^{(t)}\) when a single member leaves or joins.

Assume the information to be encrypted is a message \(M\), which could be \(d1\), ID, or KEK. Define the protocols as single encryption protocols that generate messages according to the following grammar:

\[
\begin{align*}
M & \rightarrow ID, d1, \{d1\}_{ID} \\
d1 & \rightarrow KeyGen(ID, MK, d2).
\end{align*}
\]

The group key distribution protocols (GKD) that fall within this class are called single encryption GKD protocols.

The following definitions are adapted from related security definitions [22].

**Definition 1.** An n-member GKD protocol is called correct, if for all \(t > 0\), for all simple sequences \(S^{(t)} \in (2^{[n]})^t\), \(\forall i \in S^{(t)}\), member \(i\) always knows the ID/d1 key pairs of the nodes in the path from the associated leaf node to the root in \(Tr^{(t)}\).
Definition 2. An n-member GKD protocol is called secure against single-user attacks, if for all \( t > 0 \), for all simple sequences \( \mathcal{S}^{(t)} \in (2^n)^t \), \( \forall i \notin \mathcal{S}^{(t)} \), member i cannot recover any \( \mathcal{GK}^{(t)} \) and \( ID/d_1 \) key pairs of nodes in \( Tr^{(t)} \) by \( (1D_s,d_i) \) and any rekeying messages throughout the lifetime of the group.

Definition 3. An n-member GKD protocol is called secure against collusion attacks, if for all \( t > 0 \), for all simple sequences \( \mathcal{S}^{(t)} \in (2^n)^t \), and arbitrary set of members \( \text{Col} = \{ i | i \notin \mathcal{S}^{(t)} \} \), \( \text{Col} \) cannot recover any \( \mathcal{GK}^{(t)} \) or \( ID/d_1 \) key pairs of nodes in \( Tr^{(t)} \) by \( \{(1D_s,d_i) | i \in \text{Col}\} \) and all rekeying messages throughout the lifetime of the group.

According to the above definitions, it is easy to prove the following theorems.

**Theorem 1.** The multicast key distribution protocol based on hierarchical identity-based encryption is correct and secure against single-user attacks.

**Proof.** For \( t = 0 \) and \( S^{(0)} = \emptyset \), Theorem 1 is trivially true. If the claim is true for \( t − 1 > 0 \), then it is true for \( t \) as well. For any simple sequence \( \mathcal{S}^{(t)} = (S^{(0)},S^{(1)},...,S^{(t)}) \in (2^n)^t \), there are the following five cases. □

Case 1: When a group member leaves at time \( t \), group member \( i \in (S^{(t−1)} \cap S^{(t)}) \) can only recover incremental \( ID/d_1 \) key pairs corresponding to its rekeyed node \( ID/d_1 \) key pairs in \( Tr^{(t−1)} \). According to the inductive hypothesis, \( i \) holds all the \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t−1)} \) as required by Definition 1. On the basis of the description of Member-Leave, \( i \) can recover all \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t)} \), as required by Definition 1.

Case 2: When a group member joins at time \( t \), group member \( i \in (S^{(t−1)} \cap S^{(t)}) \) can only recover the incremental \( ID/d_1 \) key pairs corresponding to its rekeyed node \( ID/d_1 \) key pairs in \( Tr^{(t−1)} \). According to the inductive hypothesis, \( i \) holds all the \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t−1)} \), as required by Definition 1. On the basis of the description of Member-Join, \( i \) still can recover all \( ID/d_1 \) key pairs of nodes in \( Tr^{(t)} \), as required by Definition 1.

Case 3: A new member \( i \) joins the group at time \( t \) (\( i \notin S^{(t−1)} \cap i \in S^{(t)} \)). On the basis of the description of Member-Join, new member \( i \) can only recover the \( ID/d_1 \) key pairs of the nodes, as required by Definition 1.

Case 4: Member \( i \) leaves the group at time \( t \) (\( i \in S^{(t−1)} \cap i \notin S^{(t)} \)). On the basis of the description of Member-Leave, member \( i \) cannot recover any \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t)} \). This satisfies Definitions 1 and 2.

Case 5: Member \( i \) is not a legitimate member at time \( t − 1 \) and \( i \notin (S^{(t)}) \cap S^{(t)} \). According to Definition 1, \( i \) cannot get any \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t−1)} \). On the basis of our protocol, \( i \) also cannot recover any \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t)} \) without \( ID/d_1 \) key pairs of the nodes in \( Tr^{(t−1)} \), which satisfies Definitions 1 and 2.

The multicast key distribution protocol based on hierarchical identity-based encryption is correct and secure against single-user attacks.

**Theorem 2.** If the multicast key distribution protocol based on hierarchical identity-based encryption is a single encryption GKD protocol, this protocol is (strongly) secure against single-user attacks if and only if it is (strongly) collusion-resistant.

This theorem was proved by Panjwani [22].

**Theorem 3.** The multicast key distribution protocol based on hierarchical identity-based encryption is secure against collusion resistance.

**Proof.** The proposed protocol is a single encryption GKD protocol. According to Theorems 1 and 2, it is secure against collusion resistance. □
4.4. Extension on Dynamic Conference

A dynamic conference is one whose membership changes frequently. The most crucial point is how arbitrary members can efficiently build a safe conference and reduce the workload of the GC and SGC. In the proposed protocol, a group member can build a safe conference with the same or other group members without the GC and SGC.

The revocation algorithm is as follows. According to the above protocol, every group member knows the structure of the key tree, public keys of all nodes, and private keys $d_1$ of nodes in the path from the associated leaf node to the root in the key tree. If some members want to build a dynamic conference, those specific nodes that cover the members are chosen, the conference key (CK), which encrypts the conference information with the public keys of these nodes, is encrypted. Referring to Figure 6, the red node of member $U_7$ wants to build a dynamic conference with the blue nodes. The set of legal members is $U = \{U_1, U_2, U_3, U_4, U_6, U_7, U_9, U_{10}\}$, and the white nodes constitute the illegal users set $R = \{U_5, U_8, U_{11}, U_{12}, U_{13}, U_{14}\}$.

![Figure 6. The key tree of a dynamic conference.](image)

There are four steps for building a dynamic conference.

- **Building a Steiner tree.** Let $ST(R)$ be a Steiner tree containing all the $R$ nodes and nodes in the path from the leaf node to the root.
- **Finding the subtrees.** Let $T$ be the whole key tree. The subtrees are $T - ST(R)$. As Figure 6 shows, the subtrees consist of nodes linked by black lines with curves.
- **Sending CK safely.** The member who sets up the conference generates a CK encrypts it with the public keys of the subtrees’ root nodes, and broadcasts it to all the members. Referring to Figure 6, the message of the conference key distribution is $\{(ID_{SGC_4}, \{CK\}ID_{SGC_4}), (ID_{U_6}, \{CK\}ID_{U_6}), (ID_{SGC_6}, \{CK\}ID_{SGC_6})\}$.
- **Recovering CK.** When the legal members receive the message, they can decrypt it and obtain CK.

Members, including those not in the group, can complete the whole process by themselves without the GC or SGC.

Assume there is an $n$-member group with $r$ illegal members who are not in the dynamic conference and $n - r$ legal members. In the revocation algorithm, the communication overload, which is determined by the number of subtrees, is $O(r\log(n/r))$.

For the sender, the computational overload is the time to build a dynamic conference, which is proportional to the number of subtrees, i.e., $O(r\log(n/r))$. For a receiver, the computational overload is the time to decrypt the distribution message. Each member only needs to decrypt once. Hence, the overload is $O(1)$.

5. Comparison

We discuss the comparison from the aspects of performance, functionality, and security.
5.1. Performance

We compare the proposed protocols with Wallber’s method [2] and Slave’s method [3]. Wallner’s method [2] is a classic central protocol based on LKH, and a lot of current protocols are based on Wallber’s method [2]. Slave’s method [3] is a newly proposed decentralized protocol based on LKH. To demonstrate the effectiveness of the proposed protocol, both a theoretical analysis and an experimental analysis of the performance is given.

5.1.1. Theoretical Analysis

New parameters are introduced to provide a better description of the performance. Let \( h \) be the height of the key tree, and \( d_l \) the number of keys in its \( l \)th floor. \( E \) and \( D \) denote encryption and decryption operations, respectively, and \( n \) is the number of group members. \( n_i \) is the number of children of node \( i \), and \( n_i^{-1} \) is the number of children of the next-level node. Table 2 shows the theoretical comparison with Wallner’s method [2] and Slave’s method [3]. Wallner’s method [2] only has one GC. In Slave’s method [3], a group is split into a plurality of subgroups, and each group has an SGC. GM means “group member”.

| Protocol | Wallner [2] | Salve [3] | Our Protocol |
|----------|-------------|-----------|-------------|
| **Storage overload** | | | |
| GC | \( 2n - 1 \) | - | \( n_i \) |
| SGC | - | \( \sum_{l=0}^{h} d_l \) | \( 2n_i - 1 \) \( 2n_i - 1 \) |
| GM | \( \log n \) | \( h \) | \( \log n \) |
| **Computational overload (Join)** | | | |
| GC | \( (2\log n - 1)E + \log nK \) | - | \( n_i^{-1}E + K \) |
| SGC | - | \( (2h - 1)E + hE \) | \( (2\log n_i - 1)E + \log n_i K \) |
| GM | \( \log nD \) | \( hD \) | \( \log nD \) |
| **Computational overload (Leave)** | | | |
| GC | \( \log nE + (\log n)K \) | - | \( n_i^{-1}E + K \) |
| SGC | - | \( hE + hE \) | \( \log n_i E + \log n_i K \) |
| GM | \( \log nD \) | \( hD \) | \( \log nD \) |
| **Communication overload** | | | |
| Join | \( 2\log n \) | \( 2h \) | \( 3\log n \) |
| Leave | \( \log n \) | \( h \) | \( 2\log n \) |

As shown in Table 2, the overload performance depends on the height of the key tree. If the number of group members in Wallner’s method [2] is the same as that of the subgroup in Salve’s method [3], then these two methods have the same overload. Salve’s method [3] applies LKH in each subgroup; however, our protocol only requires GC/SGC to manage the next-level nodes, which reduces the workload at each level. In the communication overload, the computational overload of rekeying messages generated by our protocol is distributed to multiple SGCs. The length of the rekeying message is the sum of all the group controllers, which is consistent with the other two schemes. To distribute some private keys in the secure channel, our protocol must double the communication overload.

5.1.2. Experimental Analysis

The structures of key trees of Wallner’s method [2], Salve’s method [3], and our protocol depend on the application environment. To provide better comparability, we assume that the key trees are binary trees. The purpose of the proposed protocol is to balance the workload of the GC and the SGCs. Therefore, our experiments focused on the computational overload. The process of member joining is similar to that of member leaving. This section discusses the computational overload of member joining.
In Figure 7, the vertical axis is the number of encryption, which means the operation of generating KEK, which is an important factor of computational overload. These three protocols are based on LKH. When the numbers of group members are the same, the total encryption times of the three protocols are the same. The proposed protocol achieves the optimum load balancing effect, the workload of the GC is shared by some SGCs. Due to the binary key tree, the GC in the proposed protocol just needs to deal with two nodes, so the workload of the GC is constant. There are two types of SGCs: SGC-SGC and SGC-U. An SGC-SGC means the subgroup controller’s next-level nodes are not the group members, otherwise it is an SGC-U. In Figure 7, the three-color histogram indicates the computation overload of the GC, the SGC-U, and the SGC-SGC in the proposed protocol. When the number of group members is 30 and 50, the experiments assume they have the same numbers of SGCs, and the workload of SGC-SGC is the same. As the number of members increases, each SGC should maintain more members, and the workload of the SGC-U increases. We changed the number of SGCs in the last three histograms, and found that the workloads of the SGC-SGC and the SGC-U increases.

![Figure 7. Computational overload of member joining.](image)

Figure 7 shows the total work of the GC and the SGC. Figure 8 reflects the workloads of each SGC, GC, and GM. Assume the key tree is a standard binary tree. As shown in Figure 8, the workloads of the GC and the SGC are constant, and the workload of the GM depends on the height of the key tree. It is consistent with the above theoretical performance analysis.

![Figure 8. Computational overload of GC, SGC, and GM when a member joins.](image)

5.2. Functionality and Security

This section presents a comparison of the proposed protocol with the methods from [3,6,14,15], the method from [14], and the method [15], where the studies presented
protocols based on ABE. The method of [3] and the method of [6] proposed decentralized protocols based on LKH and blockchain, respectively. These are classic protocols applied in different scenarios, and are well-suited to the analysis of functionality and security.

As shown in Table 3, in terms of reliability, the method of [14] and the method of [15] use the technique of ABE. A group member can encrypt the group key with attributes, and a member who satisfies those attributes can decrypt the ciphertext. If the membership changes, the attributes must be changed to cover only legal members. This protocol does not increase overload for the GC or other group members. If the GC cannot work, group members can still share the group key by attributes—it is reliable; however, at the beginning, the protocol must build a large enough attribute library. Method [3] divided a group into many subgroups, where each group has an SGC, and these are isolated from each other. Once an SGC fails, a group managed by it cannot work; thus, the method of [3] has weak reliability. In the method of [6], management can be performed by every group member; the failure of a member will not affect others and the reliability is strong. In our protocol, the GC/SGC only needs to manage the next-level node, which greatly reduces the work of the GC and the SGC. We also provide an SGC replacement mechanism without extra work. Thus, our protocol is reliable.

Table 3. Comparison with other schemes.

| Protocol     | Reliability | Group Type | Security                                           |
|--------------|-------------|------------|----------------------------------------------------|
| Nabeel [14]  | Strong      | Multiple   | Unable to resist collusion attacks                 |
| Rajkumar [15]| Strong      | Multiple   | Secure against collusion attacks                   |
| Salve [3]    | Weak        | Single     | Secure against collusion attacks                   |
| Lei [6]      | Strong      | Single     | Secure against collusion attacks                   |
| Our protocol | Strong      | Multiple   | Secure against collusion attacks                   |

Protocols based on public key encryption can generally realize multi-group communication, such as between subgroups. Any group member can encrypt the communication/conference key by the public key of the subgroup or member. In contrast, protocol-based symmetric encryption has difficulty in doing this job. Hence, the method of [14], the method of [15], and our protocol can realize multi-group communication.

In security, Salve [3], Lei [6], and our protocol have been shown to resist collusion attacks, unlike most current protocols based on ABE. However, the method of [15] overcomes this security problem by using the LKH model.

6. Conclusions

The GC is one of the factors that restrict the reliability of the group key distribution protocol. In this paper, we propose a hierarchical multicast key distribution protocol that allows multi-level servers to manage group communication, greatly reducing the workload of the GC. The proposed protocol contains an SGC replacement mechanism that takes very little work to ensure that group communication still can work even if an SGC fails. Our protocol overcomes the single point of control issue that exists in traditional LKH-based protocols.

The assessment of the presented protocol reveals that future research is to further reduce the communication overload while ensuring reliability.

Author Contributions: Conceptualization, J.L. (Jie Li) and Y.W.; methodology, J.L. (Jie Li) and J.L. (Jing Liu); software, Y.W.; validation, S.Y. and Y.W.; formal analysis, S.Y. and J.L. (Jing Liu); investigation, J.L. (Jie Li); resources, S.Y.; data curation, Y.W.; writing—original draft preparation, J.L. (Jie Li); writing—review and editing, J.L. (Jie Li) and Y.W.; visualization, Y.W. and J.L. (Jing Liu); supervision, S.Y.; project administration, Y.W.; funding acquisition, J.L. (Jie Li). All authors have read and agreed to the published version of the manuscript.
Funding: This research was funded by the Open Foundation of Engineering Research Center of Cyberspace grant number NAME OF FUNDER grant number No.11, National Natural Science Foundation of China (Grant No. 61863036), the Projects of Yunnan Provincial Science and Technology Department(Grant No. 202001BB050076), and the Projects of Provincial Industrial Internet Security Situation Awareness Platform.

Data Availability Statement: Not applicable.

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

References
1. Hardjono, T.; Donsetti, L.R. Multicast and Group Security; Artech House: Boston, MA, USA, 2003.
2. Wallner, D.; Harder, E.; Agee, R. Key Management for Multicast: Issues and Architectures; RFC 2627; Internet Society: Reston, VA, USA, 1999.
3. De Salve, A.; Di Pietro, R.; Mori, P.; Ricci, L. A logical key hierarchy based approach to preserve content privacy in decentralized online social networks. IEEE Trans. Dependable Secur. Comput. 2017, 17, 2–21. [CrossRef]
4. Sun, F.; Zang, W.; Huang, H.; Farkhadinov, I.; Li, Y. Accelerometer-Based Key Generation and Distribution Method for Wearable IoT Devices. IEEE Internet Things J. 2021, 8, 1636–1650. [CrossRef]
5. Li, X.; Wang, Y.; Vijayakumar, P.; He, D.; Ma, J. Blockchain Based Mutual-Healing Group Key Distribution Scheme in Unmanned Aerial Vehicles Ad-Hoc Network. IEEE Trans. Veh. Technol. 2019, 68, 11309–11322. [CrossRef]
6. Lei, A.; Cruickshank, H.; Cao, Y.; Asuquo, P.; Ogah, C.P.A.; Sun, Z. Blockchain-based dynamic key management for heterogeneous intelligent transportation systems. IEEE Internet Things J. 2017, 4, 1832–1843. [CrossRef]
7. Rao, R.V.; Selvamani, K.; Kanimozhi, S.; Kannan, A. Hierarchical group key management for secure data sharing in a cloud-based environment. Concurr. Comput. Pract. Exp. 2019, 31, 4866.
8. Pande, A.; Joshi, D.Y.; Joshi, D.; Wadhwa, D. Analysis of Tree structure for Secure Group Communication Using LKH Approach. Int. J. Comput. Sci. Eng. 2019, 7, 1130–1136. [CrossRef]
9. Anand, A.; Conti, M.; Kaliyar, P.; Lal, C. TARE: Topology Adaptive Re-kEying scheme for secure group communication in IoT networks. Wirel. Netw. 2020, 26, 2449–2463. [CrossRef]
10. Pandi Vijayakumar, M.A.; 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. 2016, 17, 1015–1028. [CrossRef]
11. Joshi, M.Y.; Bichkar, R.S. A Centralized Key Table based Communication Efficient Group Key Management Protocol. Int. J. Comput. Netw. Inf. Secur. 2015, 7, 49–55. [CrossRef]
12. Tang, S.; Xu, L.; Liu, N.; Huang, X.; Ding, J.; Yang, Z. Provably secure group key management approach based upon hyper-sphere. IEEE Trans. Parallel Distrib. Syst. 2014, 25, 3253–3263. [CrossRef]
13. Tsitaitse, T.J.; Cai, Y.; Ditta, A. Secure self-healing group key distribution scheme with constant storage for SCADA systems in smart grid. Wirel. Pers. Commun. 2018, 101, 1749–1763. [CrossRef]
14. Nabeel, M.; Bertino, E. Poster: towards attribute based group key management. In Proceedings of the 18th ACM conference on Computer and Communications Security, Chicago, IL, USA, 17–21 October 2011; pp. 821–824.
15. Rajkumar, N.; Kannan, E. Attribute-Based Collusion Resistance in Group-Based Cloud Data Sharing using LKH Model. J. Circuits Syst. Comput. 2020, 29, 2030001. [CrossRef]
16. Zhong, H.; Luo, W.; Cui, J. Multiple multicast group key management for the Internet of People. Concurr. Comput. Pract. Exp. 2017, 29, 3817–3831. [CrossRef]
17. Kung, Y.H.; Hsiao, H.C. GROUPIT: Lightweight Group Key Management for Dynamic IoT Environments. IEEE Internet Things J. 2018, 5, 5155–5165. [CrossRef]
18. Alphand, O.; Amoretti, M.; Claeyts, T.; Dall’Asta, S.; Zanichelli, F. IoTChain: A Blockchain Security Architecture for the Internet of Things. In Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 15–18 April 2018; pp. 1–6.
19. Pal, O.; Alam, B. Key management scheme for secure group communication. In Advances in Data and Information Sciences; Springer: Berlin/Heidelberg, Germany, 2020; pp. 171–177.
20. Boneh, D.; Boyen, X.; Goh, E.J. Hierarchical identity based encryption with constant size ciphertext. In Annual International Conference on the Theory and Applications of Cryptographic Techniques; Springer: Berlin/Heidelberg, Germany, 2005; pp. 440–456.
21. Aboba, B.; Simon, D. Ppp eap tls Authentication Protocol; RFC 2716; Internet Society: Reston, VA, USA, 1999.
22. Paniwani, S.K. Private Group Communication: Two Perspectives and A Unifying Solution. Ph.D. Thesis, UC San Diego, San Diego, CA, USA, 2007.