Efficient, Authenticated, and Fault-Tolerant Key Agreement for Dynamic Peer Groups

Li Zhou and Chinya V. Ravishankar

Department of Computer Science & Engineering
University of California, Riverside
Riverside, CA 92521, USA
{lzhou,ravi}@cs.ucr.edu

Abstract. We present an efficient authenticated and fault-tolerant protocol (*AFTD*) for tree-based distributed key agreement. Our approach is driven by the insight that when a Diffie-Hellman blinded key is updated, in a tree-based method, it suffices to send the update to a small subset of the group, instead of entire group, as current methods require. Our scheme distributes each updated public key to a relatively small subgroup, called its *trust set*, greatly improving performance. Moreover, we use a threshold secret sharing method to distribute the function of the trusted authority across trust sets, thereby guaranteeing key authentication, enhancing fault-tolerance, and protecting our protocol from impersonation attacks. Our performance analysis suggests that our scheme significantly reduces the communication overhead and storage requirement.
Key Words: Secure Group Communication, Key Agreement, Key Authentication

1 Introduction

As a result of the increased popularity of group-oriented applications, such as pay-TV, distributed interactive games, teleconferencing and chat rooms, there is a growing demand for security services to achieve secure group communication. A common method is to encrypt messages with a group key, so that entities outside the group cannot decode them. A satisfactory group communication system would possess the properties of *group key security*, *forward secrecy*, *backward secrecy*, and *key authentication/integrity* [1–3]. *Group key security* ensures that it is computationally infeasible for non-group members to discover the group key. *Forward secrecy* ensures that a passive adversary cannot infer new group keys from old group keys. *Backward secrecy* ensures that a passive adversary cannot infer past group keys from the current group key. *Key authentication/integrity* ensures that public keys of group members cannot be modified by adversaries.

Broadly, there are two approaches for generating such group keys: centralized key distribution and distributed key agreement. Centralized key distribution uses a dedicated key server, resulting in simpler protocols. However, centralized methods fail entirely once the server is compromised, so that the central key

server makes a tempting target for adversaries. In addition, centralized key distribution is not suitable for dynamic peer groups, in which all nodes play the same function and role, so that it is unreasonable to make one the key server, placing all trust in it. In contrast, distributed key agreement requires each group member to contribute a share to generate the group key, and result in more complex protocols.

To achieve forward and backward secrecy, the group key must be updated on every membership change. This method is referred to as *group rekeying*. To reduce the number of rekeying operations, Wong et al. [4] proposed a logical data structure called a *key tree* that reduces the rekeying overhead from $O(n)$ to $O(\log n)$, where n is the number of nodes in the group. Based on this idea, Kim et al. proposed a tree-based key agreement protocol, *TGDH* [1], which is a combination of key tree and well-known Diffie-Hellman key exchange to generate and maintain the group key.

Unfortunately, *TGDH* suffers from two drawbacks. As explained in Section 3.1, although *TGDH* uses digital signatures, it is still prone to impersonation attacks since it is unable to assure the long-term secrecy of verification keys of those digital signatures. Also, when nodes join or leave, *TGDH* requires that the updated public keys of nodes on the key path of sponsor (defined in Section 4.3) be broadcast to the whole group on group rekeying. This is reasonable when a broadcast channel is available, but causes excessive communication overhead otherwise, especially in very dynamic groups.

1.1 Our Work

In this paper, we propose a novel **A**uthenticated, **F**ault-tolerant **T**ree-based **D**iffie-Hellman key agreement protocol, *AFTD*, based on two key ideas. First, as we point out in Section 3, it is gross overkill to broadcast updated public keys to all group members for recomputing the group key when a node n_i joins or leaves. As we show, it suffices to send each update to a much smaller subset of nodes in the tree, called its *trust set*. Second, we achieve robust key authentication by distributing the function of trusted authority among the nodes in $TS(n_i)$, using a threshold cryptographic scheme. Any k members of a node's trust set can serve as its public key certificate authority. Our performance analysis shows this distributed trust authority mechanism can significantly reduce the communication overhead and storage requirements.

AFTD has the following salient features:

- Since it sends the updated public keys to much smaller subsets of nodes, instead of broadcasting to whole group, it reduces the communication overhead from $O(n^2)$ to $O(n \log n)$ for initialization, and from $O(n \log n)$ to $O(n)$ for rekeying. It also reduces the storage requirement for blinded keys from $O(n)$ to $O(\log n)$. This feature is particularly useful when a broadcast channel is unavailable.
- It incorporates key authentication services into a tree-based key agreement protocol, and is secure against impersonation attacks.

Notation	Description
n	number of group members
SK	the system secret key
PK	the system public key
SS_{M_i}	the system secret share held by member M_i
k	number of shares SS_{M_i} required to reconstruct the certificates
M_i	i-th real group member, $i \in \{1, \ldots, n\}$
V_i	i-th virtual group member
p	p is a prime number
K_{M_i}	i-th real group member's (Diffie-Hellman) secret key, $i \in \{1, \ldots, n\}$
K_{V_i}	i-th virtual group member's (Diffie-Hellman) secret key
BK_{M_i}	i-th real group member's (Diffie-Hellman) *blinded* (or public) key, $i \in \{1, \ldots, n\}$
BK_{V_i}	i-th virtual group member's (Diffie-Hellman) *blinded* (or public) key
E_i	i-th real group member's RSA public (or verification)key, $i \in \{1, \ldots, n\}$
D_i	i-th real group member's RSA secret (or signing)key, $i \in \{1, \ldots, n\}$
α	a generator of Z_p^*
h_{M_i}	the height of the group member M_i in the key tree
$TS(M_i$ or $V_i)$	M_i or V_i's trust set

Table 1. Our Notation

– It is applicable to very dynamic groups.

The rest of this paper is organized as follows. We survey related work in Section 2. Section 3 motivates our work and defines our intrusion model. We present our solution in Section 4. In Section 5, we illustrate the advantages of our protocol by performance analysis and comparison. Finally we make a conclusion in Section 6.

Throughout the paper, we use the notation in Table 1 to describe security protocols and cryptographic operations.

2 Related Work

The use of logical key trees for scalable group key management was independently proposed by Wong et al. [4] and Wallner et al. [5]. Key trees reduce the number of rekeying operations to $O(\log n)$, which was shown to be optimal in [6].

Key trees were first proposed for centralized group key distribution, but Kim et al. [1] adapted it to a fully distributed, contributory key agreement protocol *TGDH*. Key trees are generally represented as binary trees, and created separately by each group member. Each leaf node is associated with a *real* group member, while each non-leaf node corresponds to a subgroup of group G, and is considered a *virtual member*. In Figure 1, virtual member node V_4 corresponds to the subgroup that contains the two real group members M_3 and M_4.

In *TGDH*, every node on the key tree has a Diffie-Hellman key pair based on the prime p and generator α. A real member M_i has key pair $\{K_{M_i}, BK_{M_i} =$

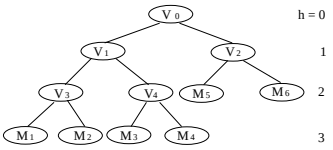


Fig. 1. Key Tree. M_i and V_i are real and virtual members, respectively.

$\alpha^{K_{M_i}} \bmod p\}$, where K_{M_i} is the secret key, and BK_{M_i} is the public key, or also called as *blinded key.* Similarly, virtual member V_i has the key pair $\{K_{V_i}, BK_{V_i} = \alpha^{K_{V_i}} \bmod p\}$. These Diffie-Hellman key pairs are used to compute the group key. Consider a node M_v whose left child is M_{lv} and right child node is M_{rv} (to simplify the description, we do not distinguish real members from virtual members here). M_i's secret key can be computed in the usual Diffie-Hellman fashion as follows:

$$K_{M_v} \equiv (BK_{lv})^{K_{rv}} \equiv (BK_{rv})^{K_{lv}} \bmod p \tag{1}$$

Since all blinded keys are well-known, each group member can compute the secret keys of all nodes on its key path, comprising the nodes from the leaf node up to the root. The root node's secret key K_{V_0} is known to all group members, and becomes the group key. Figure 1 shows a group with six members. Group member M_2 knows the key pairs of M_2, V_3, V_1 and V_0. V_0's secret key is the group key.

As discussed in Section 3.1, *TGDH* is a simple protocol, but has some drawbacks. First, it requires every updated blinded key to be broadcast to the whole group on group rekeying, causing significant overhead. Second, it is vulnerable to impersonation attacks. In this paper we solve this problem by designing a distributed trust mechanism.

Steiner et al. [7, 2] considered the problem of key agreement in dynamic peer groups and proposed a family of Group Diffie-Hellman (*GDH*) protocols by extending the classical two-party Diffie-Hellman key exchange to obtain a multiparty version. Furthermore, Ateniese et al. [8] proposed a new multiparty authenticated key agreement protocol based on Steiner's *GDH* protocols, which offers key authentication/integrity, key confirmation, and non-repudiation of group membership. However, some flaws in this protocol have been found by Pereira et al. [9].

Lee et al. [10, 11] have designed several tree-based distributed collaborative key agreement protocols for dynamic peer groups, and try to reduce the rekeying complexity by performing it at finite intervals. This approach groups multiple join and leave requests, and processes them at the same time. They also present an authenticated tree-based key agreement protocol, which extends the two-party authenticated Diffie-Hellman protocol proposed in [12]. As the success of their authenticated key agreement is partially based on a certificate authority, their protocol will encounter the same problems as centralized trust mechanisms.

In [13], Kong et al. provide robust and ubiquitous security support for mobile ad-hoc networks. In their scheme, they distribute the certificate authority functions through a threshold secret sharing mechanism, in which each entity holds a secret share, and multiple entities in a one-hop neighborhood jointly provide certificate services.

Our distributed trust mechanism differs from theirs in several aspects. First, our goals are different. Their goal is to provide a ubiquitous certificate service in the whole ad-hoc networks, and they are not concerned about secure group communication or group key generation and update issues. In our scheme, we focus on key authentication in a group environment, and on ensuring that group keys are generated and updated correctly. Briefly, the protocol in [13] focuses on point-to-point secure communication, while our scheme focuses on secure group communication, which is generally any-to-any communication. Second, the nodes that offer valid certificates are different in our respective scheme. In [13], any k-coalition of nodes can provide the public key certificates for any node. In our scheme, the public key certificate for any node n_i is entrusted to a much smaller subset, called the *trust set* for n_i (Section 5.1.2), such that any k members from this set can offer n_i's public key certificate. Thus, our scheme distributes the function of a trusted authority to appropriate trust sets and localizes the public keys much more.

3 Motivation and Attack Model

3.1 Motivation

The tree-based group Diffie-Hellman protocol *TGDH* [1] is a simple and reasonably efficient approach for the establishment of ephemeral keys for group sessions. New keys are established for each group session and at each node join or leave. Unfortunately, *TGDH* suffers from two significant drawbacks.

First, although *TGDH* uses authenticated channels, it still seems vulnerable to impersonation attacks. To provide authenticated blinded keys, *TGDH* suggests that every protocol message be signed by its sender using some strong public signature method such as DSA or RSA, and then verified by all receivers using the sender's public key. However, *TGDH* is a session key generation protocol, and does not address the long-term security of DSA or RSA keys. Unfortunately, while it may be safe to assume that a group member's DSA or RSA secret is preserved in the short run, but it is unsafe to assume that adversaries cannot compromise those keys in the long run. Hence, these keys must be refreshed periodically from a trusted source that is available online. A centralized, online trusted authority used for this purpose would become a single point failure, and a bottleneck when the group scales to large size. Instead, we define a *trust set* for each member M_i, and distribute the function of trusted authority across trust sets so that any k members from this set can offer M_i's public key certificate, enhancing fault-tolerance. We distribute the function of trust authority in a manner similar to the scheme in [13], which is based on (k, n)-threshold secret sharing scheme. It is pointed out in [14] that scheme in [13] is not usable in a

group with malicious members since it does not provide an important property known as *verifiability*. However, we focus instead on the correct and secure generation of group keys in the face of *outsider* attacks, so that our scheme is safe from this flaw. As we argue in 3.2, no group scheme can accommodate malicious members since they can always compromise the group by disclosing the group key.

Second, *TGDH* involves excessive communication and storage overheads caused by broadcasting updated blinded keys. This problem becomes more serious if membership events are common, i.e. there are new members joining or old members leaving frequently. Storage requirement is not a problem when real group members are general computers. However, it becomes an important issue when resource limited devices, such as PDAs, are able to join the group as qualified group members. Therefore, broadcasting updated blinded keys limits the scalability of the scheme. In our scheme, instead of broadcasting these updates, we define a subset called *trust set* for each node on the key tree, and transmit the updates only to their corresponding trust sets, so that communication and storage overheads caused by broadcasting can be reduced significantly.

In this paper, we solve the key authentication problem by introducing a distributed trusted authority, which provides verifiable public keys. We are also concerned with reducing broadcast overheads. Specifically, we address two challenges:

- How can we distribute the function of the trusted authority in an efficient and scalable fashion and apply it to a tree-based key agreement protocol?
- How can we use this distributed key authentication mechanism to reduce the communication and storage overheads of broadcasting the updated blinded keys?

3.2 Intrusion Model

Detection of group key loss is an important issue that is orthogonal to our work. If a group key is lost in any secure group communication system, the system must be reconfigured to exclude compromised nodes, and a new group key generated. We do not address this issue.

The emphasis of our work is on the correct and secure generation of group keys in the face of outsider attacks mounted by non-group members as in [15, 16]. We do not address insider attacks mounted by malicious group members because they can always reveal their own private keys or the group key to non-group members, thus causing fraudulent membership events or compromising group communication. An exception is [14], which does consider insider attacks when it addresses secure member admission control using a threshold DSA scheme in a group with potentially malicious members. However, although their scheme enables the new members to verify the correctness of the partial secret shares received from group members, unfortunately, it does not consider the more serious scenario where the malicious group members may compromise the group by disclosing the group key.

Impersonation Attacks We assume that group members transmit messages on a bandwidth-limited insecure channel. Outsider adversaries can intercept, modify or even replace the messages transmitted in plaintext. We can protect group communication with the group key. However, before the group key is generated, a group member cannot use the group key to securely send out the blinded keys of nodes on their key paths, which are required by other group members to generate the group key. Besides, when a new member joins the group, it does not know the group key, so it cannot use the group key to verify whether received messages come from real group members.

To prevent outsider adversaries from impersonating group members, we require each group member to authenticate their messages with RSA signatures. However, these RSA keys may be compromised by an active adversary, given enough time. After this point, the adversary is free to impersonate the group members whose keys it has discovered, and send false information to both new nodes as well as nodes already in the group.

We address this problem in our work as follows. For each member, we also define a trust set whose members jointly authenticate its RSA public key using a threshold scheme. Consequently, receivers can verify received messages with authenticated RSA public keys. The work in [1, 15] also uses digital signatures to provide source authentication. However, this scheme fails to consider the long-term security of those public verifying keys (Section 3.1).

In our scheme, we also introduce a trusted authority with an RSA secret-public key pair $\{SK, PK\}$ to offer initial off-line public key certificates. The secret shares of the trusted authority's secret key SK will be distributed to all group members using threshold secret sharing. We may allow adversaries to discover these secret shares over the long run, but provide protection through share updating [17] to periodically renew these secret key shares.

4 Our Solution

Throughout this paper, we will use the term *node* to refer to a position in the key tree that will be assigned to a *group member*. A group member (or simply *member*) is a participant in the group, who will communicate with other members using the group key.

4.1 Overview

TGDH [1] observes that a node n_1 in the group needs to know the blinded key of another node n_2 only if n_2 is on its *co-path*, defined as the set of siblings of each node in the key path of n_1. However, we offer the key insight that even more is true. In fact, an update of a blinded key need be sent only to a small subset of the group, instead of entire group. Because of the way Diffie-Hellman key exchanges are used in a key tree to generate the group key. the blinded key of any node n_i is only needed by the leaf nodes of the subtree rooted at n_i's sibling. This group of nodes, which we call n_i's *trust set*, forms the basis for both

improved efficiency as well as key authentication. We send each node's blinded keys only to its trust set. A node's trust set is also entrusted with the task of responding to requests for its public key, and provides key certificates using a threshold cryptographic scheme. This insight is missing from earlier work.

Key Management Phases In our scheme, each group member construct a key independently. For each real group member M_i, it has two key pairs. One is a Diffie-Hellman key pair, $\{K_{M_i}, BK_{M_i} = \alpha^{K_{M_i}} \bmod p\}$, which is used to generate the group key; and the other is an RSA secret-public key pair, $\{D_i, E_i\}$, which is used to provide source authentication. Non-leaf nodes V_i are virtual members, and have only a Diffie-Hellman key pair, represented as $\{K_{V_i}, BK_{V_i} = \alpha^{K_{V_i}} \bmod p\}$.

Group key management in our approach occurs in two phases: the *initialization phase* and the *rekeying phase*. Initialization is a one-time activity that distributes appropriate public-key certificates to trust sets. While such initialization may be done in many ways, we simplify our presentation by postulating that this function is performed by a trusted authority (TA), which subsequently goes offline. Offline here means the TA will not issue renewed public key certificates for existing group members during the process of group rekeying. New members wishing to join the group may obtain initial certificate from the TA at any time prior to join.

This TA uses an RSA secret-public key pair $\{SK, PK\}$, and establishes public key certificates for each group member M_i by signing M_i's public key with its secret key SK. M_i's public key certificate $\langle M_i, BK_{M_i}, E_i \rangle_{SK}$ is now distributed to its *trust set*, as explained in Section 4.2. Since the public key PK is well known, any member of M_i's trust set can verify this certificate and obtain M_i's public key.

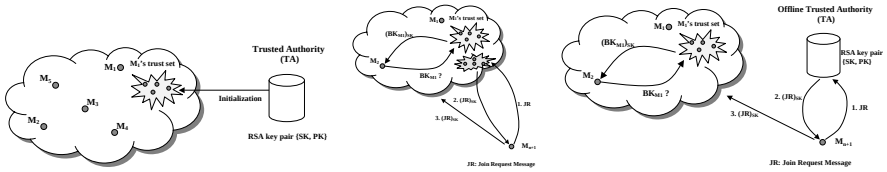
The initialization phase also distributes a secret share SK_j of the secret key SK to each group member M_j using Shamir's (k, n)-threshold sharing, which is used for creating partial public key certificates held by members of trust sets. A node M_j in M_i's trust set verifies the original certificate for M_i (signed using SK), and re-encrypts it with SK_j to create a partial certificate. Now any k members in the trust set of a given group member can offer that group member's public key certificate by group signing of certificates.

In the rekeying phase, *AFTD* includes protocols in support of the following operations:

- Join: a new member joins the group
- Leave: a member leaves the group
- Interval Multicast: a member sends messages to a subgroup of members

4.2 Initialization Phase

Assume the group G initially has n real group members $M_1, M_2, \ldots, M_n$. In the following subsections, we describe how to distribute the function of the trusted



(a) Initialization Phase

(b) Rekey Option 1: Certification by Peers

(c) Rekey Option 2: Certification by Offline Trusted Authority

Fig. 2. *AFTD* overview

authority to appropriate subgroups (*trust set*) so that any k member nodes in an appropriate subgroup can offer the corresponding valid certificate. Here "valid" means the certificate has been signed by the system secret key SK.

Distributing the system secret key shares SK_i Our design uses Shamir's (k, n)-threshold scheme [18], which provides an elegant construction of a (k, n)-threshold scheme using Lagrange interpolation. First, the trust authority randomly selects a $(k-1)$-degree polynomial $f(x) = SK + a_1 \cdot x + \cdots + a_{k-1} \cdot x^{k-1}$, such that the shared secret is $f(0) = SK$. Each group member obtains a secret share $SS_{M_i} = (f(M_i) \bmod m)$. For any k group members $\{M_1, M_2, \ldots, M_k\}$, Lagrange interpolation yields

$$SK \equiv \sum_{i=1}^{k} (SS_{M_i} \cdot l_{M_i}(0)) \equiv \sum_{i=1}^{k} SK_i \pmod{m} \tag{2}$$

where $l_{M_i}(0)$ are the Lagrange coefficients[1].

Obtaining valid certificates: The certificate X for any node is served by the node's *trust set* (Section 4.2), with each member in that trust set providing a partial certificate X^{SK_i}. With any k partial certificates, the requesting member can compute the valid certificate as follows [13]:

$$X^{SK_1} \cdot X^{SK_2} \cdots X^{SK_k} = X^{(\sum_{i=1}^{k} SK_i)} = X^{SK} \tag{3}$$

Thus, these k members can work like a trusted authority, and jointly offer the certificate. (We use the *t-bounded coalition offsetting* algorithm proposed

[1] Defined as $l_{M_i}(x) = \frac{(x-M_1)\cdots(x-M_{i-1})(x-M_{i+1})(x-M_k)}{(M_i-M_1)\cdots(M_i-M_{i-1})(M_i-M_{i+1})(M_i-M_k)}$.

in [13] to ensure that equation (3) is valid.) This approach has the nice feature that the system secret key SK is never revealed to any member node nor to any subset of member nodes. They can jointly reconstruct X^{SK}, but never SK itself. While this method can be unsafe if group members can be compromised [14], this difficulty does not arise in our case, as elaborated in Section 3.1.

Furthermore, *AFTD* improves the fault-tolerance of the system. Shamir's threshold scheme ensures that any set of $k-1$ or less secret shares cannot jointly obtain any information about the system secret key *SK*. Thus if any set of $k-1$ or less secret shares have been discovered by adversaries, the system secret key *SK* is still safe from adversaries.

Defining Trust Sets In this section, we discuss how to define the trust set for each node on the key tree. At the beginning, each group member is assigned a member ID. No special requirements are needed for this assignment, except that each member ID must uniquely identify its corresponding member. Each group member will be associated with a leaf node of the key tree in ascending order, as shown in Figure 1.

Compared with [13], our scheme enhances certificate localization by defining a trust set for each member's certificate. Only the members of this trust set need maintain the node's certificate, and any k of them can offer a valid certificate for that member.

To define trust sets, the group is firstly split into k-member clusters. The members in the last cluster may have more than k group members when n is not a multiple of k. The upper part of Figure 4 shows a 7-member group. When $k = 2$, the group is divided into 3 clusters, and the last one has three members.

Definition 1 (Trust Set). *The trust set of member M_i or V_i is the set of nodes in the union of all clusters that contain one or more leaf nodes of the subtree rooted at M_i or V_i's sibling node, and represented as $TS(M_i)$ or $TS(V_i)$.*

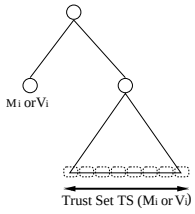


Fig. 3. Trust Set of M_i or V_i

Figure 3 illustrates this definition. In Figure 4, $TS(V_2)$ is the set of nodes in the first two clusters, which contain all leaf nodes $\{M_1, M_2, M_3, M_4\}$ of the subtree rooted at V_2's sibling node V_1.

When the number of clusters in $TS(M_i)$ is less than two, we improve fault tolerance by including in $TS(M_i)$ the first adjacent cluster formed by the leaves of the sibling of M_i.

Distributing the Certificates to Trust Sets After distributing the system secret key shares to all group members, the trusted authority distributes the stored public key certificates to appropriate trust sets. As for a non-leaf node V_i, we randomly select a real group member M_j that corresponds to one of V_i's leaf node to compute BK_{V_i} and transmit BK_{V_i} to $TS(M_i)$ with M_j's RSA signature.

After initialization, the trusted authority works offline and is used only to initialize new members joining the group. The system can provide key authentication service to renewed RSA keys without the help of the trust authority, since its function has been fully distributed to appropriate trust sets.

4.3 The Rekeying Phase

To achieve forward and backward secrecy (Section 1), the group key must be updated whenever new members join or old members leave. Each new member M_J must also obtain a system secret share SS_{M_J} and some certificates so that it can offer certificate services. In this section, we present protocols to deal with these issues.

Localized Self-initialization When an uninitialized new member M_J joins the group, a k-coalition of old members which are share holders $\{M_{J_1}, M_{J_2}, \cdots M_{J_k}\}$, can jointly offer a system secret share SS_{M_J} to M_J:

$SS_{M_J} \equiv \sum_{i=1}^{k} SS_{M_{J_i}} \cdot l_{M_{J_i}}(M_J) \equiv \sum_{i=1}^{k} SS_{J_i} \pmod{m}$.

The Lagrange coefficients and SS_{J_i} are known by M_J, so $SS_{M_{J_1}}$ can be derived directly. Here we use the *shuffling* scheme presented in [13] to maintain the secrecy of M_{J_i}'s share. For details, please see [13].

The Join Protocol Assume that the group has n members $\{M_1, M_2, \ldots, M_n\}$, and that a new member M_{n+1} wishes to join the group. M_{n+1} is required to authenticate itself by presenting a join request signed with the system secret key SK. M_{n+1} may obtain a signature on its join request either by establishing credentials with the offline trusted authority, or by enlisting the cooperation of at least k nodes from the group G willing to recognize and certify M_{n+1}'s request.

The new member M_{n+1} now broadcasts this signed join request to the group. When the other group members receive this request, they independently determine M_{n+1}'s insertion node [1] in the key tree, which is always the shallowest rightmost node, or the root node when the key tree is well-balanced.

The Join Sponsor The join protocol also requires a *join sponsor* node M_S [1] which is a real group member responsible for coordinating the join. We define the join sponsor to be the rightmost leaf node in the subtree rooted at the insertion node.

No keys change in the key tree at a join, except the blinded keys for nodes on the key path for the sponsor node. The sponsor simply recomputes the group key, and sends updates for blinded keys on its own key path to their corresponding trust sets. Each member M_j of the sponsor node's trust set creates a new partial share SS'_j of the secret key SK (Section 4.3), and forwards it to M_{n+1}, which combines them to obtain its new secret share SK_{n+1}. The new node M_{n+1} also sends its signed certificate to the members of its trust set $TS(M_{n+1})$, and gets the public keys needed for generating the group key. The join works as shown in Algorithm 1.

Algorithm 1 Join Protocol in *AFTD*

1: The new member M_{n+1} broadcasts the signed join request to the group.
2: Group members determine the insertion point, and update their key trees by creating a new intermediate node and promoting it to become the parent of both the insertion node and M_{n+1}.
3: Each group member adjusts the clusters in its key tree by adding M_{n+1} to the smallest cluster adjacent to the insertion point, or to the cluster on its right one in case of a tie. If the size of the modified cluster goes up to $2k$, split it into two clusters.
4: The sponsor M_S computes the new group key, and sends the updated blinded keys of nodes on its key path to their corresponding trust sets. These messages are signed by the sponsor M_s.
5: The members in these trust sets request the sponsor M_S's certificate from $TS(M_S)$ to verify the updated blinded keys they received.
6: M_{n+1} obtains its secret share SK_{n+1} from $TS(M_S)$.
7: M_{n+1} sends its valid public key certificates to its trust set, and gets the public keys needed for generating the group key.

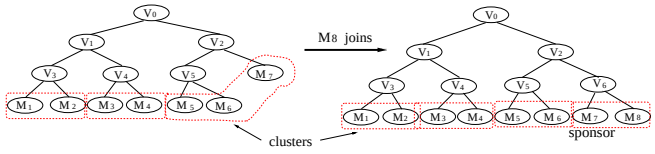


Fig. 4. Join Process in AFTD

In Figure 4, M_8 joins a 7-member group, and $k = 2$. The join sponsor M_7 creates a new intermediate node V_6 in the key tree and promotes it to become

the parent of M_7 and M_8. The sponsor M_7 computes the new group key, sending the updated BK_{V_6} and BK_{V_2} to their corresponding trust sets $\{M_5, M_6, M_7, M_8\}$ and $\{M_1, M_2, M_3, M_4\}$ respectively. Finally as the size of the third cluster is extended to $2k = 4$, it splits into two clusters: $\{M_5, M_6\}$ and $\{M_7, M_8\}$.

Leave Protocol Assume that we have n members and a member M_L wishes to leave the group. First M_L initiates the leave protocol by sending a leave request. When the other group members receive the request, they independently determine the sponsor node, which is defined as in [1] to be the right-most leaf node of the subtree rooted at the leaving member's sibling node. The leave protocol works as shown in Algorithm 2.

Algorithm 2 Leave Protocol in *AFTD*

1: The former sibling node of M_L is promoted to replace M_L's parent node.
2: The size of the cluster that formerly contained M_L is decreased by one, and merges with an adjacent cluster if its size drops below k. The new cluster may split if its size is $2k$ or larger.
3: The sponsor M_S picks a new secret key K'_{M_s}, computes the new group key, and sends the updated blinded keys of nodes on its key path to their corresponding trust sets. These messages are signed by the sponsor M_s.
4: The members in these trust sets request M_S' certificate from $TS(M_S)$ to verify the updated blinded keys they received.

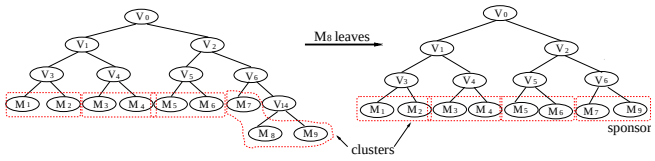


Fig. 5. Leave Process in AFTD

In Figure 5, M_8 leaves a 9-member group where $k = 2$. The sponsor M_9 picks a new secret key K_{M_9} and computes the new group key, sending updated BK_{M_9}, BK_{V_6} and BK_{V_2} to their corresponding trust sets $\{M_5, M_6, M_7, M_9\}$, $\{M_5, M_6, M_7, M_9\}$ and $\{M_1, M_2, M_3, M_4\}$ respectively.

Interval Multicast Protocol *AFTD* can also realize secure interval multicast, in which a group member wants to send data to a subgroup of group G. This problem is discussed by Gouda et al. [19], who describe a new use of key trees. They are concerned about using the existing subgroup keys in the key tree to

securely multicast data to different subgroups within the group. Unlike their approach, which depends on a centralized key server to maintain the unique key tree and manage all keys, *AFTD* solves this problem in a distributed fashion.

Let the sending member be M_0, and the destination set be $SG = \{M_{R_1}, M_{R_2}, \ldots, M_{R_m}\}$. Establishment of secure interval multicast key proceeds as in Algorithm 3.

Algorithm 3 Interval Multicast Protocol of *AFTD*

1: M_0 first determines the smallest set of subtrees in the key tree whose leaves cover SG. Let these subtrees be rooted at nodes $\{N_1, N_2, \cdots, N_t\}$.
2: $\forall N_i$, $M_0 \longrightarrow TS(N_i)$: asking for N_i's blinded key certificate
3: $\forall N_i$, $TS(N_i) \longrightarrow M_0$: N_i's blinded key certificate
4: $\forall M_i \in SG$, $M_i \longrightarrow TS(M_0)$: asking for M_0's blinded key certificate
5: $\forall M_i \in SG$, $TS(M_0) \longrightarrow M_i$: M_0's blinded key certificate

After these operations, the sender M_0 and the receivers can establish secure channel(s) between them by generating subgroup key(s) of M_0 and SG_i:

$$(\alpha^{K_{M_0}})^{K_{N_i}} = \alpha^{K_{M_0} \cdot K_{N_i}}$$

Secret Keys Updating In *AFTD*, each group member is required to update its blinded key before each group session, or during a session when it is selected as a sponsor on members' leaving. The source authentication of the updated blinded keys is guaranteed by the sender's RSA signature. Further, to ensure the long-term secrecy of the RSA keys, *AFTD* requires that each group member renew its RSA key pair periodically and sent to its trust set by current RSA secret key. When new public keys are received, all former keys can be safely deleted. We do not address the renewal of RSA keys, since this is well-studied.

Secret Shares Updating Our scheme is based on (k, n)-threshold secret sharing, which allows at most $k-1$ secret shares to have been compromised without compromising the system. However, after a long enough period of time, an adversary may compromise more than $k-1$ secret shares. So the group members periodically update their system secret shares to invalidate compromised secret shares. We adopt the proactive secret share update algorithm proposed in [17].

4.4 Security of Trusted Authority

The trusted authority is on-line during initialization, but remains offline subsequently. During initialization, the trusted authority distributes valid key certificates and secret shares of its secret key SK, so that the function of key authentication can be realized and distributed across appropriate trust sets. Since the duration of initialization is relatively short, it is safe for us to use a single trusted authority at that time.

During the rekeying phase, the trusted authority may be approached by new group members for authentication and creation of valid key certificates for them. In this mode, the trusted authority works offline, in that it only communicates with new group members, making compromises of TA unlikely.

4.5 The Number of Rekeying Messages Received

Our scheme has the nice property that each group member handles nearly the same number of messages due to rekeying operations. On a rekeying event, all members in the trust sets of the nodes on the sponsor's key path will receive an updated blinded key.

Let us consider the union of the trust sets of the nodes on a key path. As in [1], we use the term *co-path*, defined as the set of siblings of each node on the key path of member M_i. The nodes on a co-path have disjoint subtrees, so that the set of leaf nodes for these subtrees are also disjoint. Thus, the leaf nodes of each node in the co-path fall into clusters with minimal overlaps.

Because of the way we define trust sets (Section 4.2), the trust sets of the nodes on the key path of member M_i have small overlap. Consequently, each group member receives nearly the same number of rekeying messages in our scheme. For example, in Figure 4, V_6 and V_2 are on the key path of the sponsor M_7. V_5 and V_1 are on the co-path of M_7, which have disjoint subtrees, so that V_6 an V_2 have disjoint trust sets. V_6's trust set is $\{M_5, M_6, M_7, M_8\}$, and V_2'trust set is $\{M_1, M_2, M_3, M_4\}$. Each member in these trust sets will receive one updated blinded key.

5 Performance Analysis

In this section, we present an analysis of our proposed authenticated tree-based key agreement protocol, and compare it to *TGDH*. We consider communication overheads, computation overhead and storage requirements. The communication overhead is measured by the number of the messages.

5.1 Communication Overheads

The Initialization Phase In the initialization phase of *AFTD*, the trusted authority or group members distributes the certificates ($\langle M_i, BK_{M_i}, E_i\rangle$ or $\langle V_i, BK_{V_i}\rangle$) of each node in the key tree to its trust set. Since every node in each trust set receives a message, the overall communication overhead of the first phase is measured by the number of nodes in all trust sets. For each node V_i, its trust set is defined as the set of nodes in the union of clusters that contain one or more leaf nodes of the subtree rooted at its sibling node. If n is the group size and h_{V_i} is the height of V_i, there are at most $2^{\lg n - h_{V_i}}$ leaf nodes on the subtree of V_i's sibling node. Since the cluster size is k, the number of shares needed to reconstruct the certificate, these nodes will fall into at most $\lceil \frac{2^{\lg n - h_{V_i}}}{k} \rceil$ clusters.

Hence, the size of V_i's trust set is no more than $\lceil \frac{2^{\lg n - h_{V_i}}}{k} \rceil \cdot k < 2^{\lg n - 1 - h_{V_i}} + 2k$ nodes. As the maximum number of nodes in each level h_{V_i} is $2^{h_{V_i}}$, the communication overhead of each level is at most $2^{h_{V_i}} \cdot (2^{\lg n - h_{VM_i}} + 2k)$. Therefore, we can compute the overall communication overhead in the initialization phase as follows:

$$C_{Initial} = \sum_{i=1}^{\lg n} 2^i \cdot (2^{\lg n - i} + 2k) = O(n \log n) \tag{4}$$

In contrast, in the initialization phase of *TGDH*, all blinded keys need to be broadcast. If n is the group size, there are $(2n-2)$ blinded keys in the key tree, since BK_{V_0} needs not be computed. Furthermore, *TGDH* combines $(2n-2)$ updated blinded keys in one message and broadcasts, so that are $(2n-2)$ times larger than in *AFTD*. Therefore, the overall communication overhead of *TGDH* at the initialization stage is $O(n^2)$, significantly higher than that of *AFTD*.

The Rekeying Phase In the rekeying phase of *AFTD*, when a new member joins or an old member leaves, keys for nodes on the sponsor node's key path must be updated and multicast. Hence for each level of the key tree, only one node's blinded key has been updated and must be multicast to its trust set. Since the size of the trust set of the nodes on level h_{V_i} is at most $\lceil \frac{2^{\lg n - h_{V_i}}}{k} \rceil \cdot k < 2^{\lg n - h_{V_i}} + 2k$ nodes, we can compute the overall communication in this stage as follows:

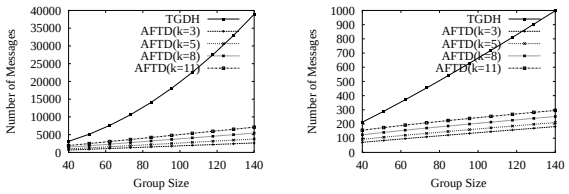
$$C_{Rekey} = \sum_{i=1}^{\lg n} (2^{\lg n - i} + 2k) = O(n) \tag{5}$$

In the rekeying phase of *TGDH*, in contrast, the blinded keys of nodes on the sponsor node's key path need to be broadcast. Since there are at most $\lg n$ nodes on a key path, the overall communication overhead of *TGDH* in this stage is $O(n \log n)$.

Figure 6 (a) and (b) show the effects of different thresholds k on the communication overheads of *AFTD*. They also compare the communication overhead of our scheme to that of *TGDH*.

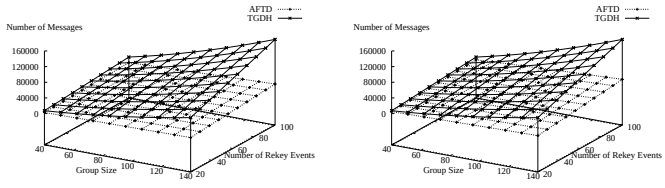
Figure 7 (a) and (b) compare the total communication overhead of *AFTD* with that of *TGDH*, where $k = 5$ and 11 separately. Total communication overhead is defined as the combined communication overhead of initialization phase and multiple rekeying events. Clearly, the communication overhead of our scheme is significantly smaller than that of *TGDH*, especially in large dynamic group scenarios.

The value of k is a system-dependent parameter, and represents a trade-off between system security and fault-tolerance. Furthermore, as seen from the above figures, the communication overhead of our scheme is insensitive to the threshold k.



(a) Initialization Phase (b) Rekeying Phase

Fig. 6. Effects of Threshold k on Communication Overhead



(a) $k = 5$ (b) $k = 11$

Fig. 7. Total Communication Overhead From Rekeying

Number of Messages Received on Rekeying On an rekeying event, blinded keys of nodes on the key path of sponsor node must be updated and transmitted. As explained in Section 4.5, each group member receives nearly the same number of rekeying messages in our scheme, because of the way trust sets are defined. Thus, the average number of messages on an rekeying event is a reasonable measure of communication overhead.

Figure 8 compares the average number of messages received by each group member on an rekeying event. As the group size increases, the average number of messages received decreases to approximately one in our scheme, while it remains at $\lg n$ in *TGDH*. For example, when the group size is 128, each group member receives around two rekeying message in our scheme, but about seven in *TGDH*. This demonstrates the scalability of *AFTD* in terms of the load experienced by group members.

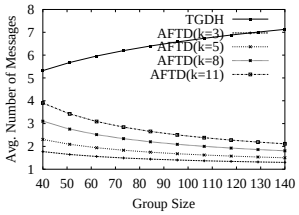


Fig. 8. Average Number of Messages

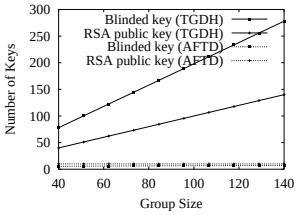


Fig. 9. Storage Requirement ($k = 5$)

5.2 Computation Overhead

Computation is required in tree-based key agreement protocols for group key generation as well as for signing rekeying messages with the sponsor's RSA secret key D_s. Since we use Diffie-Hellman to generate the group key exactly as *TGDH* does, the two methods generate the same CPU overhead for key generation.

Signed transmission of the blinded keys for rekeying in *TGDH* requires a single RSA signing because it concatenates $(2n-2)$ blinded keys(in improved *TGDH*, $\lg n$) into a single message and broadcasts it. In our scheme, the sponsor node can also concatenate $\lg n$ updated blinded keys into one message, sign it once and send it to different trust sets separately. As we pointed out in Section 5.1, the overhead of transmitting these updated blinded keys is much smaller in our scheme than in *TGDH*. Further, since *TGDH* does not provide any key authentication mechanism for RSA public keys, it can not guarantee that every group member can correctly verify the blinded keys they receive. Our method guarantees key authentication, and incurs some additional overhead. However, incorporating key authentication into *TGDH* would also incur at least the same overheads.

5.3 Storage Requirements

In *TGDH*, each group member stores the blinded keys of all nodes on the key tree, as well as all RSA public keys. Storage requirements are highest when the key tree is well-balanced, for which there are n real member nodes and $(n-1)$ virtual nodes. Thus each group member needs to store $(2n-2)$ blinded keys (the root node's blinded key needs not to be computed) and n RSA public keys.

In our method, each group member only needs to know the blinded keys of $\lg n$ nodes to generate the group key, and $2k$ RSA public keys to verify the signature. To improve fault tolerance, each member needs to store the blinded keys of $O(\log n)$ nodes. Therefore, the total storage requirement for each member in our scheme is $O(\log n)$ blinded keys and $2k$ RSA public keys. See Figure 9.

Table 2 summaries the performance comparison between *TGDH* and *AFTD*.

	Communication Overhead		Storage Requirement	
	Initialization Phase	Rekeying Phase	Blinded Key	RSA Public Key
$TGDH$	$O(n^2)$	$O(n \log n)$	$O(n)$	n
$AFTD$	$O(n \log n)$	$O(n)$	$O(\log n)$	$2k$

Table 2. Performance of $TGDH$ and $AFTD$ Compared

6 Conclusion

In this paper, we have presented $AFTD$, an efficient, authenticated and fault-tolerant tree-based key agreement protocol. Our scheme improves current schemes in several ways. First, other protocols do not consider the long-term security of digital signature keys, making them vulnerable to impersonation attacks. Our approach avoids this difficulty by ensuring the long-term security of such keys through a fully distributed approach using a threshold cryptographic scheme to distribute them appropriately within the group. We are also able to minimize the communication overheads and storage requirements of our key agreement protocol by noting that updates of public Diffie-Hellman keys need not be propagated to all group members. Central to our technique is a threshold secret sharing based method to distribute the function of trusted authority to appropriate trust sets. The public Diffie-Hellman keys are multicast to the corresponding trust sets instead of being broadcast to the whole group. Our performance analysis shows that our approach can reduce the communication overhead from $O(n^2)$ to $O(n \log n)$ in the initialization phase and from $O(n \log n)$ to $O(n)$ in the rekeying phase; the storage requirement for each member can be reduced from $O(n)$ to $O(\log n)$.

References

1. Kim, Y., Perrig, A., Tsudik, G.: Simple and fault-tolerant key agreement for dynamic collaborative groups. In: Proceedings of the 2000 ACM Conference on Computer and Communication Security, Athens, Greece (2000)
2. Steiner, M., Tsudik, G., Waidner, M.: Key agreement in dynamic peer groups. IEEE TRANSACTIONS on Parallel and Distributed Systems **11** (2000)
3. Perrig, A.: Efficient collabortive key management protocols for secure automonomous group communication. In: In Proceedings of CrypTEC'99. (1999)
4. Wong, C., Gouda, M., Lam, S.: Secure group communication using key graphs. In: Proceedings of the ACM SIGCOMM'98, Vancouver, Canada (1998)
5. Wallner, D., Harder, E., Agee, R.: Key management for multicast: Issues and architecture. In: Internet Draft, draft-wallner-key-arch-01.txt. (1998)
6. Snoeyink, J., Suri, S., Varghese, G.: A lower bound for multicast key distribution. In: Proceedings of the INFOCOMM 2001, Anchorage, Alaska (2001)
7. Steiner, M., Tsudik, G., Waidner, M.: Cliques: A new approach to group key agreement. In: Proceedings of the IEEE Internaitonal Conference on Distributed Computing Systems, Amsterdam, Netherlands (1998)

8. Ateniese, G., Steiner, M., Tsudik, G.: New multiparty authentication services and key agreement protocols. IEEE Journal of Selected Areas in Communications **18** (2000)
9. Pereira, O., Quisquater, J.: A security analysis of the cliques protocols suites. In: In Proceedings of the 14-th IEEE Computer Security Foundations Workshop. (2001)
10. Lee, P., Lui, J., Yau, D.: Distributed collaborative key agreement protcols for dynamic peer groups. In: In Proceedings of International Conference on Network Protocols (ICNP 2002). (2002)
11. Lee, P., Lui, J., Yau, D.: Distributed collaborative key agreement protcols for dynamic peer groups. Technical report, Dept. of Computer Science and Engineering, Chinese University of Hong Kong (2002)
12. Song, B., Kim, K.: Two-pass authenticated key agreement protocol with key confirmation. In: In Proceedings of Indocrypt2000. (2002)
13. Kong, J., Zerfos, P., Luo, H., Zhang, L.: Providing robust and ubiquitous security support for mobile ad-hoc networks. In: In Proceedings of the 9th International Conference on Network Protocols (ICNP2001). (2001)
14. Narasimha, M., Tsudik, G., Yi, J.H.: On the utility of distributed cryptography in p2p and manets: the case of membership control. In: Proceeding of the International Conference on Networking Protocols (ICNP'03). (2003)
15. Amir, Y., Kim, Y., Nita-Rotaru, C., Tsudik, G.: On the performance of group key agreement protocols. In: Proceedings of the ICDCS 2002. (2002)
16. Amir, Y., Nita-Rotaru, C., Stanton, J., Tsudik, G.: Scaling secure group communication systems: Beyong peer-to-peer. In: Proceedings of the DISCEX 2003, Washington DC (2003)
17. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. extened abstract, IBM T.J. (1995)
18. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979)
19. M.G.Gouda, Huang, C., E.N.Elnozahy: Key trees and the security of interval multicast. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria (2002)