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 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.

Keywords: 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* [1,2,3]. In this paper, we focus on *Key authentication/integrity*, which ensures that public keys of group members cannot be modified by adversaries. 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, thus it is unreasonable to make one the

N. Mitrou et al. (Eds.): NETWORKING 2004, LNCS 3042, pp. 759–770, 2004.

key server, placing all trust in it. In contrast, distributed key agreement requires each member to contribute a share to generate the group key, resulting in more complex protocols.

The group key is updated on every membership change for forward and backward secrecy, a method called *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 group size. Based on this idea, Kim et al. proposed a tree-based key agreement protocol, *TGDH* [1], which is a combination of key tree and Diffie-Hellman key exchange to generate and maintain the group key.

Unfortunately, *TGDH* suffers from two drawbacks. As explained in Section 3, it remains prone to impersonation attacks, and uses more messages than necessary.

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 explained 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. It suffices to send each update to a much smaller subset of nodes in the tree, called its *trust set* $TS(n_i)$. 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 scheme can 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.

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 and demonstrate performance analysis and comparison in Section 5. Finally we make a conclusion in Section 6.

2 Related Work

Key trees [4,5] were first proposed for centralized key distribution, while Kim et al. [1] adapted it to a distributed key agreement protocol *TGDH*. Every group member create a key tree separately. Each leaf node is associated with a *real* group member, while each non-leaf node corresponds to a subgroup of group G, considered a *virtual member*. In Figure 3, virtual member V_4 corresponds to the subgroup that contains 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, used to generate the group key. Secret-public key pair $\{K_{M_i}, BK_{M_i} = \alpha^{K_{M_i}} \bmod p\}$ is for real member M_i, and $\{K_{V_i}, BK_{V_i} =$

$\alpha^{K_{V_i}} \bmod p\}$ is for virtual member V_i. Public key BK_{M_i} is also called as *blinded 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 $K_{M_v} \equiv (BK_{lv})^{K_{rv}} \equiv (BK_{rv})^{K_{lv}} \bmod p$.

With all blinded keys 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. In Figure 3, 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.

Steiner et al. [6,2] proposed a family of Group Diffie-Hellman (*GDH*) protocols for dynamic peer groups. Based on them, Ateniese et al. [7] proposed a new multiparty authenticated key agreement protocol, 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. [8].

Lee et al. [9,10] have designed several tree-based distributed key agreement protocols, reducing the rekeying complexity by performing interval based rekeying. They also present an authenticated key agreement protocol. As the success of their scheme is partially based on a certificate authority, their protocol will encounter the same problems as centralized trust mechanisms.

In [11], 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 two aspects. First, our goals are different. Second, the nodes that offer valid certificates are different [12].

3 Motivation and Attack Model

TGDH [1] is a simple and efficient approach for the establishment of ephemeral keys for group sessions, however, it 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. Since adversaries can compromise those keys in the long run, these keys must be refreshed periodically from a trusted source that is available online. Our approach is to 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 [11], which is based on (k, n)-threshold secret sharing scheme. It is pointed out in [13] that scheme

in [11] is not usable in a group with malicious members since it does not provide an important property known as *verifiability*. However, we focus on the correct and secure generation of group keys in the face of *outsider* attacks mounted by non-group members as in [14,15]. 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. Thus our scheme is unaffected by the flaw pointed out in [13]. Detailed description of intrusion model appears in [12].

Second, *TGDH* involves excessive communication and storage overheads caused by broadcasting updated blinded keys. This problem becomes more serious if membership events are common. Storage requirement also becomes an important issue when resource limited devices, such as PDAs, are able to join the group as qualified group members. In our scheme, instead of broadcasting these updates, they are transmitted to a smaller subset of nodes (*trust set*), so that communication and storage overheads can be reduced significantly.

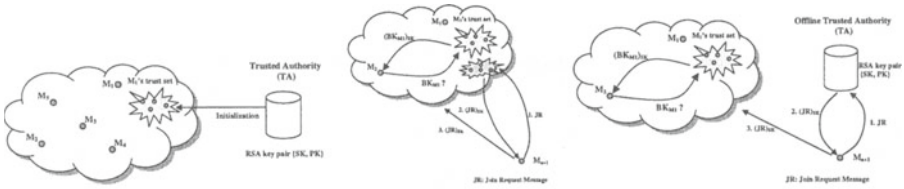
4 Our Solution

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. Each real group member M_i has two key pairs: 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 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 $\{K_{V_i}, BK_{V_i} = \alpha^{K_{V_i}} \bmod p\}$.

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



(a) Initialization Phase

(b) Rekey Option 1: Certification by Peers

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

Fig. 1. *AFTD* overview

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*. 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 [16], 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 three operations: join, leave and interval multicast.

4.2 Initialization Phase

Assume the group G has n real group members $M_1, M_2, \ldots, M_n$ initially. We describe how to distribute the function of the trusted 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 with the system secret key SK.

Distributing the system secret key shares SK_i. Our design uses Shamir's (k, n)-threshold scheme [16]. First, the TA 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 $\{M_1, M_2, \ldots, M_k\}$, La-

grange interpolation yields $SK \equiv \sum_{i=1}^{k}(SS_{M_i} \cdot l_{M_i}(0)) \equiv \sum_{i=1}^{k} SK_i \pmod{m}$, 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*, 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 $X^{SK_1} \cdot X^{SK_2} \cdots X^{SK_k} = X^{(\sum_{i=1}^{k} SK_i)} = X^{SK}$ [11]. Thus, these k members can work like a trusted authority, and jointly offer the certificate. (We use the *t-bounded coalition offsetting* algorithm proposed in [11] to ensure that the above equation 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 [13], this difficulty does not arise in our case, as explained in Section 3.

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

Defining Trust Sets. At the beginning, each group member is assigned a unique member ID and associated with a leaf node of the key tree in ascending order. To define trust sets, the group is first 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 3 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 M_i or V_i (See Figure 2) 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 $TS(V_i)$.*

In Figure 3, $TS(V_2)$ 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. Then the TA 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 TA, 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

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

M_J must also obtain a system secret share SS_{M_J} and some certificates so that it can offer certificate services.

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 as follows:

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

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 [11] to maintain the secrecy of M_{J_i}'s share.

The Join Protocol. Assume that a new member M_{n+1} wishes to join a n-member group which contains $\{M_1, M_2, \ldots, M_n\}$. M_{n+1} is required to authenticate itself by presenting a join request signed with 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.

When the other group members receive this request, they independently determine M_{n+1}'s insertion node [1] in the key tree, which is the shallowest rightmost node, or the root node when the key tree is well-balanced. They also independently determine a real member called *join sponsor*M_S [1] to take responsible for coordinating the join, which is 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, 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.

In Figure 3, 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

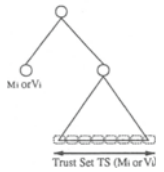


Fig. 2. Trust Set of M_i or V_i

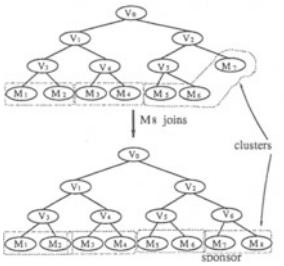


Fig. 3. Join Process in AFTD

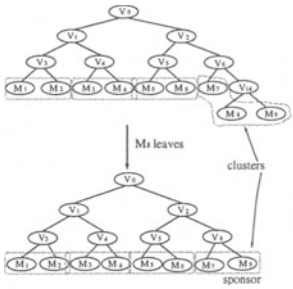


Fig. 4. Leave Process in AFTD

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 a member M_L wishes to leave a n-member 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.

In Figure 4, 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. [17], 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. For the detailed algorithm, please refer to [12].

Updating Secret Keys & Secret Shares. In *AFTD*, each group member is required to update its Diffie-Hellman keys before each group session, or during a session when it is selected as a sponsor on a member's leaving. 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 each group member to renew its RSA key pair periodically, and send it to its trust set securely using its current RSA secret key. *AFTD* adopts the proactive secret share update algorithm in [18] to periodically update the system secret shares to invalidate compromised secret shares.

4.4 Security of Trusted Authority

The trusted authority, which may be distributed, is on-line during initialization, but remains offline subsequently. During initialization, the TA 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 the TA 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

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. As in [1], we use the term *co-path* for 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 they are defined(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 3, 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's trust set is $\{M_1, M_2, M_3, M_4\}$. Each member of these trust sets will receive one updated blinded key.

5 Performance Analysis

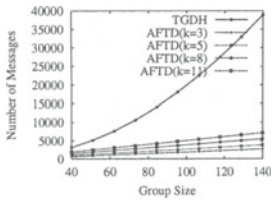
5.1 Communication Overheads

The Initialization Phase. The communication overhead is measured by the number of the messages. In the initialization phase of *AFTD*, the trusted authority distributes the certificates of each node in the key tree to its trust set.

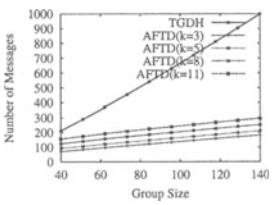
Since every node in each trust set receives a message, the overall communication overhead of the first phase is measured by the number of nodes in all trust sets.

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

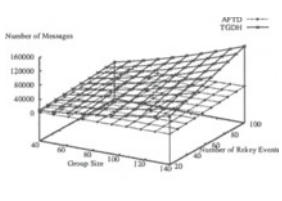
The Rekeying Phase. In the rekeying phase of *AFTD*, when a new member joins or an old member leaves, keys for nodes on the sponsor node's key path must be updated and multicast. Hence for each level of the key tree, only one node's blinded key has been updated and must be multicast to its trust set. 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 $C_{Rekey} = \sum_{i=1}^{\lg n} (2^{\lg n - i} + 2k) = O(n)$.



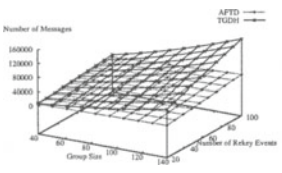
(a) Initialization Phase



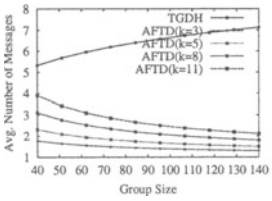
(b) Rekeying Phase



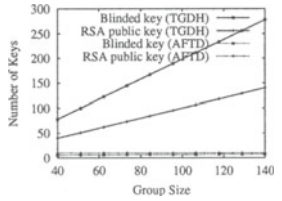
(c) Total Communication Overhead($k = 5$)



(d) Total Communication Overhead($k = 11$)



(e) Avg Number of Messages



(f) Storage Requirement ($k = 5$)

Fig. 5. Communication & Storage Overheads

Table 1. Performance of *TGDH* and *AFTD* Compared

	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	$O(n \log n)$	$O(n)$	$O(\log n)$	$2k$

Figure 5(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 5(c) and (d) 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 tradeoff between system security and fault-tolerance. Further, as seen from the above figures, the communication overhead of our scheme is insensitive to k.

Number of Messages Received on Rekeying. 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 5(e) 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.

5.2 Computation Overhead and Storage Requirements

Table 1 compares the performance of *TGDH* and *AFTD*. Due to space limitation, readers are referred [12] for a detailed analysis of computation and storage requirements.

6 Conclusion

In this paper, we have presented *AFTD*, an efficient, authenticated and fault-tolerant tree-based key agreement protocol. Central to our technique is a threshold secret sharing based method to distribute the function of trusted authority to appropriate trust sets. Our performance analysis shows that our approach can significantly reduce the communication and storage overheads.

Acknowledgement. This work is supported in part by grants from Tata Consultancy Services, Inc., and the Fault-Tolerant Networks program of Defense Advanced Research Projects Agency, under contract F30602-01-2-0536.

References

1. Kim, Y., Perrig, A., Tsudik, G.: Simple and fault-tolerant key agreement for dynamic collaborative groups. In: Proceedings of the CCS'00. (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: 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. Steiner, M., Tsudik, G., Waidner, M.: Cliques: A new approach to group key agreement. In: Proceedings of the ICDCS'98, Amsterdam, Netherlands (1998)
7. Ateniese, G., Steiner, M., Tsudik, G.: New multiparty authentication services and key agreement protocols. IEEE Journal of Selected Areas in Communications **18** (2000)
8. Pereira, O., Quisquater, J.: A security analysis of the cliques protocols suites. In: Proceedings of the 14-th IEEE Computer Security Foundations Workshop. (2001)
9. Lee, P., Lui, J., Yau, D.: Distributed collaborative key agreement protcols for dynamic peer groups. In: Proceedings of the ICNP'02. (2002)
10. 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)
11. Kong, J., Zerfos, P., Luo, H., Zhang, L.: Providing robust and ubiquitous security support for mobile ad-hoc networks. In: Proceedings of the ICNP'01. (2001)
12. Zhou, L., C.V.Ravishankar: Efficient, authenticated, and fault-tolerant key agreement for dynamic peer groups. Technical Report 88, Dept. of Computer Science and Engineering, University of California, Riverside (2003)
13. 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 ICNP'03. (2003)
14. Amir, Y., Kim, Y., Nita-Rotaru, C., Tsudik, G.: On the performance of group key agreement protocols. In: Proceedings of the ICDCS'02. (2002)
15. Amir, Y., Nita-Rotaru, C., Stanton, J., Tsudik, G.: Scaling secure group communication systems: Beyong peer-to-peer. In: Proceedings of the DISCEX'03, Washington DC (2003)
16. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979)
17. M.G.Gouda, Huang, C., E.N.Elnozahy: Key trees and the security of interval multicast. In: Proceedings of the ICDCS'02, Vienna, Austria (2002)
18. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. extened abstract, IBM T.J. (1995)