Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication*

Adrian Perrig
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

adrian_perrig@cs.cmu.edu

Abstract

This paper introduces a new family of protocols addressing collaborative group key agreement for secure group communication in autonomous groups. I present three protocols with varying degrees of security and efficiency. The first protocol provides a non-authenticated key agreement and is suitable for applications with low security requirements. The second and third protocols add a Diffie-Hellman-based authenticated key agreement to provide collaborative authentication. In particular, the third protocol uses Günther's concept of implicitly-certified public keys to achieve higher efficiency. A major advantage of the protocols is that they allow efficient "join" and "leave" operations, while preserving perfect forward and backward secrecy. These protocols improve previously proposed schemes in the following ways: first, they can be used for autonomous group key agreement, where no central server is necessary and no member has a special role. Also, the complexity is drastically reduced: compared to best currently used techniques, the number of rounds for the initial key agreement are reduced from n to $\log(n)$, and the bandwidth requirements are reduced from $O(n^2)$ to $O(n)$, where n is the number of members. In addition, I present new primitives that enforce rights management policies in the group (such as sender authorization).

Keywords: Secure group communication protocol, collaborative group key agreement, key distribution.

1 Introduction

With the explosive growth of the Internet and the shift of traditional communication services to the Internet, group communication becomes increasingly important. Simultaneously, there is a growing demand for security. Consequently, not only does Internet communication need to be secure, but also Intranet communication faces an increasing need for security for the following reasons: although companies install firewalls, high security applications need additional protection, because a firewall is usually the only serious security mechanism a company has — once an intruder has bypassed it, the remaining systems are simple to compromise. There is ample evidence that it is easy to find security leaks in large corporations. Also, large companies have contractors, agreements with other companies, temporary employees, cleaning personnel, all with access to the "secure" network behind the firewall, where one can, for example, install a tunneling device. Therefore, even Intranet communication needs additional security.

We need efficient secure group communication protocols. The parameters for these protocols include: the number of members that can be concurrently present in a group, how frequently members are expected to join or leave the group, how many senders can be present in a group, whether a group is autonomous or administered by a central server, and the security and policy requirements. In this paper, I present a protocol to establish a group key in small to medium-sized, autonomous groups, which may have highly dynamic memberships. Examples include telephone and video conferences, remote consultation and diagnosis systems for medical applications, contract negotiation, multi-party games, collaborative work places, electronic commerce environments such as on-line real-time auctions, and information dissemination of stock quotes.

Most currently proposed protocols [2, 10, 16, 17, 3, 8, 1] use a dedicated server or group controller, which results in simpler protocols, but at the cost of requiring members to trust the server. However, a central server or group controller incurs management overhead and is a central point of failure. Conversely, in an autonomous group, if

*This publication was supported in part by Contract Number 102590-98-C-3513 from the United States Postal Service. The contents of this publication are solely the responsibility of the author and do not necessarily reflect the official views of the United States Postal Service.

a member fails, it will be excluded eventually without obstructing group communication. Another drawback of server-based protocols is that electing one group member as the group controller creates an asymmetry among the members, producing problems when the group controller fails, wants to leave, or is evicted from the group. Also, server-based schemes require that all members trust the server. In addition, requiring a server connected to the Internet may inhibit its use for some applications, i.e., multi-party games, because of the required management overhead.

The major contributions of the protocols presented in this paper are the following:

- They do not rely on a central server or on a special member (a group controller).
- The communication, computation, and time complexities are one order of magnitude faster than state of the art techniques.
- They use efficient and scalable "join" and "leave" mechanisms.

The outline of the paper is as follows: Section 2 gives background information on the problem of secure group communication, and reviews previous work. Section 3 presents the protocols, along with novel security properties. In Section 4, I discuss implementation issues. The protocol complexity and security analysis are presented in Sections 5 and 6. Finally, I conclude and present ideas for future work.

2 Background

The general goal of secure group communication is to establish a common secret key (also referred to as a group key), among all group members for confidential communication. Generally, a secret group key is established by a group key agreement protocol (GKAP). Joining members and leaving members pose the problem of *backward* and *forward secrecy.* A protocol provides *perfect backward secrecy* if a member joining the group at time t does not gain any information about the content of messages communicated at times $t' < t$. A protocol provides *perfect forward secrecy* if a member leaving the group at time t does not gain any information about the content of messages communicated at times $t' > t$. Protocols need to provide these properties to provide secure group communication in dynamic groups (dynamic implies that the membership can change through joins and leaves).

A necessary condition for a GKAP to provide backward secrecy is that all information a new member receives must be unrelated to any previous group information. In particular, a new member can not use the keys he or she receives to obtain more information about previous keys than an outsider would have. Similarly, a necessary condition for a protocol to provide forward secrecy is that after a member leaves, he or she can not obtain key information for subsequent group operations. The leaving member will then not have any more information about the group than an outsider would.

Besides confidentiality, a GKAP needs to provide integrity and authentication to be secure. Message *integrity* guarantees that the message was not altered or sent by an outside attacker. Member *authentication* ensures that only valid users can join the group.

Many existing protocols are vulnerable to the *collusion attack.* A collusion attack is when members exchange private key data to decrypt future group messages, even though they were expelled. The *multi-join* problem is related: a member might "collude" with itself by joining the same group twice under different identities. The issue of a member colluding with an outside member is not as serious a problem since, as long as the legal member is in the group, the outsider will be able to send and receive group messages through his or her friend. In protocols where individual group members authenticate each other, however, one group member might infiltrate an outsider into the group, who would normally not be allowed to join the group. The problem here is that the outsider might stay in the group, even after his or her friend was evicted.

The following terminology, which is partly adapted from [9, 1], is used throughout the paper:

A *group key agreement protocol* (GKAP) is a key establishment technique in which a shared secret key is derived by two or more specified parties as a function of information contributed by each party, such that no party can predetermine the resulting value. After a successful run of the protocol all members share an identical secret key. GKAP's should also have the following desired properties.

- A key agreement protocol (KAP) is *contributory* if each party equally contributes to the key and guarantees its freshness.
- A GKAP provides *implicit key authentication* if each group member is assured that no outsider knows the secret group key, unless aided by a dishonest group member.
- A contributory key agreement protocol provides *key integrity* if every member is assured that its particular secret key is a function of only the individual contributions of all members and not of any outside information. In particular, extraneous contribution to the group key cannot be tolerated even if it does not assist an attacker with any additional knowledge.
- A KAP provides *key confirmation* if every member is assured that its peers are in possession of the common group key.
- A secure group communication protocol provides *key independence* if no sequence of group keys provides

any information on previous or future group keys. This property is a necessary condition for forward and backward secrecy.

As pointed out in [1], key integrity and key authentication are independent concepts. A protocol violates key integrity as soon as outside information influences the key, which does not necessarily have an impact on the security of the key. Conversely, an insecure GKAP might leak a secret key and violate key authentication, while providing key integrity.

Previous work

Previous work in secure group key communication protocols mainly focused on server-based systems [2, 10, 16, 17, 3, 8]. Early work on server-based schemes uses a central server, which shares a secret key with each member and uses unicast and encryption to communicate new keys securely with each member individually [7, 10]. To achieve scalability Mittra proposes a server hierarchy [10]. Wallner et. al. propose a key tree hierarchy [16], which is built by the central server and which makes key updates efficient by broadcasting a message of size $O(\log N)$ where N is the number of members. My approach uses a similar key tree hierarchy, but the key tree is generated by the members and no central server is necessary. Wong et al. extends the centrally managed key tree hierarchy to provide secure multicast service [17]. Chang et al. propose another method for efficient key update [3], which reduces the total number of keys in the group from N (in key tree schemes) to $2\log(N)$. Unfortunately this scheme is not robust against collusion attacks — members can remain in the group (even after they were expelled) by exchanging their keys. A careful inspection of the protocol reveals that the keys of a leaving member are still used for subsequent group operation, which violates forward secrecy.

Protocols for smaller groups have not received as much attention as those for large groups. Since most groups are expected to be small groups in practice[1], efficient protocols for small groups are important. For small groups, we can remove the requirement for a central server and construct collaborate group key authentication protocols.

In collaborative group key agreement protocols, Burmester and Desmedt describe star-based, tree-based, broadcast, and cyclic protocols for contributory group key agreement [2]. All protocols are variations on the Diffie-Hellman key exchange. They do not address the problems of dynamic groups, namely no “join” and “leave” operations are presented. In addition, the tree-based scheme uses a group “chair” which is at the root of the key hierarchy and is responsible to establish the common group key. I improve this scheme by eliminating the need for a trusted group chair, adding dynamic membership, and resolving practical implementation issues.

Ateniese et al. address dynamic membership issues [1]. Their system is also based on a variant of the Diffie-Hellman key agreement. The scheme, however, uses a group controller and needs N protocol rounds to establish a common key in a group of N members. The total bandwidth used is $O(N^2)$. I improve these bounds to $\lceil \log(N) \rceil$ number of rounds and $O(N)$ bandwidth requirements.

[1]Assuming that an analogue of Zipf’s law holds for the distribution of the group size, we expect to see a small number of large groups and a large number of small groups. As a *Gedanken Experiment*, we can verify the previous statement by considering telephone calls. We expect to see a high number of calls between two persons, fewer between three, etc.

3 Efficient Key Agreement Protocols for Autonomous Groups

This section describes three novel contributory group key agreement protocols for dynamic communication groups. The first protocol presented does not authenticate the individual group members, whereas the subsequent two do. All protocols are based on a binary key tree hierarchy.

3.1 Assumptions

The protocols function under the following assumptions. First, authenticated members are not malicious and are well-behaved. Therefore, two members trust each other after mutual authentication. Also, all members trust the Certificate Authority (CA) or the Key Authentication Center (KAC).

The design of the protocol depends on the failure model. As described by Powell [12], there are two basic failure models: the *crash failure* or *fail-stop* model and the *arbitrary failure* model, also referred to as *Byzantine failure* model. The arbitrary failure model describes the “worst-case”, where a program can behave in an arbitrary way. Examples for group communication systems in this setting are [11, 15, 13]. The protocols in this paper assume that members fail as in the fail-stop model, where members fail by halting. This is a reasonable model for practical application domains. Contrary to the arbitrary failure model, synchronous, reliable communication is assumed — meaning that a message will be received within a given delay. Also, although IP multicast is inherently unreliable, the protocols rely on an underlying protocol which provides reliable multicast [18, 5]. All the protocols assume that all members know the structure of the key tree and the position of each member in the tree; more details on the key tree are given in Section 4.

3.2 Notations

I use the following notation to describe the protocols:

N: Number of members that are in the group or that want to join it initially.

$E_k(M)$: Encryption of plaintext M with key k.

$D_k(C)$: Decryption of ciphertext C with key k.

$H(M)$: Hash of message M.

$S_A(M)$: Sign message M with A's key.

$A \rightarrow B : M$: A sends message M to B.

$A \rightarrow * : M$: A broadcasts message M.

In the description of the protocols, I use the following notation for binary trees. Figure 2(a) is an example of a key tree. I use d to denote the depth of the key tree. The root is at level 0 and the lowest leaves are at level d. The nodes are denoted as $\langle l, i \rangle$, where l stands for the level and i for the node number in the level, where $1 \leq i \leq 2^l$ since each level l hosts at most 2^l nodes. (This is in the case of a perfectly balanced binary tree. As we will see later, the tree is not always balanced, but to simplify the discussion of the protocols, I assume that the key tree is balanced, hence the number of members $N = 2^d$. In Section 4 I show how the protocol can handle an arbitrary number of members and also unbalanced trees.) For each node $\langle l, i \rangle$, there is a corresponding key $K_{\langle l,i \rangle}$. The members are always placed at a leaf node. I also use the notion of *subtree*: in a binary tree, every non-leaf node has a right and left subtree.

3.3 Non-authenticated group key agreement protocol (NAGKA)

The first protocol establishes a common group key without user authentication. The lack of authentication can be useful in many settings, for example where the group members prefer to remain anonymous, or where the members do not share a commonly trusted third party. The protocol is a variation of the non-authenticated Diffie-Hellman 2-party key agreement [4], extended to group key agreement. The scheme is described in detail in the boxes labeled Protocol 1 and Procedure 1. Figure 1 shows the key tree in a group with 4 members. After the protocol, all group members share the root key and know only the keys on the path from their leaf node up to the root. The key tree hierarchy is similar to the one proposed by Wallner et al. [16], but they use a central server to establish the key hierarchy, which initially sends the keys by unicast to the individual members. I adopted the key tree hierarchy because it provides perfect forward and backward secrecy, and it is robust against collusion attacks, with little overhead (every member only needs to know $\log(N)$ keys).

As an overview, the protocol works as follows: The key tree is constructed from the leaves up to the root. Initially each member chooses a random number, for the key value of its leaf node, which collectively form the lowest level of the key tree. For the construction of each subsequent layer, two members establish a key through the key agreement procedure, one member from the left subtree and the other from the right subtree. They both broadcast their part of the Diffie-Hellman key agreement, which allows all the members of the subtree to compute

Initialization: There are $N = 2^d$ members $\{M_i | (1 \leq i \leq N)\}$ that participate in the group, where d is the depth of the key tree. An appropriate prime p and generator α of $\mathbb{Z}_p^*(2 \leq \alpha \leq p-2)$ are selected and published. Each user M_i selects a random number $r_i \in \mathbb{Z}_p^*$ and sets the keys of level d to $K_{\langle d,i \rangle} = r_i$.

Iterate over j: $(1 \leq j \leq d)$
In each step all keys $K_{\langle l,v \rangle}$ of level $l = d - j$ are constructed, $1 \leq v \leq 2^l$.
Run the key agreement Procedure 1 for each
$v : (1 \leq v \leq 2^l)$, passing (j, v, l) as arguments.

Finally: All members M_i share the root key and know only the keys on the path from their leaf node up to the root.

Protocol 1: Non-authenticated group key agreement protocol (NAGKA)

Key_agreement_1(j : **int**, v : **int**, l : **int** **)** **:** $K_{\langle l,v \rangle}$
j stands for the current round, v is the node number of the current level l.
The leftmost leaf node of the subtree rooted at node $\langle l, v \rangle$ is $v_l = (v-1) \cdot 2^j + 1$ and the rightmost leaf is $v_r = v \cdot 2^j$. Choose a random member of the left subtree $M_{i'}(v_l \leq i' \leq \frac{v_l + v_r - 1}{2})$ and one from the right subtree
$M_{i''}(\frac{v_l + v_r + 1}{2} \leq i'' \leq v_r)$. The left key of the node $\langle l, v \rangle$ has the index $v' = 2v - 1$; the right index, $v'' = 2v$. This is shown in Figure 2(b).

1. $M_{i'} \rightarrow * : \alpha^{K_{\langle l+1,v' \rangle}} \mod p$
2. $M_{i''} \rightarrow * : \alpha^{K_{\langle l+1,v'' \rangle}} \mod p$
3. All members of the subtree of node $K_{\langle l,v \rangle}$ can now compute the new key of this level

$$\begin{aligned} K_{\langle l,v \rangle} &= (\alpha^{K_{\langle l+1,v' \rangle}})^{K_{\langle l+1,v'' \rangle}} \mod p \\ &= (\alpha^{K_{\langle l+1,v'' \rangle}})^{K_{\langle l+1,v' \rangle}} \mod p \\ &= \alpha^{K_{\langle l+1,v' \rangle} \cdot K_{\langle l+1,v'' \rangle}} \mod p \end{aligned}$$

Procedure 1: Establish key $\langle l, v \rangle$ using non-authenticated key agreement

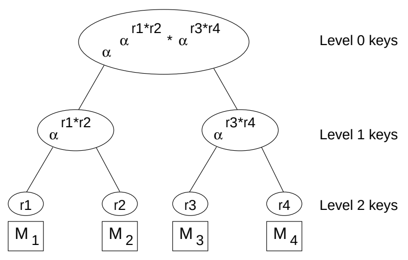


Figure 1: Non-authenticated group key tree (depth = 2)

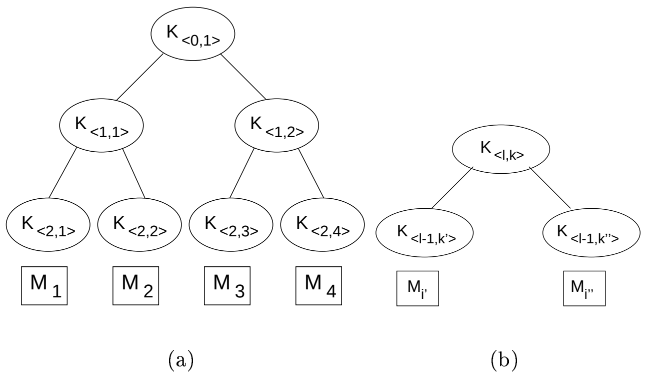


Figure 2: (a) Notation of the nodes of a group key tree of depth 2. (b) shows the key indices and members associated within one step of the key agreement, corresponding to the notation used in the protocol description.

the key of the current level. Once the root key is established, the group can start the secure communication by encrypting all messages with the root key.

Since this protocol does not provide authentication, anybody can join the group, which violates confidentiality. In addition, this protocol is based on the basic Diffie-Hellman key agreement, which is vulnerable to a man-in-the-middle attack, which also compromises confidentiality. To solve these problems, I replace the unauthenticated Diffie-Hellman key agreement with authenticated versions. Hence, the second protocol uses an authenticated version of the Diffie-Hellman key agreement, and the third protocol uses Günther's key agreement.

3.4 Authenticated group key agreement (AGKA)

For many applications mentioned in the introduction, the members need to be authenticated, e.g., members of a secure conference telephone call. I will apply an authenticated Diffie-Hellman scheme for this purpose.

The AGKA protocol is similar to Protocol 1, except that the two group members (which establish the parent key node of their current key nodes) also perform a mutual authentication when exchanging the exponents. The authentication is based on digital signatures and a public key infrastructure (PKI). When exchanging the keys, both members sign the concatenation of the exponent, the members position in the tree, and a timestamp. The details are described in the box of Protocol 2 and Procedure 2, again for the case where $N = 2^d$, for clarity.

3.5 Günther's implicitly-certified public keys

One disadvantage of the AGKA protocol is that it uses certificates and a PKI. For each key agreement, the certificates need to be exchanged, which consumes bandwidth because of the large size of certificates. Furthermore, the signature of the PKI needs to be verified, which is computationally expensive. For these reasons, the third protocol uses Günther's *identity based key agreement* [6], also known as *implicitly-certified key agreement protocol.* This protocol is reproduced here to keep this paper self-contained. Also, the original paper does not detail all the steps necessary for the key agreement.

This protocol is similar to Protocol 1, except that in each step the members $M_{i'}$ and $M_{i''}$ perform mutual authentication.

One-time initialization: Each member M_i registers his or her public key K_{M_i} at a Certificate Agency (CA) which then issues the certificate which is public and assumed to be known by all the other members. An appropriate prime p and generator α of $\mathbb{Z}_p^*(2 \le \alpha \le p-2)$ are selected and published. Each user M_i selects a random number $r_i \in \mathbb{Z}_p^*$ and sets the keys of level d to $K_{\langle d,i \rangle} = r_i$.

Iterate over j: $(1 \le j \le d)$
In each step all keys $K_{\langle l,v \rangle}$ of level $l = d - j$ are constructed, where $1 \le v \le 2^l$.
Run the key agreement Procedure 2 for each
$v : (1 \le v \le 2^l)$, passing (j, v, l) as arguments.

Finally: All members M_i share the root key and have all the keys on the path from their leaf node up to the root. All members know that every other member was at least authenticated once by another member.

Protocol 2: Authenticated group key agreement protocol (AGKA)

Key_agreement_2(j : **int**, v : **int**, l : **int** **)** : $K_{\langle l,v \rangle}$
j stands for the current round, v is the node number of the current level l.
The leftmost leaf node of the subtree rooted at node $\langle l, v \rangle$ is $v_l = (v-1)\cdot 2^j + 1$ and the rightmost leaf is $v_r = v \cdot 2^j$. Choose a random member of the left subtree $M_{i'}(v_l \le i' \le \frac{v_l+v_r-1}{2})$ and from the right subtree $M_{i''}(\frac{v_l+v_r+1}{2} \le i'' \le v_r)$. The left key of the node $\langle l, v \rangle$ has the index $v' = 2v-1$, the right index is $v'' = 2v$.

1. $M_{i'} \rightarrow * : \alpha^{K_{\langle l+1,v' \rangle}} \bmod p, i', t', S_{M_{i'}}(\alpha^{K_{\langle l+1,v' \rangle}} \bmod p, i', t')$ with t' being a recent time stamp.
2. $M_{i''} \rightarrow * : \alpha^{K_{\langle l+1,v'' \rangle}} \bmod p, i'', t'', S_{M_{i''}}(\alpha^{K_{\langle l+1,v'' \rangle}} \bmod p, i'', t'')$ with t" being a recent time stamp.
3. Every member in the subtrees can verify the validity of the digital signature and compute the new key

$$\begin{aligned} K_{\langle l,v \rangle} &= (\alpha^{K_{\langle l+1,v' \rangle}} \bmod p)^{K_{\langle l+1,v'' \rangle}} \bmod p \\ &= (\alpha^{K_{\langle l+1,v'' \rangle}} \bmod p)^{K_{\langle l+1,v' \rangle}} \bmod p \\ &= \alpha^{K_{\langle l+1,v' \rangle} \cdot K_{\langle l+1,v'' \rangle}} \bmod p \end{aligned}$$

In case one signature is invalid, they will notify the group and the dishonest member is expelled.

Procedure 2: Establish key $\langle l, v \rangle$ using mutual authentication

Günther's authentication scheme is based on a trusted key authentication center (KAC), similar to a certificate authority (CA) in a PKI. The KAC pre-authenticates a user, issues his or her private and public information, and makes sure that the identity identifiers are globally unique.

This authentication scheme is split up into three phases: First, an initial set-up phase where the KAC distributes its public information; second, a pre-authentication phase where each member sets up his or her public/private information with the KAC; and finally, the authentication phase where two members perform mutual authentication. During the mutual authentication phase, both members exchange their public information and a blinded nonce. If both members can compute a common secret key, it implies that they were both pre-authenticated through the KAC. Since the "name" of each member is used to reconstruct the user's "certificate" which is used during the authentication, the scheme is also called identity-based key agreement. The box of Procedure 3 shows the details of Günther's implicitly-certified public keys.

3.6 Authenticated group key agreement using Günther's scheme (AGKA-G)

This protocol, AGKA-G, is similar to AGKA, except that instead of public key certificates, it uses Günther's scheme for mutual authentication. An additional difference, however, is that the messages exchanged in the mutual member key agreements do not allow the other members in the same subtree to generate the node key. Therefore the two members use unicast to exchange their public information and to establish the new node key. To pass the new key on to the other members residing in the same subtree, both members encrypt the new key with the root key of their respective subtree, and broadcast it to the members in the same subtree. For example, in Figure 2, if M_1 and M_3 perform the key agreement for the root key $K_{\langle 0,1\rangle}$, M_1 will use $K_{\langle 1,1\rangle}$ to encrypt the new key $K_{\langle 0,1\rangle}$ before sending it to the other members in the same subtree. In the same way, M_3 uses $K_{\langle 1,2\rangle}$ for encryption.Again, I describe AGKA-G in Protocol 3 for the case where the binary key tree is balanced ($N = 2^d$).

3.7 Join and leave procedures

Since the protocols are used for dynamic communication groups, mechanisms to allow members joining and leaving the group are necessary, without compromising forward and backward secrecy, and key independence. For both the join and leave procedures, I assume a general binary tree structure, not necessarily balanced, as shown in Figure 3. First, I discuss the case of a new member joining the group. The box labelled Procedure 5 contains the details of the join protocol.

In this key agreement protocol, there is a trusted third party, the KAC. A and B set up their identities and do the key agreement with authentication.

Initialization: The KAC chooses and publishes a prime p and generator α of $\mathbb{Z}_p^*$. KAC selects $x \in \mathbb{Z}_p^*$ randomly, not divisible by the largest prime factor of $p-1$, which is its private key. $y = \alpha^x$ is also published.

Pre-authentication: A sends the KAC its identity information I_A which the KAC checks for uniqueness and then issues the corresponding private key.
$A \rightarrow T : I_A$
The KAC then chooses randomly k_a such that $\gcd(k_a, p-1) = 1$ and computes $P_a = \alpha^{k_a} \mod p$ and $k_a^{-1} \mod (p-1)$. Then the KAC computes $a = k_a^{-1} \cdot (H(I_A) - x \cdot P_a) \mod (p-1)$, where H is a cryptographic hash function. Furthermore k_a is private to the KAC, a is sent back to A as its secret key. P_a and I_A are both published.
KAC$\rightarrow A : a, P_a$ (a is passed through a secure channel.)
B goes through the same procedure to receive his or her secret key b and public information P_b and I_B.

Key agreement:

1. $A \rightarrow B : I_A, P_a$
2. $B \rightarrow A : I_B, P_b$
3. A chooses secret r_a at random, B picks r_b
4. $A \rightarrow B : (P_b)^{r_a} \mod p$
5. $B \rightarrow A : (P_a)^{r_b} \mod p$
6. Both can now compute the common key
$K = \alpha^{k_a \cdot a \cdot r_b + k_b \cdot b \cdot r_a}$
We walk through A's steps, B's are analogous.
A knows $a, I_B, P_b = \alpha^{k_b} \mod p, r_a, (P_a)^{r_b} \mod p = \alpha^{k_a \cdot r_b} \mod p, \alpha,$ and $\alpha^{k_a} \mod p$. The first part of the product is simple to compute from the known information:
$\alpha^{k_a \cdot a \cdot r_b} \mod p = ((P_a)^{r_b})^a \mod p = \alpha^{k_a \cdot a \cdot r_b} \mod p$
The computation of $\alpha^{k_b \cdot b \cdot r_a} \mod p$ is as follows.
$\alpha^{k_b \cdot b} \mod p = \alpha^{H(I_B) - x \cdot P_b} \mod p = \frac{\alpha^{H(I_B)}}{(\alpha^x)^{P_b}} \mod p$. If we now raise the final term to the r_ath power, and multiply, we get: $K = \alpha^{k_a \cdot a \cdot r_b} \cdot (\alpha^{k_b \cdot b})^{r_a} \mod p = \alpha^{k_a \cdot a \cdot r_b + k_b \cdot b \cdot r_a} \mod p$. We can see how the public information I_B, as well as the private r_a is used to compute K.

Procedure 3: Günther's implicitly-certified public key

Since the join procedure needs to provide perfect backward secrecy, the joining member cannot receive any of the old keys in the key tree. Hence all the keys from the leaf node of the new member up to the root need to be changed. The new member joins at the closest node to the root. In the example shown in Figure 3(a), the new member M_5 joins at node $\langle 1,2\rangle$. Member M_4 is moved one level down to accommodate M_5, as shown in Figure 3(b). All keys from the new member up to the root need to be newly established, in the case shown in Fig-

This protocol is similar to Protocol 2, except that Günther scheme, described in Procedure 3, is used for mutual authentication, instead of certificates and PKI.

One-time initialization: Each member M_i registers his or her identity I_{M_i} at the trusted third party T, which then issues its corresponding private information m_i and the public information P_{M_i}. The public information known by each member is: α, α^t, p, q.

Iterate over j: $(1 \leq j \leq d)$
In each step all keys $K_{\langle l,v\rangle}$ of level $l = d - j$ are constructed, where $1 \leq v \leq 2^l$ keys in total.
Run the key agreement Procedure 4 for each v : $(1 \leq v \leq 2^l)$, passing (j, v, l) as arguments.

Finally: All members M_i share the root key and have all the keys on the path from their leaf node up to the root.

Protocol 3: Authenticated group key agreement protocol using Günther's identity-based key agreement

Key_agreement_3(j : **int**, v : **int**, l : **int** **)** : $K_{\langle l,v\rangle}$
j stands for the current round, v is the node number of the current level l.

The leftmost leaf node of the subtree rooted at node $\langle l, v\rangle$ is $v_l = (v-1)\cdot 2^j + 1$ and the rightmost leaf is $v_r = v \cdot 2^j$. Choose a random member of the left subtree $M_{i'}(v_l \leq i' \leq \frac{v_l+v_r-1}{2})$ and from the right subtree
$M_{i''}(\frac{v_l+v_r+1}{2} \leq i'' \leq v_r)$. The left key of the node $\langle l, v\rangle$ has the index $v' = 2v - 1$, the right index is $v'' = 2v$.

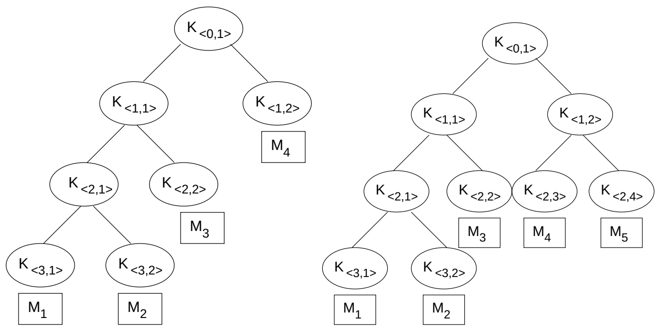
1. $M_{i'} \rightarrow M_{i''} : I_{M_{i'}}, P_{M_{i'}}$
2. $M_{i''} \rightarrow M_{i'} : I_{M_{i''}}, P_{M_{i''}}, (P_{M_{i'}})^{K_{\langle l+1,v''\rangle}} \mod p$
3. $M_{i'} \rightarrow M_{i''} : (P_{M_{i''}})^{K_{\langle l+1,v'\rangle}} \mod p$
4. From this information, both members can compute the key of the current level $K_{\langle l,v\rangle}$ following Protocol 3.
5. To verify that both members computed the same key, $M_{i'}$ and $M_{i''}$ exchange the following messages.
   $M_{i'} \rightarrow M_{i''} : E_{K_{\langle l,v\rangle}}(i')$
   $M_{i''} \rightarrow M_{i'} : E_{K_{\langle l,v\rangle}}(i'')$
   If authentication fails, the valid member will contact the other members to expell the dishonest outsider.
6. Both $M_{i'}$ and $M_{i''}$ then send the new key $K_{\langle l,v\rangle}$ encrypted to all members below the key node.
   $M_{i''} \rightarrow * : E_{K_{\langle l+1,v''\rangle}}(K_{\langle l,v\rangle})$
   $M_{i'} \rightarrow * : E_{K_{\langle l+1,v'\rangle}}(K_{\langle l,v\rangle})$

Procedure 4: Establish key $\langle l, v\rangle$ using authenticated the Günther key agreement

ure 3(b) the keys $K_{\langle 2,4\rangle}$, $K_{\langle 1,2\rangle}$, and $K_{\langle 0,1\rangle}$ need to be newly established. I defer the discussion of how the new member finds a position in the tree to Section 4.

Similarly, since the leave procedure needs to provide perfect forward secrecy, all the keys that the leaving member knows need to be changed, which prevents him or her from knowing any of the new keys. The box labelled Procedure 6 contains the details of the leave protocol. When the member at node $\langle l, i\rangle$ leaves the group, the node $\langle l - 1, \lceil \frac{i}{2} \rceil\rangle$ gets replaced by $\langle l, i \pm 1\rangle$ (by moving the entire subtree one level up). All keys from the deleted node up to the root need to be updated: $K_{\langle l', \lceil \frac{i}{2^{l'}} \rceil\rangle} (0 \leq l' \leq l - 1)$.

For example, Figure 3(b) shows the key tree right before M_5 leaves, then the node $\langle 1, 2\rangle$ is deleted and $\langle 2, 3\rangle$ moves one level up, as shown in Figure 3(a).



(a) Before Join (b) After Join

Figure 3: (a) Key tree before join operation (b) and after joining the new member M_5.

Initialization: There are N current group members. The new user is inserted at node $\langle l, i\rangle$ in the key tree, which is at the leaf closest to the root. Then there was a member at $\langle l - 1, \lceil \frac{i}{2} \rceil\rangle$ which gets "pushed" to $\langle l, i - 1\rangle$. All keys from the leaf key up to the root need to be updated:
$K_{\langle l', \lceil \frac{i}{2^{l'}} \rceil\rangle} (0 \leq l' \leq l)$. The new user M_i selects a random number $r_i \in \mathbb{Z}_p^*$ and sets the key of level l to $K_{\langle l,i\rangle} = r_i$.

Iterate over j: $(1 \leq j \leq l)$
In each step the key $K_{\langle l',v\rangle}$ is constructed, where $l' = d - j$ and $v = \lceil \frac{i}{2^j} \rceil$.
Procedure 1 is run with j, v, l' as arguments, to establish key $K_{\langle l',v\rangle}$.

Procedure 5: "Join" procedure in NAGKA

Initialization: The leaving member is at node $\langle l, i\rangle$ in the key tree. All keys $K_{\langle l', \lceil \frac{i}{2^{l'}} \rceil\rangle} (0 \leq l' \leq l - 1)$ need to be updated. The peer subtree is moved one level up.

Iterate over j: $(1 \leq j \leq l - 1)$
In each step the key $K_{\langle l',v\rangle}$ is constructed, where $l' = d - j$ and $v = \lceil \frac{i}{2^j} \rceil$.
Protocol 1 is run with j, v, l' as arguments, which establishes key $K_{\langle l',v\rangle}$.

Procedure 6: "Leave" procedure in NAGKA

3.8 Additional Security Services

The key tree's main role in the above description is to provide forward and backward secrecy. I propose to use the key tree for additional innovative security and policy features. A new type of service in the domain of rights management is sender authorization, where only specific members are allowed to send group messages, and the remaining members are allowed only to receive. This property is enforced by placing all senders on the left hand side of the binary key tree and verifying their sender permissions when constructing the key tree. In AGKA, the left-hand members know the key $K_{\langle 1,1 \rangle}$, whereas the right-hand members know only $\alpha^{K_{\langle 1,1 \rangle}}$. To expoit this asymmetry to sign every message sent, I use the El-Gamal signature scheme [14, 9]. All the receivers verify the sender authorization upon receipt of the message. The same idea extends to produce a group signature. The key tree structure also allows construction of efficient time-bounded memberships. By placing in the same subtree all members who need to leave at a certain point in time, all of them can leave in one step by cutting off the entire subtree in the key tree. These primitives, together with the tree structure, express and enforce powerful rights management policies. These methods are not restricted to the protocols described in this paper, but are applicable to any key-tree scheme, in particular to previously proposed server-based key tree schemes [16, 17].

4 Implementation Details

In the key agreement protocols we have seen so far, I stated two assumptions: all members know their position and the other member's position in the key tree, and second, the number of members is $N = 2^d$.

To realize the first assumption, we need an algorithm to place new members deterministically in the key tree, before running the key agreement protocol. We achieve this property through a multi-phase protocol, where in an initial phase all members who wish to join announce themselves, then in a second phase all members agree on who will be in the group, and in a final phase all members build the same tree by inserting all members in lexicographic order, for example. These issues can be easily resolved by using a group controller, but which would introduce other problems, which I mentioned in the introduction.

To relax the second assumption, the protocols need to work for the case $N \neq 2^d$. As we have seen in the examples of the "join" and "leave" protocols, the only necessary requirement for the protocols is that the tree is binary, meaning that every node either has two children or is a leaf.

In the description of the key agreement procedures (boxes labeled Procedure 1, 2, and 4), I mentioned that the members of the key agreement are chosen randomly out of the subtrees, rooted at the node of the key which is to be established. To choose the two members, I use the following method. Choosing a member at random is not necessary; every member sends the key agreement message with a random delay, since every member in the subtree can perform the key establishment. When a member M_i in a subtree receives the key agreement message sent by another member $M_{i'}$ in the same subtree, M_i does not need to perform the key agreement in the current round. This results in favorable load-balancing properties, because the members having faster computers and better network connectivity will be able to react quicker to a key agreement in a level and therefore do more work. Thus these protocols are ideally applicable in settings with large differences in speed and network connectivity among members.

A similar method allows an efficient "join" mechanism, where the problem is how the joining member finds his or her position in the key tree. This problem is solved by using a multi-phase protocol: during the initial join phase, the new member sends his or her membership request; in a second "mating" phase, group members send "offer" messages, offering the new member to join the group at their node. Since the key tree needs to stay balanced, only the members closest to the root reply (since every member has a complete map of the tree, this is a simple operation). The new member might get multiple offers, which allows him or her to choose the joining position. If multiple members wish to join, they can generate their own key tree, which gets grafted into the main key tree. This process is extremely efficient and allows a large number of members to join concurrently.

Another implementation issue is the transition from Diffie-Hellman keys (which result from the key agreement) to symmetric keys. Because of the mismatch in the number of bits between the length of the Diffie-Hellman key $K = \alpha^x \mod p$ and the secret key size required by the symmetric algorithm, we can use a hash function on K to deliver the symmetric key bits. A cryptographic hash function, such as SHA-1 [14, 9] would work and yields 160 bits, enough to initialize a shorter symmetric key. Also, I do not specify a particular symmetric algorithm, since any algorithm that satisfies the security requirements will work. Similarly, the choice of a signature scheme is open.

5 Security Analysis

The key advantage of the authenticated protocols AGKA and AGKA-G, is that they achieve mutual authentication without complete member-to-member authentication (involving $\frac{n(n-1)}{2}$ messages), and without a dedicated member checking all the other members. The property that all members are authenticated if the protocol finishes successfully, is proven informally in the appendix. To understand the proof intuitively, we consider the two boundary cases: First, only one member is corrupt; and second, all but one member are corrupt. In the first case, the corrupt member will be involved in an authenticated key agreement with a good member at least once, in which case the corrupted member is discovered. In the second case, the valid member will be involved in an authenticated key agreement at least once, detecting the corrupt user. The proof shows that for each subtree of the key tree that was generated successfully, either all members in that subtree are good or all members are corrupt.

In case an authentication fails, the certified member broadcasts a signed report pointing out the intruder. The subtree containing the intruder is then cut off the key tree and all the users below it are expelled. This may appear draconian, but the other members in that subtree must also be intruders, otherwise the corrupt member would have been discovered at an earlier level.

Another important property of the protocol is that any member only knows the keys in the key tree which lie on the path from its node up to the root, and no others. This property is important to ensure forward secrecy in the case of members leaving the group. Since the key tree is constructed bottom-up and by the properties of the key agreement protocols, any member gets only the keys on the path from its leaf to the root.

In the appendix, I sketch a proof that the authenticated key agreement protocols provide implicit key authentication against passive attackers. A frequent active attack in the context of Diffie-Hellman key agreements is the man-in-the-middle attack. I have mentioned earlier that the unauthenticated NAGKA protocol is vulnerable against this type of attack. Due to the mutual authentication of members in the authenticated protocols, this problem is not an issue for AGKA and AGKA-G.

Due to the intractability of the Diffie-Hellman problem, my protocols provide key independence. In addition all keys that a joining member receives were not previously used, and similarly, all keys that a leaving member knows are changed and do not influence the subsequent group operation. For these reasons, the protocols provide perfect forward and backward secrecy. Therefore, collusion does not compromise the security of the group.

As can be seen from inspecting the final key (as shown in Figure 1), we can see that the key agreement protocols are contributory and provide key integrity.

6 Complexity analysis

I analyze the complexity of my protocols based on communication and computation overhead. The metrics for communication overhead are the number of messages transmitted and their size. To characterize computation overhead I measure the number of exponentiations, inverses, multiplications, hashes, de/encryptions, signature generations and verifications of a group key agreement of N members.

I compare the protocols to the state of the art authenticated group key agreement protocols A-GDH.2 and SA-GDH.2 described by Ateniese et al. [1].

I observe the case of a group key agreement among N members, where the key tree of depth d is balanced, therefore $N = 2^d$. In this analysis, I neglect the phase where the new members negotiate their positions in the key tree, since the protocol we are comparing against also report only the time of the key agreement. Similarly to Ateniese et al. 's analysis [1], the certificate distribution overhead is not taken into consideration. This gives an advantage of AGKA over AGKA-G, since the latter sends the information necessary to construct the identity-based certificate.

Table 1 shows the results of the comparison. The protocols presented in this paper have the property that if $N = 2^d$, there are $N - 1$ mutual key agreements in total. For each "min" entry, we assume that the member was involved in only one mutual key agreement (level $d - 1$), and for each "max" entry we assume that that member was involved in d key agreements. The first part of the table describes the communication costs involved. If the network is the bottleneck, these parameters are important. With $|K|$ we refer to the length of a key. The second part of the table describes the computational overhead. Signature computations are by far the most expensive operations, followed by exponentiations and decryptions. We can see that the AGKA-G really has by far the lowest overheads by not needing to compute or verify signatures, through the use of implicitly-certified public keys. The drawback is that more messages need to be exchanged, four times as many as in AGKA. Clearly, the protocols described improve by an order of magnitude the current state of the art protocols for contributory group key agreement for dynamic groups. One of the drawbacks, however, is the large number of broadcasts used. These broadcast messages are used to distribute the new keys to the other members of the same subtree. If the number of broadcasts turns out to be a problem, it would be possible to replace the randomized matching of participating members in a key agreement by a deterministic algorithm. This modification would allow the two members to exchange the key agreement messages through unicast, followed by a deterministic distribution of the new key by unicast using the structure of the tree.

7 Conclusions and Future Work

The paper presents three novel key agreement protocols for group key communication. The protocols satisfy many security properties, including perfect forward and backward secrecy. All of the protocols require only $\lceil \log(N) \rceil$ rounds for the group key agreement involving N members. The bound is achieved by constructing a binary key tree and letting individual users establish new keys in parallel. This is a vast improvement over the

	NAGKA	AGKA	AGKA-G	A-GDH.2	SA-GDH.2
Communication costs					
rounds	$d = \lceil \log(N) \rceil$	d	d	N	N
broadcasts	$2(N-1)$	$2(N-1)$	$2(N-1)$	1	1
total msgs	$2(N-1)$	$2(N-1)$	$8(N-1)$	N	N
total bandwidth	$2(N-1)\|K\|$	$2(N-1)\|K\|$	$8(N-1)\|K\|$	$\frac{N^2+N}{2} - 1$	N^2
msgs sent per M_i (max)	d	d	$4d$	1	1
msgs sent per M_i (min)	1	1	1	1	1
msgs recvd per M_i (max)	d	d	$4d$	2	2
msgs recvd per M_i (min)	d	d	$d+3$	2	2
Computation costs					
exp per M_i (min,max)	$d+1, 2d$	$d+1, 2d$	$5, 5d$	$1, N$	N, N
exp total	$(d+2)N$	$(d+2)N$	$10(N-1)$	$\frac{N^2+3N}{2} - 1$	N^2
inverse per M_i (min,max)	-	-	-	-	$1, 1$
inverse total	-	-	-	-	N
mult/div per M_i (min,max)	-	-	$2, 2d$	$1, N-1$	$2N-2, 2N-2$
mult total	-	-	$4(N-1)$	$2N-2$	$2N^2 - 2N$
hash per M_i (min,max)	-	-	$1, d$	-	-
en/decryptions (min,max)	-	-	$d+2, 3d$	-	-
signature verif	-	d	-	-	-
signature gen (min,max)	-	$1, d$	-	-	-

Table 1: Complexities of key agreement protocols

linear number of rounds of previously proposed protocols for dynamic groups. Also, the protocols reduce the bandwidth consumption from $O(N^2)$ to $O(N)$. In addition, I have improved on today's known group key communication protocols by removing the necessity of a central server or group key controller thus enhancing the applicability to new domains. Another contribution is that we can exploit the structure of the key tree to design rights management policies; other key tree based group communication protocols only use the key tree to achieve forward and backward secrecy. In addition, I also describe implementation issues, and in particular show how to optimize the key agreement protocol to achieve load balancing among the members.

I am currently building, and will distribute a Java library that implements the protocols described. These libraries will simplify the implementation of applications which require efficient and secure group communication.

The future work on the protocols is to reduce the number of broadcasts and to explore how to reduce the need for reliable multicast.

8 Acknowledgements

I would like to thank particularly to Dawn Song and Yang-hua Chu for the numerous discussions about secure multicast. Many of the ideas presented were spawned during our meetings. Furthermore, I am grateful to Doug Tygar for his continued help and support. Finally, I would like to thank the anonymous reviewers for their feedback.

References

[1] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated Group Key Agreement and Friends. In *5th ACM Conference on Computer and Communications Security*, pages 17–26. ACM, November 1998.

[2] Mike Burmester and Yvo Desmedt. A Secure and Efficient Conference Key Distribution System. In Alfredo De Santis, editor, *EUROCRYPT94*, pages 275–286. LNCS 950, 1994.

[3] Isabella Chang, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques. INFOCOM 1999, September 1998.

[4] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[5] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of the ACM SIGCOMM 95*, pages 342–356, Boston, MA, August 1995.

[6] Christoph G. Günther. An identity-based Key-Exchange Protocol. In *EUROCRYPT89*, 1989.

[7] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Specification / Architec-

ture. Technical Report RFC-2093 and RFC-2094, IETF, July 1997.

[8] David A. McGrew and Alan T. Sherman. Key Establishment in Large Dynamic Groups Using One-Way Function Trees, May 1998. Published at http://www.cs.umbc.edu/~sherman/Papers/itse.ps.

[9] Alfred J. Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[10] Suvo Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *ACM SIGCOMM*, September 1997.

[11] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A Fault-Tolerant Multicast Group Communication System. *CACM*, 39(4):54–63, April 1996.

[12] David Powell. Group Communication. *CACM*, 39(4):50–53, April 1996.

[13] Michael K. Reiter. Distributing Trust with the Rampart Toolkit. *CACM*, 39(4):71–74, April 1996.

[14] Bruce Schneier. *Applied Cryptography (Second Edition).* John Wiley & Sons, 1996.

[15] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: A Flexible Group Communication System. *CACM*, 39(4):76–83, April 1996.

[16] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key Management for Multicast: Issues and Architectures. Technical report, IETF, September 1998. draft-wallner-key-arch-01.txt.

[17] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure Group Communications Using Key Graphs. In *ACM Sigcomm 98*, 1998. University of Texas at Austin.

[18] X. Rex Xu, Andrew C. Myers, Hui Zhang, and Raj Yavatkar. Resilient Multicast Support for Continuous-Media Applications. In *NOSSDAV*, 1997.

A Proofs of chosen properties

Theorem 1: Weak implicit key authentication
A passive adversary can gain no knowledge about any of the keys in the key hierarchy.

Proof sketch. We prove this property by a reduction to the Diffie-Hellman problem and the discrete-logarithm problem. We show that if the passive adversary finds one key of the key hierarchy, then he either can solve an instance of the Diffie-Hellman or the discrete-logarithm problem.

The proof works by induction on the depth of the key tree. We assume that the attacker receives all messages.
Base case. The attacker cannot figure out any of the keys $K_{\langle d,k \rangle}$ in the lowest level d since each member choses a random number r_i and no messages were sent yet at this stage of the protocol.
Induction. Knowing that the adversary does not know any key in a level between l' and d, we show that he cannot figure out any key established in level l'. For this we analyze the key agreement of key $K_{\langle l',k \rangle}$. $M_{i'}$ and $M_{i''}$ exchange $\alpha^{K_{\langle l'+1,k' \rangle}}$ and $\alpha^{K_{\langle l'+1,k'' \rangle}}$ in public. Because of the intractability of computing the discrete-logarithm, neither $K_{\langle l'+1,k' \rangle}$ nor $K_{\langle l'+1,k'' \rangle}$ is compromised. The attacker can also not compute the new key $K_{\langle l',k \rangle} = \alpha^{K_{\langle l'+1,k' \rangle} \cdot K_{\langle l'+1,k'' \rangle}}$ due to the intractability of the Diffie-Hellman problem. Similar as in the previous case, the symmetric encryption of the new key with the lower-level key does not compromise the security either.

Therefore, the adversary cannot compute any of the keys of the tree. □

In order to show that the authenticated protocols are secure against an active attacker, the proof becomes more complicated. The first step would be to build a model of the capabilities of the adversary, which suffers from concerns for completeness. Then we need to show that any group of active attackers would not be able to break the properties of the protocol. Together with Dawn Song, we are currently in the process of applying model checking techniques to Strand spaces to verify the correctness of the protocol. For this paper we contend with the model of only a passive attacker.

Theorem 2: All members are authenticated
If the protocol finishes successfully, then all members are either valid or they all are intruders. If there is at least one valid member joining the group, then the protocol only progresses if all other members are also certified.

Proof sketch. We assume that the protocol finished successfully. Therefore all members have broadcasted their final message successfully, containing their membership number encrypted with the root key. Because all members have submitted their validation of the group, all authentications between a valid member and another member must have been successful. This implies that if one of the members of a mutual authentications was certified, then the other one was as well. Therefore all mutual authentications must have been either between two valid members or two invalid members. From this we can imply that all members in the group are either valid or all are intruders. □