Efficient Security for Large and Dynamic Multicast Groups

Germano Caronni†, Marcel Waldvogel‡, Dan Sun‡, Bernhard Plattner‡

†Sun Microsystems Inc., Internet Commerce and Security, Palo Alto, USA, gec@acm.org*

‡Computer Engineering and Networks Laboratory (TIK), ETH Zürich, Switzerland, {waldvogel,sun,plattner}@tik.ee.ethz.ch

Abstract

Proposals for multicast security that have been published so far are complex, often require trust in network components or are inefficient. In this paper we propose a series of novel approaches for achieving scalable security in IP multicast, providing group-wide privacy and authentication. They can be employed to efficiently secure multi-party applications where members of highly dynamic groups of arbitrary size may participate.

Supporting dynamic groups implies that newly joining members must not be able to understand past group communications, and that leaving members may not follow future communications. Key changes are required for all group members when a leave or join occurs, which poses a problem if groups are large. The algorithms presented here require no trust in third parties, support either centralized or fully distributed management of keying material, and have low complexity ($O(\log N)$ or less). This grants scalability even for large groups.

In this paper we discuss the requirements for secure multicasting, present our approaches, and evaluate their properties, based on an experimental implementation.

1 Introduction

With IP multicasting being offered in the Internet, multi-party applications are fast becoming an important class of distributed applications, as is demonstrated with the popularity of the experimental Mbone multicast service and the applications it supports. Today, the most important class of applications using a multicast transport service are collaborative multimedia applications, such as vic or vat [MB94]. However, it is apparent that many distibuted applications may be implemented in an efficient way by taking advantage of multicast services, e.g. fault tolerant distributed systems relying on redundant storage of data, distributed databases and distributed file systems, or massively parallel distributed applications where the actual identity of participants is not known to all involved parties. In all cases, redundant distribution of information could be done efficiently with multicasting, or by using anycasts. Other examples are airtraffic control systems, in which groups of airplanes associated to one or a set of control centers would exchange positional information among each other and with the control centers. Another class of applications needing multicast services are those whose primary task is to distribute information to a set of receivers; stock data distribution and audio or video distribution services clearly belong to this class, as could Usenet news postings.

Besides using a multicast service for an efficient implementation, in all these applications there is a need to exchange information among the members of small (in the case

[0] Work started at ETH

of a simple multi-party desktop conference), medium (e.g. in a distance-education scenario) or very large groups (audio distribution). In addition, the groups involved often are highly dynamic: Members may join or leave the group frequently and at any time, and — depending on the application — knowledge about actual group membership may be unnecessary. IP multicast was designed to cope with the requirements outlined above; however, this statement only applies directly to the basic IP best-effort multicast service. It is well known that scalability is not easily achieved as soon as reliability is added to the set of requirements. In fact, the current thinking is that reliable multicast should be dealt with in a specific application context [McC92], rather than in a universal way.

It is the purpose of this paper to investigate how secure multicasting can be provided as a universal service, preserving the properties of scalability and flexibility as offered by the basic IP multicast service. We maintain and will demonstrate that such solutions exist; our techniques, however, are not only applicable to IP multicast — they may also be used e.g. with connection-oriented multicast services as found in ATM [ATM95].

Like many unicast applications, most of the multi-party applications listed above will only be successful if privacy and authenticity of participants can be provided effiently. To this end, cryptographic mechanisms are deployed. Consider, for example, a stock data distribution service, which distributes its information to a large number of customers around the globe. It is obvious that only those people who have subscribed to the service should be able to receive this information. If a new customer subscribes, he should be able to receive stock data immediately, but not to understand information which was released before the time of his subscription. Conversely, a customer canceling his subscription should not be able to process information beyond the time of cancellation. By consequence, the purpose of this paper will be to discuss key management schemes which guarantee that at each instance in time only actual group members will be in possession of the cryptographic keys needed to participate. A naive solution would be to create a new session key when a new member joins the group, and to securely distribute the key to all members of the group, using unicast security mechanisms. However, such a solution would not scale, as it requires that the new session key be encrypted individually for each participant.

Even though multicast routing itself implements a kind of closed user group, the property of closedness is rather weak: Multicast routing protocols known to date are designed to distribute multicast datagrams to a set of links hosting group members, i.e. to grant, and not to prevent access to information. This is most prominent with routing protocols based on flooding algorithms, such as DVMRP [DPW88], and generally with approaches using reverse path broadcasting/multicasting [DC90], which distribute multicast datagrams quite generously to a set of potential recipients being much larger than the actual set of group members. A similar situation exists in satellite-based communication: While satellites have the advantage of potentially being able to reach a large number of recipients using broadcast communication, they do not offer any intrinsic technique which would permit to narrow the set of recipients to those that belong to a specific group. Cryptographic mechanisms to restrict the real flow of information will therefore be of primary importance if tightly controlled closed user groups are to be created.

In this paper we propose a series of novel approaches for achieving efficient security in IP multicast, enabling secure multi-party applications in which members of highly dynamic groups of arbitrary size may participate. Our approaches allow all group members to establish a mutually shared secret, which can be used to provide group-wide privacy, message authenticity or any other property relying on a shared secret. Transition from one key management approach to another in a running system is possible. All approaches can offer perfect forward secrecy [Dif90], require only a small amount of calculations and storage from the participants, and avoid investing trust into third party components such as e.g. routers. Depending on the choosen approach, after a setup phase, unidirectional communication is sufficient to manage group membership, and no inter-participant communication may be required.

For optimal understanding, knowledge about cryptography basics is advantageous

([Sch96, Sim92]).

The remainder of the paper is organized as follows: Section 2 presents related work, Section 3 will discuss the schemes and their relation, Section 4 evaluates the results, shows preliminary measurements and discusses impacts of security attacks. Section 5 concludes the paper and explores further work.

2 Related Work

Although a number of cryptographic techniques have been proposed to secure group communications in broadcast or multicast scenarios, very few of them are targeted to a large group setting with highly-dynamic membership without 3rd party trust or are complex and inefficient in dealing with this issue.

According to the primary algorithms used or objectives achieved, five approaches are identified and are compared below in terms of their security, efficiency and capability to deal with the key management in large groups with frequent membership changes.

2.1 Key Predistribution Scheme

As described by Matsumito in [MI87], the name "Key Predistribution Scheme" comes from the fact that the secret information related to all possible groups are distributed to involved members before operation. In [MI87], a linear scheme was presented as a realization method. There, predistribution takes place with the aid of any two-party key distribution primitive. Afterwards a member can read out the common key for a specified group from its key list according to the identities of all participating members. A group membership change means switching to a different group — involving a look up on the predistributed key list for a different common key. This scheme is secure as long as the group key distribution centers are trusted. Storage size required for each user grows exponentially with the number of group participants.

A variation of this scheme was presented by Berkovits [Ber91]. The idea is based on "k out of n" secret sharing schemes, in which each participant gets a share in the secret, then any k of the n participants can pool their shares and reconstruct the secret. An example using polynomial interpolation was presented as well as a related vector formulation. As an extension of [Ber91], both interactive and non-interactive models were defined in [BSH$^+$93].

2.2 Secure Lock

The implementation of Secure lock is based on the Chinese Remainder Theorem. Here, the group session key is secured such that only the keys of authorized users, as described in [CC89], can retrieve the session key. Using this secure lock, only one copy of the ciphertext is sent, and the number of secret keys held by each user is minimized (1 for public-key based, and $O(n^2)$ for private-key based approaches).

The weakness of this scheme is its need to associate one large number (relatively prime to all other group members' numbers) for each participant, and that retrieving the group session key is an expensive operation. These conditions confine this protocol to being used only within small groups.

To prevent the replay or masquerading attacks in the original approach in [CC89], an extra timestamp and a checksum are suggested to be encrypted in the Secure Lock by L. Gong in [GS95].

2.3 Spanning Tree Distribution

To securely spread the conference key from the conference chair to all group members in a spanning tree, [BD96] proposed the use of a secure-path graph. To construct this secure graph, adjacent nodes in the spanning tree first establish a shared secret using any two-party key distribution primitive. Then, starting from the chair, each entity sends to its adjacent nodes (along the spanning tree) information which links the conference key with the exchanged key. Using this information, the entities can compute the conference key. Whenever the membership changes, the spanning tree needs to be extended or pruned to make sure that only the group members can get the updated conference key. The honesty of insiders plays a key role in this protocol, as they could easily prevent whole sub-trees from taking part in the group, by denying them correct keying material.

The propagation of information in the tree can also be done in parallel by using broadcasts. Comapred to the sequential case, this decreases key change delays and required intermediate computations, but is at the cost of bandwidth. The usage of spanning trees avoids the traffic implosion in the network.

2.4 Extended Diffie-Hellman Key Exchange

A natural extension to Diffie-Hellman [DH76] was presented by Ingemarsson et al. in [ITW82]. Here, the members are arranged in a logical ring, all members joining at the same time, and having to participate in $n-1$ rounds (where n is the number of group members). In a given round, every participant raises the previously-received intermediate key value to the power of its own exponent and forwards the result to the next participant. After $n-1$ rounds everyone holds the same key. This protocol involves high latency, and is only suitable for static key distribution.

A more efficient protocol has been proposed by Burmester and Desmedt [BD95], which uses broadcast messages and executes in only three rounds. In the first round, each participant selects its random exponent r_i, computes $z_i = \alpha^{r_i} \pmod p$, and broadcasts z_i. Secondly, each participant computes and broadcasts $X_i = (z_{i+1}/z_{i-1})^{r_i} \pmod p$. In the last round, each participant can compute the conference key $K_i = (z_{i-1})^{nr_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdot \ldots \cdot X_{i-2})) \pmod p$. This key is identical for all participants. Although this protocol is roughly as fast as RSA and as secure as the Diffie-Hellman problem, it is difficult to deploy in a dynamic group. All members have to keep transient states for possible changes in the group membership, otherwise each join or leave has to be considered as a new group and all three rounds need to be redone. Again, the cooperation of all participants, involving n reliable broadcast messages, is required.

A more recent work was described by Steiner et al. [STW97] and presented the capability to distribute session keys in dynamic groups. While this protocol provides a way to distribute a session key in highly dynamic groups, the solution does not scale well to large groups, where the group manager has to perform $O(n)$ exponentiations for each group change, and messages get prohibitively large.

2.5 Scalable Multicast Key Distribution

Ballardie [Bal96], presents a scalable multicast key distribution scheme, which is based on a Core Based Tree [Bal97] multicasting architecture. Here, a group key distribution center (GKDC) is controlling access to the group. Later on, some functionality of the GKDC is passed on to the other routers which are on the path from joining members to the GKDC. Each joining router is provided with a group access packet containing an access control list and group keying materials (K_{GRP}). The scheme requires absolute trust in router components, and no group key changes can be done, except by establishing a new group.

More recently, Mittra [Mit97] proposed a solution to deal with the scalability issues in highly dynamic large groups. A secure distribution tree, composed of a number of smaller secure multicast "subgroups" is used to enable secure multicasting. The subgroups are arranged in a hierarchy to create a single virtual secure multicast group, and use independant keying material. Intermediate agents are members of two subgroups, and perform a re-encryption operation for messages passing the borders. The necessity to change keying material upon a change in membership is limited to a specific subgroup. Upon a join, the new subgroup key K'_{GRP} encrypted with the original K_{GRP} is multicast to the current subgroup and unicast to the joining member via the separate secure channel; for a leave operation, a message containing n copies of K'_{SGRP} (assuming n remaining members in the subgroup) is multicast, where each copy of K'_{SGRP} is encrypted with the member's secret key. The solution requires full trust in the subgroup agents. In terms of the message size transmitted and the key deciphering involved for a join or leave, this protocol is complicated.

2.6 Group Key Management Protocol

Harney et al. described the Group Key Management Protocol (GKMP) [HM97b, HM97a], in which a group controller is assigned to a group. The group controller processes all join requests individually, and forwards the single group key to the joining members. This is basically a unicast key distribution protocol and provides no solution for changing the group key when membership changes. Perfect forward secrecy is not addressed.

Existing protocols for secure multicasting are limited to distribute session keys in static and/ or small groups. For dealing with the group key distribution in a large group with frequent membership changes, some good explorations have been done in [Mit97, STW97]. Issues to be improved are scalability, reduction of computational complexity and reduction of trust in dedicated nodes (e.g. network components), and the necessity for group members to interoperate for the generation of a group-wide secret. We will propose a new set of protocols, demonstrating the ability to successfully handle these issues in large and highly dynamic groups.

3 Secure Multicasting

In the solutions presented here, changes to the group's membership are possible with minimal involvement of dedicated nodes and group members. The approaches cope with several properties inherent to multicast and broadcast environments: There is an unreliable (and in the case of IP also unordered) transmission channel, and the transmissions may be one-way, with no or only a minimal return channel, to reflect the nature of broadcast environments – likely users of secure multicasting. Last but certainly not least, it is important that as little trust as possible should be necessary towards third party entities such as routers or other intermediate systems. While those third party components may be trusted to distribute a session directory, certified public key material, or access control information signed by a group memeber, they should never be able to gain access to actual keying material and payload.

As seen earlier, it is important to have a system which — even with large groups and frequent joins or leaves — neither is susceptible to implosion nor enables users to understand what was transmitted at times they were not part of the group, either before they joined or after they left or were expulsed. Additionally, any third party recording ongoing transmission and later capturing the secrets held by a participant must not be able to understand its recordings. This is known as "perfect forward secrecy" [Dif90]. To completely achieve this, also the unicast connections need to be setup using ephemeral secrets.

This section is organized as follows: First, the general architecture is discussed, followed by the detailed descriptions of the three key management approaches (Tree-based, Centralized Flat, and Distributed Flat), explaining the properties they make available to large, dynamic groups. The presented schemes cover a wide range of applications and security needs: From very tight control in the centralized approach to extreme tolerance to system and network failures in the completely distributed scheme. A selection of advanced topics will conclude the discussion.

3.1 Architecture

First, the common components are identified and explained, then their interactions during all the operations are shown.

3.1.1 Components

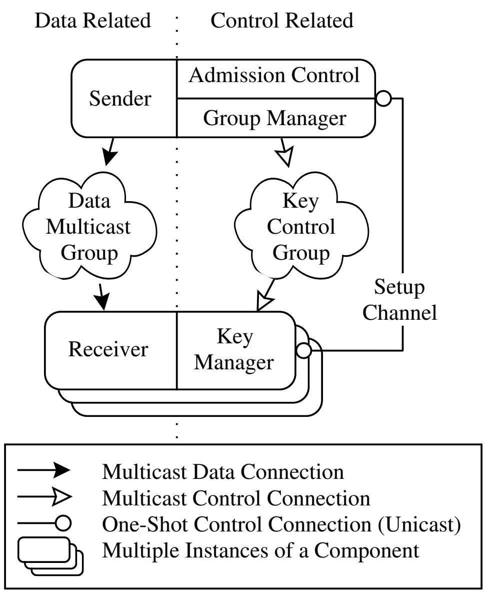


Figure 1: Secure multicasting components in a single sender, multiple recipients scenario

Figure 1 shows the basic architecture for the simplest scenario, forming the basis of the descriptions: A single sender and any number of participants (multiple senders and group collaboration will be explained below). Fundamental and common functions are explained here, while individual extensions and modifications will be pursued later. Generally, the components can be separated into two groups: (1) a group of data related components, covering components very similar to those of current insecure multicast or broadcast communication architecture. It consists of the sender, recipients, and one or more Data Multicast Groups. (2) a group of control (or key management) related components, which includes all components involved in the key agreement and key exchange process.

Sender The application prepares data as it would for non-secure transmission, then encrypts (and possibly authenticates) the packets using the current Traffic Encryption Key (TEK), received from the Group Manager.

Recipient Receives the data from the Data Multicast Group and decrypts it according to the TEK given by the local Key Manager. Later steps in the application data processing will not notice any differences resulting from the encryption or authentication of data.

Data Multicast Group Any multicast, broadcast, or anycast channel delivering the secured packets from the sender(s) at least to the intended receivers. It will be used to transport the bulk of the application's data.

Group Manager Receives, admits, and processes join and leave requests from participants and sends out the messages to have Key Managers perform the necessary key changes.

Admission Control Is queried by the Group Manager to find out who is to be admitted. This function can also be delegated to a human, e.g. a chairperson.

Key Manager Receives and decodes the rekeying requests from the Group Manager, passing the resulting TEK to the Receiver.

Setup Channel Join requests from new members are usually received through this unicast connection, or via another out-of-band mechanism. This channel is only needed to bootstrap a join request and to perform authentication between the new participant and the Group Manager. Although this looks like a source for implosion problems, the distributed approach (presented below) is not prone to them and the problem can be mitigated in the centralized key management schemes, as will be seen in Section 3.7.

Key Control Group Any multicast, broadcast, or anycast channel delivering the packets from the Group Manager to at least the intended receivers. Traffic consists of new keying material which needs to be distributed to the participants Key Managers. Transmissions over this channel have to be received by every participant, which can be achieved by (1) implementing components of any reliable multicast mechanism (such as those discussed [FJM+95, PSB+95, PTK94]), as was done in our experimental realisation of the system, or (2) performing retransmits on a regular basis with a limited history of key changes, resulting in a soft state approach. The latter approach is desirable for scenarios without return channel and especially feasible if the loss rate is known (e.g. through bandwidth reservation [BCS93]) or through known good or reliable transmission channels.

If for any reason a receiver should be unable to receive a packet in reasonable time, the fallback solution is to contact the Group Manager again. This can also be done using an out-of-band channel when there is no return channel.

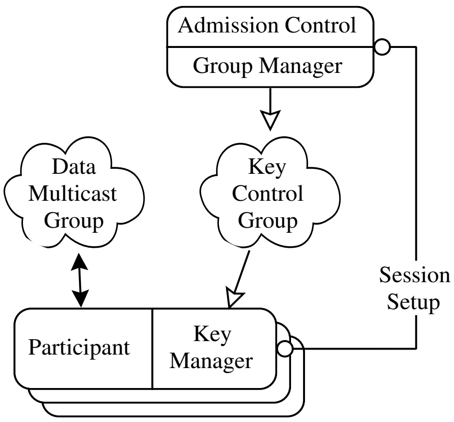


Figure 2: Group collaboration scenario

Often, there is more than one sender, and senders and receivers cannot be distinguished. Also, any receiver is free to send data encrypted or authenticated using the current TEK, and in a group collaboration environment every member of the group holds both roles at the same time, resulting in a situation as shown in Figure 2. This is a transformation of Figure 1 where sender and recipient were integrated, and the Group Manager has been isolated. All of the schemes also work in that scenario, the distributed key management scheme is even very well suited for it. If senders and receivers are treated equally, they will be referred to using the term “participant”.

3.1.2 Basic Operations on the Group

To transmit the Traffic Encryption Key (TEK) secretly, a number of Key Encryption Keys (KEKs) are used to encrypt the control traffic containing the TEK. To distinguish the keys, each key consists of a unique ID, a version, a revision, and the keying material proper. The usage of the version and revision fields is explained in the leave and join descriptions, respectively.

The abovementioned components and keys will be involved in different activities:

Group Creation The Group Manager is configured with group and access control information. Additionally, the group parameters are published using a directory service.

Single Join The new participant’s Key Manager sends its request to the Group Manager, which checks whether this participant is allowed to join. If yes, the Group Manager assigns a unique ID to him, and selects a series of KEKs which will be transmitted to the newcomer. The selection of KEKs will be discussed separately for each key management scheme.

The Group Manager now increases the *revision* of all keys (TEK and KEKs) to be transmitted to the participant by passing the keying material through a one-way function (e.g. a cryptographically secure hash), then sends the keys out to the new participant. It also informs the sender(s) to use the new TEK. The other participants will notice the revision change visible in ordinary data packets, and also pass their TEK through the one-way function. Since the function is not reversible, the newcomer has no way to determine the key used beforehand.

Single Leave There are three ways to leave a group:

Silent Leave A receiver just stops participating in the group without telling anyone. No action is needed.

Voluntary Leave A receiver announces that it’s leaving. Depending on the policy, its keying material can be made unusable through a leave message as described below, the leave message may be delayed until another leave has to be performed, or no action is done, allowing the receiver to continue listening, if it wishes so.

Forced Leave If the Admission Control feels a need to forcibly exclude a participant, a leave message is to be sent out. Also, participants may ask the Admission Control to exclude a member. It is up to the admission policy how to deal with such requests.

To exclude a member, all keys known to it need to be replaced with entirely new keying material. To make all remaining participants aware of this change, the key’s *version* number is increased.

The Group Manager sends out a message with new keying material which can be decrypted by all the remaining participants’ Key Managers, but not the member which just left. Additionally, it frees the slot previously utilized by the leaving participant, making it available for reuse. As soon as all participants throw away prior keying material, perfect forward secrecy for the past traffic is assured.

Multiple Join, Multiple Leave, Group Merge, Group Split These functions have a number of dependencies on the chosen scheme and will thus be detailed there.

Group Destruction The Group Manager notifies all remaining participants of the destruction, closes all network connections, destroys all keying material and frees all memory. As soon as all parties have thrown away their keying material, perfect forward secrecy covering all traffic against third party opponents is guaranteed.

3.2 Centralized, Tree-Based Key Management

Thightest control over the individual participants can be achieved by this centralized approach, which is thus suitable for applications with high security demands. It is very easy to implement and maintain, and poses very little load on the network and the receivers. All keying material is managed centrally by the Group Manager, where all joining participants have to register. To store the keying material, a binary tree is used. The participants are represented by leaves therein. For simplicity of the explanation assume that the tree is fully balanced. The example in Figure 3 depicts such a tree with a maximum of 16 group members, and a depth of 4.

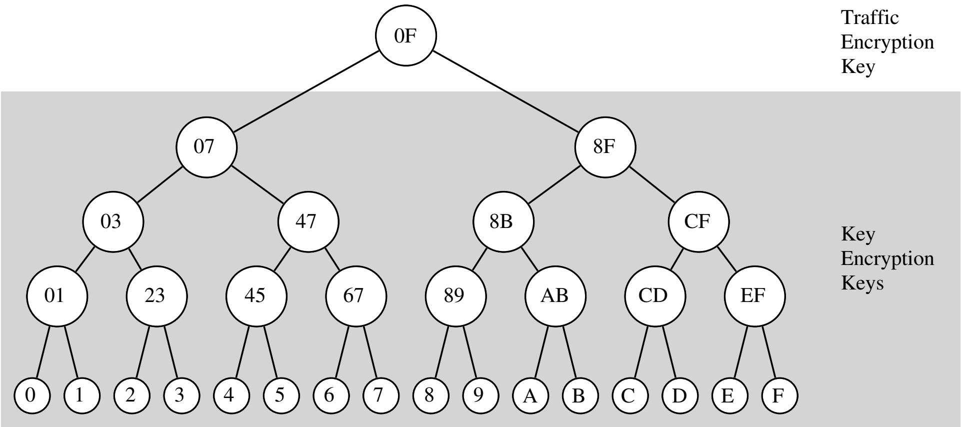


Figure 3: Binary hierarchy of keys. Labels in hexadecimal define the range of participants knowing this key

During a setup phase, which includes admission control, each participant establishes a shared secret with the Group Manager. This shared secret is known only by the Group Manager and the individual participant, and is used as the lowest level Key Encryption Key (KEK). The Group Manager stores it in the leaf node associated with this participant, and uses it whenever a truly private communication with this participant is required — such as during the join operation. Its revision is increased after each use to insure perfect forward secrecy. The nodes in the binary tree held by the Group Manager contain further KEKs, used to achieve efficient communication of new keying material when the membership of the group changes. These nodes do not represent actual systems or intermediate entities, but only hold keys for a hierarchy of virtual sub-groups of different sizes.

Each participant holds a different subset of keys from the tree, more specifically those keys that are in the path from the participants leaf to the root node, which is used as the Traffic Encryption Key (TEK). These intermediate Key Encryption Keys are used if a message should only be understood by a part of the group, e.g. a message encrypted with KEK 47 is understood by participants $4 \ldots 7$. This enables the transmission of new keys to only a limited set of Receivers, thereby disabling others to decrypt specific messages.

Each encrypted payload and key change message includes a reference to its key's version and revision number, such that key changes and out-of-order delivery can be implicitly detected by the Receivers. Version changes are always escorted by a separate message from the Group Manager, where the new key is provided in a secure manner. Revision changes can be resolved locally, for unknown versions the retransmission request function of any reliable multicast scheme [FJM+95, PSB+95, PTK94] can be used or the participant can wait passively for the next resend of the lost message.

Join On a join operation, the participant's Key Manager unicasts its request to the Group Manger, which checks with Admission Control and assigns an ID (say 4), where the participant's individual key is stored (usually the unicast session key already employed for the join request). The ID is used such that the bit-pattern of the ID defines the traversal of the tree, leading to a unique leaf. As an alternative to the explicit assignment of IDs, it is possible to use the IP address (or a function thereof) of participants as IDs. The Group Manager increases the revision of all the keys along the path from the new leaf to the root (Key Encryption Keys 45, 47, 07, and the Traffic Encryption Key $0F$), puts them through the one-way function and sends the new revision of the keys to the joining participant, together with their associated version and revision numbers. At the same time, all senders are informed of the revision change in a preferrably reliable manner, so they start using the new TEK. The receivers will know about this change when the first data packet indicating the use of the increased revision arrives. This creates less traffic and can make the revision change more reliable.

Leave To perform a leave operation, the Group Manager sends out a message with new keying material which can only be decrypted by all remaining participants' Key Managers. Additionally, it frees the slot utilized by the leaving participant, making it available for reuse at the next join.

Assume C is leaving. This means that the keys it knew (Key Encryption Keys CD, CF, $8F$, and the Traffic Encryption Key $0F$) need to be viewed as compromised and have to be changed in such a way that C cannot acquire the new keys. This is done efficiently by following the tree from the leaf node corresponding to the leaving participant to the TEK stored in the root node, and encrypting the new node keys with all appropriate underlying node or leaf keys. For our example, the tree in Figure 3 shows that the new Key Encryption Key CD_{new} (replacement for CD) needs to be received by D, CF_{new} by participants D, E and F, $8F_{new}$ by $8 \dots B, D \dots F$, and the new Traffic Encryption Key $0F_{new}$ by every participant except C. Instead of encrypting the new keys individually for each of the intended participants, we take advantage of the existing hierarchy:

- CD_{new} is encrypted for D, the only recipient in need of it.
- CF_{new} is sent twice, each copy encrypted with one of its two children keys, the existing EF and the new CD_{new}, so it can be decrypted by the intended recipients $D \dots F$.
- $8F_{new}$ is similarly encrypted for those knowing $8B$ or CF_{new}.
- $0F_{new}$ is finally encrypted for those holding key 07 or key $8F_{new}$.

This results in the following message being sent out:

$E_D(CD_{new})$	
$E_{EF}(CF_{new})$	$E_{CD_{new}}(CF_{new})$
$E_{8B}(8F_{new})$	$E_{CF_{new}}(8F_{new})$
$E_{07}(0F_{new})$	$E_{8F_{new}}(0F_{new})$

Along the path to the leaving node's leaf, all new keys except the bottom two rows will be encrypted for their two children. The new key in the leaver's parent node will be encrypted once. This results in $2W - 1$ keys being sent out, where W represents the depth of the hierarchy and also the length of the ID. Thus, even for a huge group with 4 billion participants ($W = 32$) and 128 bit keys, a single message of around 1200 bytes[1] multicast to everyone in the group establishes the new secrets. Processing this multicast message will require at most W decryption operations from the participants, with an average of less than 2 decryptions.

Multiple Leaves Intuitively, this can be extended to multiple leaves. The simplest and most obvious is the exclusion of a subtree, but it can be generalized to any arbitrary group of nodes. Using a single message for multiple leaves takes advantage of path overlaps, so several keys will only need to be created and sent out once per message instead of once per leave operation. This can be used to efficiently coalesce multiple leave (and join) operations into a single message.

Colluding participants can be reliably excluded by either sequential exclusions of them, or by grouping them together into a multiple leave operation.

Multiple Joins Similarly, if several joins happen in short succession, the revision of the TEK and the KEKs shared between the newcomers only need to be increased once, if newcomers can be allowed to decipher a small amount of data sent out before they were admitted (usually only a fraction of a second). If frequent joins are to be expected, the architecture may be changed such that the acutal senders are responsible for revision increases of the used TEK. They may increase the revision in regular, short intervals (such as half a second), thus creating a limited window for newcomers to read past traffic, but at the same time removing the need for the Group Manager to reliably keep in contact with the senders. If leaves and joins happen interleaved, they can both be grouped individually.

Group Merge To merge two independent groups, their two trees can be joined by adding a new root node, which becomes the new TEK for the joint group. The former TEKs become the KEKs for the second level. The new TEK is then sent out encrypted twice, once for each of the previous TEKs, together with the information that the tree has grown a level, resulting in a unified group. One has to keep in mind that the TEK is treated exactly like the KEKs when it comes to key changes, the only difference is that it is also used to encrypt traffic.

This insertion of an additional hierarchy level can also be used to grow a group, if the previously assigned ID space is exhausted because of the unexpected number of participants.

Group Split If the above group is to be split again into it's original subgroups, the top layer with the common TEK can be removed, resulting in two separate trees. Of course, it is also possible to split groups that have been intermingled, then each of the two new Group Managers (which can be the same machine) performs a Group Leave operation on the foreign members.

3.3 Centralized Flat Key Management

Instead of organizing the bits of the ID in a hierarchical, tree-based fashion and distributing the keys accordingly, they can also be assigned in a flat fashion (Figure 4). This has the advantage of greatly reducing database requirements, and obievates the sender from the need of keeping all participants in memory. It is now possible to exclude participants without knowing whether they were in the group in the first place.

[1] One Traffic Encryption Key with 32 bits each for key id, version, and revision encrypted for two groups, $W - 1$ Key Encryption Keys with 31 bit version and 1 bit revision encrypted for two sub-groups and one leaf Key Encryption Key, encrypted for a single node. One bit revision is enough for KEKs, since the higher revisions are always sent out in secure unicast connections.

TEK		
KEK 0.0	KEK 0.1	ID Bit #0
KEK 1.0	KEK 1.1	ID Bit #1
KEK 2.0	KEK 2.1	ID Bit #2
KEK 3.0	KEK 3.1	ID Bit #3
Bit's Value = 0	Bit's Value = 1	

Figure 4: Flat ID assignment

The data structure held by the Group Manager is a simple table, with $2W + 1$ entries. One entry holds the current TEK, the other $2W$ slots hold Key Encryption Keys. W represents the amount of bits in the participant ID, which normally will be equal to its network address. For each bit in the network address, two keys are available. Each participant knows W of those keys, depending on the value of the single bits in its address. All keys have associated version and revision numbers as in the tree scenario above.

The table contains $2W$ KEKs, two keys for each bit $b \in W$, corresponding to the two values $v \in \{0, 1\}$ that bit can take. The key associated with bit b having value v is referred to as K$b.v$ ("Bit Keys"). While the keys in the table could be used to generate a tree-like keying structure (e.g. by starting with the key associated with the highest-order address bit, and combining this with the key of the next level and so on, to create the shared secrets of ever diminuishing subtrees), they can also be used independently of each other.

The results are very similar to the Tree-Based Control from Section 3.2, but the key space is much smaller: For an ID length of W bits, only $2W + 1$ keys (including TEK) are needed, independent of the actual number of participants. The number of participants is limited to 2^W, so a value of 32 is considered a good choice. For IPv6 and calculated IDs, a value of 128 should be chosen to avoid collisions. This still keeps the number of keys and the size of change messages small. Besides reducing the storage and communication needed, this approach has the advantage that nobody needs to keep track of who is currently a member, yet the Group Manager is still able to expel an unwanted participant.

Join To join, a participant contacts the Group Manager, where it is assigned a unique ID and receives the keys corresponding to the ID's bit/value pairs, after previous revision increment. The ID may also be derived form the network address. As an example, a newcomer with (binary) ID 0010 would receive the TEK and the Key Encryption Keys K3.0, K2.0, K1.1, and K0.0 over the secure setup channel, after their revision was increased.

Leave All keys known to the leaving participant (the TEK and W KEKs) are to be considered invalid. They need to be replaced in a way intractable to the leaver, but easily computable for all remaining participants. The Group Manager sends out a multicast message consisting of two parts: Firstly, it contains a new TEK encrypted for each of the valid KEKs so that every participant with at least a single bit of difference with the leaver's ID can calculate the new TEK. Secondly, it contains a new replacement KEK encrypted with both the old KEK and the new TEK for each of the invalid KEKs, so that every participant remaining in the group can update the KEKs it previously had, but does not gain any further knowledge about the keys the other participants have. An example for the message generated when the participant with (binary) ID 0110 leaves is shown in Figure 5.

Multiple Joins The revision numbers of all involved keys only need to be incremented once. Then, the senders have to be informed about the new revision to use.

E(KEK 0.0_{new})	$E_{KEK\ 0.1}$(TEK)	ID Bit #0
$E_{KEK\ 1.0}$(TEK)	E(KEK 1.1_{new})	ID Bit #1
$E_{KEK\ 2.0}$(TEK)	E(KEK 2.1_{new})	ID Bit #2
E(KEK 3.0_{new})	$E_{KEK\ 3.1}$(TEK)	ID Bit #3
Bit's Value = 0	Bit's Value = 1	

The new KEKs are encrypted using a function of the old KEK and new TEK

Figure 5: Centralized Flat: Message to exclude participant 0110

Multiple Leaves When considering the union of all keys owned by all leaving participants as invalid, this will soon result in all, or almost all, of the keys being unusable. Even if not all of the keys are tainted, a large number of legitimate participants will be unable to recover the new TEK. This can be overcome by executing it similar to the tree-based leave. Because keys are not organized in a hierarchical fashion in Centralized Flat, "imaginary" keys are created in the hierarchy, derived from the keys known to the participants: The individual (lowest-level, leaf) imaginary KEK in the hierarchy is calculated as a function (e.g. a simple exclusive-or) of all W KEKs known to that node. The next higher imaginary KEK is equivalent to the function applied to a subset of size $W-1$ of its real keys, e.g. the KEKs corresponding to the highest $W-1$ ID bits, and so on.

When working with these imaginary keys, the Multiple Leave algorithm from Section 3.2 can be applied as is. As an additional bonus, the order of the KEKs can be rearranged arbitrarily, as long as the subset relation described above still holds. This will result in a shorter message at the expense of additional processing cost for the Group Manager.

Note that — unlike in the Centralized Tree approach — expelling colluding participants can not easily be done in the flat approach. Here, they can share their key tables, and thus cover a subgroup defined by the KEKs they do not have in common. Every participant sharing each of his individual KEKs with at least one of the colluding parties is indistinguishable from them in terms of keying material that he holds. Most other approaches known to us are unable to exclude colluding participants — short of re-creating the whole group without them. With out flat approach, excluding colluding participants is possible by overspecifying the range, i.e. considering all keys held by the colluding participants to be tainted. This will usually exclude a certain amount of valid participants as well, and they will have to re-register with the group manager.

Group Merge Merging two groups can be achieved by the two Group Managers agreeing on a single fresh set of keys (KEKs and TEK). Each Group Manager then sends out the new key encrypted with the equivalent old key, then one of the Group Managers resigns its position.

This only works if participants can keep their IDs. This strengthens the need for 'coordinated' ID assignment, e.g. by using something derived from the network addresses.

A similar mechanism can be used to recover from the failure of a Group Manager. After a new manager has been designated, he just collects the key tables from a few selected group members, and is thus able to reconstruct the full set of $2W$ Key Encryption Keys.

Group Split Splitting the group is done analogously to the procedure described in Section 3.2: Each of the new groups performs a multiple leave for the non-members. The main difference to note is that groups that have been merged cannot take advantage of the simplification mentioned in Section 3.2's description of Group Split.

3.4 Distributed Flat Key Management

The main concerns with centralized approaches is the danger of implosion and the existence of a single point of failure. It is thus attractive to search for a distributed solution for the key management problem. This solution was found in completely distributing the key database of the Centralized Flat approach, such that all participants are created equal and nobody has complete knowledge. As in the Centralized Flat approach above, each participant only holds keys matching his ID, and the collaboration of multiple participants is required to propagate changes to the whole group. There is no dedicated Group Manager, instead, every participant may perform admission control operations.

While some participants will be distinguished as *Key Holders*, performing some authoritative funtion, this function a) is only needed to improve performance on version changes, b) is assigned naturally to the creator of the newest version of the key, and c) can be taken over at any time by any other participant knowing the key, if that node should seem to have disappeared.[2] The duties of a Key Holder are to heartbeat the key and to perform key translations. They will be detailed in the description of the operations below.

Since there is no Group Manager knowing about the IDs in use, the IDs need to be generated uniquely in a distributed way. Apparent solutions would be to use the participant's network address directly or to apply a collision-free hash function.

This scheme is the most resilient to network or node failures because of its inherent self-healing capability, but is also more vulnerable to inside attacks than the others. It offers the same security to break-in attacks as the schemes discussed above; thanks to its higher resilience to failures, it can be considered stronger against active attacks.

First Participant The first participant in the group will find that no heartbeat exists and start to create its own keys (the TEK and W of the $2W$ KEKs), the ones it would have received from the Group Manager in the Centralized Flat scheme. Then it starts a heartbeat announcing itself and the fact that it is Key Holder for the keys it just generated. The heartbeat contains for each key the key's ID (bit/value pair as described in Section 3.3), version, revision, and creator's address. In this early phase where no previous common key exists, multiple creations of the same key are resolved as described below, except that a unicast connection is opened between the Key Holders to establish a previous key.

Join All further joins will see the heartbeat and select a previous participant (from the sender address of packets, the list of key creators from the heartbeat, or expanding multicast rings) who is willing to admit them.[3] This *introducer* will send the newcomer the keys the two of them share (the TEK and the applicable KEKs, all with increased revision). KEKs which are needed by the newcomer and do not already exist, are created as in the initial Since the ID can be calculated from the network address, it is easy to select participants having the remaining keys (the introducer, having more knowledge about the group, can assist the newcomer).[4]

Before the leave operation is described, a number of concepts are introduced, which help to understand how the system works with no centralized control and a number of participants performing operations at the same time. This knowledge will also make it easier to follow the description of the join operation.

Heartbeat Each Key Holder performs a regular heartbeat sending out a message containing its view of the newest keys and a short history of previous keys, as an automatic retransmission in case some messages were lost, in a format analogous to those described in Section 3.3. Each participant who recently has created a key, will consider

[2] If no remaining participant has that key, nobody needs to be Key Holder for it.

[3] Of course, the newcomer has to make sure that the introducer is trustworthy, i.e. both sides perform access control

[4] These additional key contributors can perform a simplified access control procedure if the newcomer includes a MAC with the TEK

itself a Key Holder, until it has received a heartbeat superseding his (i.e. having every key at least as new as his own). This results in a small number of messages being sent out in a regular fashion, in addition to the rekeying messages needed by Centralized Flat. If a Key Holder should stop announcing its function, any other participant knowing that key can take over. The participants willing to take over should use a non-flooding election scheme to decide.[5]

Key Merging Since multiple parties may create new keys at the same time, each has to include its own ID to assure uniqueness. Additionally, it has to include on which key (version, revision, version creator) the new key is based, since this also is the key it is encrypted with. This allows the participants to implicitely (i.e. without sending additional messages) agree on a common key and also be able to understand any traffic that was encrypted using both the individual and the merged keys. See Figure 6 for examples.

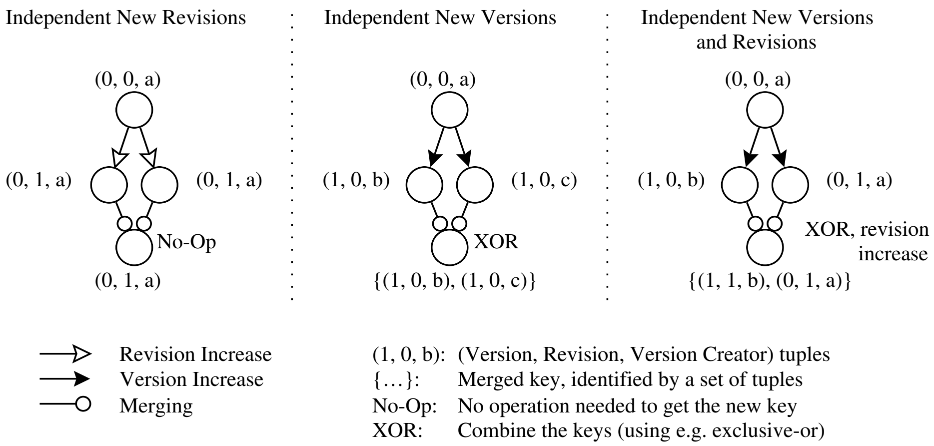


Figure 6: Different Key Merging situations

Multiple new revisions There is no conflict, since the key is the same.

Multiple new versions Any participant seeing that the same version has been created by several Key Holders, can combine these keys into a single new key which can be easily calculated from the base keys (e.g. using exclusive-or). The merged key's ID will be the set of ID tuples. Any Key Holder of a base key should consider itself as a Key Holder of the merged key.

New versions and new revisions Any participant seeing a revision increase on a key that has been superseded, should increase the revision of the new key accordingly to assure perfect forward secrecy. Any Key Holder for the new key may re-encrypt the new key with the new revision of its base key, to make life easier for the newcomer.[6]

Key Superseding A Key Holder stops performing a heart-beat, if its message is superseded. A message with key K is to be considered superseded, if any of the following keys are being announced: (a) a newer revision, (b) a newer version, which bases on K or any key superseding it, (c) a merged key which includes K, or (d) K is a merged key and it is being announced by a contributor to that key which has higher priority (e.g. higher network address).

[5] E.g. expanding multicast rings where the participant with higher priority (e.g. higher network address) wins. Additionally, the replacement Key Holder might want to perform a Leave for the old Key Holder.

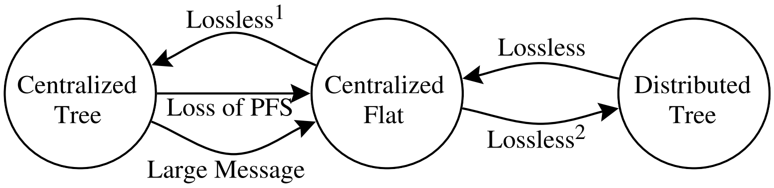
[6] Otherwise, the newcomer needs to contact some of its introducers again

Leave Now, the leave operation works analogous to the description in Section 3.3, with the participant taking care of someone's leave ("excluder") becoming Key Holder of this new version, announcing the new key and who has left (to update the other participants' Admission Control). Since the excluder will not know all keys whose version needs to be increased, the current Key Holder of these Keys will perform the version increase; it works as a "key relay". Participants wishing to leave also can initiate this operation through a key relay (without supplying the new keying material, which they are not supposed to know).

The other operations such as multiple joins and leaves and group merges can be performed analogous to the description in Section 3.3 when making use of the relays, since no participant is supposed to know more than its share of keys.

3.5 Transitions

As we have seen, the three schemes are closely related. It is thus worth exploring the possiblities to change between the schemes at run-time. The possible transitions are show in Figure 7.



[1] No security gain for old participants: Colluding old participants still cannot be expelled, participants joining after the transition can.

[2] Previous Group Manager still knows all keys and thus cannot be expelled.

Figure 7: Transitions between the three schemes

The transitions between the two flat schemes are simple, because they use the same data structure. Towards the centralized flat approach, the transition happens by appointing a new Group Manager and giving him all the keys, in the other direction it can be done even after the Group Manager ceases to exist, and can thus also be viewed as a backup solution or to create a basis to elect a new Group Manager. Its only requirement is that each participant must be able to perform access control functions, or needs to trust another participant in doing this.

This transition pair is most attractive because a heterogeneous approach combining the advantages of both schemes can easily be created: Centralized Flat is used whenever possible to simplify the participants' operation, except when the Group Manager gets overloaded or becomes dysfunctional.

The transition between the two centralized schemes is more complex, as it involves changes in the key structure. A hierarchy can be generated from the flat table in the way described in Section 3.3's Multiple Leave. Keys derived from this hierarchy are then used to populate the tree data structure held by the Group Manager.

The transition from Centralized Tree to Centralized Flat is more difficult, and depends on the internal design of the keying material generator in the Group Manager. If the keying material is generated such that perfect forward secrecy of the system is assured, a transition basically involves the notification of each participant, carrying his new keying material. Buf if a limited amount of perfect forward secrecy is sufficient, another generation process can be utilized instead. Here, the Group Manager holds $2W$ generating secrets, one for each branching on each level of the tree. A short multicast message from the Group Manager to the participants is then sufficient to reveal the generating secrets to those entitled to undestand them, and leads to the table data structure. Finding a solution to this which retains perfectforward secrecy is under investigation.

3.6 Untrusted Senders

If there are multiple Senders, which are only partially trusted, i.e. they should not be able to decrypt traffic sent by other Senders, each Sender has to have its own TEK, transmitted from the Group Manager through an individual secure channel. The Receivers can acquire the TEKs in several ways:

Independent TEKs All the Receivers get a complete set of TEKs, one for each Sender. This increases the size of a key change message and also the storage needed in receivers.

Related TEKs The Group Manager sends the Receivers a Master TEK, from which they derive the Sender's TEK in a algorithmic way. This function must not be reversible by any Sender. A good choice would be to feed the contatenation of the Master TEK and the Sender's ID through a cryptographic hash. The size of the message doesn't increase, but there is more processing overhead on the Receivers' side. Since the keys can be generated on the fly, only the keys for the currently active Senders the Receiver is interested in need to be calculated and stored.

3.7 Reducing the Load on the Group Manager

Asymmetric cryptographic operations are very costly (Table 2). To reduce the load they pose on the Group Manager in the centralized approaches, it can be off-loaded to any number of Session Setup nodes. If required, any trusted participant with bi-directional bandwidth capacity can become a Session Setup node (to be found using increasing MCast rings). See Figure 8. Session Setup nodes perform asymmetric authentication and symmetric session key exchange, then forward the information to Admission Control, which either sends it to the Group Manager or returns a reject message.

Since the main goal of the centralized approaches is to have a tight control, it is not feasible to have everything distributed and tight access control checking is needed. Trustworthy Session Setup entities should be chosen wisely.

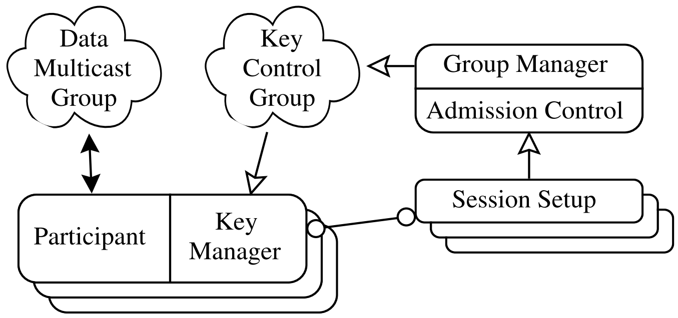


Figure 8: Offloading expensive asymmetric operations by replicating Session Setup

4 Evaluation

The three presented schemes behave differently in terms of offered functionality, achieved performance, and how they deal with security threats. These properties will now be explored.

4.1 Offered Functionality

Table 1 compares the properties for each scheme. Most properties are self-explanatory, the others are described here:

Multiple leaves Multiple leaves are more difficult in the approaches using flat datastructures. Having multiple invalidated fields causes the table to become sparse, thus the normal mechanisms can not be used. Forcing out collaborating entities is difficult.

Easily recoverable If the group manager or other group members suddenly disappear, the flat approaches can recover from this situation by either electing a new group manager in the centralized approach, or shifting key holders in the distributed approach. This does not involve the cooperation of the whole group, but only a few participants. Thus failure recovery or self-healing can be achieved.

Assigned IDs While the centralized flat approach can work with assigned IDs, it may be unwanted to remember the assignment of IDs, and thus the use of IDs defined by the network (or a function thereof) may be preferred.

Exclusion of colluding participants This is possible in the Flat schemes, but will also exclude a number of valid participants, which will need to join again.

Property	Tree	Centralized Flat	Distributed Flat
Allows establishment of group-wise key to achieve privacy and/or authenticity	yes	yes	yes
Perfect forward secrecy	yes	yes	yes
Dynamic join and leave can be handled	yes	yes	yes
Trust in third parties required	no	no	no
Designed for one central controlling entity	yes	yes	no
Controlling entity must know all participants	yes	no	no
Multiple leaves	yes	difficult	difficult
Exclusion of colluding participants	yes	difficult	difficult
Joining and separation of groups	easy	yes	yes
Setup implosion is an issue	yes	yes	no
Return channel required during operation	no	no	yes
Assigned IDs or Network IDs	both	both	network
Single point of failure	yes	yes	no
Easily recoverable	no	yes	yes
Small database	no	yes	yes
Involvement of multiple parties for leave/join	no	no	yes

Table 1: Properties of different schemes

4.2 Useability

While the centralized approaches are better suited for broadcasting and high-security applications, the distributed approach fits more into dynamic conferencing without a dedicated session chair. While memory requirements for the group manager are significantly higher in the tree scenario (see memory consumption below), this allows for an additional level of control, and may thus be necessary anyway, and worth its cost in certain applications.

The multitude of available features, such as perfect forward secrecy, self-healing, no need for participants to cooperate or return channels to the manager, the possibility to make a transition form one scheme to the other, migrate control and no required trust in third parties allow these approaches to fulfill many different basic needs. They compare favorably to existing approaches in terms of simplicity, reliability, computational requirements and achieved security.

4.3 Achieved Performance

Ressource usage is a critical point in all applications that offer cryptographic functions. Relevant costs (both for the group manager and the participants) are:

- CPU consumption
- Memory consumption
- Communication bandwith
- Typical end-to-end operation delay

While parts of the system are being implemented, implementation specific figures are not yet ready, but will be provided once it has been determined how to measure them in real-world environments. In view of the simplicity of the presented architecture, a sound assessment of the involved costs can be made. The upper bounds given as concrete values are so far confirmed by our implementation, and are appropriate for a Sun "Ultra 1" workstation. The following two tables, Table 2 and Table 3, highlight the required amount for each primitive function to achieve a join or leave operation. Data is given for the group manager and the participants for both the Centralized Tree and Centralized Flat model.

Function	Cost per Function	Join Operation			Leave Operation	
		GM	Newcomer	Participants	GM	Participants
DH Agreement	$< 100ms$	1	1	–	–	–
RSA Signature	$< 200ms$	1	1	–	(1)[a]	–
RSA Verify	$< 50ms$	1	1	–	–	(1)
Key Generation	$< 0.05ms$	1	–	–	$W-1$	–
Hash	$< 0.01ms$	$W-1$	–	$1\ldots(W-1)$[b]	–	–
Encryption	$< 0.01ms$	$W-1$	–	–	$2W-3$[c]	–
Decryption	$< 0.02ms$	–	$W-1$	–	–	$1\ldots(W-1)$[d]

[a]If asymmetric authentication required, e.g. if denial of service by participants is an issue

[b]Operation needs to take place eventually, latest at the next leave of concern to this participant. Mean over all participants is below 2

[c]Includes double encryption of new keys

[d]Mean for all participants is below 2

Table 2: CPU Usage — Tree

W indicates the depth of a tree (equal to $log_2(N)$), or the size of a table in the flat case, a typical value is 32. Algorithms used are MD5 for hashing and MAC computation, and IDEA for encryption operations. As can be seen in the concrete costs, key setup for IDEA in decryption mode is more expensive than it is for encryption mode. This has to be taken into account as the internal key schedules usually will not be cached by the group manager. Particpants may precompute and cache them for their own keys if required.

All function counts in the tables are given as atomic. They may involve multiple encryptions or hash calculations, whose costs have been given in the concrete figures. Thus $W - 1$ hash operations would require less than $(W - 1) * 0.01ms$. The cost also includes key setup times for encryption/decryption algorithms.

An additioal cost, incurred by all participants covers memory management, tree traversal, MAC computation for outgoing messages etc. A conservative estimate of the expected costs per operation for each participant places this below $0.03ms$.

The costs for the first three operations in the table may be delegated to a dedicated replicated setup component that does only the asymmetric computations and access control verification. This saves the central group manager component most of the load for the joining of new participants.

Function	Cost per Function	Join Operation			Leave Operation	
		GM	Newcomer	Participants	GM	Participants
DH Agreement	$< 100ms$	1	1	–	–	–
RSA Signature	$< 200ms$	1	1	–	(1)[a]	–
RSA Verify	$< 50ms$	1	1	–	–	(1)
Key Generation	$< 0.05ms$	1	–	–	W	–
Hash	$< 0.01ms$	$W+1$	–	$1\ldots(W+1)$[b]	–	–
Encryption	$< 0.01ms$	$W+1$	–	–	$2W$[c]	–
Decryption	$< 0.02ms$	–	$W+1$	–	–	$1\ldots(W+1)$[d]

[a] If asymmetric authentication required, e.g. if denial of service by participants is an issue

[b] Operation needs to take place eventually, latest at the next leave of concern to this participant. Mean over all participants is below 2

[c] Includes double encryption of new keys

[d] Mean for all participants is $1 + W/2$

Table 3: CPU usage — Centralized Flat

In the case of the distributed flat approach, the costs of the centralized flat appraoch apply, but some participants additionally incur the costs of the group manager in the central flat approach. In the best case, the sum of the additional costs is the same as the cost of the group manager.

For all scenarios, additional periodic costs may incur. To achieve perfect forward secrecy, the group manager may choose to update its own secret value (used to establish a shared secret with joining participants, for example a Diffie-Hellman key) regularly, e.g. once an hour. This would not change anything for current participants, it would just put a small additional load on the group manager.

Memory consumption is very different in the tree vs. flat scenarios. For the tree, the group manager needs to hold all N participants, and an additional $N-1$ KEK nodes. This corresponds to a storage of about 40 bytes per tree node or leaf, in an uncompressed tree, or two times this figure for each prospective participant. The tree can be sparsely populated and compressed. It can also be grown at run-time, so the group manager need not commit to a certain size in the beginning. In the tree scenario, memory requirements for each participant amount to W times 40 bytes, or less than 10kB even for IPv6 IDs. In the flat scenarios, the memory requirement for each participant and the Group Manager is small. Some additional information may need storage, such as key ownership, but total cost is below 20kB in all cases. This makes the approach usable on platforms with comparatively reduced ressources, such as embedded systems.

On the communication side, join operations in centralized scenarios induce no additional traffic, and participants are notified of key revision changes implicitly, by the reception of messages encrypted with a higher revision number. A leave operation causes a message of typically $2W * 40$ bytes to be sent, or about 1-2 kB. This message may need to be retransmitted in one of the reliable multicast implemenations, increasing the pariticpants delay until he receives the updated keying material. In the distributed scenario, multiple exchanges are required, resulting into $2W$ multicast messages in the worst case. This may also involve a few unicast messages to cover gaps between unrelated subgroups.

4.4 Co-operation

This approach builds a complete framework, but it doesn't stand alone. It works nicely on top of (unicast) security architectures such as the one mandatory for IP version 6 [Atk95]. We are working on an integration into SKIP [CLA$^+$96], which is available in source for a number of platforms.

Our schemes also work atop any reliable multicast protocol (e.g. [FJM$^+$95, PSB$^+$95, PTK94]). It can also work without any, but this will increas the load on the Group Manager. It also can take advantage of the proposed Integrated Services architecture for the Internet and the associated resource reservation protocol, RSVP [BCS93] to limit the work that has to be done by the reliable multicast protocol and thus reduce latencies.

Our schemes do not rely on specific cryptographic algorithms and protocol, but can use any of a number of them providing a basic functionality. So even if one of them should have to be considered weak or even broken, these components are easy to change and this framework will continue to work.

5 Conclusions and Further Work

In this paper we presented a complete framework for secure multicasting. The core of the framework consists of three approaches which have different properties, but rely on the same basic philosophy. All our approaches organize the space of keys that will eventually to be assigned to group members in a unique way, without actually generating the keys as before they are needed. Only when new group keys need to be established, they are generated and distributed to only the members of the group affected by a change. Our organization of the key space assures that all operations on groups may be executed with a complexity of $O(logN)$ or less, where N is the size of the group, and the complexity is measured in the size and number of messages exchanged, and the number of cryptographic operations to be performed by any of the participants.

Our three approaches differ in some important aspects. Among others, they offer the system designer a choice between

- centralized or distributed key management,
- no or some trust in other participants,
- varying degrees of load on the participants, and
- tight control of the group or failsafe distributed operation.

As discussed in the introductory section, various authors have published work on secure multicasting schemes. Some of the properties as presented in Table 1 are also offered by their approaches, but we are not aware of any scheme that has all these properties while maintaining the efficiency of ours.

Some considerations deserve further studies. Although a preliminary implementation is available and working, we still lack experiments with large and distributed groups; to this end, the integration of our experimental software into currently available platforms is planned, such as SKIP [CLA$^+$96] and ISAKMP/Oakley [Orm97]. The possibility of a hot switching between the approaches presented as discussed in Section 3.5 is a recent discovery, and needs to be considered in detail. Specifically, an efficient translation algorithm between the tree (Section 3.2) and the flat data structure (Section 3.3) needs to be found and analyzed. Furthermore, we anticipate that batching of leave operations may be made more efficient with optimal grouping of the participants leaving within some time interval.

References

[Atk95] R. Atkinson. Security architecture for the internet protocol. RFC 1825, August 1995.

[ATM95] ATM Forum. *UNI Signalling 4.0*, 1995.

[Bal96] A. Ballardie. Scalable multicast key distribution. RFC 1949, May 1996.

[Bal97] A. Ballardie. Core based trees (CBT version 2) multicast routing. RFC 2189, September 1997.

[BCS93] R. Brade, D. Clark, and S. Shenker. RSVP: A new resource reservation protocol. *IEEE Network*, September 1993.

[BD95] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology: Proceedings of Eurocrypt '94*, number 950 in Lecture Notes in Computer Science, pages 275–286, Berlin, 1995. Springer-Verlag.

[BD96] M. Burmester and Y.G. Desmedt. Efficient and secure conference-key distribution. In *Security Protocols Workshop*, pages 119–129, 1996.

[Ber91] S. Berkovits. How to broadcast a secret. In *Advances in Cryptology: Proceedings Eurocrypt '91*, pages 535–541, April 1991.

[BSH+93] C. Blundo, A.D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In *Advances in Cryptology: Proceedings of CRYPTO '92*, pages 471–486, 1993.

[CC89] G.H. Chiou and W.T. Chen. Secure broadcasting using the secure lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[CLA+96] G. Caronni, H. Lubich, A. Aziz, T. Markson, and R. Skrenta. Skip: Securing the internet. In *Proceedings of the IEEE Fifth Workshop on Enabling Technologies (WET ICE)*, 1996.

[DC90] S.E. Deering and D.R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8:85–110, May 1990.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[Dif90] W. Diffie. Authenticated key exchange and secure interactive communication. In *Proceedings of "8th Worldwide Congress on Computer and Communications Security and Protection" SECURICOM 90*, pages 300–306, 1990.

[DPW88] S.E. Deering, C. Partridge, and D. Waitzman. Distance vector multicast routing protocol. RFC 1075, 1988.

[FJM+95] S. Floyd, V. Jacobson, S. McCanne, L. Zhang, and C. Liu. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of ACM SIGCOMM '95*, pages 342–356, September 1995.

[GS95] L. Gong and N. Shaham. Multicast security and its extension to a mobile environment. *Wireless Networks*, (1):281–295, 1995.

[HM97a] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) architecture. RFC 2094, July 1997.

[HM97b] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) specification. RFC 2093, July 1997.

[ITW82] I. Ingemarsson, D.T. Tang, and C.K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, September 1982.

[MB94] M.R. Macedonia and D.P. Brutzman. MBone provides audio and video across the internet. *IEEE Computer*, 27(4):30–36, April 1994.

[McC92] Steve McCanne. A distributed whiteboard for network conferencing. http://http.cs.Berkeley.edu/˜mccanne/unpublished.html, 1992.

[MI87] T. Matsumoto and H. Imai. On the key predistribution system — a practical solution to the key distribution problem. In *Advances in Cryptology: Proceedings of CRYPTO '87*, pages 185–193, 1987.

[Mit97] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of ACM SIGCOMM '97*, pages 277–288, September 1997.

[Orm97] H.K. Orman. The OAKLEY key determination protocol. Internet Draft (work in progress), draft-ipsec-ietf-oakley-02.txt, 1997.

[PSB+95] S. Paul, K. Sabnani, R. Buskens, S. Muhammad, J. Lin, and S. Bhattacharyya. RMTP: A reliable multicast transport protocol for high-speed networks. In *Proceedings of the Tenth Annual IEEE Workshop on Computer Communications*, September 1995.

[PTK94] S. Pingali, D. Towsley, and J. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of SIGMETRICS '94*, 1994.

[Sch96] B. Schneier. *Applied Cryptography*. Jon Wiley & Sons, New York, 2nd edition, 1996.

[Sim92] G.J. Simmons, editor. *Contemporary Cryptography: The Science of Information Integrity*. IEEE Press, 1992.

[STW97] M. Steiner, G. Tsudik, and M. Waidner. Cliques: A protocol suite for key agreement in dynamic groups. Research Report RZ 2984 (#93030), IBM Zürich Research Lab, December 1997.