The VersaKey Framework: Versatile Group Key Management

Marcel Waldvogel, Germano Caronni, *Member, IEEE*, Dan Sun, *Student Member, IEEE*, Nathalie Weiler, *Student Member, IEEE*, Bernhard Plattner, *Member, IEEE*

Abstract—

Middleware supporting secure applications in a distributed environment faces several challenges. Scalable security in the context of multicasting or broadcasting is especially hard when privacy and authenticity is to be assured to highly dynamic groups where the application allows participants to join and leave at any time.

Unicast security is well-known and has widely advanced into production state. But proposals for multicast security solutions that have been published so far are complex, often require trust in network components or are inefficient. In this paper, we propose a framework of new approaches for achieving scalable security in IP multicasting. Our solutions assure that that newly joining members are not able to understand past group traffic, and that leaving members may not follow future communication.

For versatility, our framework supports a range of closely related schemes for key management, ranging from tightly centralized to fully distributed and even allows switching between these schemes on-the-fly with low overhead. Operations have low complexity ($O(\log N)$ for joins or leaves), thus granting scalability even for very large groups. We also present a novel concurrency-enabling scheme, which was devised for fully distributed key management.

In this paper we discuss the requirements for secure multicasting, present our flexible system, and evaluate its properties, based on the existing prototype implementation.

Keywords— **Secure multicasting middleware, Tree-based key distribution, Multicast key distribution schemes, Distributed key management, Concurrent key distribution.**

I. Introduction

With the increasing ubiquity of the Internet and the growing popularity of IP multicasting, multi-party communication is fast becoming a requirement for 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 taking advantage of multicast transport services are collaborative multimedia applications and conferencing services [1]. This usage will grow and include new applications such as fault-tolerant, distributed database systems [2] or massively-parallel supercomputers made of workstations [3].

Besides the basic need to exchange information among the members of a group, the requirements of specific applications differ greatly. Resulting groups come in very different sizes: small (in the case of a simple multi-party desktop conference), medium (e.g., distance-education scenario), or very large groups (e.g., broadcasting of a major sports event). In many applications, group members may also decide to join or leave the group frequently and at any time. Best-effort IP multicast service was specifically designed to address these requirements, and does this very well.

But it is missing additional features that have to be provided by other means: Quality of Service and resource reservation issues are being covered by numerous schemes such as [4], [5]. Reliable transmission of data and concurrency resolution are generally considered to be application-specific, if overhead is to be minimal [6], [7]. But currently the provision of privacy and authenticity for group members, e.g. by cryptographic means, is still missing. Current solutions often require human intervention (manual keying is common), or restrict the dynamics provided by multicasting and required by many applications.

In this paper, we investigate how secure multicasting can be provided as a universal service in an application-transparent middleware, while 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 in other environments, e.g. with connection-oriented multicast services as found in ATM [8] or even one-way broadcast services.

Like many unicast applications, a large group of multi-party multi-media applications will only be successful if privacy and authenticity of participants can be provided efficiently. Consider, for example, a tele-education service, which distributes its program 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 it. If a new customer subscribes, she should be able to receive data immediately, but not to understand information which was released before the time of her subscription. Conversely, a customer canceling his subscription should not be able to process information beyond the time of cancellation.

Similarly, consider a teleconference meeting between managers of a virtual corporation which need some outside expert opinions during their meeting, but do not want this expert to learn about the other topics they are discussing.

By consequence, this paper will 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 naïve solution would be to create a new session key whenever a member leaves the group, and to securely distribute the key to each member of the group, one by one. 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 dis-

Marcel Waldvogel, Nathalie Weiler, and Bernhard Plattner are with the Computer Engineering and Networks Laboratory, ETH Zürich, Switzerland (<*last name*@tik.ee.ethz.ch>). Germano Caronni is with the Network and Security group of Sun Labs, Palo Alto, California, U.S.A. (<gec@acm.org>). Dan Sun is with OpenCon Systems, New Jersey, U.S.A. (<kathysunwang@hotmail.com>).

tribute 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 [9], and generally with approaches using reverse path broadcasting/multicasting [10], which distribute multicast datagrams quite generously to a set of potential recipients which is much larger than the actual set of group members. 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.

We argue that a solution for secure multicasting must offer the following properties:

- Groupwide privacy and authenticity, including the unability of newcomers to read past traffic.
- Efficient distribution of keying material in large groups with frequent membership changes (minimize traffic and computation effort for all parties involved).
- No trust in intermediate or third party components.
- Avoid multicast implosion.
- No restriction of the services offered by the underlying multicast infrastructure (e.g. avoid unicasts and relaying).
- Minimize knowledge needed by participating entities.
- Minimize attack vulnerabilities.

Additionally, the system should address the following issues:

- Provide Perfect Forward Secrecy [11].
- Cope with system and network failures (failure recovery and/or resilience).
- Work with (mostly) one-way traffic, such as satellite broadcasts.
- Allow sender authentication (as opposed to group-wide authentication).

In this paper, we present three closely related schemes for key distribution and management, ranging from tightly centralized to completely distributed. Each of them already meets most of the requirements above. For the case that requirements change during the life-time of a group (e.g. unexpected growth), we also provide for a set of efficient transitions from one scheme to another. This yields a truly versatile framework that achieves scalable security in IP multicast, enabling secure multi-party multimedia applications in which members of large and highly dynamic groups may participate.

Our approaches allow all group members to establish a mutually shared secret, which can be used to provide group-wide privacy and message authenticity, or any other property relying on shared secrets. The system can offer perfect forward secrecy [11], requires only a small amount of calculations and storage from the participants, can be made highly resilient to component and network failures, and avoids the need for trust into third party components such as routers. It is independent of the security algorithms used, so it can work together well with IP Security (IPsec [12]) encryption and authentication mechanisms.

The remainder of the paper is organized as follows: Section II presents related work, Section III introduces the three key management solutions, and Section IV explains the transitions between them. Section V then evaluates the functionality and performance of VersaKey. Section VI draws conclusions and explores further work.

II. Related Work

Although a number of cryptographic techniques have been proposed to secure group communication in broadcast or multicast scenarios, very few of them are targeted at a large group setting with highly dynamic membership without third party trust, and if they do, they are complex and inefficient in dealing with this issue.

The existing approaches or applications concerning multicast key management can be separated into two classes. Those offering dynamic operations are able to change group keying material on the fly. Static solutions, forming the second class, require the establishment of a new group to cope with membership changes. Manual keying, still being the prevalent solution to multicast key management as e.g. used in the MBone applications, is considered an insufficient key management solution.

A. Static Key Management Approaches

The static approaches distribute an unchanging group key to members as they join. They provide no solutions for changing the key when the group membership changes other than establishing a new group from scratch.

For IP multicast security, several key management schemes are proposed, e.g. the Group Key Management Protocol (GKMP) [13], [14], the Simple Key-Management for Internet Protocols (SKIP) [15], the Internet Key Exchange (IKE) [16], making use of the Internet Security Association and Key Management Protocol (ISAKMP) [17] and the the Oakley Key Determination Protocol [18], and the Scalable Multicast Key Distribution Scheme (SMKD) [19]. None of them provides a solution for key change upon membership changes or for Perfect Forward Secrecy (PFS). The properties of all presented schemes are summarized in Table I.

B. Dynamic Key Management Approaches

In order to prevent the joining members from understanding the past traffic and the left members from listening to future messages, dynamic changes of the session key must be possible without rebuilding the whole group. Among the existing dynamic approaches, centralized and distributed schemes can be distinguished depending on if they rely on a designated central entity.

A few schemes can be enumerated as centralized dynamic approaches, like Key Pre-distribution [20], Fiat-Naor Broadcast Encryption, [21], Secure Lock [22], the spanning tree-based scheme [23] and [24]. All of them require a designated centralized controller to take care of distributing and/or updating keying material. However, they also share the inherent drawbacks: possible setup implosion, single point of failure and relatively large database for the keying material.

To reduce the storage at the user's end and the message length broadcast by a center for dynamically changing privileged subset of users, several schemes were presented by Fiat and Naor [21].

Wallner et al. [25] propose a key management scheme for multicast communication which requires each of the N users to store $log(N)+1$ keys. In order to remove a user from the group, a new group key must be generated. Unlike in the Fiat-Noar

TABLE I
PROPERTIES OF DIFFERENT SCHEMES

Property	Static Approaches (GKMP, SMKD, SKIP, ISAKMP/Oakley)	Centralized Approaches (Pre-distribution, Secure Lock, Fiat-Naor, Spanning tree, Iolus)	Distributed Approaches (Cliques)	VersaKey
Group-wide key	yes	Iolus: no; others: yes	yes	yes
Dynamic join and leave handled	no	yes	yes	yes
Scalability	no	Iolus/Spanning tree: yes; others: no	yes	yes
Perfect forward secrecy	no	no	no	yes
Centralized entity required	yes	yes	variable	variable
Trust in third parties required	SMKD: yes; others: no	Iolus: yes; others: no	no	no
Trust in other participants	no	Spanning tree: yes; others: no	yes	no[a]
Memory with each entity required	small	Pre-distribution: huge; others: small	small	small[b]
High Delay in key distribution	no	Spanning tree: yes; others: no	Initial setup: yes; otherwise: no	no

[a] Distributed Flat: yes, but untrusted participants can be safely ignored
[b] Except group manager in Centralized Tree: large

broadcast encryption schemes, the number of transmissions required to rekey the multicast group is small. However, in this scheme every group member must assure that he receives all the update messages sent by the group manager. A similar approach has been proposed in [26].

Secure lock is implemented based on the Chinese Remainder Theorem. Here, the group session key is secured in a way that only the keys of authorized users can retrieve it. This scheme requires the association of one large number (relatively prime to all other group members' numbers) with each participant. In addition, the retrieval of the group session key is an expensive operation. These conditions confine this protocol to being used only within small groups.

The spanning tree [23] needs to be extended or pruned, whenever the membership changes, to make sure that only the group members can get the updated conference key. The delay in distributing a conference key along the spanning tree makes this approach not applicable for frequent changes of membership.

Iolus deals with the scalability issues in highly dynamic large groups by decomposing large groups into subgroups. Thus, a group membership change can be handled in the respective subgroup without affecting any other subgroups. While improving scalability, the absence of a global group key requires the introduction of secure agents, one for each subgroup, to relay messages and perform "key translation". In addition to requiring full trust into each subgroup agent, extra delays in message delivery must be accepted.

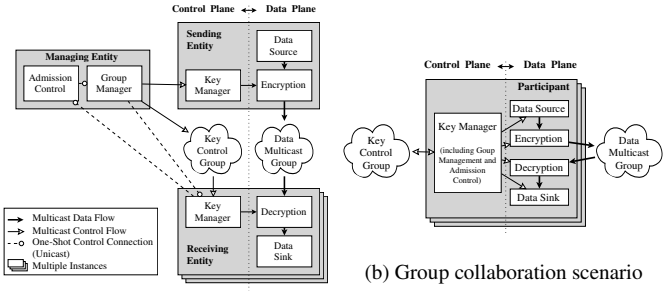
Cliques, described by Steiner et al. [27], is a natural extension to the Diffie-Hellman key exchange protocol and presents the capability to distribute session keys in dynamic groups. The group controller can be either fixed with a designated node or transferred to the newly joint member. While this 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.

As summarized in Table 1, most 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 [24], [27]. However, several issues must be improved: the reduction of computational complexity, decrease 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 now present several schemes that demonstrate the ability to successfully handle these issues in large and highly dynamic groups.

III. SECURE MULTICASTING ALGORITHMS

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 wide-scale distribution 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 decrypted 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



(a) Broadcast-like scenario

(b) Group collaboration scenario

Fig. 1. Two Possible Multicast Scenarios

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" [11]. To completely achieve this, also the unicast connections need to be set up using ephemeral secrets.

This section is organized as follows: First, the general architecture and components of the framework are 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 concludes the discussion.

A. Components and Group Operations in Multicast Scenarios

Figure 1(a) illustrates the basic architecture for a simple scenario consisting of a single sending entity and any number of receiving entities. Generally the components are 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 data source, data sink, encryption and decryption units and the data multicast group(s). (2) a group of control (or key management) related components, which includes all components involved in the key agreement and key exchange process. Note that in the centralized approaches described below, it is possible to locate instances of the admission control component on different machines, thus mitigating a potential implosion problem.

The outline of the multicast data flow from the sending entity to one of the receiving entities is as depicted in Figure 1(a): The data source is fed to the encryption unit to be multicast to the addressed data multicast group. The receiving entity performs the necessary decryption and hands its result on to the data sink. The control related components provide the necessary keys to the encryption and decryption units.

An overview of the roles of the different components in Figure 1(a) during group management operations are shown in Table II (for the distributed approach explained below, the duties of the group manager are shared by all participants). Further possible operations concern the group setup: creation, destruction, merging, and splitting of groups. They are highly dependent on the key management scheme and will therefore be discussed in the corresponding sections. Also, the exclusion of multiple colluding participants is to be treated differently in some of the schemes.

The components have been described for a simple scenario. However, there often is more than one sender, and senders and receivers may not be distinguishable. Also, any receiving entity is free to send data encrypted or authenticated using the current group-wide symmetric key, 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 1(b). This group collaboration scenario arises from a transformation of Figure 1(a) where sending and receiving entity were integrated, yet the group manager remains isolated. All of the schemes also work in this scenario, and the later presented distributed key management scheme (cf. Section III-D) is very well suited for it. If senders and receivers are treated equally, they will be referred to using the more generic term *participant*.

In the following two subsections, we will illustrate additional aspects, namely the properties of keying material, and the basic operations in the groups.

A.1 Identification of Keying Material

We distinguish two types of keys. Firstly, we need a key to encrypt, decrypt, and possibly authenticate the data traffic. For this purpose, the *Traffic Encryption Key (TEK)* is given by the local key manager to the appropriate unit. Secondly, a number of *Key Encryption Keys (KEKs)* are used to encrypt the control traffic in the key control group, ultimately containing the TEK.

To distinguish the keys, each key is addressed through a *key selector*, consisting of (1) a unique ID which will stay the same even if the secret keying material changes, and (2) a version and revision field, reflecting updates in the keying material (cf. Figure 2). The version is increased whenever new keying material is sent out by the group manager on a leave, while the revision is increased whenever the key is passed through a one-way function, eliminating the need for sending update messages on joins.

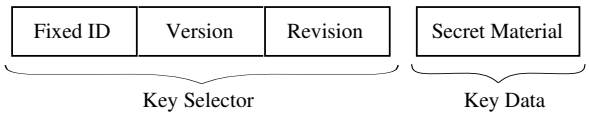


Fig. 2. Structure of a Key

A.2 Basic Operations on the Group

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

TABLE II
INTERACTIONS OF THE DIFFERENT COMPONENTS DURING THE OPERATIONS

Components	Operations		
	Join		Leave
	Single	Multiple	
Participant key manager	update keying material (4)[a]		update keying material (3)
Key manager of entity/-ies requesting operation	request (1) update keying material (4)		no comprehensibility of the keying material update (3)
Group manager	change keying material, notification of the joining entity (3)	common handling of several requests (3)	change keying material (3)
Admission control	asymmetric cryptographic operations, check of access rights (2)		change of access rights for leaving entity (1) notification of the group manager(2)[b]

[a] The numbers in parentheses indicate the sequence of steps.
[b] This is policy-dependent. In case of a voluntary leave, the keying material may be kept the same.

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:

1. Silent Leave: A receiver just stops participating in the group without telling anyone. No action is needed.
2. 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.
3. 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.

B. Centralized, Tree-Based Key Management

In our first approach, we proposed and implemented a centralized, easy maintainable scheme which achieves tightest control over the individual participants [28]. It is suitable for applications with high security demands, and poses very little load on the network and the receivers. All keying material is managed centrally by the group manager, where all joining entities have to register. To store the keying material, a tree is used in which all participating entities are represented by its leaves. For simplicity of the explanation assume that a fully balanced binary tree is used. The example in Figure 3 depicts such a tree with a maximum of 16 group members (address length W of 4 bits).

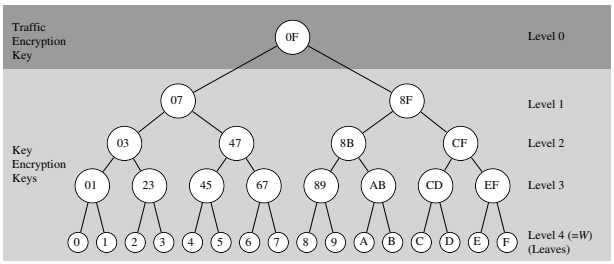


Fig. 3. Binary Hierarchy of Keys. Labels in hexadecimal define the range of participants knowing this key.

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 KEK. The group manager stores it in the leaf node associated with this par-

ticipant, and uses it whenever only this individual participant should understand a message — such as for unicast traffic during this participants join operation. Its revision is increased after each use to insure perfect forward secrecy. Besides incrementing the revision field, the keying material is passed through a one-way function, so that earlier traffic can not be recovered by the newcomer. 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 hold keys for a hierarchy of virtual subgroups of different sizes.

Each participant holds a different subset of keys from the tree, more specifically all those keys that are in the path from the participants leaf to the root node, which is used as the TEK. These intermediate KEKs 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 participants, 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 participants. 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, thus reducing the amount of messages and decryptions needed compared to other independently proposed schemes [25], [26].

B.1 Centralized Tree Operations

B.1.a 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 participant's address (IP address and port number, 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.

B.1.b 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 a future 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 $8 \ldots B, D \ldots F$, 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 $D \ldots F$.
- $8F_{new}$ 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.

B.1.c 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.

B.1.d 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

[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.

be changed such that the actual 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.

B.1.e 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.

B.1.f 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.

B.2 Evaluation for Improvement

This centralized tree based approach is well suited for broadcasting and high-security applications. If we consider the leaving operation for a huge group with 4 billion participants ($W = 32$) and 128 bit encryption keys, a single multicast message of around 1200 bytes is sufficient. It contains all the new keys, appropriately encrypted, that are necessary for the exclusion of a single participant. Processing this multicast message will require at most W decryption operations from the other participants, with an average of less than two decryptions.

Our scheme achieves the objective of establishing group-wise keys to obtain privacy and authenticity, while guaranteeing perfect forward secrecy without any trust in third parties. Joining and separation of groups are easy. However, setup implosion is an issue. Furthermore, the central unit which must be known by all participants is a single point of failure in the system. The relatively large key management database ($O(N)$, with N being the number of participants) is another minor disadvantage of this scheme. To cope better with these issues, we will now modify Centralized Tree key management into a completely distributed key management using a flat key structure, called *Distributed Flat* ($D^\flat$). This approach is well suited for dynamic conferencing applications without a dedicated session chair. Since there are scenarios which require a dedicated session chair, we first introduce an intermediate solution, *Centralized Flat* ($C^\flat$) key management, which copes better with the memory allocation for the key space than the centralized, tree-based approach (cf. Section III-B), yet preserves the simplicity of the centralized approach.

C. 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 storage requirements, and obviates the group manager from the need of keeping all participants in memory. As long as a participant's ID is known, it can be thrown out without the need to have kept any further state (and whether it is currently part of the group at all).

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

Fig. 4. Simple Key Assignment for a Flat ID

In the simplest case, the data structure held by the group manager is a 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. Often, this ID will be taken from the participant's network address, e.g. IP address and port number, in order not to have to keep track of the assigned IDs, since this is already unique. For each bit in the ID, two keys are available. Each participant knows one of those keys, depending on the value of the single bits in his ID. He holds $W + 1$ keys in total. 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, they can also be used independently of each other.

The results are very similar to the Tree-Based Control from Section III-B, 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.

C.1 Centralized Flat Operations

C.1.a 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.

C.1.b 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.

$E(KEK\ 0.0_{new})$	$E_{KEK\ 0.1}(TEK)$	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

Fig. 5. Centralized Flat: Message to exclude participant 0110

C.1.c 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.

C.1.d 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 III-B 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.

C.1.e 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.

C.1.f Group Split. Splitting the group is done analogously to the procedure described in Section III-B: 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 III-B's description of Group Split.

C.2 Group Manager Authentication

In the case of a Centralized Flat key management scenario, a very interesting solution offers itself to the problem of authenticating the group manager, similar to the scheme sketched in [29] which was independently proposed. It can also be extended to permit authentication of data sent from a single source that is co-located with the group manager (as in broadcast scenarios).

To protect against a malicious insider "hijacking" the role of the group manager, traffic from the group manager must be authenticated such that no insider can fake the authentication. Obviously, the TEK can not be used for this. The traditional solution is the use of asymmetric authentication, e.g. RSA, where the sender signs a message, or, to offset processing cost, the MACs of several messages. Receivers can then verify the signature without being able to generate it.

Due to the special nature of the distribution of the KEKs, one can do away with the costly asymmetric authentication altogether. By using all $2W$ KEKs as key to a MAC, $2W$ MACs are generated. When a receiver obtains these MACs together with the key change message that has thus been authenticated, he can check all the MACs for which he holds the KEKs. Everybody holds a different set of KEKs, so only the receiver, or the group manager, are able to create a valid set of MACs. All receivers can verify that the message originated from the manager, but no single receiver can fraudulently create such a message. Due to the symmetric nature of the used mechanism, receivers will not be able to prove the receipt of an authentic message to third parties – but that is not a requirement for the present application.

D. Distributed Flat Key Management

The main concerns with centralized approaches are 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, so the collaboration of multiple participants is required to propagate key changes to the whole group. There is no dedicated group manager, instead, every participant may perform admission control and other administrative functions.

While some participants will be distinguished as *key holders* for some time, performing some authoritative function, 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. If no remaining participant has that key, nobody needs to be key holder for it. The duties of a key holder are to heartbeat the key and to perform key translations. These operations 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 first apply a collision-free hash function on it.

This scheme is highly 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.

D.1 Join Dynamics

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 Centralized Flat. 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 III-C), 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 follows, except that a unicast connection is opened between the key holders to establish a previous key.

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 III-C. Each participant who recently has created a key, will consider 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. A simple example for such a scheme are expanding multicast rings where the participant with higher priority (e.g. higher network address) wins.

D.2 Distributed Flat Operations

Before the operations will be 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.

Parallel Operation Since multiple parties may create new keys at the same time, each has to include its own ID to assure uniqueness, and the ID of the keys it is based on, since only this allows picking the correct key for decryption. Additionally, there needs to be an algorithm to unify the results of multiple parallel operations. Instead of trying to sequentialize all changes, we have devised a simple, yet efficient *continuous consensus protocol*, which allows each participant to always use the most current information.

A sample set of parallel operations for a single KEK is depicted on Figure 6, showing three different snapshots of development happening to the same KEK. Initially, a single key is created ("root" circle), then two independent operations (lines) lead to two new keys (shown in the left snapshot). As soon as any given participant receives the second key change message, both claiming to replace the same key, it needs to merge these changes. Our solution is to put the resulting keys from both messages as elements into the *Active Set*, the set of current keys (shaded area in Figure 6. More formally, the Active Set consists of all the current leaf nodes in the key inheritance graph. Whenever the current KEK needs to be used to encoding a new key with, the active set is used (middle snapshot). If a message announcing this change is received, it will push all its "parent" keys out of the Active Set, since they will no longer be leafs. The result of two more key changes occuring concurrently is shown in the right snapshot (one of the creators of a new key had only received a single key change message at the time it sent out its message).

As we have seen, the creation of a new key element will remove leaf status from all current elements of the Active Set, shrinking it to the single new element. Thus it can be easily seen that the size of the Active Set is bounded by the number of parallel operations within a round-trip time.

The key selector for a KEK is thus the enumeration of the selectors of the contributing keys; the keying material used is by combining the individual secrets (e.g. using exclusive-or). By keeping a short history containing the key elements received during a round-trip time, any message received can be decoded, because it can only reference key elements from this history.

Resolution of parallel operations is only necessary for version changes. Concurrent revision changes will result in the same result, independently of who performs it. In fact, only two revisions are ever needed for a single KEK element. The first message encrypted with any KEK is revision 0 of the current TEK. To achieve forward secrecy, later joiners must be prevented from deciphering this message. This is done by sending the newcomer revision 1 of the current KEK elements through an encrypted unicast channel. Successor keys will also be encrypted using revision 1 and the current TEK. Inclusion of the current TEK has already been done in Centralized Flat to prevent expelled members from learning the new KEK, here the use of the current TEK when encrypting KEKs also helps ensure forward secrecy.

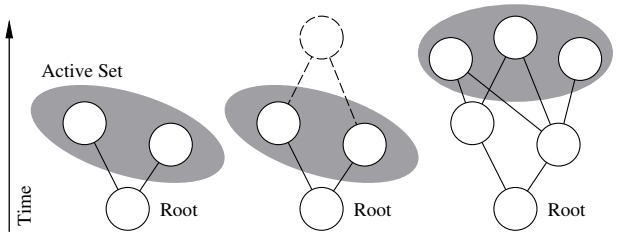


Fig. 6. Parallel Key Changes: Key inheritance hierarchy snapshots

Heartbeat When Distributed Flat is not run on top of a reliable multicast scheme, 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 III-C. The message format is detailed in Figure 7. Each participant who recently has created a key, will consider 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.

Key coordinate: KEK row/column or TEK;
Current Active Set: set of (K, P, M) tuples
(K=key selector, P=parent key selector set,
M=key material, encrypted with the keys from P);
Selected elements from the history in the same format;
Message authentication code using current TEK;
Message authentication code using an older TEK;

Fig. 7. Heartbeat Message Format

D.2.a 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 III-C), 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.

D.2.b 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.[2] 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 operation. 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). These additional key contributors can perform a simplified access control procedure if an access control token is supplied by the first introducer. Simplified pseudo-code can be found in Figure 8.

Although *admission control* issues are out of the scope of this paper, it can be noted that when connecting to a further participant to get some of the remaining keys, a token proving the successful previous admission can simplify this step.

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

Newcomer:
Wait for packets containing table geometry and participant addresses;
Select members sharing keys and contact them to obtain keys,
performing admission control on them;
Introducer:
Wait for request from newcomers;
Perform admission control and establish shared secret;
Send TEK and shared KEKs on an encrypted connection;

Fig. 8. Joining a Distributed Flat Group

D.2.c Leave. The leave operation works analogous to the description in Section III-C, 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 them new keying material which they are not supposed to know). Pseudo-code for this operation can be found in Figure 9.

Expeller:
Mark table entries known by expelled participant as forbidden;
Create new TEK and new KEKs for marked keys shared with the expelled;
Send out message with encrypted new keys, list of marked entries,
and updates to admission control (if necessary);
Key Holder receiving that message:
Relay message: Create new keys for KEKs owned and send them out;

Fig. 9. Leaving a Distributed Flat Group

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

E. Collusion and Recovery

In the Tree approach, expelling colluding participants does not involve more work than expelling the same number of non-colluding participants. In the Flat approaches, colluding members sharing their keys thus become immune to individual expels of members. To exclude supposedly colluding members, the union of their keying material has to be excluded at once. This union may also include unsuspecting participants who happen to share each of their individual KEKs with at least one of the "bad guys". In this section, we will analyze the impact of mass exclusion and present ways to reduce it.

We have also seen that two carefully chosen participants with complementary IDs know—if they co-operate—all the group's keys, and thus can only be expelled by re-creating the group, they become "resistant". One solution to this problem is to widen the matrix, thus increasing the minimal number of participants. Altough the schemes have been described in terms of bits, it can be generalized to *symbols* with any number of values V, e.g. by combining several bits into one symbol. For the same size ID, this will reduce the number of symbols W and thus the number of keys each participant will hold. At the same time, this will increase the number of keys a colluding group needs to hold

to V per symbol, requiring at least V conspirators with carefully chosen IDs to become resistant.

For a typical output of a hash function used for ID creation (128 bits), the matrix sizes may thus range from 2^1 by 128 to 2^{128} by 1 (width$\times$height).

For Centralized Flat, increasing V has the drawback that more storage is needed at the group manager (the participants are not affected). For Distributed Flat, storage is not increased, but increasing V will weaken the connectivity network, so more relay operations are needed to perform leave operations. Thus, V should be chosen based on estimates of the total group size and the expected number of colluding participants.

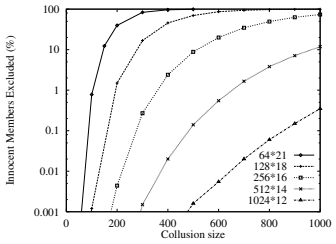


Fig. 10. Expelling Colluding Groups for Different Matrix Sizes (log scale)

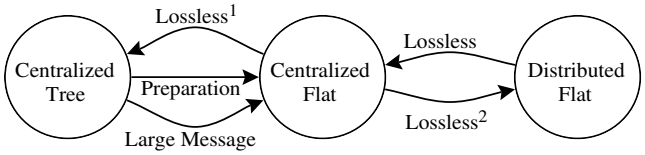
The probability for any legitimate member to be wrongly excluded when expelling large collusions is shown in Figure 10 on a log scale. The lines represent the behavior for different table sizes, represented by their size (V columns by W rows). They represent typical table sizes that can be achieved if the ID is the output of a 128 bit hash function.

As can be seen, even with modest table sizes, it is possible to exclude large amounts without major interference with the group operation. Setting the table width to the maximum collusion size keeps re-joins below 1%, increasing it slightly massively reduces re-joins. Slight paranoia, i.e. assuming collusions where there in fact aren't is thus a viable policy.

To further reduce the impact for the mis-excluded participants, the group manager (Centralized Flat) or admitting member (Distributed Flat) may include a token in its original response. This token may later be used to obtain the current versions and revisions of the TEK and the same KEK set again from the original introducer. Limited admission control needs to be performed, but only to check whether the admissability has been revoked. Such a token may also be used as the basis for an encryption key, to avoid another establishment of a shared secret. If the introducer does not want to keep state for each successful admission, the token can be constructed as follows: $H_S(\text{time} \mid \text{Peer ID} \mid \text{Group ID} \mid \text{Peer Network Address})$, where $\mid$ is the concatenation operator, H_S is a MAC keyed with S, a key only known by the introducer, the only state to be kept. The party wanting to be re-admitted can then submit the information used to build the token, authenticated by using the token as a key to a MAC covering the submitted data.

IV. Transitions

As we have seen, the three schemes discussed are closely related. Not only is it possible to have the schemes working together in a hybrid fashion (i.e. one part of the key space is managed by one scheme, while another, possibly overlapping, part is managed by another scheme), it is also possible to switch between them at run-time quite easily, adapting to the application's needs whenever required. Useful transitions are identified in Figure 11.



[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 cannot be easily expelled.

Fig. 11. Transitions Between the Three Schemes

A. Flat-Flat Transitions

Switching between the two flat schemes is simple, because they use the same data structure. This transition pair is therefore very attractive, allowing a heterogeneous approach combining the advantages of both schemes: Centralized Flat is used whenever possible to simplify the participants' operation.

To perform the switch towards Distributed Flat, the group manager notifies the group of the change, assists in electing the first set of key holders, and then forgets all the keys. Should the group manager be dysfunctional, the remainder of the group can agree on the transition and perform the election among themselves. For that, any election scheme can be used, such as [30].

To get towards Centralized Flat, a new group manager is appointed, which starts collecting all the keys from the current key holders and builds a complete table. The new group manager should be chosen carefully, since there will be no way to expel it short of re-creating the group.

B. Centralized-Centralized Transitions

The transitions between the two centralized schemes are somewhat more involved, as they require changes in the organization of the keys. To create a hierarchy from the flat table, apply the following scheme to each KEK in the newly-created hierarchy: The lowest-level (leaf) 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. All the participants similarily create the W tree keys they should know. This key calculation can be done lazily, only when a key is actually needed. As this transition obviously does not strengthen the system against previously unsuspected collusions, it is advisable to gradually replace all the auto-generated keys after the transition.

The transition from tree to centralized flat is easy, if keys for the flat structure are sent out when participants join the tree. Each participant thus gets a "sleeping" flat structure, which is stored until the transition takes place. While the transition is occurring, the participants combine each KEK in it with the current TEK (e.g. by hashing them or using exclusive-or), obtaining the new KEKs. This process is necessary to ensure that previously

expelled participants cannot sneak back into the group during the transition.

If a transition from Centralized Flat to Centralized Tree and back should be possible, the KEKs should be passed through a one-way function on the first change.

V. Evaluation

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

A. Offered Functionality

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

Multiple leaves Dealing with multiple leaves is more difficult in the approaches using flat datastructures. Having multiple invalidated fields causes the table to become sparse, thus the mechanisms of the Centralized Tree approach cannot 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 of VersaKey, but may also exclude a number of valid participants, which will need to re-join.

B. 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.

C. 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 bandwidth
- Typical end-to-end operation delay

Parts of VersaKey (especially, the Tree and Distributed Flat approaches) have been implemented, for specific measurements see Section V-D. In view of the simplicity of the presented architecture, a sound assessment of the involved costs can be made for all approaches. The upper bounds given as concrete values are so far confirmed by our implementation, and are appropriate for a Sun "Ultra 1/170" workstation. The following two tables, Table IV and Table V, 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.

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 revision increments and MAC computation, and IDEA for encryption operations. As can be seen in the 'Cost per Function' column, 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. Please note that computational costs of cryptographic functions as outlined here are worst case measurements. Hand optimized code and better performing platforms may offer significantly shorter processing times. Gains of a factor up to five have been observed.

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 additional 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 can 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. Because of the simplistic admission control used, the current implementation of VersaKey does not allow more than 20 joins per second. However, more joins are possible, if this admission control component is adequately enhanced.

In the case of the Distributed Flat approach, the costs of the Centralized Flat approach 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.

TABLE III
PROPERTIES OF THE VERSAKEY SCHEMES

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

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 resources, 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 consisting of $2W$ new encrypted keys each at 24 bytes — if we assume the key length to be 128 bits — to be sent, or about 1-2 kB. This message may need to be retransmitted in one of the reliable multicast implementations, increasing the participants 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.

D. Measured System Behavior

The following measurements cover the Tree approach of VersaKey. To perform the measurements, a small distributed environment was set up, incorporating our prototype. The implementation uses a growing tree structure, and lossless communication of key change data is assumed.

The depicted scenario consists of a group of 20000 participants, with one dedicated sender and group manager and two dedicated admission control machines. Admission control may be performed at a rate of 20 participants per second in total. This limit has been chosen by assuming that it requires an establishment of a shared secret using Diffie-Hellman key agreement. 25% (5000) of the participants are ready to join at the beginning of the test which runs for 7200 seconds. For each of these 7200 seconds, each non-member may initiate a join operation with a probability of 1%. At the same time, the group manager is excluding every participant with a probability of 0.1%, and 0.01% of the participants definitively leave the test setup in each second.

Most prominent in this scenario is the overload on the admission components (cf. Figure 12). For the first 30 minutes, admission is catching up with the 5000 participants that want to join from the beginning, and the one additional percent that comes in every second. Soon after admission control catches up, and no joins remain pending, leave and join actions balance each other out, due to the nature of the chosen scenario. Erosion of participants, by those that leave permanently becomes visible.

The amount of operations required by the group manager and the participants are significantly different. The depicted amount of operations per second stands for the number of atomic operations required due to leave and join operations. The peak of 700 operations per second for the server is caused by a peak of 30 leaving and 20 joining participants at the same time. Due to the essentially random leaving behavior in the experiment, the fictious client with id '0' that was chosen as reference point experienced peaks of up to 60 necessary operations per second. This happens when the amount of participants that leave from a closely related branch has disproportiate size. Otherwise the client load middles out nicely, on a level reflecting the logarithmic nature of this key management scheme.

The observed network peak traffic of approx. 1000 messages

TABLE IV
CPU USAGE — TREE

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 V
CPU USAGE — CENTRALIZED FLAT

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 $1 + W/2$

per minute, with a message size of 728 bytes, results in a load of below 100 kbit/sec on the entire group. This is a worst case scenario measurement both in terms of performed operations on the involved machines, and in terms of produced messages. Here, all joins and leaves were assumed to be alternating, requiring the maximum amount of work on the key tree, and no grouping of e.g. leave operations was performed. By grouping leaves into one operation per second, the average number of messages could be reduced by a factor of over 20, with an average message size of less than 4000 bytes, resulting in a net gain of a factor of four on the network load.

VI. CONCLUSIONS AND FURTHER WORK

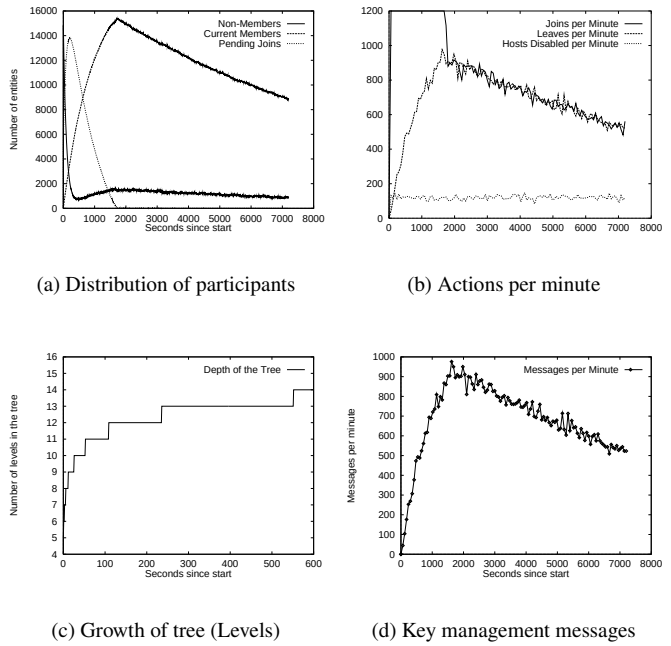
In this paper we presented the VersaKey middleware framework for secure multicasting. The core of the framework consists of three approaches which have different properties, but rely on the same basic principle. All our approaches organize the space of keys that will eventually be assigned to group members in a unique way, without actually generating the keys 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 user of the middleware 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 III 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 two preliminary implementations are available and working, we still lack experiments using real-world large, distributed groups; to this end, the integration of our experimental software into currently available IPsec platforms is planned. More specifically, one VersaKey key management approach is beeing joined with the successor of SKIP [15], to provide transparent security to



(a) Distribution of participants (b) Actions per minute

(c) Growth of tree (Levels) (d) Key management messages

Fig. 12. Measurements for the Tree Approach of VersaKey

group communications in the context of the Internet Protocol Suite. At the same time, efforts are going on to extend our approach of the continous consensus protocol used for reconcilation of key changes in distributed environments, and to develop a distributed scheme that is more collusion resistant.

While a detailed analysis on security issues can be found in the Appendix of [31], we believe this warrants further study. Enhanced and efficient admission control is a challenge on its own and requires further studies. Furthermore, we anticipate that batching of leave operations may be made more efficient with optimal grouping of the participants leaving within some time interval. Procedures on how to optimally allocate the IDs are under investigation.

ACKNOWLEDGEMENTS

We thank Roman Hofer for his ideas to simplify and unify key merging. We also thank the anonymous reviewers for their constructive criticisms, which helped improve the paper.

REFERENCES

[1] M.R. Macedonia and D.P. Brutzman, "MBone provides audio and video across the Internet," *IEEE Computer*, vol. 27, no. 4, pp. 30–36, April 1994.

[2] Angelo R. Bobak, *Distributed and multi-database systems*, Artech House, 1996.

[3] Andrew S. Grimshaw, Wm. A. Wulf, and the Legion team, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, no. 1, January 1997.

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

[5] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Adaptive packet marking for providing differentiated services in the internet," in *Proceedings of ICNP '98*, October 1998.

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

[7] M. Handley and J. Crowcroft, "Network text editor (NTE): A scalable shared text editor for the MBone," in *Proceedings of ACM SIGCOMM '97*, September 1997, pp. 197–208.

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

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

[10] S.E. Deering and D.R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

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

[12] R. Atkinson, "Security architecture for the Internet protocol," RFC 1825, August 1995.

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

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

[15] 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.

[16] D. Harkins and D. Carrel, "The internet key exchange (ike)," RFC2409, 1998.

[17] Douglas Maughan, Mark Schertler, Mark Schneider, and Jeff Turner, "Internet security association and key management protocol (ISAKMP)," Internet-Draft, March 1998.

[18] H.K. Orman, "The OAKLEY key determination protocol," http://www.ietf.org/internet-drafts/draft-ipsec-ietf-oakley-02.txt, 1997.

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

[20] 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*, 1987, pp. 185–193.

[21] Amos Fiat and Moni Naor, "Broadcast encryption," in *Advances in Cryptology: CRYPTO '93*, 1993, number 773 in Lecture Notes in Computer Science, pp. 480–491.

[22] G.H. Chiou and W.T. Chen, "Secure broadcasting using the secure lock," *IEEE Transactions on Software Engineering*, vol. 15, no. 8, pp. 929–934, August 1989.

[23] M. Burmester and Y.G. Desmedt, "Efficient and secure conference-key distribution," in *Security Protocols Workshop*, 1996, pp. 119–129.

[24] S. Mittra, "Iolus: A framework for scalable secure multicasting," in *Proceedings of ACM SIGCOMM '97*, September 1997, pp. 277–288.

[25] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee, "Key management for multicast: Issues and architectures," http://www.ietf.org/internet-drafts/draft-wallner-key-arch-01.txt, September 1998.

[26] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam, "Secure group communications using key graphs," in *Proceedings of ACM SIGCOMM '98*, September 1998.

[27] 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.

[28] Germano Caronni, Marcel Waldvogel, Dan Sun, and Bernhard Plattner, "Efficient security for large and dynamic multicast groups," in *Proceedings of the IEEE 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98)*, June 1998.

[29] Ran Canetti and Benny Pinkas, "A taxonomy of multicast security issues," http://www.ietf.org/internet-drafts/draft-canetti-secure-multicast-taxonomy-00.txt, Mai 1998.

[30] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers, "A secure and optimally eficient multi-authority election scheme," in *Proceedings of EUROCRYPT '97*, 1997, Lecture Notes in Computer Science, pp. 103–118.

[31] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, and Bernhard Plattner, "The versakey framework: Versatile group key management," TIK Technical Report TIK-57, TIK, ETH Zürich, September 1998, ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report57.ps.gz.