CITI Technical Report 01-9

Efficient Kerberized Multicast in a Practical Distributed Setting

Giovanni Di Crescenzo
Telcordia Technologies, Inc., NJ, USA
giovanni@research.telcordia.com

Olga Kornievskaia
Center for Information Technology Integration
aglo@citi.umich.edu

Abstract

Many of today's distributed applications are based on group communication. Given the openness of today's networks, communication among group members must be secure and, at the same time, efficient.
In this paper we consider a realistic distributed setting modeling general networks, including the Internet, that suggests the use of Kerberos, and, more specifically, a distributed mode of operation of Kerberos, called *crossrealm authentication protocol*, as a desirable choice for distributed applications.
We design an efficient protocol for secure group communication via multicast, using Kerberos. While developing the main components of our protocol, we construct an efficient paradigm for crossrealm authentication protocols that decreases communication over the Internet, makes most operations local, and reduces the workload of the involved components. We also design extensions of single-center multicast encryption schemes to multiple-center schemes. Our main protocol is obtained by combining these two contributions.

August 21, 2001

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

Efficient Kerberized Multicast in a Practical Distributed Setting

Giovanni Di Crescenzo
Telcordia Technologies, Inc., NJ, USA
giovanni@research.telcordia.com

Olga Kornievskaia
University of Michigan, Ann Arbor
Center for Information Technology Integration
aglo@citi.umich.edu

1 Introduction

Audio and video conferencing, data casting, and collaborative applications are among the next-generation applications emerging on the Internet that are based on group communication. When communication is conducted over a wide-area network such as the global Internet, security is crucial as messages traverse many links that are prone to attacks. Group members must have the means to communicate securely, in a private and authentic manner. While peer-to-peer security is a well-developed field, and several practical security protocols and primitives have been proposed, the problem of designing practical and secure group communication protocols is not as well explored.

In this paper we study, in a practical distributed setting motivated by real-life and developed applications, the problem of secure group communication. We now describe the setting we consider, the tools used and previous results, and finally describe our results.

Model considered. In this paper we consider security protocols for a practical distributed setting. Specifically, we consider several users that are divided into realms or Intranets, all of these being connected through the Internet. Moreover, in each Intranet there is a server or center responsible for several operations related to security management within its own Intranet. We assume that the communication through the Internet is not secured by default. This setting is sufficiently general, as almost any type of network can be characterized as described, and, yet, offers a useful structure in the form of a trusted server inside every Intranet.

Tools used and previous results. Two basic tools we use are: Kerberos protocol and multicast encryption schemes.

Kerberos, a widely used authentication mechanism, is a key distribution protocol that states how to share a secret between two participants. Crossrealm authentication is an extension to the original protocol that states how to deal with authentication in a distributed environment. One interesting feature of Kerberos is that almost all of its design is based on symmetric-key cryptography and does not suffer from computational overhead or open status of certain public-key cryptography techniques (as, for instance, digital signature revocation). Kerberos relies on the existence of a trusted server, which may limit its application to settings where this assumption is realistic. On the other hand, the setting we consider already satisfies this assumption. This, among other factors, has strongly motivated our choice of the Kerberos system. Although Kerberos (originally designed as a third party protocol) has extensions to distributed settings, we found out that the efficiency of some of these extensions can be questioned, thus leaving room for possible improvement.

Our results. In devising a practical and secure solution to our problem, we rule out the use of public-key crypto because of its enormous cost relative to symmetric-key crypto. Public-key techniques are suitable in a setting with no trusted centers. However, as our setting includes trusted centers, a combination of efficient extensions of Kerberos to a distributed environment and efficient extensions of multicast encryption schemes offers more practical solution of our problem.

Globally speaking, the contribution of our paper is the construction of a practical protocol for secure group communication via multicast using Kerberos.

In developing this solution, we have designed extensions to Kerberos and used some recently obtained extensions to multicast encryption schemes. In particular, we have designed a new practical paradigm for crossrealm Kerberos authentication that significantly decreases the amount of slow communication over the Internet, makes most operations local, and reduces the workload of the involved components. The scheme extends efficiently to handle multicast encryption. Moreover, we use some new extensions of multicast encryption schemes to the case of multiple centers, which is relevant in our setting.

Our main protocol is then obtained as a careful combination of the new paradigm for crossrealm Kerberos authentication with the schemes for multi-server multicast encryption. While performing this combination, we pay special attention to communication and storage complexity.

Organization of the paper. The rest of the paper is organized as follows. In Section 2, we present an overview of the Kerberos authentication protocol. In Section 3, we present an overview of crossrealm authentication in Kerberos. In Section 4, we describe multicast encryption schemes, where we recall the notion and basic solution approach and then show, in Sections 4.1 and 4.2, the two main paradigms of key management schemes for multicast encryption. Readers familiar with the background material should skip to Section 5, where we introduce an efficient crossrealm authentication protocol. In Section 6, we design extensions of known multicast encryption key management protocols to the environment of many multicast servers. In Section 7, we show how to obtain our protocol for secure group communication by carefully integrating the previous schemes, namely, the new crossrealm Kerberos authentication protocols and the new multi-server multicast encryption schemes.

2 Overview of Kerberos

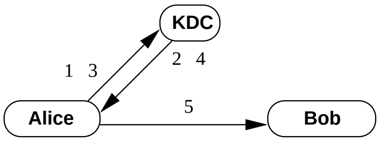
Kerberos [9] is a network authentication system based on the Needham-Schroeder protocol [8]. To avoid quadratic explosion of prior key agreement, Kerberos relies on a trusted third party, referred to as a Key Distribution Center (KDC). *Alice*, a Kerberos principal, and *Bob*, a Kerberized service, each establish shared secrets with the KDC.

Authentication is achieved when one party proves knowledge of a shared secret to another. Kerberos authentication process makes use of two special data structures: a ticket and an authenticator. A ticket is created by a KDC. It contains a secret key intended for two entities that will engage in authentication. The ticket and the secret key are given to a requestor. To authenticate the requestor creates an authenticator by using the secret key received from the KDC. Kerberos authentication is illustrated in Figure 1.

Let us consider an example, where *Alice* wishes to authenticate to *Bob*. At login, *Alice* receives a ticket granting ticket, TGT, from the KDC. She uses her password to retrieve a session key encrypted in the reply. The TGT allows *Alice* to obtain tickets from a Ticket Granting Service, TGS, for other Kerberized services. To access a Kerberized service, *Alice* authenticates to the TGS. She presents her TGT and constructs the authenticator, $\{\mathrm{T}\}_{K_{A,TGS}}$. If authentication is successful, *Alice* receives a service ticket, $\{Alice, Bob, \mathrm{K}_{A,B}\}_{K_B}$. To authenticate to *Bob*, *Alice* constructs an authenticator, $\{\mathrm{T}\}_{K_{A,B}}$, proving to *Bob* that she knows the session key inside of the service ticket.

3 Overview of crossrealm authentication in Kerberos

In this variant of Kerberos, there are several KDCs, one for each realm. In addition to assuming that there is a shared key between each user and their local KDC, this protocol assumes that there is a shared key between any two KDCs that participate in the crossrealm authentication. Even if the original Kerberos protocol doesn't specify how secret keys are established between participating KDCs, any of the solutions in the public-key cryptography literature for sharing keys among two parties can be used. An



LOGIN PHASE:	ONCE PER SESSION
1. *Alice* $\rightarrow$ *KDC*:	"Hi, I'm *Alice*"
2. *KDC* $\rightarrow$ *Alice*:	TGT = $\{Alice, \mathrm{TGS}, \mathrm{K}_{A,TGS}\}_{K_{TGS}}$, $\{\mathrm{K}_{A,TGS}, \mathrm{T}\}_{K_A}$
ACCESSING SERVICES:	EVERY TIME BEFORE TALKING TO A SERVICE
3. *Alice* $\rightarrow$ *TGS*:	*Alice*, *Bob*, TGT, $\{\mathrm{T}\}_{K_{A,TGS}}$
4. *TGS* $\rightarrow$ *Alice*:	TKT = $\{Alice, Bob, \mathrm{K}_{A,B}\}_{K_B}$, $\{\mathrm{K}_{A,B}, \mathrm{T}\}_{K_{A,TGS}}$
5. *Alice* $\rightarrow$ *Bob*:	"Hi, I'm *Alice*", TKT, $\{\mathrm{T}\}_{K_{A,B}}$

Figure 1: **Kerberos authentication.** Two phases are shown: initial authentication and service ticket acquisition. KDC is the Kerberos Key Distribution Center. TGS is the Ticket Granting Service. Most implementations combine these services. K_{TGS} is a key shared between the TGS and the KDC. K_A is a key shared between *Alice* and the KDC, derived from *Alice's* password. $\mathrm{K}_{A,TGS}$ is a session key for *Alice* and TGS. $\mathrm{K}_{A,B}$ is a session key for *Alice* and *Bob*. T is a timestamp used to prevent replay attacks. In each step, the encrypted timestamp serves the role of an authenticator. Note the version of Kerberos in this figure does not include an optional pre-authentication in the initial authentication. Furthermore, the steps shown describe only one way authentication, *Alice* to *Bob*. Mutual authentication is achieved with an additional message from *Bob* to *Alice*.

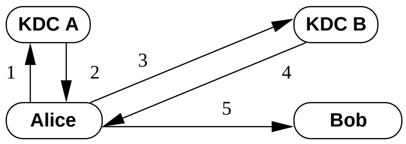
example of an use of such a solution can be found in an extension of Kerberos, called PKCROSS [5].

Once again, suppose *Alice* wishes to communicate with *Bob*. Crossrealm authentication proceeds as illustrated in Figure 2. As before, there is an initial login phase where *Alice* authenticates to her local KDC (this phase is not shown in the picture as it is identical to the original Kerberos protocol). *Alice* uses her TGT to request a remote ticket granting ticket (RTGT) for the remote realm. A local KDC Kerberos authenticates the request by verifying the validity of the ticket and the authenticator. The KDC constructs the RTGT and encrypts it with a key shared between the two KDCs; in our example, it's KDC A and KDC B. *Alice* presents the RTGT to KDC B and requests a service ticket for *Bob*. KDC B checks that RTGT is valid and that *Alice* included the corresponding authenticator, $\{\mathrm{T}\}_{K_{A,KDCB}}$. It proceeds by issuing a service ticket.

4 Multicast encryption with a single server

The word *multicast* is often associated with a variety of communication and networking topics; in this paper we consider *multicast encryption*. Informally, multicast encryption schemes are methods for performing secure communication among a group of users. In such schemes the group structure is dynamically changing, as a consequence of additions and deletions of members. Moreover, they are divided into two phases: an off-line initialization phase, in which a center communicates privately with each group member, and an on-line communication phase, in which the only communication means between center and group members is through a multicast channel, which the center (or any other party) uses to broadcast messages to all others.

Given that the communication is secured through symmetric-key encryption we consider the usual paradigm of sharing a random key among the group members. Once established, this *group key* is used to encrypt and decrypt messages exchanged among



1-2	Kerberos login
3. *Alice* $\rightarrow$ *KDC A*:	*Alice*, *Bob@B*, TGT, $\{\mathrm{T}\}_{K_{KDCA}}$
4. *KDC A* $\rightarrow$ *Alice*:	RTGT $= \{$*Alice@A*, $\mathrm{K}_{A,KDCB}\}_{K_{KDCA,KDCB}}$, $\{\mathrm{K}_{A,KDCB}, \mathrm{T}\}_{K_{A,KDCA}}$
5. *Alice* $\rightarrow$ *KDC B*:	*Alice@A*, *Bob@B*, RTGT, $\{\mathrm{T}\}_{K_{A,KDCB}}$
6. *KDC B* $\rightarrow$ *Alice*	TKT $= \{$*Alice@A*, *Bob@B*, $\mathrm{K}_{A,B}\}_{K_B}$, $\{\mathrm{K}_{A,B}, \mathrm{T}\}_{K_{A,KDCB}}$,
7. *Alice* $\rightarrow$ *Bob*:	"Hi, I'm *Alice@A*", TKT, $\{\mathrm{T}\}_{K_{A,B}}$

Figure 2: **Crossrealm authentication.** Initial Kerberos authentication (steps 1-2) is not shown in the figure. Also, a secret key, $K_{KDCA,KDCB}$, shared between the KDCs is assumed to be established prior to clients' requests. *Alice* authenticates with her local realm by presenting the TGT and constructing the authenticator for which she uses $K_{A,KDCA}$, key shared between *Alice* and the KDC A. *Alice* receives a remote ticket granting ticket, RTGT, encrypted with $K_{KDCA,KDCB}$. *Alice* proceeds to request a service ticket from the remote KDC. The last two steps of the protocol are equivalent to the last two steps of the original Kerberos.

members of the group. The problem is then reduced from a multicast encryption to a *key management problem*. The latter requires a scheme for managing keys among the group members in such a way that the following invariant is maintained: all group members have knowledge of the group key and all others have no information about it, where the group membership is not fixed, but varies through operations such as addition and deletions of users.

Key management protocols in the literature can be divided into two types: collaborative and non-collaborative. *Collaborative* key management refers to a method of managing the group key between the group members without any outside assistance; collaborative key management protocols have been proposed, for instance, in [10, 1, 6]. *Non-collaborative* key management, which we consider in this paper, refers to a method that relies on the existence of a *center* that is responsible for generating, distributing and updating the group key and other keys in the system.

In the literature, all non-collaborative key management architectures consider the scenario of a single center managing N users that comprise a multicast group. This center, which we call C, is responsible for managing keys and distributing them to users U_i, for $i = 1, \ldots, N$, both in an initial phase and while they are joining and leaving the group.

Joining a group is inexpensive if no backward secrecy is required. Typically, when a user wants to join a group, he contacts a center and receives a group key encrypted with a secret key established previously (like a password in Kerberos). If backward secrecy is required, then a new group key is created and distributed to all group members including the newly joined user. When a user wants to leave a group, the group key must be changed in order to satisfy the stated invariant. A center then generates a new group key and distributes it to the group. Multicast encryption schemes differ in how the key update operation is managed. In this paper we assume that backward secrecy is not needed and thus the join operation is simple. We mostly concentrate on describing what happens when a user leaves a group.

The multicast encryption schemes presented in the literature are compared based on several complexity measures, such as communication complexity, storage complexity both for the user and the center, and time complexity. Although all such complexity measures are important, given the type of applications multicast encryption schemes are usually employed for, it is often of crucial importance to design a scheme having small communication complexity (for instance, at most polylogarithmic in the number of users, for each update operation). For more extensive discussions, we refer the reader to [2, 3]. In the rest of this section we briefly review the two main families of multicast encryption schemes in the literature: minimal storage schemes and tree-based schemes. Their performance, in the various complexity measures, is summarized in the table in Figure 3.

4.1 Minimal storage schemes

A typical scheme in this family works as follows. Each user U_i holds two keys: a group key K_g and a unique secret key K_i shared between this user and the center. The center holds two keys: a group key K_g and a secret key l. This secret key is used to generate the keys for all the users in the group by combining it with a pseudo-random function f, for instance, as $K_i = f_l(i)$. We now describe how the center manages keys when users join or leave the group, the latter case being the non-trivial one.

When a user, say U_{N+1}, joins the group, the center just computes a key $K_{N+1} = f_l(N+1)$ and gives it to U_{N+1}; then the center privately sends the group key to U_{N+1} by first encrypting it using key K_{N+1}.

When a user leaves the group, the center chooses a new group key $K_{g'}$, and, for each user U_i who has not left the group, the center encrypts $K_{g'}$ using K_i as a key and transmits the result to U_i. The communication cost of this scheme is linear in the number of clients in the group. The storage cost of both user and center is constant.

4.2 Basic tree schemes

These schemes reduce the communication cost of key update to logarithmic but increase the storage cost. The center creates a balanced tree. For simplicity, we assume that for n members, $n = 2^t$ for some natural number t. A key is associated with each node in the tree. Leaf nodes are associated with individual keys known to a particular user. Each user is given all the keys associated with nodes on the path from the leaf node to the root. Thus, each user U_i holds $\log n + 1$ keys. The root node is considered to be a group key for the group since it is known to all the users. The center stores all the keys in the tree, requiring a storage cost of $2n + 1$ keys. We now describe how the center manages keys when users join or leave the group; again, the latter case is the non-trivial one.

When a user joins the group, the center adds a new leaf to the tree, associates a new key and the new member with the new leaf, and distributes all keys on the path from this leaf to the root to the new user.

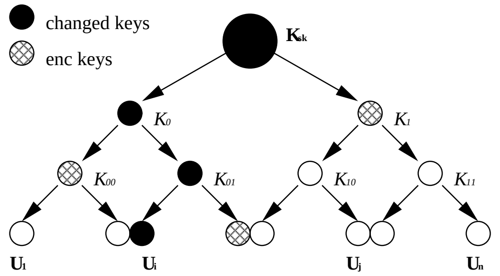
When a user leaves the group, the center changes all the keys on the path from the removed leaf to the root. The center generates $\log n$ new keys. Each of the keys is encrypted under the key of the unaffected sibling node on the path and multicasted to all users. Overall, only $\log n$ encrypted keys are sent.

Figure 3 shows an example of key assignments in the basic tree scheme for $N = 8$. In addition to 8 leaf nodes, there are 7 other keys in the system. User U_3 stores 4 keys: K_{010}, K_{01}, K_0, and K_{sk}. If user U_3 leaves, then keys K_{01}, K_0, and K_{sk} must change. The new key K_{01}' is encrypted with K_{011}. The new key K_0' is encrypted with K_{00} and K_{01}'. The new session key is generated next and encrypted with K_0' and K_1. This technique guarantees that new keys will be known only to the rest of the group and not to the removed user.

This scheme takes advantage of the fact that some keys are shared between multiple clients and are not on the affected path. The details of the update protocol are described in [2, 12, 13]. An improved binary tree scheme is described in [3], which generalizes from binary trees to a-ry trees and applies a minimal storage scheme on the groups of users of size m at the bottom of the tree. In the version of this scheme that achieves efficient communication $O(\log n)$ and user storage $O(\log n)$, the center storage is improved from linear to $O(n/\log n)$.

5 Efficient crossrealm authentication protocol

In Section 3, we showed how users from different Kerberos realms can authenticate each other. We note that the crossrealm authentication protocol that was discussed requires the user to contact



Evaluation parameters	**Minimal storage**	**Basic tree**	**Improved tree**
user storage	2	$\log n + 1$	$\log n$
center storage	2	$2n - 1$	$\frac{n}{\log n}$
communication	$n - 1$	$\log n$	$\log n$
computation	n	$\log n$	$\log n$

Figure 3: **Basic tree scheme.** The figure shows the tree created by the center in the basic tree scheme, for N = *8*. A highlighted path represents the affected path after a user deletion. Keys of the affected nodes are shaded. The table above compares three multicast encryption schemes.

the remote KDC before being able to contact the other user. As a consequence, in the (typical) situation in which each user is in her own Intranet and the two Intranets are connected through Internet links, the crossrealm authentication protocol introduces network delays associated with the traffic going through Internet links, which are potentially much slower than Intranet links. The *fake ticket protocol* described in this section reduces the network delay seen by the client.

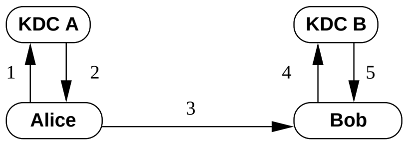
Our protocol maintains the same properties of the original crossrealm authentication protocol. Specifically, our protocol provides a mechanism for two parties to establish a secret key and authenticate one another in the process. For now we make an assumption that the participating KDCs share a secret key. The same assumption is made in the original crossrealm authentication protocol. Additionally, we assume that clients trust KDCs to make requests on their behalf. The KDC is a trusted entity (recall that this is a basic assumption of Kerberos itself), so this assumption is reasonable.

Figure 4 gives details of the protocol, including the flow of messages among the participants. First, we give intuition behind the protocol; then we go over each of the steps in detail. As in the original crossrealm protocol, *Alice* makes a request to her local KDC for a service ticket. The local KDC replies to the user with a service ticket for *Bob*. *Alice* continues and makes a request to *Bob*, as in the original Kerberos protocol.

Notice that in this protocol *Alice*'s local KDC can not issue a valid service ticket for *Bob*, as he is not in the same realm as *Alice*, therefore, *Alice*'s KDC does not know the shared key between *Bob* and *Bob's* local KDC. Instead, *Alice's* local KDC issues a *fake* ticket. When a KDC issues any kind of a ticket to an entity, the requestor has no way of verifying the validity of the ticket because the ticket is encrypted with a key that is not known to that entity. *Alice* can not know that the ticket she has is not encrypted with the proper key. *Alice's* KDC generates a session key to be used by *Alice* and *Bob* and gives this key to *Alice.*

Upon receiving a service ticket from *Alice*, *Bob* is unable to decrypt it because he doesn't share any secrets with *Alice* or *Alice's* KDC. So *Bob* contacts his local KDC and requests verification of the ticket. The KDC extracts the session key from the ticket and returns it to *Bob.*

In step 1, *Alice* makes a request for a service ticket



1. *Alice* $\rightarrow$ *KDC A*: *Alice*, *Bob@B*, TGT, $\{T\}_{K_{A,KDCA}}$
2. *KDC A* $\rightarrow$ *Alice*: FTKT = $\{$*Alice@A*, *Bob@B*, $K_{A,B}\}_{K_{KDCA,KDCB}}$, $\{K_{A,B}, T\}_{K_{A,KDCA}}$
3. *Alice* $\rightarrow$ *Bob* : "Hi, I'm *Alice@A*", FTKT, $\{T\}_{K_{A,B}}$
4. *Bob* $\rightarrow$ *KDC B*: FTKT, TGT, $\{T\}_{K_{B,KDCB}}$
5. *KDC B* $\rightarrow$ *Bob*: TKT = $\{$*Alice@A*, *Bob@B*, $K_{A,B}\}_{K_B}$, $\{T\}_{K_{B,KDCB}}$

Figure 4: **Fake ticket protocol** The figure shows the flow of messages between participants in the protocol.*Alice* authenticates to the KDC A by presenting the TGT and creating the corresponding authenticator. KDC A generates a fake service ticket, (FTKT), encrypts the session key with the key it shares with *Alice*. She authenticates to *Bob* with the ticket, (TKT), and the authenticator. *Bob* forwards to the ticket to the KDC B which after authenticating *Bob* by validating his TGT and the included authenticator creates a real service ticket.

from her local KDC. The format of the message is as in previous protocols. The local KDC authenticates the client and generates a session key to be used by *Alice* and *Bob*. It encrypts the session key with the key it shares with *Alice*. Also it generates a ticket and sends the reply to *Alice*. The ticket is encrypted with a key that is shared between *Alice's* KDC and *Bob's* KDC, $K_{KDCA,KDCB}$.

In step 3, upon receiving the reply from her KDC, *Alice* creates a request to *Bob* that is identical to the last step of the Kerberos protocol (see Figure1). The request contains the service ticket for *Bob* that *Alice* has just received and the authenticator, $\{T\}_{K_{A,B}}$, proving to *Bob* that she knows the session key inside of the service ticket. The request is marked as crossrealm authentication request.

In step 4, when the server gets the message that contains this special type of a ticket, *Bob* forwards the ticket to his local KDC B. *Bob* authenticates to the KDC B by presenting his ticket and the authenticator that proves that *Bob* knows the session key that goes with the ticket.

In step 5, after successful authentication of *Bob*, KDC B decrypts the ticket and retrieves the session key. KDC B also checks that the service ticket was intended for *Bob* before revealing the session key inside of the ticket. Furthermore, it checks that the ticket is still valid by checking the expiration time. KDC B creates a real service ticket for *Alice* and *Bob* and returns this ticket to *Bob*.

Security analysis. The proposed protocol is a variation of the original crossrealm protocol. Thus, if the original protocol is secure then so is the *fake ticket* protocol. At each step of the protocol the sender (and optionally the receiver) is authenticated with the corresponding tickets and authenticators. Furthermore, in order to prevent *Charlie* from capturing a *fake* ticket and asking the KDC to reveal the session key inside of it, we require that the party authenticated in step 4 of the protocol be the same principal for which the *fake* ticket was issued.

5.1 Comparison of protocols

In the previous sections, we presented two crossrealm authentication protocols. One was previously known and one was introduced in this paper. Both protocols achieve the goal of establishing a secret key between communicating parties and authenticating each other in a setting where parties are divided into realms. Each realm has a server responsible for various operations within a realm, such as key management.

This section looks at how the protocols introduced in this paper compare to the protocol previously introduced in the literature. We consider the following comparison criteria: communication delays, software compatibility, workload at the client side, workload at the KDCs (local and remote) for each of the protocols, and efficient multicast applicability.

5.1.1 Communication delays

To compare communication cost, we identify two types of messages: Intranet (fast) and Internet (slow). Overall, we observe that the *fake ticket* protocol is the most favorable. Each execution of the original crossrealm protocol requires three Internet messages, while the *fake ticket* protocol requires only one. Intranet messages are assumed to be fast, so the difference in their numbers (in our case two messages) should not affect overall performance.

5.1.2 Software compatibility

Each of the crossrealm protocols requires certain modifications to Kerberos. To support crossrealm operations in Kerberos, client software needs to modified to handle communication with the remote KDC. The *fake ticket* protocol doesn't requires client side modifications. In practice, it is very important that the client side software remains unchanged.

Each of the protocols requires modifications to the KDC. In the original crossrealm proposal, the remote KDC must recognize remote ticket granting tickets. In practice, this is not a major modification. In the *fake ticket* protocol the ability to generate fake tickets is required. Furthermore, the support for converting fake tickets into regular tickets is needed. In practice, the support for additional functionality is implemented in terms of new services that run on the same machine as the KDC and thus do not require modification to already existing code.

In addition to these modifications, the *fake ticket* protocol requires modifications to the server software.

5.1.3 Client workload

We measure the workload of a component in terms of the number of cryptographic operations and network messages. The original crossrealm protocol produces the biggest client workload. If the client machine is not powerful enough to do encryption operations, then overall performance of the system degrades. In the original crossrealm protocol the client is required to perform seven cryptographic operations (encryption/decryption) and process five messages. The *fake ticket* protocol requires four cryptographic operations and three messages.

5.1.4 KDC workload

In case of the remote KDC, all protocols produce the same number of encryptions, which is not surprising as the remote KDC is the only one that can generate a valid service ticket for the server. This operation takes three encryptions. The request for a service ticket is authenticated by a ticket and a nonce. In order to decrypt a ticket and a nonce, a KDC performs two decryption operations.

5.1.5 Efficient multicast applicability

As it will become clear in Section 7, the *fake ticket* protocol can be extended to allow multicast encryption by preserving its efficiency in all parameters. The original crossrealm protocol can not be easily combined with multicast encryption to produce a protocol that is reasonably efficient.

6 Multicast encryption with many servers

As we pointed out in Section 3, the solution that assumes a single server responsible for all the clients doesn't scale to the environments such as an Internet connecting several Intranets (as well as the Internet itself). The previously reviewed schemes for multicast encryption consider only a single center. In this section, we discuss several protocols for the more realistic case where there are multiple centers, each managing a group of users. In Table 1 we summarize performance characteristics of each of the schemes.

Imagine a dynamically-changing group of n users (namely, with users being added to and removed from the group); as before each client is denoted as U_i. A group is broken down into smaller groups of size m. Each such group is managed by a center, denoted as C_j.

The security requirements that a *multi-server secure multicast encryption scheme* should satisfy can be defined essentially in the same way as those of a single-server scheme. The only difference is that in the various operations of a multi-server protocol, the users may potentially interact with many servers (rather with only one), and the servers may have to coordinate in order to perform all necessary operations. As a result, users managed by different centers can belong to the same group.

A straight forward approach in constructing a multi-server multicast encryption scheme, given a single-server one, elects one out of the many servers as a principal server and then uses it as the server when executing the single-server scheme. The inconvenience of such a scheme is that for every operation, each client must communicate to a server in a specific Intranet, which, in general, requires Internet messages to be sent, degrading overall performance.

Instead, we would like multi-server schemes to satisfy some basic efficiency requirements. First, we require every user to communicate only to the *local* server in the same Intranet during the various management operations of the scheme. Moreover, we require both the total communication among the servers and the storage complexity of all servers to be minimized. This list of requirements is not exhaustive: it is not hard to think of other efficiency requirements that might be necessary for certain specific applications.

6.1 A simple construction

A simple construction groups all users into a single binary tree, as in the single server tree based schemes, and then requires that the same tree data structure is maintained by each of the centers. Addition of a user requires the user to communicate only with its local center, who later communicates the join to the other centers. Similarly, deletion of a user is handled locally by the center associated with the departing user, as for a single center scheme, and later the center informs all other centers of the deletion.

Although obtaining locality of key updates (the communication for each update is efficient, as it is logarithmic in n), this solution is inefficient in terms of center storage. Each center is required to store all the client keys, including the keys of the clients it doesn't manage. The total number of keys stored among all the centers for a group is $(2n-1)\cdot\frac{n}{m}$, assuming there are $\frac{n}{m}$ centers, each managing m out of n clients.

6.2 A second construction, extending tree-based schemes

To improve on the previous scheme, we observe that each of the centers doesn't need to keep the information about the users that don't belong to the same Intranet as that center. So if each center manages m users, then each of the centers applies the single center binary tree scheme to store information about their users only. As a consequence, each user stores only $\log m+2$ keys; a center stores the keys in the tree for m clients. The root of each tree is a key shared by m users and not all n users. Thus each client additionally stores a session key for the group, K_{sk}. Each of the centers stores a secret key, K_{mk}, not known to the clients, that is used to distribute a new session key.

When a user leaves, his local center creates a new session key for the whole group. Distribution of the session key to the local users is done by applying the single center tree-based protocol. The shared key among the centers is not affected, so the center uses it to encrypt the new session key and broadcasts it to the rest of the centers. Upon receiving the message and decrypting it, each of the centers encrypts with the corresponding root key for the local tree and broadcasts it. The update operation takes $\log m+\frac{n}{m}$ messages. The total storage among all centers is $2(m+1)\frac{n}{m}$.

We note that this scheme improves over the previously described scheme both in center and in user storage, but has a more expensive communication complexity for each key update.

Parameters	Replicated tree	Master key	Weighted tree
User storage	$\log n + 1$	$\log m + 1$	$O(\log n)$
Center storage	$2n - 1$	$2m$	$O(m + \log \frac{n}{m})$
Communication	$\frac{n}{m} \log n$	$\log m + \frac{n}{m}$	$O(\log n)$

Table 1: **Multi-server multicast encryption schemes.** The table summarizes the complexity values for each of the multi-server multicast encryption schemes. Values for center storage are of a single center. To get the total center storage among all the centers each of the values need to by multiplied by number of centers, assumed for the first two case to be $\frac{n}{m}$.

6.3 A third construction, based on coding theory algorithms

One possible extension of the previous scheme is to create a better data structure for the keys shared among the servers. Specifically, a binary tree having centers as leaf nodes could help. Note that the resulting data structure of the entire system contains several binary trees, each in a different Intranet, built over the users in that Intranet and having a center as a root; and, moreover, one binary tree over each of the centers. Overall, the data structure is in fact a single binary tree; however, unlike the case in which a binary tree is obtained by applying a single center tree-based scheme, here the tree can be unbalanced if the relative numbers of users in each Intranet differ significantly from each other.

Consequently, the problem of finding the optimal balancing, given as information the number of users in each Intranet, naturally rises as a way to minimize the average number of keys to be changed in the next update operation (assuming that the next user leaving is random). Interestingly, this problem can be seen to be equivalent to the well-known coding theory problem of finding codes for variable-length sources that minimize the average codeword length. A classical algorithm constructing such codes, based on the is due to Huffman (see, e.g., [7]). This algorithm, given their frequencies, constructs a tree which minimizes the average length of the codewords.

To construct an optimal multicast tree for our scheme, we first construct a tree in each Intranet using a single center tree-based scheme. Then, each center is associated with a codeword. The number of users in that Intranet divided by the number of total users is considered as the frequency of that codeword. We run Huffman's algorithm on these inputs and obtain a minimal tree whose structure shared among all clients and all servers and added to all trees previously constructed for each Intranet. This gives rise to a single, global tree.

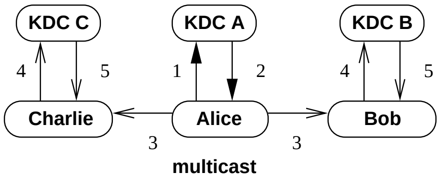
Addition and deletion of users is handled as in the single center tree-based scheme, but on the global tree.

We note that in this scheme each user is given at most $\log n$ keys; some come from the tree constructed for that Intranet, and some from the execution of Huffman's algorithm, the relative sizes according to how many users are in the same Intranet. The communication per update is now only logarithmic in n, and is, in fact, optimal assuming that the next deletion is from a randomly chosen user. The total center storage is also efficient, although the exact expression depends on the relative numbers of users in each Intranet; in the case when all Intranets have the same number m of users, each server stores only $\log(\frac{n}{m}) + m$ keys.

7 Integrating Kerberos with multicast encryption schemes

In this section we combine two parts of this paper into a single protocol for secure group communication via multicast using Kerberos-based protocols. This protocol is designed into the practical distributed setting considered in this paper; namely, users are divided into realms or Intranets, which are connected through the Internet.

We show how to combine both of the crossrealm protocols with any of the discussed multi-server multicast encryption schemes. For simplicity of description, we discuss only tree-based multi-server multi-



1. *Alice* $\rightarrow$ *KDC A*:	*Alice*, *Group*, TGT, $\{\mathrm{T}\}_{K_{A,KDCA}}$
2. *KDC A* $\rightarrow$ *Alice*:	GTKT = $\{Alice@A,\ Group,\ \mathrm{K}_{gk}\}_{K_{mk}}$,
	$\{\mathrm{K}_{gk},\ \mathrm{K}^{A}_{help},\ \mathrm{T}\}_{K_{A,KDCA}}$,
3. *Alice* $\rightarrow$ *Group*:	"Hi, I'm *Alice*", GTKT, $\{\mathrm{T}\}_{K_{gk}}$
4. *Bob* $\rightarrow$ *KDC B*:	GTKT, TGT, $\{\mathrm{T}\}_{K_{B,KDCB}}$
4. *Charlie* $\rightarrow$ *KDC C*:	GTKT, TGT, $\{\mathrm{T}\}_{K_{C,KDCC}}$
5. *KDC B* $\rightarrow$ *Bob*:	GTKT = $\{Alice@A,\ Group,\ \mathrm{K}_{gk},\ \mathrm{K}^{B}_{help}\}_{K_B}$, $\{\mathrm{T}\}_{K_{B,KDCB}}$
5. *KDC C* $\rightarrow$ *Charlie*:	GTKT = $\{Alice@A,\ Group,\ \mathrm{K}_{gk},\ \mathrm{K}^{C}_{help}\}_{K_C}$, $\{\mathrm{T}\}_{K_{C,KDCC}}$

Figure 5: **Kerberized multicast encryption protocol**

cast encryption schemes. To combine the protocols we note that:

- the key $K_{A,KDCA}$ in the crossrealm protocols and the key in the multicast encryption scheme shared between *Alice* and the server can be the same key;
- the key $K_{A,B}$ shared between *Alice* and *Bob* can be replaced by the group key that is shared during the execution of the (key-distribution phase of the) multicast encryption scheme;

Each of the proposed algorithms has the following properties:

- Allows for secure group key distribution to participants in the group. We consider a *join by invitation* where the initiator of the group contacts (invites) other entities to participate in a group communication.
- Achieves sender authentication in a group membership setting, which allows this algorithm to be used for the authenticated client join operation. To achieve sender authentication, the join algorithm described below can be easily modified.

7.1 Kerberized multicast with fake tickets

We add the requirement that a key is shared among all centers in a preliminary stage. Note that before, analogously, a key was shared between KDC A and KDC B; this key was computed using public-key cryptography techniques (e.g., the two-party Diffie-Hellman key-exchange [4]). For the above protocol we can use any extension of these techniques to more than two parties (e.g., the n-party generalized Diffie-Hellman key-exchange [11]

Figure 5 shows how *Alice* can send a message to the *Group*, which in this example includes *Bob* and *Charlie*, according to this protocol.

In step 1, *Alice* authenticates to her local KDC and requests a ticket for the multicast group, denoted *Group* in the Figure 5. *Alice* presents her Kerberos credentials: TGT and authenticator.

In step 2, *Alice's* KDC creates a group key, K_{gk}, and a group ticket, $\{Alice@A,\ Group,\ \mathrm{K}_{gk}\}_{K_{mk}}$. K_{mk} represents a key shared between the KDCs. KDC A also generates other multicast group keys, K_{help} for *Alice*. KDC A, as well as other KDCs, need to know membership information. One solution could be for *Alice* to include the information in the request. Then this membership information would be included in the group ticket; thus, other KDCs would have access to the membership list and be able to generate the necessary multicast keys for other members.

In step 3, *Alice* retrieves the group key from the reply and creates the authenticator using that key. She multicasts the message to the group. The message includes the group ticket and the corresponding authenticator.

In step 4, each of the members contacts their local KDCs as in the *fake* ticket protocol in order to validate and interpret the group ticket. All the requests are Kerberos authenticated. Each of the KDCs generates other multicast group keys from the group key if needed and converts *fake* tickets into regular tickets, by reencrypting the information with the appropriate keys.

In the last step, each of the participants receives the needed multicast keys and is assured that the message came from *Alice*.

Note that the protocol works regardless of whether the group members belong to the same realm as *Alice* or multiple realms, as in the example shown in Figure 5.

7.2 Integration with original crossrealm

The original crossrealm protocol fails to provide the same efficiency when used to secure group communication. Figure 6 shows how *Alice* can send a message to the *Group* according to this protocol. As in the previous example, the group consists of *Bob* and *Charlie*.

In step 1, *Alice* authenticates to her local KDC and request a ticket for the multicast group, simply named *Group* in the Figure. *Alice* presents her Kerberos credentials: TGT and authenticator. The request could include the membership list or *Alice* could include it in the request. From this list, the KDC infers which of the remote KDC are involved.

In step 2, KDC A first generates a group key, K_{gk}. Then it issues to *Alice* needed RTGTs. In our example, *Alice* receives remote ticket granting tickets for KDC B and KDC C. Each of the remote ticket granting tickets would need to include a newly created session key. Additionally, KDC A generates all other group keys required (which depend on the particular multicast encryption scheme used).

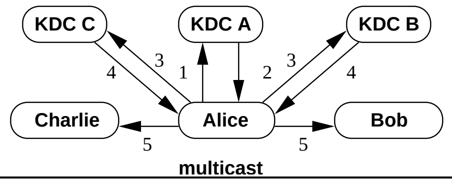
In step 3, *Alice* performs crossrealm authentication to all the remote KDCs. In each remote authentication request, *Alice* requests a ticket for a member of the group that belongs to the realm. Each of the remote KDCs authenticates *Alice* by checking the RTGT and the corresponding authenticator. Each of the KDCs retrieves a group key from the RTGT and creates the requested ticket with that group key. For example a ticket to *Bob* would be $\{Alice@A,\ Bob@B,\ \mathrm{K}_{gk}\}_{K_B}$. Also, for each member K_{help} multicast group keys are generated and included in the ticket.

In step 4, *Alice* awaits the replies from the remote KDCs. She checks the authenticators by validating the timestamps of the replies.

In step 5, *Alice* prepares the message to the group. The message includes all the tickets she received in the last step. *Alice* creates the authenticator, $\{\mathrm{T}\}_{K_{gk}}$. She needs only one authenticator for all the tickets because the secret key inside each of the tickets is the same, namely K_{gk}. Each of the recipients can validate the ticket and the authenticator, then retrieve the group key from the ticket.

7.3 Discussion

Integration with the original crossrealm authentication protocol requires the client to contact all of the remote KDCs involved; i.e., $2 \cdot m$ message, where m is the number of KDCs. Furthermore, a join message multicast by *Alice* to all members is linear in size because it includes a ticket for each of the members. The use of *fake* tickets allows us to multicast a message with only a single ticket. *Alice* is not required to make any (potentially time consuming) requests. In terms of network delay, in the first protocol we have to account for $2 \cdot n$ Intranet messages. Due to the network latency of Internet message we assume that $n \cdot \alpha \leq m \cdot \beta$, where α is the network delay within an Intranet and β is the network delay on the Internet, while $m \leq n$.



1. *Alice* $\rightarrow$ *KDC A*:	*Alice*, *Group*, TGT = $\{Alice, \mathrm{K}_{A,KDCA}\}_{K_{KDCA}}$, $\{\mathrm{T}\}_{K_{A,KDCA}}$
2. *KDC A* $\rightarrow$ *Alice*:	RTGT1 = $\{Alice@A, \mathrm{K}_{gk}, \mathrm{K}_{A,KDCB}\}_{K_{KDCA,KDCB}}$, $\{\mathrm{K}_{A,KDCB}\}_{K_{A,KDCA}}$, RTGT2 = $\{Alice@A, \mathrm{K}_{gk}, \mathrm{K}_{A,KDCC}\}_{K_{KDCA,KDCC}}$, $\{\mathrm{K}_{A,KDCC}\}_{K_{A,KDCA}}$, $\{\mathrm{K}_{gk}, K^{A}_{help}, \mathrm{T}\}_{K_{A,KDCA}}$
3. *Alice* $\rightarrow$ *KDC B*:	*Alice@A*, *Bob@B*, RTGT1, $\{\mathrm{T}\}_{K_{A,KDCB}}$
3. *Alice* $\rightarrow$ *KDC C*:	*Alice@A*, *Charlie@C*, RTGT2, $\{\mathrm{T}\}_{K_{A,KDCC}}$
4. *KDC B* $\rightarrow$ *Alice*	TKT1 = $\{Alice@A, Bob@B, \mathrm{K}_{gk}, \mathrm{K}^{B}_{help}\}_{K_B}$, $\{\mathrm{T}\}_{K_{A,KDCB}}$
4. *KDC C* $\rightarrow$ *Alice*	TKT2 = $\{Alice@A, Charlie@C, \mathrm{K}_{gk}, \mathrm{K}^{C}_{help}\}_{K_C}$, $\{\mathrm{T}\}_{K_{A,KDCC}}$
5. *Alice* $\rightarrow$ *Group*:	"Hi, I'm *Alice*", TKT1, TKT2 $\{\mathrm{T}\}_{K_{gk}}$

Figure 6: **Crossrealm multicast encryption protocol**

8 Conclusion

In this paper we propose an efficient protocol for secure group communication via multicast, using Kerberos. We consider a practical distributed setting where users reside in different Intranets (realms) connected through the Internet. This setting is sufficiently general as almost any type of network can be characterized as described, and, yet, offers useful structure especially the presence of a trusted server inside every Intranet. We considered an extension to the original Kerberos protocol that enables crossrealm operations, identified its inefficiencies, and proposed a new paradigm for crossrealm Kerberos authentication that significantly decreases slow communications over the Internet, makes most operations local, and reduces the workload of the involved components. Furthermore, the paper describes new extensions of multicast encryption schemes to the case of multiple centers. Finally, we combine the new crossrealm protocol with those for the multiserver multicast encryption.

9 Acknowledgments

We thank the anonymous reviewers for their helpful comments. We also thank Peter Honeyman for his valuable comments.

References

[1] K. Becker and U. Wille. Communication complexity of group key distribution. In *Proceedings of the 5th ACM Conference on Computer and Communication Security*, pages 1–6, San Francisco, CA, November 1998.

[2] R. Canetti, J. Garay, G. Itkis, D. Miccianco, M. Naor, and B. Pinkas. Multicast security: A taxonomy and efficient authentication. In *IEEE INFOCOMM*, 1999.

[3] R. Canetti, T. Malkin, and K. Nissim. Efficient communication storage tradeoffs for multicast encryption. In *Proceedings of "Advances in Cryptology – EUROCRYPT'99", Lecture Notes in Computer Science*, Springer Verlag, 1999.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transaction on Information Theory*, 22:644–654, November 1976.

[5] M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld. Public key cryptography for cross-realm authentication in kerberos, May 2001. Internet draft.

[6] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the*

7th ACM Conference on Computer and Communication Security, CCS'00, pages 235–244, November 2000.

[7] F. MacWilliams and N. Sloane. *The theory of error-correcting codes.* Elsevier Science, 1977.

[8] R. Needham and M. Shroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993 – 999, December 1978.

[9] C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[10] A. Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *CryptTEC*, 1999.

[11] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to groups. In *Proceedings of the 3rd ACM Conference on Computer and Communications in Security, CCS'96*, pages 31–37, March 1996.

[12] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures, June 1999. RFC 2627.

[13] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In *Proceedings of the ACM SIGCOMM'98*, pages 68–79, September 1998.