Policy-Based Security Management for Large Dynamic Groups: An Overview of the DCCM Project[1]

Peter T. Dinsmore, David M. Balenson, Michael Heyman, Peter S. Kruus,
Caroline D. Scace, and Alan T. Sherman[2]
Cryptographic Technologies Group
NAI Labs, The Security Research Division
Network Associates, Inc.
{peter_dinsmore, david_balenson, mheyman, pkruus, cscace, asherman}@nai.com

Abstract

The Dynamic Cryptographic Context Management (DCCM) project efficiently provides security for very large, dynamically changing groups of participants. The DCCM system has two novel distinguishing characteristics. First, policy plays a key role in DCCM. Groups at all levels have policies. These policies are represented; they are negotiated; they are managed; and a cryptographic context—an unambiguous set of mechanisms and configuration—is created to make particular interactions possible subject to these policies. Second, DCCM implements a scalable key management system based on One-way Function Trees (OFT) that can handle group sizes up to 100,000 members and can dynamically handle members entering and leaving groups.

1. Introduction

The Dynamic Cryptographic Context Management (DCCM) project [2] efficiently provides security for very large, dynamically changing groups of participants. For example, command and control of tactical military forces requires several types of protection among a very large group of participants, perhaps from different countries or from different Armed Forces units, grouped together under one command for some time period or for a specific exercise. By "large," we mean groups with number of members ranging from 10,000 to 100,000 or more. By "dynamic," we mean new members may be added to the group at any time and existing members may be evicted from the group, thereby requiring immediate changes to some of the security provisions. Members need not be humans; they can be a variety of communicating entities, including sensors, mobile client workstations, server workstations, or network nodes.

The DCCM system has two novel distinguishing characteristics. First, policy plays a key role in DCCM. Groups at all levels have policies. These policies are represented; they are negotiated; they are managed; and a *cryptographic context*—an unambiguous set of mechanisms and configuration—is created to make particular interactions possible subject to these policies. Second, DCCM has a scalable key management system that can handle group sizes up to 100,000 members and can dynamically handle members entering and leaving groups.

This paper presents:

- The DCCM view of a secure group: its organization; lifecycle; and operations;
- Secure group policy and a Cryptographic Context Negotiation Template to represent it;
- Policy negotiation via a Cryptographic Context Negotiation Protocol;
- The One-way Function Tree (OFT) method for large group key management; and
- The architecture of a demonstration system incorporating the DCCM concepts.

2. Secure groups

A secure group is a collection of members who are authorized to access a set of information. Secure group mechanisms enable the members, and only the

[1] This work was supported by the Defense Advance Research Projects Agency (DARPA) under Air Force Research Laboratory (AFRL) Contract No. F30602-97-C-0277.

[2] Sherman is also with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County (UMBC).

members, to access the information. DCCM supports group policies that require group confidentiality and/or group integrity of the information.

Group confidentiality protects the group information from disclosure to non-group members. In order to provide confidentiality in a dynamically changing group, security mechanisms must provide forward and backward secrecy. Forward secrecy means that evicted members cannot access future information. Backward secrecy means that new members cannot access past information.

Group integrity protects information from modification by non-group members. Members of a secure group can verify that information has not been modified by anyone outside of the group, and that it originated from an authorized group member. Group integrity does not allow a member to determine which individual group member created a message, *i.e.,* individual source authentication.

DCCM does not provide mechanisms to support individual source authentication of information or to maintain the availability of information.

2.1. Group organization

DCCM recognizes and supports a range of models for the organization of a secure group, from a strict hierarchical model to a broad flat model. A hierarchical group is based on inherent structure of the members forming a group. This structure is typically found in groups created for organizational collaboration, such as an industrial consortium or a military coalition. The security policies and supporting infrastructure for the group also take advantage of existing structure and relationships between members. For example, there may be multiple authentication policies and servers for a group, based on the existing policies within the hierarchy or across multiple hierarchies.

A flat group has no group structure or relationships between group members. It is typically used for individual collaboration or information distribution, such as web broadcasts or pay-per-view. DCCM allows each member to have its own security policy, or to subscribe to a known security policy. In a flat model, certain members may advertise a policy designed to meet certain goals that other members with similar goals can use as well.

The most basic entity in a DCCM secure group is the *participant.* A participant is a single entity that is involved in secure group communication. Typically a participant represents a person, but it can represent a sensor, a Personal Digital Assistant (PDA), or a piece of software. DCCM supports all security services for all participants, and assumes that all participants have the same group communication capabilities. DCCM fully supports the many-to-many model of group communications, meaning all participants can both send and receive information. DCCM can also support the one-to-many model (one sender with many receivers) within its support of the many-to-many model.

A group of participants sharing a secure group communication mechanism for a specified period of time for a common purpose is a *session.* A session implies that all of the participants share the same security mechanisms, share the same security policy for those mechanisms, and share a common security configuration, including cryptographic keys, for the enforcement of the policy. For example, a single lecture broadcast from a semester long class would be a session.

A DCCM *project* is a set of sessions occurring over time. All of the sessions within a project will use the same cryptographic context, or policy, and support the same set of participants. The project is the unit of administration for access control for DCCM. DCCM administers the list of participants in a project and enforces an access control policy between project members and non-members. Within a project, any project participant is free to participate in any session announced for the project. All project members are authorized for all sessions within that project. Project members may choose not to participate, but that is not a security relevant decision. Multiple sessions can occur simultaneously within a project, and a participant can join more than one of these sessions.

The highest level of organization of secure groups in DCCM is a *system.* A system is the supporting infrastructure for a set of related participants that transcends individual projects. The system maintains a single authentication database that is used across multiple projects. The DCCM authentication mechanism utilizes a system base key, or shared secret, that is established when a participant joins the system and authenticates for the first time, typically with a public key mechanism. DCCM amortizes the high cost of the public key authentication over the creation of multiple projects.

2.2. Group lifecycle

All groups go through several phases during their life. The first phase of DCCM, even before any groups are formed, is the induction of participants into a DCCM system. As previously noted, participants authenticate themselves once to a DCCM system and establish a shared key that is used for all future secure operations.

During the next phase, a DCCM project is created from the set of participants authenticated to the system. One of the participants takes on the role of project initiator. The project initiator specifies a list of participants and a proposed security policy for the project. The DCCM system infrastructure uses the proposed policy in negotiation (see Section 4) with each of the proposed participants to derive a cryptographic context for the project.

Once the cryptographic context and membership are established for a project, the key management mechanism establishes the group keys and distributes them to all of the participants (see Section 5, Key management). Participants then use the keys for protecting their group communications. DCCM mechanisms support cryptography at any level in the network stack. Typically, group cryptography is found at the network, transport, or application level.

The group is actively managed during its lifetime (see Section 2.3, Group operations). Participants may be added or removed from the group; keys may expire; and eventually the group is dissolved.

2.3. Group operations

DCCM defines the following group operations for each project:

- **Add**. Participants can be added to a project. There are three security relevant aspects to adding participants to an existing project. First, the project initiator applies an access control policy for the project to determine if the participant can be added. Second, the project policy must specify whether or not new participants are to have access to communications or information generated before they joined the project. If not, a new project key has to be generated when a new participant joins the project. Finally, the project policy must specify how to negotiate the project cryptographic context when new participants are added. Because the negotiation process ensures that the project context is fully compliant with the policies of all participants, a new participant may require a change that is mutually exclusive with an existing member. An even more subtle problem during late additions is that existing members may actually have a different policy depending on the project membership. A new member may require a fresh negotiation for the project cryptographic context.
- **Remove**. Participants can be removed from a project. A remove is used when a participant is no longer participating, but still authorized for a group. There is no change to the group key.
- **Evict**. When participants are evicted from a project, their authorizations are revoked and the group key is changed to prevent them from obtaining further access to any project information.
- **Freeze**. Freezing a participant is a special case of removing a participant. When a participant is frozen, they are not removed from the group keying mechanism, but the system does not send any further key updates to the participant. This operation is useful for a member who knows they will be unable to receive or protect key information for a period of time.
- **Thaw**. Thawing a participant resumes the distribution of key update messages to the participant. A thaw reverses a freeze.
- **Resync**. Resynchronizing a participant resends the latest key update information for that participant only. In addition, it will thaw a frozen participant. A resync operation is used when a participant unexpectedly stops receiving key update information or suspects that its keying material is out of date.

In addition to the project level operations, a participant may also be evicted from the system, which results in an eviction from any projects containing the participant. A participant may choose to join or leave sessions within a project, but those are not security relevant operations. By definition in DCCM, all participants in a project are authorized for all sessions within that project. That authorization is constant, regardless of a participant's choice to exercise it.

3. Policy

A security *policy* is a set of rules specifying how to protect information. Policies can be described by *whom* they cover and by *what* they cover. In DCCM, every organizational entity has a policy. When a participant joins the DCCM system, they bring with them their security policy for the protection of their information. It may be a policy specific to them, such as in a flat group model, or the policy may be an organization policy covering all of the members of some hierarchy.

Policies can cover different security attributes at different levels of abstraction. For example, a high-level policy would state that confidentiality must be protected; a mid-level policy would state that strong encryption must be used; and a low-level policy would specify that triple-DES encryption in CBC mode must be used. Policies can also specify allowable behavior as a range of options. For example, a policy might

state that *at least* moderate strength integrity mechanisms must be used.

A policy that contains a range, or set of allowable actions, cannot be enforced by multiple participants with any expectation of interoperability. Interoperability can only be achieved when it can be guaranteed that all of the participants will enforce the policy exactly the same. DCCM accomplishes interoperability by distributing a policy that is completely unambiguous; there are no ranges or options. A policy that specifies a singular instantiation in DCCM is referred to as a *context*, specifically a *cryptographic context*.

There are several problems, however, in creating a cryptographic context that can be enforced by all participants, regardless of their initial policy and their security capabilities. The first problem is specifying a common syntax for representing policy information. There are other policy representations under research, such as SPSL [9] and GSAKMP [12], but neither of them is designed to express group policy including multiple levels of abstraction.

The second problem is specifying common semantics for the interpretation of a policy. Many common security terms and descriptions can be implemented or interpreted in different ways. This situation holds at both high levels and low levels of abstraction. For example, what is *strong* encryption? Does it imply specific algorithms, specific key lengths, specific protocols, or all of them? Is 64 bits of key length strong, or is 128? At low levels of abstraction, does 3DES mean CBC mode and three key EDE?

Without common semantics there is no way to support a common mapping or translation between policy abstractions and implementing mechanisms across the various participants in a project. DCCM addresses this problem with a *Cryptographic Context Negotiation Template* (CCNT) [3].

The CCNT is the common thread for policy representation, distribution, and negotiation in DCCM. It captures policy information at a low level of abstraction to ensure common semantics and interoperability between participants. Mapping from high level constructs to a CCNT can be performed locally; outward representation of policy is made only at the level of the CCNT.

The CCNT represents policy as an *n*-dimensional space. Each axis in the space represents a different aspect of the policy. Each value along an axis is a specific mechanism or configuration for that attribute. For example, *confidentiality* is the vertical axis in Figure 1, and it has the possible values of *3DES, CAST, IDEA,* and *RC4*. DCCM contexts, which are unambiguous sets of mechanisms, are single points in the *n*-dimensional space. Contexts can be proscribed by only a single value on any axis. DCCM policies, which are ranges or sets of allowable mechanisms, are sets of points in this space. Policies are proscribed by ranges of values or separate discrete values on multiple axes. Note that a policy is not required to have an intersection with every axis in the space. Policies can have a "don't care" with respect to that attribute.

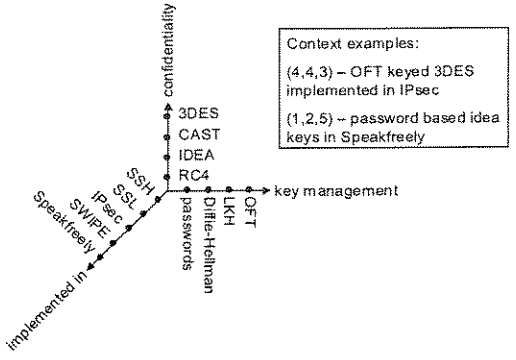

Figure 1: Example cryptographic context negotiation template

The CCNT is designed to represent a broad view of group security policy. Possible axes include:

- Data confidentiality;
- Key management ;
- Implementation mechanism;
- Temporal secrecy (forward secrecy, backward secrecy);
- Key lifetime;
- Key recovery;
- Group authentication;
- Layer for group security;
- Source authentication;
- Integrity;
- Required data throughput;
- Eviction; and
- Error handling.

The axes are not built into DCCM, but are specified in a context file. Axes can be easily added to the policy space by changing the context file; no other changes are required to the system.

4. Policy negotiation

A project context is created through a negotiation protocol during project creation in the DCCM system. The Cryptographic Context Negotiation Protocol (CCNP) [4] creates a fully specified cryptographic context for the project that fulfills the individual

policies of the group participants. Viewed graphically in Figure 2, cryptographic context negotiation finds the intersection between the allowable policies of the project participants.

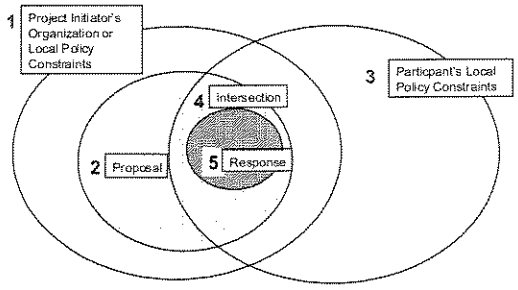

Figure 2: Cryptographic context negotiation

The process starts with the project initiator. The project initiator proposes a policy (area 2) for the project. Whatever is proposed is constrained by the initiator's own local policy (area 1). Each participant finds the intersection (area 4) between the proposed policy and their own local policy (area 3) and chooses a response (area 5) from the intersection. The initiator chooses the final project context from the intersection of the responses.

Careful selection of the proposal by the project initiator can effect different negotiation strategies. If the initiator creates a proposal that is actually a context, *i.e.*, there is only one choice for each axis, then negotiation becomes a directive action. The proposal *is* the policy (context) for the project; accept it as it is or do not join the project. A broad range of policy values (up to the full set of their local policy) proposed by the initiator increases the likelihood of an intersection occurring between a wide range of participants. The smaller the set in the proposal, the higher the chances that some participants will be unable to comply with or enforce the project's policy, and will, therefore, be unable to join the project.

5. Key management

The DCCM project requires a scalable method for establishing session keys for large dynamic groups. The method has to support efficient establishment of a shared secret key, as well as changing this key when group members are added or evicted. The following types of methods were considered:

- *Simple linear methods*, such as a Simple Key Distribution Center (SKDC), scale poorly to large groups, but are easy and straightforward to implement and employ with small-to-moderate size groups.
- *Information-theoretic methods*, such as Blundo's Symmetric Polynomials (BSP) [7] and Chiou-Chen's Secure Lock (CSL) [10], require large amounts of space per user and scale exponentially with the number of members in a group.
- *Multi-party methods based on Diffie-Hellman key agreement*, such as the Group Diffie-Hellman (GDH) method developed under the DARPA-sponsored Cliques project [1], and Burmester-Desmedt methods [8], require slow public-key operations and typically scale linearly with the size of a group. These methods seem better suited for small-to-moderate size groups of 10's or 100's of members. The GDH algorithm is especially attractive when distributed or decentralized control is needed. The Burmester-Desmedt method has the unique quality of constant delay (relative to group size) during initial key establishment.
- *Distributed, fault-tolerant systems* developed by Reiter *et al.* [14], and related *dynamic virtual private network (DVPNs)* developed by Rodeh *et al.* [15] include group key distribution techniques that are better suited for small-to-moderate size groups, but incorporate highly desirable fault-tolerant characteristics.
- *Hierarchical, tree-based methods* include Wallner *et al.*'s Logical Key Hierarchy (LKH) [11][16]. These methods represent group members as the leaves, and the group key as the root of a logical tree, and update keys via encrypting node keys down the tree. Time, space, and broadcast complexity all grow logarithmically relative to the size of a group, and hence these algorithms scale best to very large groups.

For further discussion of related work and additional references, see [2] and [6].

5.1. One-way function trees (OFT)

The hierarchical methods are best suited for the DCCM project. While exploring the use of LKH, the DCCM project developed a new hierarchical method for large dynamic group keying based on the novel application of One-way Function Trees (OFTs) [5][6] [13]. The OFT method, like LKH, represents group members as the leaves and the group key as the root of a logical tree. However, rather than "pushing" the group key down the tree, the OFT method "pulls" the group key up the tree, using one-way functions.

5.1.1. OFT structure. As shown in Figure 3, an OFT is a binary tree, each node x of which is associated with two cryptographic keys: a node key k_x and a blinded node key $k'_x = g(k_x)$. The blinded node key is computed from the node key using a one-way function g; it is blinded in the sense that a computationally limited adversary cannot find k_x from k'_x.

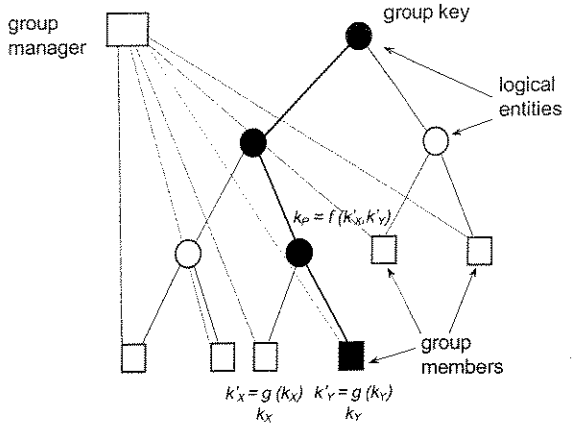

Figure 3: A one-way function tree

A group manager maintains a one-way function tree. Each leaf is associated with a member of the group. The manager uses a symmetric encryption function E to communicate securely with subsets of group members, using unblinded keys as encryption keys as explained below.

A randomly-chosen key is assigned to each member. This key is shared with the manager (via an external secure channel), and the key is assigned as the node key of the member's leaf. A variety of choices are possible governing who chooses the keys. In particular, the key could be chosen by the manager, member, or a combination thereof (see Section 5.1.3).

Each internal node p of the tree has exactly two children. The interior node keys are defined by the rule:

$$k_p = f(g(k_x), g(k_y)), \qquad (1)$$

where x and y denote the left and right child of the node p, respectively. The function g is one-way, and can be based on a cryptographic hash function such as MD5 or SHA-1. The function f does not need to be one-way; it needs to mix its inputs—the bitwise exclusive-or function *XOR* is a fast, simple, and effective choice. The node key associated with the root of the tree is the group key, which the group can use to communicate with privacy among group members and/or authentication of group membership.

The security of the system depends on the fact that each member's knowledge about the current state of the key tree is limited by the following invariant:

> *OFT security invariant* – each member knows the unblinded node keys on the path from its node to the root, and the blinded node keys that are siblings to its path to the root, and no other blinded or unblinded keys.

This invariant is maintained by all operations that add members to the group, and by all operations that delete members from the group.

McGrew and Sherman [13] discuss the properties of f and g in detail and give some preliminary observations regarding the security of OFT. A rigorous proof of the necessary and sufficient properties of f and g needed to satisfy formal security requirements for OFT remains an open problem.

5.1.2. OFT operations. The operations of adding and evicting members rely on the communication of new blinded key values, from the manager to all members, after the node key associated with a leaf has changed. To maintain security, each blinded node key must be communicated only to the appropriate subset of members. If the blinded key k'_x changes, then its new value must be communicated to all of the members who store it. These members are associated with the descendants of the sibling of x, and they know the unblinded node key k_y, where y is the sibling of x. To provide the new value of the blinded key to the appropriate set of members, and keeping it from other members, the manager encrypts k'_x with k_y before broadcasting it to the group. (Here and throughout, we shall use the verb "broadcast" in the sense of "group broadcast"—sending a message from the group manager to all members of the group.)

Adding a member—when a new member joins the group, an existing leaf node x is split, creating new nodes *left(x)* and *right(x)*. The member associated with x becomes associated with *left(x)*, and the new member is associated with *right(x)*. Both members are given new keys. The old member receives a new key because its former sibling knows the old blinded node key and could use this information in collusion with another group member to find an unblinded key that is not on his path to the root. The new values of the blinded node keys that have changed are broadcast securely to the appropriate subgroups, as described in the previous paragraph. The number of blinded keys that must be broadcast to the group is equal to the distance from x to the root plus two. In addition, the new member is given its set of blinded node keys. In order to keep the height h of the tree as low as possible,

when a new member is added, the leaf closest to the root is split.

Evicting a member—when the member associated with a leaf x is evicted from the group, the member assigned to the sibling of x is reassigned to the parent p of x and given a new leaf key value. If the sibling y of x is the root of a subtree, then p becomes y, moving the subtree closer to the root. In this case, one of the leaves of this subtree is given a new key (so that the evictee no longer knows the blinded key associated with the root of the subtree). The new values of the blinded node keys that have changed are broadcast securely to the appropriate subgroups, allowing all members to construct the new group key. The number of keys that must be broadcast is equal to the distance from x to the root.

Initialization—group initialization is the process through which the group establishes an initial group communications key. For the OFT method, this process involves two steps. First, the group manager broadcasts some information to the group members needed to apply the OFT key-updating procedures. Second, the members compute a shared group communications key, which is needed to begin secure group communications.

Group initialization is separate from group induction. During group *induction*, each member establishes an individual group base key known only by the member and the group manager. Group *initialization* assumes that each member has already established an individual group base key.

In the first step of OFT group initialization, the manager broadcasts every blinded node key in the OFT to all group members. In this broadcast, each blinded node key is encrypted by the unblinded key of the sibling node, so that only members in the sibling subtree can learn the blinded node key. All members receive the entire broadcast, which consists of a sequence of encrypted blinded node keys.

5.1.3. OFT properties. This section comments briefly on the security, resource usage, and salient characteristic features of the OFT method.

The security properties of OFT stem from the system invariant stated above, from the strength of the component one-way function, and from the random selection of leaf keys. In short, evicted members cannot read future messages, even with collusion by arbitrarily many evicted members, and newly admitted group members cannot read previous messages.

Evicted members have some information about the key tree but not enough to directly compute any unblinded node key. After a member is evicted, the keys along the path from the member's node to the root change. After this change, the evictee knows only the blinded keys of the siblings of the nodes along the path from the evictee to the root. These blinded nodes are insufficient to directly compute any unblinded key.

Interestingly, OFT is a centralized, member-contributory method. OFT is centralized in the sense that the group manager plays a special trusted role. OFT is member contributory in the sense that each leaf can contribute entropy to the group communication key.

With the OFT method, the number of keys stored by group members, the number of keys broadcast to the group when new members are added or evicted, and the computational efforts of group members, are logarithmic in the number of group members. The hierarchical nature of OFT distributes the computational costs of re-keying among the entire group, so that the manager's computational burden is comparable to that of a group member. Table 1 below summarizes the salient resource usage of adding or deleting a member with OFT in terms of time, memory, number of bits broadcast, and number of random bits needed. In the table, n is the group size, K is the size of a key in bits, and h is the height of the OFT ($h = lg\ n$ when the tree is balanced). Either the member or the manager could generate the random bits needed at the leaves. See [6] and [13] for more details, including comparisons with other key establishment methods such as SKDC and LKH.

Table 1: Summary of resource usage of adding or deleting a member with OFT

Resource Measure	Group Member Cost	Group Manager Cost
Time	*h*	*h*
Memory	*hK*	*2nK*
Bits broadcast	*0*	*hK + h*
Random bits generated	*0*	*K*

5.2. Key management summary

OFT was chosen as the group keying method for the DCCM project. While the simple SKDC, Group Diffie-Hellman, and other group keying methods may often be appealing for moderate size groups, many applications will likely demand a method that scales logarithmically in total delay and member memory usage. For such applications, especially if the add-member is more frequent than the evict-member operation, the OFT and other hierarchical methods look attractive for their constant-time add-member. LKH and OFT are similar methods, with LKH offering relatively simpler security semantics, and with OFT requiring fewer bits to transmit for re-keying. If it is

critical for the application to minimize the number of bits broadcast, the number of random bits generated, or if a member-contributory method is needed, then OFT may be the method of choice.

6. Implementation architecture

The DCCM concepts described in this paper, in particular OFT, the CCNT, and the CCNP, have been instantiated in a set of toolkits for use in producing demonstrations of providing security for very large groups. Two toolkits were produced, a group key toolkit implementing the OFT algorithm, and a DCCM toolkit implementing the DCCM group policy management.

6.1. Constraints

The design of the DCCM toolkit assumed several constraints. These constraints are a result of the existing structure on the Internet and the state of the current deployment of multicast technologies. The constraints are:

- No individual participant addressing. The design must function without the ability to address a participant individually on the network. Participants may have unique identities, but they may not be network addressable due to private addressing schemes hidden behind a firewall or network address translation box. A participant can address a known server individually and set up a connection, but the participant must initiate the connection.
- Firewalls will block unknown ports. DCCM must not use unknown or nonstandard ports for its communications. If unknown ports are used and participants are protected by a firewall, then the communications will be dropped until specialized configuration of the firewall can be performed.
- Multicast communication is unreliable. Multicast does not provide a connection paradigm. DCCM cannot assume that messages arrive at any point in the DCCM system. DCCM cannot assume that messages arrive in order.
- Multicast communication is subject to fragmentation. Fragments may arrive out of order or not at all. For simplicity, the DCCM design assumes that a message cannot be larger than the standard ethernet MTU of 1500 bytes.

6.2. High-level architecture

The functionality of DCCM is provided through several key software components that can be placed in different parts of an overall system depending on the application and its use of group security mechanisms. As shown in Figure 4, there is a single system component that services multiple instances of the other components.

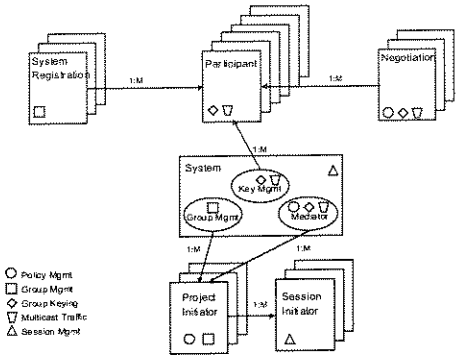

Figure 4: DCCM high-level architecture

The *System* component provides the long-term state of a DCCM system. It maintains the internal DCCM participant list, role information, and authentication shared secret. It enforces project contexts through its implementation of the key management mechanism for a project. The System component also supports the creation and maintenance of individual projects by storing the project membership list and mediating the project context negotiation process.

The *Participant* component provides the long-term state for an individual participant. It manages the participant's registration in a DCCM system, the shared secret, a list of projects for the participant, and the group keys for the projects that include the participant. The participant enforces the cryptographic context, specified in the CCNT, for each project in which it participates.

A participant is represented during policy negotiation by a *Negotiation* component. An instantiation of the negotiation component represents a single policy during project formation. All of the participants with the same policy are represented by the same negotiation component. This procedure limits the number of negotiators involved in a negotiation. Depending on the organization of the members joining a project, the negotiator may be used to represent different policies. In a hierarchical organization, the negotiation component will represent the organization's policy, and a participant will not have a

choice of negotiators. In a flat organization with autonomous participants, the negotiators will represent various public policies that participants will select from. If none of the policies offered is acceptable, a participant can be its own negotiator.

A participant joins a DCCM system through a *System Registration* component. There can be multiple registration components, each understanding a specific authentication and enrollment policy. In a DCCM system comprised of previously unrelated organizations, the organization's existing authentication infrastructure can be used by a registration component for each organization. The registration component can also enforce enrollment policy. For example, it can limit the negotiation components used by participants, thus specifying the allowable policies. The registrar can also limit the roles allowed for a participant. Finally, a registrar can remove or evict a participant from the system, and force their removal from all projects as well.

The *Project Initiator* component supports the entity that creates a project, typically one of the participants. It allows the creation of the initial policy proposal, specified in a CCNT; manages the negotiation protocol (CCNP) with the mediator in the system component and the negotiators; and manages the project membership list with the system component. Once the project is created, the project initiator will have the authority to perform group operations.

The *Session Initiator* supports the creation of sessions within a project. The session initiator is the interface between the DCCM keying mechanism and whatever security mechanism is used for the session. The session initiator will communicate the group session key to the mechanism at session establishment, and will then send key updates as they occur. There can be multiple sessions within a project.

6.3. Demonstration system

A demonstration system has been implemented for DCCM utilizing the architectural components. This provides one example of how to utilize the DCCM components in a simple system architecture. As shown in Figure 5, the demonstration system, the components are organized into a single client image and a server. For the simplicity, the demonstration includes a single registrar in the server. All of the roles for a participant are included in the client.

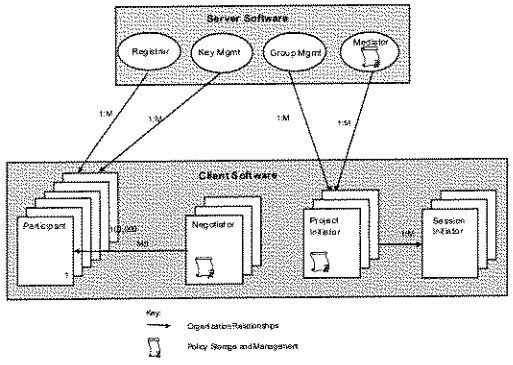

Figure 5: Demonstration architecture

7. Conclusions

Providing policy-based security for large dynamic groups is a challenging problem. It is also a challenge to envision maintaining the security of a group with 100,000 members. There are applications where a secure group of that size is meaningful, and DCCM provides solutions to several important aspects of the problem. DCCM introduces the concept of a group *cryptographic context* that enforces secure interoperability among group members. A Cryptographic Context Negotiation Template is used to represent, distribute, and negotiate group security policies. A Cryptographic Context Negotiation Protocol is used to create a group cryptographic context for a project. During project operation, the One-way Function Tree key mechanism achieves very efficient re-key operations. Its overall cost in terms of time and memory grows logarithmically with the size of the group.

Future work should expand both the policy and key management dimensions of DCCM. Future policy work should focus on representing policy at higher levels of abstraction, formalizing the languages used for policy representation, negotiating group policy at higher levels, and projecting abstract policy onto concrete mechanisms. Policies need to be expanded to represent applications where local autonomy of some mechanisms is desired or necessary; interoperability of security mechanisms may be necessary at the communication level, but not at higher levels.

Future group key management work should provide additional security mechanisms for group security, such as individual source authentication. There is also significant opportunity to enhance group key management techniques to efficiently handle surges in membership and large changes in membership.

8. Acknowledgments

The DCCM project was performed at NAI Labs, The Security Research Division of Network Associates, Inc. (formerly the Advanced Research and Engineering (AR&E) Division of Trusted Information Systems, Inc. (TIS)). Support for the DCCM project was provided by the Defense Advanced Research Projects Agency (DARPA) through Air Force Research Laboratory (AFRL) Contract No. F30602-97-C-0277.

The DCCM system was originally conceived and proposed by Dennis Branstad. During the life of the project the team included David Balenson, Dennis Branstad, Peter Dinsmore, Michael Ferguson, Michael Heyman, Peter Kruus, David McGrew, Matthew Mundy, Caroline Scace, Alan Sherman, and Jay Turner. This paper is based on the work of this talented team, and portions have been excerpted from project reports and briefings. The OFT method for large group key management was developed by David McGrew and Alan Sherman, with contributions from Michael Harding. The demonstration implementation was performed by Michael Ferguson, Michael Heyman, Peter Kruus, Matthew Mundy, and Caroline Scace.

9. References

[1] Ateniese, G., M. Steiner, and G. Tsudik, "Authenticated group key agreement and related protocols," *Proceedings ACM Conference on Computer and Communications Security*, 1998.

[2] Balenson, D., D. K. Branstad, D. A. McGrew, and A. Sherman, *Dynamic Cryptographic Context Management (DCCM) Report 1: Architecture and System Design,* TISR #0709, TIS Labs at Network Associates, Inc. (June 2, 1998).

[3] Balenson, D., D. K. Branstad, D. A. McGrew, J. W. Turner, and M. Heyman, *Dynamic Cryptographic Context Management (DCCM) Report 2, version 2: Cryptographic Context Negotiation Template,* TISR #0745-2, TIS Labs at Network Associates, Inc. (February 24, 1999).

[4] Balenson, D., D. K. Branstad, P. Dinsmore, M, Heyman, and C. Scace, *Dynamic Cryptographic Context Management (DCCM) Report 3: Cryptographic Context Negotiation Protocol,* TISR #0757, TIS Labs at Network Associates, Inc. (February 24, 1999).

[5] Balenson, D., McGrew, D, and A. Sherman, "Key management for large dynamic groups: one-way function trees and amortized initialization," NAI Labs, *Advanced Security Research Journal*, v1n1 (Fall 1998), pp 29-46.

[6] Balenson, D., D. McGrew, and A. Sherman, "Key management for large dynamic groups: one-way function trees and amortized initialization," Internet Draft (work in progress), draft-balenson-smug-groupkeymgt-oft-02.txt, October 1999.

[7] Blundo, C., A. de Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung," *Advances in Cryptology: Proceedings of Crypto92*, E. F. Brickell, ed., LNCS 740, Springer-Verlag (1992), 471–486.

[8] Burmester, M. and Y. Desmedt, "A secure and efficient conference key distribution system," *Advances in Cryptology: Proceedings of Eurocrypt94*, A. De Santis, ed., LNCS 950, Springer-Verlag (1994), 275–286.

[9] Condell, M., C. Lynn, and J. Zao, "Security Policy Specification Language," Internet Draft (work in progress), draft-ietf-ipsec-spsl-00.txt, October 21, 1998.

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

[11] Harney H. and E. Harder, "Logical key hierarchy protocol," Internet Draft (work in progress), draft-harney-sparta-lkhp-sec-00.txt, March 1999.

[12] Harney, H. and E. Harder, "Group Secure Association Key Management Protocol," Internet Draft (work in progress), draft-harney-sparta-gsakmp-sec-00.txt, April, 1999.

[13] McGrew, D., and A. Sherman, "Key establishment in large dynamic groups using one-way function trees," TIS Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD (May 1998).

[14] Reiter, M., K. Birman, and R. van Renesse, "A security architecture for fault-tolerant systems," TR 93-1354, Cornell University (June 3, 1993).

[15] Rodeh, O., K. Birman, and M. Hayden, "Dynamic virtual private networks," TR 97-1654, Cornell University (1997).

[16] Wallner, D., E. Harder, and R. Agee, "Key management for multicast: Issues and architectures," Internet Draft (work in progress), draft-wallner-key-arch-01.txt, September 15, 1998.