MULTIPLE LAYER ENCRYPTION FOR MULTICAST GROUPS

Alain Pannetrat
Alain.Pannetrat@eurecom.fr
Institut Eurecom, Sophia Antipolis, France.

Refik Molva
Refik.Molva@eurecom.fr
Institut Eurecom, Sophia Antipolis, France.

Abstract We propose a scalable multicast access control framework targeted for large dynamic groups, where members are added and removed frequently. Access control is provided by the original use of the counter-based block cipher mode of operation to encrypt traffic. This scheme uses intermediary elements in the network that contribute individually to the encryption, providing confidentiality, backward and forward secrecy and also containment, a property that limits the impact of the compromise of member access keys.

Keywords: Multicast Security, Encryption.

1. Introduction

This work describes a framework designed to provide access control in a large dynamic multicast[Dee89] group using encryption techniques. Consider, for example, the broadcast of pay-per-view TV over the Internet. In such an application, we need to ensure that only selected recipients are permitted to access the video. The set of recipients may nevertheless be very large and dynamic. As we will see, this complicates the design of an access control protocol. This work does not cover issues such as authentication of multicast data and other multicast security issues. We refer the reader to [HT00] for general presentation of multicast security issues, solutions and challenges.

We focus on a *1-to-n* multicast scenario, where there is one source and many recipients. This approach can be concretely extended to a few sources by using one source as a proxy, however our work does not aim to provide a mechanism where all recipients of the multicast group are potentially also a source. As highlighted in [HC99], the *1-to-n* scenario is well suited to large commercial multicast applications such as pay-per-view broadcasting. Through the remaining of this work we will use the following conventions: we call *recipient*, any entity capable of receiving multicast packets from a certain group, regardless of any cryptographic protection that is applied to the data inside the packets; we call *member*, a recipient who has been given cryptographic keys that enable him to access the content of the received multicast packets.

The primary goal of a multicast access control mechanism is to allow only members to access the content of multicast packets while disallowing other recipients to do so. We assume the existence of one or several entities called membership managers which *add* or *remove* members from the group based on a certain policy (which is beyond the scope of this work). Consequently, the group of members changes *dynamically* through time as members are added or removed from the group. The dynamic nature of the group of members imposes two critical requirements on the access control mechanism: *forward and backward secrecy.* Backward secrecy is defined as the impossibility for a member who is added to the group to access past data while forward secrecy is defined as the impossibility for a member who is removed from the group to access future data. The two main security objectives of a multicast access control framework are thus:

R1 - Confidentiality.

R2 - Backward and forward secrecy.

Each time a member is added or removed from the group, the encryption scheme that protects the data needs to be re-keyed to guaranty forward and backward secrecy. The main challenge is to provide a re-keying mechanism for a large multicast group that is *scalable.* As described by the same authors in [MP99], the 3 main scalability requirements of a multicast access control framework can be summarized as:

R3 - Processing scalability: the processing load supported by the entities in the framework should be independent of the group size.

R4 - Membership scalability: when a member is added or removed from the group it should not affect the rest of the group.

R5 - Group-wise scalability: no operation should require the whole group to be treated as a set of distinct individuals.

The requirements **R1** to **R5** together form the core design goals of a multicast access control framework. However, as highlighted in [MP99], there is an aspect that is often overlooked in multicast access control frameworks: *member compromise.* Generally, the more entities that participate in a security protocol, the greater the chances of a compromise. Thus if we design a multiparty security protocol that scales to a large group of members, we need to be concerned by such an issue: what happens if the access keys of a legitimate member are given to another recipient or stolen by an "hacker", published on a web site or in a newsgroup ? Consequently another requirement should be added to a multicast access control framework:

> **R6 - Containment:** The compromise of one (or several) member(s) should not cause the compromise the entire group.

There are two main directions followed by multicast access control proposals: *key graph* based approaches and *re-encryption tree* based approaches. Key graphs were first proposed by WONG ET AL. [WGL98] and WALLNER ET AL. [WHA98] (see also [CVSP98; MS98; CGI$^+$99]). These authors construct a tree T where each vertex represents a random distinct key and which has as many leaves as members in the group. Each leaf is associated to a member and each member receives a set of keys corresponding exactly to the keys on the path from the root of T to its corresponding leaf. The root of the tree is the key used to encrypt the traffic. When a member is removed from the group the root and all other keys on the path from the root to the leaf representing the departing member are invalidated. These keys are changed and re-broadcasted to the group by encrypting them with other remaining valid keys in the tree in a careful manner such that only the remaining members may update their subset of invalidated keys. We refer the reader to [WGL98] for precise definitions and related algorithms. The main strength of this approach is that it is simple because it does not rely on intermediary elements in the network and it has a reasonable overhead (logarithmic in the size of the group). However, when a member is removed from the group, then all other members are affected since at least one of the keys they hold, the root key, is invalid. As a consequence *all* remaining members in the group need to receive a message to update their access parameters. Thus this scheme does not offer membership scalability (R4). It does not offer containment either, since the compromise of the keys held by a legitimate recipient allows anyone to access the group from anywhere within the scope of the multicast group.

The other approach is based on the same principle that multicast routing protocols use to scale to large groups: involve the intermediary elements in the network. Here, the intermediary elements modify the encryption of the data going through the multicast network, such that the recipient group is partitioned to subsets with different access parameters, thus restricting all scalability issues to an arbitrary small subset of the group. The first proposal to follow this idea was the IOLUS framework [Mit97] and was used later in IGKMP[HCD00] and in Cipher Sequences [MP99]. We refer to these family of schemes as *re-encryption trees*, since they all use a tree of intermediary elements to perform transformations on the multicast data. Since each subset of the group uses a different key to access the group, there is a dependency between the location of a member in the network and the access parameters that it uses to access the group. If an access key is exposed, it has a limited impact because it is only useful in a subset of the group. Thus, this family of solutions provides containment (R6).

A strong drawback of IOLUS and IGKMP is that they trusts all intermediate elements with the security of the group. A solution to this problem was provided by Cipher Sequences[MP99]. Cipher Sequences allow the intermediate elements to perform security transforms without being trusted, thus satisfying a final requirement for a multicast confidentiality framework:

> **R7 - Limited trust in intermediary elements:** the compromise of some intermediary elements in the network should not compromise the group, or provide access to the protected data.

However, the main drawback of Cipher Sequences is that they rely on asymmetric cryptographic transforms as opposed to other schemes which use classical symmetric encryption. Thus Cipher Sequences cannot be used for bulk data encryption but are instead restricted to key distribution, which typically involves short messages.

1.1. Contribution and outline of this work

The solution we propose here belongs to the family of *re-encryption trees*. The main contribution of this work is the definition of a multicast confidentiality framework that uniquely combines 2 qualities:

1 It satisfies all the requirements described above including R7 like Cipher Sequences.

2 It can be applied for bulk data encryption as in IOLUS or IGKMP by relying on efficient cryptographic techniques.

This work is organized as follows. In the first section, we will look at some interesting cryptographic primitives that we use in our framework. Then, we present our framework based on multiple layers of encryption, or L-layer trees. In the following section we analyze the security of our construction, and discuss its scalability and relate it to the list of seven requirements we presented above. Finally, we present a potential improvement of our scheme with a shorter message expansion.

2. Cryptographic primitives.

Our framework uses a multiple key version of the counter based block cipher mode of operation (CTR-Mode) as described in [BDJR97] and can alternatively be constructed with any general stream cipher. CTR-mode has been proposed for standardization to NIST as an official AES mode of operation in [LRW00]. In this section we will briefly recall CTR-mode and present our own multiple key extension that is used in the framework. We denote "$\oplus$" as the binary XOR operation.

2.1. CTRM encryption scheme.

In [BDJR97] Bellare et al. describe and analyze various cipher modes of operation. We will briefly recall their work on the CTR-mode, which we use in our own scheme. Let $f_a(.)$ describe a l-bit pseudorandom permutation such as DES or AES[oST01] where a is the encryption key. The CTR-mode scheme $\text{CTRM}_{f_a} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined as follows:

- the function $\mathcal{K}$ flips coins and outputs a random k bit key a.
- the function $\mathcal{E}(\sigma, x)$ is defined as:

```
split $x$ in $n$ blocks of $l$ bits: $x = x_1, ..., x_n$
for $i = 1, ..., n$ do $y_i = f_a(\sigma + i) \oplus x_i$.
return $(\sigma, y_1 y_2 ... y_n)$.
$\sigma \leftarrow \sigma + n$
```

- the function $\mathcal{D}(\sigma, y)$ is defined as:

```
split $y$ in $n$ blocks of $l$ bits: $y = y_1 y_2 ... y_n$
for $i = 1, ..., n$ do $x_i = f_a(\sigma + i) \oplus y_i$
return $x = x_1 x_2 ... x_n$
```

Note: The state *or counter* σ is maintained by the encryption algorithm across consecutive encryptions with the same key. The decryption algorithm is stateless.

The authors of [BDJR97] have shown that there is a tight reduction between the security of the CTRM-scheme and the security of the primitive block operation f_a (we refer the reader to their work for details).

This scheme has many advantages. First it's paralellizable because the encryption of each block is independent of another. Second, the decryption can under certain circumstances be "prepared" in advance. Since the state is incremented in a predictable way across several messages, it means that the receiver can pre-compute some of the values of f_a to reduce online computations. Finally, this scheme uses the XOR operation which is commutative, a property that we will show to be useful.

2.2. Multiple encryptions.

The commutative nature of the XOR operation makes CTRM$=(\mathcal{K},\mathcal{E},\mathcal{D})$ interesting for a special form of multiple encryption. Normally if we encrypt a message x several times with a set of keys $a_1,...,a_m$ we would compute $(\sigma_m,y)=\mathcal{E}_{a_m}(\sigma_m,...\mathcal{E}_{a_2}(\sigma_2,\mathcal{E}_{a_1}(\sigma_1,x)...)$ but we proceed slightly differently, by leaving the counters $\sigma_1,...,\sigma_m$ outside the consecutive encryptions. We define CTRM$^{(m)}=(\mathcal{K}^{(m)},\mathcal{E}^{(m)},\mathcal{D}^{(m)})$ with m independent keys as as follows:

- $\mathcal{K}^{(m)}$ chooses m random keys : $a_0,a_1,...,a_m$.
- $\mathcal{E}^{(m)}_{a_1,a_2,..,a_m}(\sigma_1,...,\sigma_m,x)$ is defined as:

```
split $x$ in $n$ blocks of $l$ bits: $x=x_1,...,x_n$
for $i=1,...,n$ do $y_i=f_{a_1}(\sigma_1+i)\oplus...\oplus f_{a_m}(\sigma_m+i)\oplus x_i$.
return $(\sigma_1...\sigma_m,y_1y_2...y_n)$.
for $j=1,...,m$ do $\sigma_j\leftarrow\sigma_j+n$
```

- $\mathcal{D}^{(m)}_{a_1,...,a_m}(\sigma_1,...,\sigma_m,y)$ is defined as:

```
split $y$ in $n$ blocks of $l$ bits: $y=y_1y_2...y_n$
for $i=1,...,n$ do $x_i=f_{a_1}(\sigma_1+i)\oplus...\oplus f_{a_m}(\sigma_m+i)\oplus y_i$
return $x=x_1x_2...x_n$.
```

We note immediately that $\mathcal{E}_a=\mathcal{E}^{(1)}_a$ and $\mathcal{D}_a=\mathcal{D}^{(1)}_a$. This form of multiple encryption has the following interesting properties:

Fact 2.1 *For any permutation π of $\{1,...,m\}$ we have*

$$\mathcal{E}_{a_{\pi(1)},...,a_{\pi(m)}}(\sigma_{\pi(1)},...,\sigma_{\pi(1)},x)=\mathcal{E}_{a_1,a_2,..,a_m}(\sigma_1,...,\sigma_m,x)$$

and

$$\mathcal{D}_{a_{\pi(1)},...,a_{\pi(m)}}(\sigma_{\pi(1)},...,\sigma_{\pi(1)},y)=\mathcal{D}_{a_1,a_2,..,a_m}(\sigma_1,...,\sigma_m,y)$$

This is a natural consequence of the commutativity of the XOR binary operation.

Fact 2.2 *Given a message x if we compute $\{\sigma_1,...,\sigma_{(m-1)},y\}\leftarrow\mathcal{E}^{(m-1)}_{a_1,...,a_{(m-1)}}(\sigma_1,...,\sigma_{(m-1)},x)$ and $\{\sigma,z\}\leftarrow\mathcal{E}^{(1)}_a(\sigma,y)$ then we have $\{\sigma_1,...,\sigma_{(m-1)},\sigma,z\}=\mathcal{E}^{(m)}_{a_1,...,a_{(m-1)},a}(\sigma_1,...,\sigma_{(m-1)},\sigma,y)$. A similar result holds for $\mathcal{D}^{(m-1)}$ and $\mathcal{D}^{(1)}$.*

We also recall a classical property of multiple commutative encryptions that applies to our scheme [MvOV96, Chapter 7]:

Fact 2.3 *When a message is encrypted with m keys as described above it is at least as secure as anyone of the individual encryptions.*

3. L-Layer Encryption Trees.

3.1. Definitions

To describe our *1-to-n* multicast encryption framework, we view the multicast network as a tree with the following elements:

root: The root represents the source generating data to be securely distributed to members of the group.

intermediary: An intermediary describes any node in the tree besides the root and the leaves. An intermediary element is either a multicast enabled router or proxy with embedded encryption capabilities.

leaf: The leaf represents a member, or a cluster of members receiving data from the same intermediary. These elements are expected to be physically close to each other, for example on the same LAN with a common IGMP[Fen97] router.

Our approach to secure multicast can be summarized as follows. The root produces data, encrypts it and forwards it to the multicast network. intermediaries receive data, modify the encryption and forward the result to other intermediaries until it reaches a leaf. Finally, the members in the leaves decrypt the data transmitted by the source. Since each intermediary element changes the encryption of the data, each leaf will require different access parameters to access the content, which provides a form containment as described in the introduction.

3.2. Construction

We call a tree T a *singular leaf* tree if each leaf in T has a distinct unique parent. We define a function $Depth(N)$ which for a node N returns its depth in the tree, where $Depth(root) = 0$, and we define the function $Parent(N, L)$ which returns the L^{th} parent of node N if it exists or $\emptyset$ otherwise.

We call *"L-layer tree"* the association of a multicast singular leaf tree network with a set of cryptographic transformations designed to protect the distributed data with a varying set of L layers of encryption. We associate a set of keys to the tree to perform CTRM encryptions as described previously, taking advantage of the commutative nature of the encryption scheme. Let T be a tree without sibling leaves with n intermediaries. We associate a set of $n + L$ different encryption keys $[K_1, ..., K_{L+n}]$ to the tree as follows:

root: The root receives the encryption keys $[K_1, ..., K_L]$.

intermediaries: The n intermediaries receive each a distinct key from the set $[K_{L+1}, ..., K_{L+n}]$. For example if we number the intermediary arbitrarily from 1 to n we can associate key K_{L+i} to intermediary number i. We call this key the intermediary's *encryption* key. Each intermediary N receives a secondary key K', which we will call *decryption* key, as follows:

```
if $Depth(N) < L$ then $K' \leftarrow K_{Depth(N)}$.
else $K' \leftarrow$ (the encryption key of $Parent(N, L)$)
```

leaves: The leaves each receive L keys. To clarify the notation we will call these keys $X_1, ..., X_L$, A leaf N receives these keys as follows:

```
for $i = 1, ..., L$ do
(1)   if $Parent(N, i) = \emptyset$ then $X_{(L-i+1)} \leftarrow K_{Depth(N)+i-1}$
(2)   else $X_{(L-i+1)} \leftarrow$ (the encryption key of $Parent(N, L)$)
```

Line (1) shows that if the leaf does not have an i^{th} parent then it gets one of the encryption keys used by the root and line (2) shows that if it does have an i^{th} parent then it gets the encryption key of that parent.

This key assignment may seem somewhat complex but in fact it's governed by two simple principles:

- Each intermediary gets its own encryption key and the encryption key of its L^{th} parent.
- Each leaf gets all the encryption key of its parents of level L down to 1.

The complexity only appears in the algorithm for nodes or leaves that are not deep enough in the tree to fully apply the previous two rules. In such a case, the otherwise missing keys are taken from the root. If we focus our attention on a single path of the tree extending from the root to a leaf, we can see that each key used on a node in a path is used once as an encryption key and once as a decryption key, as illustrated on figure 1.

An example of our key assignment algorithm is shown on figure 2 for a 4 layer tree.

3.3. Data Distribution

Once the tree is constructed, its components operate as follows:

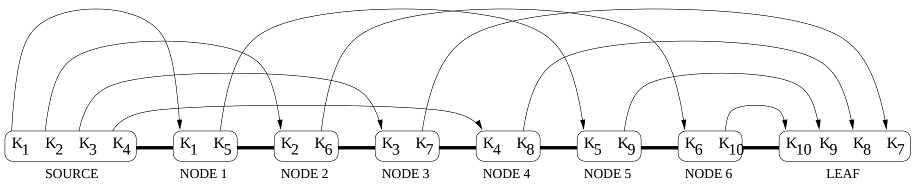

Figure 1. Key distribution on a single path in a 4 layer tree.

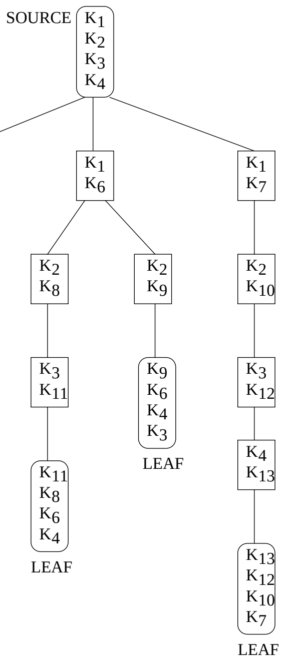

Figure 2. Key distribution on a 4 layer tree.

root: The root or source encrypts a message M by computing $(\sigma_1, ..., \sigma_L, C) = \mathcal{E}^{(L)}_{K_1,...,K_L}(\sigma_1, ..., \sigma_L, M)$ and sends the result to its children nodes in the tree.

intermediaries: Each intermediary N receives an encrypted message $(\sigma_1, ..., \sigma_L, C)$. The intermediary N performs the following operations:

1 Suppress a layer of encryption: $C' \leftarrow \mathcal{D}^{(1)}(\sigma_1, C)$.

2 Add a new of encryption: $(\tau, C'') \leftarrow \mathcal{E}^{(1)}(\tau, C')$ where τ is the internal counter of N.

3 Let $\tau_L \leftarrow \tau$ and $\tau_i \leftarrow \sigma_{(i+1)}$ for $i = 1, ..., (L-1)$. Send $(\tau_1, \tau_2, ..., \tau_L, C'')$ to the children nodes.

leaves: The leaves receive an encrypted message $(\sigma_1, \sigma_2, ..., \sigma_L, C)$ that they decrypt by computing $M = \mathcal{D}^{(L)}_{X_1,...,X_L}(\sigma_1, ..., \sigma_L, C)$.

If we recall the construction of our tree, we see that the keys are distributed to make the above algorithm work: each key used to encrypt the data is used later as a decryption key. The source and the leaves both perform L-encryptions and L-decryption. Intermediaries use Property 2 to first transform an L-encryption to a $(L-1)$ encryption, then using the same property, they transform the $(L-1)$-encryption back into an L-encryption. The combination of Property 2.1 and 2.2 allows us to decrypt a layer regardless of the order in which the encryptions were done.

As an example, we will examine how our data distribution algorithm is applied on the path of the 4-layer tree of figure 1, where C^r denotes the encryption of M at stage r in the algorithm:

Source: Computes and sends $(\sigma_1, \sigma_2, \sigma_3, \sigma_4, C^0) \leftarrow \mathcal{E}^{(m)}_{K_1,K_2,K_3,K_4}(\sigma_1, \sigma_2, \sigma_3, \sigma_4, M)$

Node 1: Suppresses a layer $(\sigma_2, \sigma_3, \sigma_4, C^1) \leftarrow \mathcal{D}^{(1)}_{K_1}(\sigma_1, C^0)$.
Then it computes and sends $(\sigma_2, \sigma_3, \sigma_4, \sigma_5, C^2) \leftarrow \mathcal{E}^{(1)}_{K_5}(\sigma_5, C^1)$.

Node 2: Suppresses a layer $(\sigma_3, \sigma_4, \sigma_5, C^3) \leftarrow \mathcal{D}^{(1)}_{K_2}(\sigma_2, C^2)$.
Then it computes and sends $(\sigma_3, \sigma_4, \sigma_5, \sigma_6, C^4) \leftarrow \mathcal{E}^{(1)}_{K_6}(\sigma_6, C^3)$.

Node 3: Suppresses a layer $(\sigma_4, \sigma_5, \sigma_6, C^5) \leftarrow \mathcal{D}^{(1)}_{K_3}(\sigma_3, C^4)$.
Then it computes and sends $(\sigma_4, \sigma_5, \sigma_6, \sigma_7, C^6) \leftarrow \mathcal{E}^{(1)}_{K_7}(\sigma_7, C^5)$.

Node 4: Suppresses a layer $(\sigma_5, \sigma_6, \sigma_7, C^7) \leftarrow \mathcal{D}^{(1)}_{K_4}(\sigma_4, C^6)$.
Then it computes and sends $(\sigma_5, \sigma_6, \sigma_7, \sigma_8, C^8) \leftarrow \mathcal{E}^{(1)}_{K_8}(\sigma_8, C^7)$.

Node 5: Suppresses a layer $(\sigma_6, \sigma_7, \sigma_8, C^9) \leftarrow \mathcal{D}^{(1)}_{K_5}(\sigma_5, C^8)$.
Then it computes and sends $(\sigma_2, \sigma_3, \sigma_4, \sigma_5, C^{10}) \leftarrow \mathcal{E}^{(1)}_{K_9}(\sigma_9, C^9)$.

Node 6: Suppresses a layer $(\sigma_7, \sigma_8, \sigma_9, C^{11}) \leftarrow \mathcal{D}^{(1)}_{K_6}(\sigma_6, C^{10})$.
Then it computes and sends $(\sigma_7, \sigma_8, \sigma_9, \sigma_{10}, C^{12}) \leftarrow \mathcal{E}^{(1)}_{K_{10}}(\sigma_{10}, C^{11})$.

Leaf: Decrypts the message $M \leftarrow \mathcal{D}^{(m)}_{K_7, K_8, K_9, K_{10}}(\sigma_7, \sigma_8, \sigma_9, \sigma_{10}, C^{12})$.

3.4. Membership Management

After describing how members access the multicast content in the previous section, we will now turn our attention to the addition and removal of members in our framework.

Add: When a recipient M wants to be added to the group, he contacts a membership manager (MM) with an authenticated secure channel. If M is allowed to access the group, then there are 2 possible scenarios:

1 M is already physically in an existing leaf F: the MM sends an authenticated secure message to the parent intermediary P of F, to change the encryption key K of P to a new value K'. Then the new key K' is sent to all the members of the same leaf and to the new member M.

2 M is not in an existing leaf: the tree is expanded to create a new leaf for M. The corresponding keys are distributed to the new intermediaries and M.

Remove: When a member needs to be removed from the group, the MM sends an authenticated secure message to the parent intermediary P of F, to change the encryption key K of P to a new value K'. Then the new key K' is sent to all the members of the same leaf except M. If the leaf is empty because the last member left, than after a certain delay, we may remove unused intermediaries from the tree.

The leaf holds L keys and needs all of them to access the data. Thus, changing just one of them when we add or remove a member provides us with forward and backward secrecy (R2).

3.5. Distributed Membership Management.

This scheme also lends itself to a certain form of decentralized key management. The tree of intermediary elements can be managed by a hierarchy of membership managers. It follows from our construction in section that an individual membership manager only needs to know L extra keys to manage a subtree on its own. More precisely, if a membership manager is selected to manage a subtree consisting of an intermediary N and all its descendants in the tree, then it needs the to know the set $Y_1, ..., Y_L$ of keys defined as follows:

```
**for** $i = 1, ..., L$ **do**
    **if** $Parent(N, i) = \emptyset$ **then** $Y_i \leftarrow K_{Depth(N)+i-1}$
    **else** $Y_i \leftarrow$ (the encryption key of $Parent(N, i)$)
```

In turn a membership manager may delegate the management of some of its own subtrees to several other membership managers.

4. Security Requirements

4.1. Encryption

To discuss the security of our construction, we will first look at *one-layer* trees before we study the general case. One layer trees are conceptually very simple, since they only use the original CTRM encryption algorithm. The source has a key K_1 and uses it to encrypt data to be sent to its children. The intermediaries decrypt the data with the key K_j that their parents used to encrypt the data and use their own key K_i to encrypt the data again for their children. The leaves use a single key to access the data. A one-layer tree is quite similar to the IOLUS framework [Mit97], and it shares one of the drawbacks of that framework: each intermediary is trusted to access the cleartext data. For now however, let's examine the security off a one-layer tree while making the hypothesis that the intermediary elements are secure.

The individual links are secured by the CTRM encryption algorithm. In our framework, an adversary has the ability to observe several links and thus the same message encrypted under different keys. We can even imagine that the adversary may modify or input new messages at different points in the tree to try to break the security of the system. In a recent work evaluating the security of public key cryptosystems in the multiuser setting [BBM00], Bellare et al. have shown essentially that if a public key cryptosystem is secure in the sense of indistinguishability, then it implies the security of the cryptosystem in the multiuser setting, where related messages are encrypted under different keys. We refer the reader to their work for further details [BBM00]. Though their work was targeted at public key cryptosystems, their results can be applied to the private key setting, and since the CTRM encryption is secure in the sense of "indistinguishability" under chosen plaintext attacks[BDJR97], we can assert the security of the whole tree by using the results of [BBM00].

Now for L-layer trees, property 2.3 tells us that they are at least as secure as a 1-layer tree if no intermediary is compromised. But, the advantage of a L-layer tree is that it remains secure even if some nodes are compromised, more precisely:

Proposition 4.1 Let $B = B_1, ..., B_p$ define a set of p compromised intermediaries in a L-layer tree. The tree remains secure as long as there exists a constant $c \in \{0, ..., L-1\}$ such that $Depth(B_i) \neq c \bmod L$ for all $i \in \{1, ..., p\}$.

Proof. This property derives from the arrangement of the keys in the tree. Let $B = B_1, ..., B_p$ define a set of compromised intermediaries in a L-layered tree T such that there exists a constant $c \in \{0, ..., L-1\}$ verifying $Depth(B_i) \bmod L \neq c$ for all $i \in \{1, ..., p\}$. From the tree T we can extract a subtree $\overline{T}$ iteratively, as follows:

```
Notations:
Let $N_0$ define the root of $T$.
Let $\overline{N}_0$ define the root of $\overline{T}$, and let $\{\overline{N}_1, ..., \overline{N}_q\}$ define the intermediary nodes of $\overline{T}$.
Construction:
$\overline{N}_0 \leftarrow N_0$
Select $\{N_1, ..., N_q\}$, the set of intermediaries $N_i$ of $T$ which verify $Depth(N_i) = c \bmod L$.
$\{\overline{N}_1, ..., \overline{N}_q\} \leftarrow \{N_1, ..., N_q\}$
for $i = 1, ..., q$ do
  if $Depth(N_i) = c$ then
      connect $\overline{N}_i$ to $\overline{N}_0$ in $\overline{T}$.
      let $\overline{\mathcal{Z}}$ be the concatenation of all leaves $\mathcal{Z}_k \in T$ such that $Depth(\mathcal{Z}_k) \leq c$.
      if $\overline{\mathcal{Z}} \neq \emptyset$ then connect $\overline{\mathcal{Z}}$ to $\overline{N}_0$ in $\overline{T}$.
  else
      let $N_j = Parent(N_i, L)$.
      connect $\overline{N}_j$ to $\overline{N}_i$ in $\overline{T}$.
      let $\overline{\mathcal{Z}}$ be the concatenation of all leaves $\mathcal{Z}_k \in T$ such that
            $(Parent(\mathcal{Z}_k, r) = N_i$ and $j \leq L)$.
      if $\overline{\mathcal{Z}} \neq \emptyset$ then connect $\overline{\mathcal{Z}}$ to $\overline{N}_i$ in $\overline{T}$.
```

The tree $\overline{T}$ represents a 1-layer tree such that none of its intermediaries $\{\overline{N}_1, ..., \overline{N}_q\}$ hold a key in common with any of those distributed to the compromised set B. Thus since there exists an independent 1-layer tree between the root and the leaves, the encryption of data in the tree remains secure (R1). $\diamondsuit$

Corollary 4.2 An obvious implication of this property is that an L-layer tree can at least withstand the compromise of any set of less than L intermediaries.

4.2. Containment

In singular parent trees, no leaf shares its direct parent with another leaf, thus each leaf receives data that is encrypted with at least one layer of encryption that is distinct from any other leaf. This distinct layer of encryption is generated by the parent intermediary node of the leaf. Thus if $\mathcal{L}$ is a leaf, then no collusion of any group of leaves $\{\mathcal{L}_1, ..., \mathcal{L}_p | \mathcal{L}_i \neq \mathcal{L}, i \in \{1, ..., p\}\}$ can break the encryption of data received in the leaf $\mathcal{L}$. An adversary in a leaf $\mathcal{L}$ who compromises the keys in a set of leaves $\{\mathcal{L}_1, ..., \mathcal{L}_p | \mathcal{L}_i \neq \mathcal{L}, i \in \{1, ..., p\}\}$ cannot use this information to access the data in his own leaf. Thus having a *singular leaf* tree is a sufficient condition to ensure a secure dependence between the location of a recipient in the network and the keys used to decrypt the received multicast data. This secure dependency provide containment (R6) since the exposure of a key will only be useful to an adversary within the same leaf and will not affect the security of the whole group.

There is no containment within a leaf, all the recipients that are physically in the same leaf use the same key to access the data, thus exposure of keying material in one leaf allows other members of the same leaf to access the data. However, unless there is a form of hardware access control installed directly on each recipient, providing containment in within a leaf is very hard: ultimately, it's difficult to stop or even detect if a member rebroadcasts decrypted data to other local recipients that are not members themselves.

4.3. Hybrid attacks

The current framework may face more complex attacks which are a combination of both the compromise of leaves and intermediary elements:

Membership extensions:. If a member M in a leaf $\mathcal{L}$ takes full control of the direct parent P of $\mathcal{L}$, it can monitor key changes in P. If the membership manager decides to remove M from the group, it will change the key K held by P and send the new updated key K'

to the other remaining members in $\mathcal{L}$ as well as P. As a consequence the removed member M will still be able to stay in the group because it learn the new value K' from P. This means that we lose forward secrecy. Recovering from such a compromise requires a key change in the parent P' of the compromised node P, which in turn requires all the leaves that have P' as an ancestor to be updated.

Containment failures:. Assume that two leafs $\mathcal{L}_1$ and $\mathcal{L}_2$ of same depth in the tree share a common ancestor node P in the tree which verifies $|Depth(\mathcal{L}_1) - Depth(P)| \leq L$. In that situation, the members in $\mathcal{L}_1$ and the members in $\mathcal{L}_2$ have $k < L$ decryption keys in common. If a member M_1 in $\mathcal{L}_1$ compromises the $L - k$ first parents of $\mathcal{L}_2$ than M_1 will know enough information to generate the set of L keys used in $\mathcal{L}_2$, by combining the k common keys with the $L - k$ compromised keys. This attacks breaks the containment property of the scheme for two leafs that are at the same depth in the tree.

5. Scalability Requirements

The processing load supported by each entity in the tree is not proportional to the group size. For the leaves and the root it depends on the parameter L which defines the number of layers in the tree, while intermediaries always perform a single decryption and a single encryption regardless of the number of layers. Thus this framework offers processing scalability (R3).

When a member is added or removed from the group, the key update remains local and only affects a leaf at a time. The number of elements in a leaf is not a scalability factor itself, because we can simply create more branches in the tree to cope with leaves that get too large. Consequently, our framework provide membership (R4) and group-wise scalability (R5).

One of the main differences in terms of scalability between key graph [WGL98] approaches and intermediary based approaches like ours, is membership scalability. In key graph approaches, the departure of a member requires the whole group to receive a message to update its access keys.

6. Reducing Expansion

In the CTRM$^{(m)}$ scheme, the encryption of a message results in an expansion of $m.|\sigma_i|$ bytes where $|\sigma_i|$ represents the size in bytes of the state value. We could use a single state chosen by the source and common to all layers of encryption as well as all elements in the tree. In other words we would rewrite the encryption algorithm as follows:

```
$\mathcal{E}^{*(m)}_{a_1,a_2,..,a_m}(\sigma, x)$:
**split** $x$ in $n$ blocks of $l$ bits: $x = x_1...x_n$
**for** $i = 1, ..., n$ **do** $y_i = f_{a_1}(\sigma + i) \oplus ... \oplus f_{a_m}(\sigma + i) \oplus x_i$.
**return** $(\sigma, y_1 y_2 ... y_n)$
$\sigma \leftarrow \sigma + n$
```

The intermediaries would use the same σ to both encryption and decryption operations. The algorithm would be simplified and the ciphertext size would be independent of the number of layers in the tree. In such a case, however, proving the security of the scheme is an open problem since the intermediaries are now stateless and cannot be modeled as independent encrypting devices, which was a requirement of the security proof found in [BBM00] upon which we relied for our scheme.

7. Conclusion

Using intermediary elements in the network we have constructed a scalable framework for multicast access control. This framework offers interesting properties such as containment, and limited trust in the intermediary elements of the network. It shows some vulnerabilities when both members and intermediary elements in the network are compromised, in particular the direct parent intermediary of a leaf in the tree.

Interesting applications of this scheme are not necessarily limited to IP-Multicast. Consider for example the use of this scheme for content distribution in next generation mobile networks. Our scheme would allow a provider to send protected data to its clients even if they are roaming in foreign or "less trusted" networks. Moreover, the risk of key piracy found for example in European Digital Video Broadcasting would be limited by the containment property of our scheme.

References

M. Bellare, A. Boldyreva, and Silvio Micali. Public-key encryption in a multiuser setting: Security proofs and improvements. In *Eurocrypt 2000*, volume LNCS 1807, pages 259–274. Springer Verlag, 2000.

Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *IEEE Symposium on Foundations of Computer Science*, pages 394–403, 1997.

R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of IEEE Infocom'99*, 1999.

G. Caronni, M. Valdvogel, D. Sun, and B. Plattner. Efficient security for large and dynamic multicast groups. In *Proceedings of IEEE WETICE'98*, 1998.

Steve E. Deering. RFC 1112: Host extensions for IP multicasting, Aug 1989.

W. Fenner. Internet group management protocol, version 2. Request For Comments 2236, November 1997. see also draft-ietf-idmr-igmpv3-and-routing-01.txt for IGMP v3.

H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of ACM SIGCOMM'99*, Harvard University, September 1999. ACM SIGCOMM.

Thomas Hardjono, Brad Cain, and Naganand Doraswamy. Intra-domain group key management protocol. Internet-Draft, work in progress, February 2000.

T. Hardjono and G. Tsudik. IP multicast security: Issues and directions. *Annales des Telecommunications*, to appear in 2000.

H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST concerning AES modes of operation: CTR-Mode encryption. In *NIST First Modes of Operation Workshop*, Baltimore, Maryland, USA, October 20 2000.

Suivo Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM'97 (September 14-18, 1997, Cannes, France)*, 1997.

Refik Molva and Alain Pannetrat. Scalable multicast security in dynamic groups. In *Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 101–112, Singapore, November 1999. Association for Computing Machinery.

David A. McGrew and Alan T. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical report, TIS Labs at Network Associates, Inc., Glenwood, MD, 1998.

Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

National Institute of Standards and Technology. Advanced Encryption Standard, 2001.

C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM 1998*, pages 68–79, 1998.

Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. Internet draft, Network working group, september 1998, 1998.