Scalable Multicast Security in Dynamic Groups.

Refik Molva, Alain Pannetrat
Institut Eurecom, Sophia-Antipolis, France.
{molva,pannetra}@eurecom.fr

Abstract

In this paper we propose a new framework for multicast security based on distributed computation of security transforms by intermediate nodes. The involvement of intermediate nodes in the security process causes a new type of dependency between group membership and the topology of the multicast network. Thanks to this dependency, the containment of security exposures in large multicast groups is assured. The framework also assures both the scalability for large dynamic groups and the security of individual members. Two different key distribution protocols complying with the framework are introduced. The first protocol is an extension of the El Gamal encryption scheme whereas the second is based on a multi-exponent version of RSA.

1 Introduction

Multi-party communications have recently become the focus of new developments in the area of applications and networking from group applications like video-conferencing to network layer multicast protocols. As part of the new issues involved with multi-party communications, security in terms of privacy and integrity has received particular attention due to the vulnerabilities inherent to multi-party architectures.

While several projects addressed the problem of key distribution [9] and digital signatures [3] among the participants of a group, the security issues related to multicast in large and dynamic groups remained comparatively unexplored. Multicast is a special case of group protocols by which a single source transmits data to multiple recipients. Like any other multi-party scheme, the inherent complexity of the underlying communication mechanisms exposes multicast protocols to vulnerabilities that have no counterpart in the unicast case as depicted in [2][6]. Possible countermeasures for those vulnerabilities are cryptographic security services ranging from authentication of group members, data confidentiality and integrity, non-repudiation of origin to access control for group membership. Next to basic security services, automatic key management is necessary for the secure provision of large recipient groups with cryptographic keying material.

In this paper we present a framework for multicast security that focuses on two issues:

- scalability in large dynamic groups: the amount of processing of each individual component of the multicast security mechanism should be independent of the group size; changes to the group membership should affect the smallest subset of the group.
- containment of security exposures through partitioning: each recipient group should be partitioned into sub-groups in order to assure that a security exposure in a sub-group does not endanger the security of other sub-groups.

As a preliminary step we present a set of cryptographic sequences with special properties that are organized in a tree. The rest of the paper considers this formal graph and applies it to a multicast tree. We show how the properties of our formal graph can be used to offer a multicast security framework that deals with security and scalability issues.

In the proposed framework, conflicting security and scalability requirements are addressed through a distributed scheme whereby intermediate components placed on the multicast transmission tree take part in the security protocol. The intermediate components share the security processing load with the source and assure the containment of security exposures at various parts of the multicast tree. The framework defines basic properties of a set of cryptographic functions that

assure data confidentiality Depending on the performance of the underlying algorithm, implementations of the framework may be suitable either for the encryption of bulk data or only for the encryption of short messages as required by key distribution. The framework is first validated with respect to security and scalability requirements. Two different implementations of the framework are then discussed. Both solutions are based on asymmetric techniques: an extension of the El Gamal algorithm [4] and a variation on RSA [8]. These algorithms which offer strong protection are only suitable for key distribution since, due to their inherent complexity, bulk data encryption with these solutions seems prohibitive.

2 Cryptographic Functions

The building blocks we have chosen to use to design data confidentiality protocols over a multicast tree are called ***Reversible Parametric Sequence*** (RPS). This section will give a formal definition of these sequences and associate them in trees. These trees will be used is further sections to describe our multicast security framework.

2.1 Reversible Parametric Sequence

Let $f : \mathbb{N}^2 \mapsto \mathbb{N}$ be a function with the following property: if $y = f(x, a)$ it is computationally infeasible to compute a knowing x and y.

Let $(a_{1 \leq i \leq n})$ be a finite sequence of n elements. Let $(S_{0 \leq i \leq n})$ be a finite sequence of $n+1$ elements defined as:

$$S_i = f(S_{i-1}, a_i), \text{ for } i > 0.$$

$$S_0, \textit{ the initial value of the sequence.}$$

Such a sequence will be called Reversible Parametric Sequence associated to f or $\mathbf{RPS}_f$ if, for all couples $(i, j) \in \mathbb{N}^2$ verifying $0 < i < j \leq n$, there exists a computable function $h_{i,j}$ such as $S_i = h_{i,j}(S_j)$.

Moreover:

- A RPS_f $(S_{0 \leq i})$ will be called a Symmetric (reversible) Parametric Sequence associated to f or $\mathbf{SPS}_f$ if $h_{i,j}$ can be computed from $\{a_{i+1}, a_{i+2}, ..., a_j\}$.
- A RPS_f $(S_{0 \leq i})$ will be called an Asymmetric (reversible) Parametric Sequence associated to f or $\mathbf{APS}_f$ if it is computationally infeasible to determine the function $h_{i,j}$ from $\{a_{i+1}, a_{i+2}, ..., a_j\}$ alone.

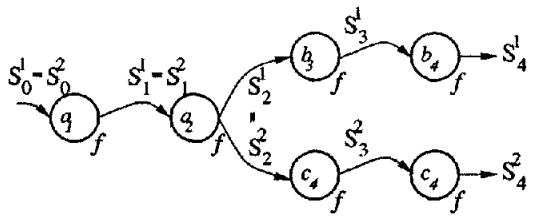

Figure 1: Two RPS_f sequences mapped over a tree.

Example: Let p be a large prime number. Let (n_i) be a set of numbers in $\mathbb{Z}^*_{(p-1)}$ and g be a generator of the cyclic group $\mathbb{Z}^*_p$. Define f as $f(x, a) = x^a \bmod p$. The following sequence is a SPS_f:

$$S_i = f(S_{i-1}, n_i)$$

$$S_0 = g$$

2.2 $\mathbf{RPS}_f$'s over a General Tree

A tree can map a family of $RPS'_f s$ which have terms that differ only after a certain rank, greater than 1. For example, if S^1 is a RPS_f defined from a parameter sequence $\{a_1, a_2, b_3, b_4\}$ and S^2 is a RPS_f defined from a parameter sequence $\{a_1, a_2, c_3, c_4\}$, a simple tree that maps S^1 and S^2 can be constructed as in figure 1. This tree illustrates the fact that S^1 and S^2 differ after rank 2.

This property can be extended to a large number of $RPS'_f s$, with a tree that branches each time at least two sequences differ. The nodes of the tree denote the f function and the edges are the elements S^i_j of the i different $RPS'_f s$ mapped over that tree. All the sequences mapped over the same tree share at least two elements, namely S_0 and S_1. These two values are the input and the output, respectively, of the root node of the tree.

In the following discussion, (S^i) will refer to the RPS_f mapped over the path that connects the root to the i^{th} leaf of a tree. The corresponding reversing functions will be denoted $h^i_{(,)}$.

3 Multicast Security

The goal of multicast security is to assure that the source of the multicast stream and the group of multicast recipients communicate securely. This can be achieved through the authentication of the message origin by the recipients and through confidentiality and integrity preventing disclosure and modification of the messages by any party other than the members of the multicast group. These services typically require the establishment of a security association between the source

and the recipients of the multicast channel. The security association defines the set of cryptographic keys and algorithms used for each service. While authentication, confidentiality and integrity of messages in the multicast stream can be assured by classical network security mechanisms akin to unicast, the establishment of a security association for a multicast channel is inherently more complex than with unicast. In the unicast case, a security association is static in that the source, the recipient and the the data flow do not vary during the association. In a dynamic multicast group, a session is an ever evolving entity as recipients can be added to or removed from the recipient group through join and leave operations, respectively. Ideally, the keying material shared by the members of the multicast security association should be updated in order to fulfill the following conditions:

1. When a user JOINS the group he should not have access to past keying material.
2. When a user LEAVES a group he should not have access to future keying material.

Hence, some keying material must change each time the set of users in a multicast group changes. It should be noted that the above conditions apply for the highest security requirements and they can be relaxed for multicast applications with less critical security requirements.

Moreover, as a group gets larger, it is not acceptable to share the same keying material between all users of the group. The security of a *large* group should not depend on its weakest member(s). If the keying material of a user is intentionally or unintentionally exposed, the security of the group should not be compromised in that only a small fraction of the recipient group should be affected by the exposure.

3.1 Scalability

Naturally, while offering security services, the multicast spirit needs to be preserved: the amount of multicast data sent by the source should be independent of the group size. This also means that the cost in space and processing of the security services at each receiver should be constant, and therefore not correlated to the group size.

Suivo Mitra has described [6] two main scalability pitfalls in multicast security:

1. The "one affects all" failure which occurs when the action of a member affects the whole group.
2. The "one does not equal n" failure which occurs when the group cannot be treated as a whole but instead as a set of individuals with competing demands.

As noted in [6], JOIN and LEAVE procedures are candidates for exhibiting such failures, if they are not carefully designed. In a straightforward multicast security protocol where each member M_i of the group has an individual key K_i used for the distribution of the group key K, at the departure of a member, the security conditions introduced in section 3 would require a new key K' to be distributed to all remaining members. This simple scenario illustrates both scalability failures: the departure of one member affects the entire group through the key update procedure and the remaining members of the group must be treated individually during the key update, since the new group key K' must be placed in a separate envelope encrypted under the key distribution key (K_i) of each remaining member. As highlighted in this example, multicast security requirements naturally call for solutions that conflict with scalability.

3.2 Conflicting Requirements

Extending the security requirements of unicast with the ones due to scalability and group dynamics, the requirements of a security protocol for data confidentiality in a dynamic multicast group can be summarized as follows:

1. Data confidentiality: the protocol should be immune to eavesdropping.
2. JOIN and LEAVE security: a new (*resp. old*) member should not have access to past *(resp. future)* data exchanged by the group members.
3. Containment: the compromise of one member should not cause the compromise of the entire group.
4. Static scalability: the processing load supported by an individual component (be it the source, an intermediate forwarding component or a recipient) should be independent of the group size.
5. Dynamic group scalability: the actions performed by an individual component should not affect the group as a whole.
6. Transparent group scalability: the group should not require to be treated as a set of distinct individuals.

At first glance these points seem to offer nests for contradictions. For example, points 3 and 5 call for the clustering of group members into different subgroups with different security parameters. However points 4 and 6 require the group to be treated as a whole. More generally, it's easy to see that the source and the recipients have opposite requirements. Scalability requires the source to consider the entire group as a single entity whereas security requires each recipient to be treated individually.

Many multicast or group security schemes have been proposed that all satisfy the first requirement. However they differ widely from one another on the remaining requirements: [1] does not make provisions for containment nor join and leave security, [9][10] do not offer scalability in large groups. Only [6] and [12] seem to address both security and scalability requirements in large dynamic groups.

3.3 Motivation for the Proposed Multicast Security Framework

The main motivation of the solution proposed in this paper is to solve the basic conflict between scalability and security akin to multicast security in order to come up with a solution that can scale up to large networks. We suggest that the conflict between scalability and security can be overcome by involving the intermediate components of the multicast communication in the security process.

Intermediate components, be they network nodes, routers, or application proxies, are inherent participants in the basic multicast transmission process. The key scalability factor in the basic multicast transmission schemes is the spread of the multicast routing and packet forwarding load over a network of intermediate nodes. Placing security mechanisms on existing intermediate components seems to be a natural extension of existing multicast protocols. Moreover, partitioning the cost of security mechanisms over the intermediate components appears to be a good way of assuring scalability. When the multicast group grows, new intermediate components are added to support new group members and the cost of security mechanisms can still be equally distributed by placing the additional security processing load due to the new members on the new intermediate components.

The involvement of intermediate components in the security process is also a premise for meeting multicast security requirements. If the security mechanisms can be made dependent on the intermediate component in which they are implemented, group members attached to different intermediate components can be treated independently or with different keying material. In addition to its relationship with the group membership, the keying material can have a relationship with the topology of the multicast network. The keying material associated with each group member can thus be a function of the intermediate component to which the member is attached. This topological dependency assures the containment of security exposures: if some keying material belonging to a group member attached to an intermediate node is compromised, this keying material cannot be exploited by recipients attached to other intermediate nodes.

We introduce our solution in two steps: first, we define a general framework for multicast data confidentiality based on distributed mechanisms involving intermediate components and preserving the scalability and security properties, then we propose actual solutions based on cryptographic functions that comply with the framework.

4 Multicast Security Framework

The proposed multicast security framework consists of a model that is an abstract definition of the components involved in the security mechanisms and the relationship between them which is an application of the functions we defined in section 2.

4.1 Model

In the abstract definition of the framework, the components of the multicast security framework form a ***tree***. The ***root*** of the tree is the multicast source and the members of the multicast group form the ***leaves*** of the tree. The intermediate nodes of the tree - referred to as ***nodes*** - correspond to the intermediate components of the multicast communication. Like the multicast scheme itself, nodes can be implemented at the application layer or at the network layer. In the case of application layer multicast, nodes can be application proxies, such as those in a hierarchical web caching structure. In the case of network layer multicast, nodes can be intelligent routers capable of performing security operations in addition to multicast packet forwarding functions.

In further abstraction, each leaf of the tree will represent the set of group members attached to the same terminal node. In the application layer case, a leaf will delimit a sub-group of members attached to a proxy. In the network layer case, a leaf will delimit a sub-network of recipient stations attached to a router. Hence a leaf will refer to a set of multicast group members with a common attachment node in the tree.

If a set of users represented by a leaf becomes too large, the leaf can easily be subdivided into several "sub-leaves" by adding new nodes. Hence the leaf size in terms of the group members it represents is not a scalability issue for algorithms that treat a leaf as a single entity.

4.2 RPS_f's over a Multicast Tree

Next, we turn to multicast by applying the previous concept of RPS_f from section 2 over a tree as a means of performing secret transforms in multicast communications. Let S_0 be the information to be transmitted over the multicast channel by the source under confidentiality. Furthermore, S_0 might either be the actual data or an encoding thereof, if possible data values are

different from possible values that S_0 can take on from the point of view of the security algorithm.

As part of the setup for a series of secure multicast transmissions, each node N_i is assigned a secret value $a_{i>1}$. Each node is capable of performing a function f as defined in the previous section. During secure multicast transmission, upon receipt of multicast data S_j from its parent node N_j, node N_i computes $f(S_j, a_i)$, and forwards the resulting value S_i as the secure multicast data to the child nodes or the leaves.

Assuming (S^i) is a RPS_f mapped over a path from the root to a leaf on the multicast tree, the leaf will eventually receive S^i_n, which is the final term of the RPS_f. The leaves in the multicast tree bear a special role in that they are able to recover the original message S_0. Each leaf is assigned a function $h^i_{(0,n_i)}$ that allows it to compute $S^i_0 = S_0$ from $S^i_{n_i}$ since $S_0 = h^i_{(0,n_i)}(S^i_{n_i})$. On the other hand, the leaves don't use function f.

The distribution of the secret a_i values to the nodes and the reversing functions to the leafs can be assured by a central server using classical unicast security mechanisms. Because of the structure of the algorithm, the central server will need to have a precise image of the tree structure. This doesn't mean, however, that the functionality of this server cannot be distributed over several network entities.

Working example: Figure 2 depicts a simple tree with three $RPS_f's$. Looking at the path from the root to leaf 3 on figure 2, we have:

- The root computes $f(S_0, a_1)$ and sends the result to its children nodes.
- N_1 receives $S^3_1 = f(S_0, a_1)$, computes and sends $f(S^3_1, a_7)$ to N_2.
- N_2 receives $S^3_2 = f(S^3_1, a_7)$ and sends $f(S^3_2, a_8)$ to leaf 3.
- Leaf 3 receives $S^3_3 = f(S^3_2, a_8)$ and recovers the original multicast data by computing $S_0 = h^3_{(0,3)}(S^3_3)$.

4.2.1 The Join Procedure.

When a user joins a group by contacting a node, two situations can arise:

1. a leaf (sub-group) attached to this node already exists.
2. there is no leaf attached to this node prior to the current join operation.

In the former situation, a RPS_f sequence (S^i) is already mapped between the source and the members in

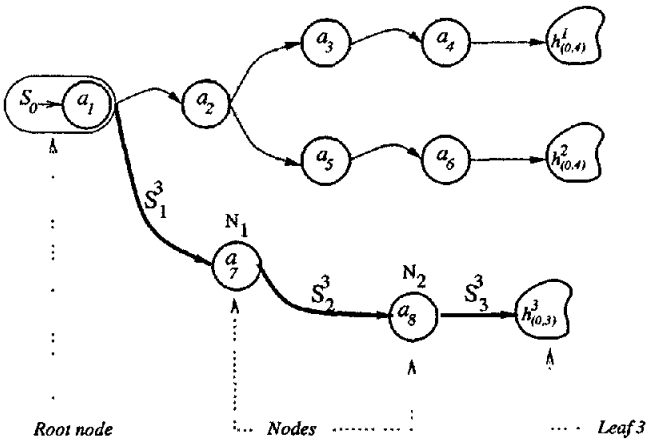

Figure 2: A simple 3 RPS_f tree.

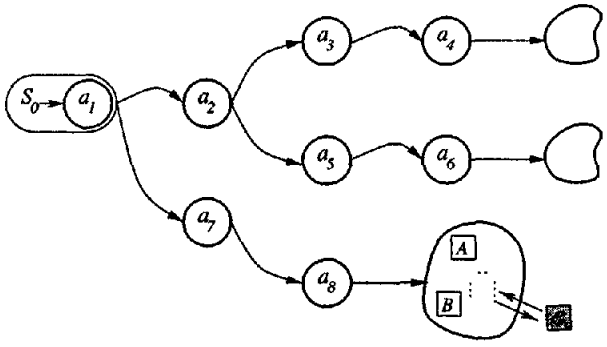

Figure 3: User C joins/leaves.

the existing leaf. The last node on the path which holds parameter $a^i_{n_i}$ will be assigned a new value $\tilde{a}^i_{n_i}$, updating the last transformation in the sequence. Hence, the corresponding new $\tilde{h}^i_{(0,n_i)}$ function will be distributed to all the member in the leaf including the new member.

In the example of figure 3 where C wishes to join the leaf including existing members A and B, the join operation will perform as follows:

1. a_8 will be substituted to $\tilde{a}_8$ in last node.
2. $h^3_{(0,4)}$ will be sent to A, B, C.

If M is the upper bound on the number of members in a leaf, a join operation requires the exchange of the following messages:

- 1 message sent to update the value in the last node on the path,
- at most $M-1$ messages sent to the current members in the leaf,
- 1 message sent to the new member.

A join operation thus requires at most $M+1$ message exchanges.

In fact, it's possible to reduce the number of messages to 3, by slightly changing the order of operations in the join procedure. Instead of changing the value $a^i_{n_i}$ in the node right away, it's possible to use the secure sequence to vehicle the new $\widetilde{h}^i_{(0,n_i)}$ function to the current members in the leaf, thus reducing the update to one message (versus an upper bound of $M-1$ messages). Then the value $a^i_{n_i}$ in the node can be changed and the new $\widetilde{h}^i_{(0,n_i)}$ function transmitted to the joining member. However this approach has a drawback: it creates a chain between the different values of $\widetilde{h}^i_{(0,n_i)}$ which potentially weakens the security of the scheme. Unless the cost of individually sending a message to each member in the leaf is more important than the security of the group, such an option should be avoided.

In the second case, the authority which receives a join request has to figure out the path from the new member to the closest node in the active tree. The path establishment method used in this case depends on the layer (application/network) at which the multicast security scheme is implemented. A similar decision has to be taken by IP multicast routing algorithms when a new router needs to be included in a multicast routing tree. Once the path to the new member is selected, the authority will assign values to the newly added nodes on the path, thus extending the RPS_f mapping. Finally the new member will receive the $h^i_{(0,n_i)}$ function needed to recover the original multicast data in the created RPS_f. Hence, he's the only member of the new leaf in the tree.

The number of messages exchanged here depends on the algorithm used to set the path between the new member and the tree. Consequently, as stated in section 3.3, this security framework would be a natural extension of multicast routing schemes. The number of messages exchanged here to create a new leaf can be assumed to be proportional to the number of messages exchanged by the multicast routing protocols when adding a new element in the multicast tree.

In many cases, it will be possible to perform the node setup ahead in time, leaving only the $h^i_{(0,n_i)}$ function to be distributed when the member effectively joins. The authority that manages the group does not need to be the root itself and its functionality can be distributed in a tree hierarchy, where each sub-authority manages a multicast subtree.

4.2.2 The Leave Procedure.

The leave procedure is similar to the join procedure. When a user leaves a leaf in the tree, the value in the terminal node is changed from $a^i_{n_i}$ to $\widetilde{a}^i_{n_i}$ and the new $\widetilde{h}^i_{(0,n_i)}$ function is distributed to the remaining nodes in the tree. In effect, the associated RPS_f sequence has its last term changed.

4.3 Evaluation of the Framework

The previous discussion has focused on the use RPS_f to achieve data confidentiality over a multicast tree. This section will show how the RPS_f construct meets the requirements established in 3.2, assuming that the intermediate nodes are trusted and secure. The implications of node compromise will be discussed in the next section.

4.3.1 Data Confidentiality

A secret message x transmitted by the source cannot be retrieved from the multicast data obtained by intruders eavesdropping on any of the links of the secure multicast tree. Retrieving x from multicast data exchanged on an intermediate link would require the computation of the inverse of f which, by definition, is computationally infeasible. Retrieving x from the multicast data transmitted to a leaf over the last hop of a multicast path is also impossible because the secret reversing function $h^i_{(0,n_i)}$ cannot be retrieved without the knowledge of at least the secret a_i values assigned to the nodes included in the path. In addition, if the multicast security scheme is based on an APS_f even the disclosure of the a_i values would not compromise data confidentiality as discussed is section 4.4. Data confidentiality is an obvious consequence of the way the model was defined. The degree of security of the one-way function f and should be evaluated on a per algorithm basis as in section 6 and 7.

4.3.2 JOIN and LEAVE Security

A new member joining a leaf gets a new reversing function $\widetilde{h}^i_{(0,n_i)}$ that cannot be used to recover the old reversing function $h^i_{(0,n_i)}$. As a consequence, past data is not accessible to a new member. Similarly, a former member using an old reversing function cannot access data that is transmitted subsequently to its departure.

4.3.3 Containment

Because of the topological dependency introduced by the model, the reversing function $h^i_{(0,n_i)}$ used in a leaf of the tree will be useless outside that leaf. An intruder will only benefit from an attack if he is located in the same leaf as the victim. This greatly reduces the impact of member compromise.

4.3.4 Static Scalability

The amount of processing per component is independent of the group size. First, in our framework, the size of messages transmitted by a node (be it the source or an intermediate component) does not depend on the number of group members but it depends only on the

size of the original secret message. Second, the number of messages transmitted by a node does not depend on the number of group members but it depends only on the number of child nodes attached to this node.

4.3.5 Dynamic Group Scalability

There are three basic actions a group member can perform, namely join, leave and receive data. The model is designed so that none of these actions affects the whole group. In fact these actions have an impact that is limited to the leaf containing the member performing these actions as shown in section 4.2.1. The "one affect all" type failure never appears.

4.3.6 Transparent Group Scalability

The "1 does not equal n" type of failure never appears over the group as a whole, instead, it is confined to the leaves in which join or leave operations occur. Since the leafs have a maximum size, this is not a scalability issue. All other operations, including re-key, address the group as a whole.

4.4 Node Compromise.

The previous section assumed that the nodes of the tree were completely secure. This has to be true for the root node of the tree but it might not be possible to make such an assumption about the intermediate nodes in the network. Hence the following section will focus on the impact of intermidiate node compromise. Two type of attacks that derive from node compromise are highlighted in this section: unauthorized membership extension and mode compromise by external users. Unauthorized membership extension happens when a former member of the secure group is able to maintain access to the data even though he has not received the new reversing function. Node compromise by external users more generally describes unauthorized access to the group by users that never became group members.

4.4.1 Unauthorized Membership Extension

If a member *Eve* in a leaf controls the last node on the path from the source to the leaf i, he can intercept changes in the last parameter of the RPS_f.

Let N be the last node on the path from the root to leaf i of the tree and a the secret held by N. N receives S^i_{j-1} from its parent node and sends $S^i_j = f(S^i_{j-1}, a)$ to the leaf elements which will use a $h^i_{(0,j)}$ reversing function to recover S_0. If the group membership manager decides that *Eve* must leave from the group, the value a in N will be changed to a new value $\widetilde{a}$ and the corresponding reversing function $\widetilde{h}^i_{(0,j)}$ will be send to all leaf members except *Eve*.

Despite its formal exclusion from the group, *Eve* can ignore the change in the router and compute S_0 from S^i_{j-1} using a and the old reversing function $h^i_{(0,j)}$ obtained trough the compromise of node N, simulating the older sequence, where $S_0 = h^i_{(0,j)}\left(f(S^i_{j-1}, a)\right)$.

This attack works whatever the nature of the sequence, APS_f or SPS_f, but requires several conditions to be met:

1. *Eve* should be a former member of the group.
2. *Eve* should be able to access the secret parameter held by node N.
3. *Eve* should have access to the data transmitted to node N by its parent node (i.e. S_{j-1}).

Moreover, updates of a_i values in nodes at a higher level will limit the scope of this attack because the resulting reversing functions cannot be retrieved based on the information gathered in a leaf or from the compromise of the last node.

A SPS_f specific attack Condition 2 described in the previous paragraph is not required if the sequence is a SPS_f. To work around condition 2, the intruder first computes:

$$\widetilde{h}^i_{(j-1,j)} \text{ form } \widetilde{a}$$

because the sequence is symmetric.
Now, using $\widetilde{h}^i_{(j-1,j)}$ and the new sequence value $\widetilde{S}^i_j$ received in the leaf, the former member computes:

$$S^i_{j-1} = \widetilde{h}^i_{(j-1,j)}(\widetilde{S}^i_j)$$

Next, the former member uses the value a obtained through the compromise of N to compute:

$$S^i_j = f(S^i_{j-1}, a)$$

Finally, using the old reversing function $h^i_{(0,j)}$ and applying it to S^i_j, we have:

$$S_0 = h^i_{(0,j)}(S^i_j)$$

where S_0 is the original multicast data.

This attack doesn't apply in an APS_f tree because by definition $\widetilde{h}^i_{(j-1,j)}$ cannot be deduced form $\widetilde{a}$.

4.4.2 Node Compromise by External Users.

If the intruder *Eve* is not even a former member of the group, an attack is still a possible if the sequence is symmetric provided that:

1. *Eve* has access to the value of the reversing function used by a legitimate member .
2. *Eve* control all the nodes on the path between him[1]and the legitimate member except the first common ancestor they have in the tree.

[1] *Eve* does not have to be in a real leaf, he can simply intercept traffic somewhere in the tree

If these conditions are met, *Eve* will be able to forge a reversing function he can use to access the group.

Instead of a lengthy formal discussion, we chose to illustrate the attack with the example scenario depicted on figure 6, where the malicious user *Eve* gets multicast data S_3^2 from node N_5. If *Eve* knows $h_{(0,3)}^3$ from a compromised user and $\{a_2, a_5, a_7, a_8\}$, he can compute:

$$S_2^2 = h_{(2,3)}^2(S_3^2)$$

because $h_{(2,3)}^2$ can be derived from a_5. Similarly,

$$S_1^3 = S_1^2 = h_{(1,2)}^2(S_2^2)$$

because $h_{(1,2)}^2$ can be computed from a_2. Then

$$S_2^3 = f(S_1^3, a_7)$$

and

$$S_3^3 = f(S_1^3, a_8)$$

yielding to

$$S_0 = h_{(0,3)}^3(S_3^3)$$

Again, this attack doesn't apply to an APS_f based tree because reversing functions associated with an APS_f cannot be derived from the parameters used in the intermediate nodes.

4.4.3 Node Compromise Summary

The distinction between an APS_f and a SPS_f is totally relevant with respect to node compromise scenarios. Unlike [6], when using $APS_f's$ our framework is immune to node compromise by external users. The framework does not however dictate the choice of an APS_f over SPS_f as one could expect because SPS_f are likely to be easier to design than APS_f.

It should be noted that the security containment property is also effective in case of node compromise. Hence, previously described node compromise scenarios don't allow the intruder to provide unauthorized access to just any other user in the network.

This section concludes the formal presentation of our secure multicast framework. The next sections present two implementations of this framework based on extensions of public key cryptographic schemes. The first scheme is an SPS_f and will therefore lend itself to further description of a concrete node compromise scenario.

5 Key Distribution

Depending on the performance of function f our framework can be used either for bulk data confidentiality or only for key distribution. Current symmetric cryptographic systems provide sufficient encryption speed but they don't exhibit the mathematical properties required to create a RPS_f. On the other hand asymmetric cryptography offers suitable properties to build a solution compliant with the framework but it doesn't offer yet the necessary performance for bulk data confidentiality. Consequently, the next two sections will describe of the framework based on asymmetric cryptography for multicast key distribution. The first scheme, derived from the El Gamal encryption algorithm, allows the creation of a SPS_f key distribution tree, whereas the second scheme, based on RSA, effectively creates an APS_f key distribution tree.

Using a RPS_f tree, the source can distribute a secret key k by initialising the sequences with $S_0 = k$ (or otherwise a function of k). The data confidentiality mechanism of the secure multicast framework will allow to securely transmit k to the members of the group. The source can frequently update k but, unlike the reversing function that is different in each leaf, k is shared among all members of the group so the exposure of k affects the group as a whole. However, unlike the reversing function that enables each member to access the multicast group, the shared key k is a short term value that can be frequently updated by the source using the secure multicast framework. Consequent values of k are independent.

6 Key Distribution using the Discrete Log.

The discrete log problem used in the El Gamal cryptosystem can be used to create a RPS_f. Let p be a large prime and let f, the node operation, be defined as $f(x, a) = x^a \bmod p$. If $y = f(u, v)$ it is computationally infeasible to compute v from (u, y).

6.1 Setup

The source of the node chooses a generator g of the cyclic group $\mathbb{Z}_p^*$ and a secret random value r in $\mathbb{Z}_{(p-1)}^*$. The nodes and the root are assigned $a_{i>0}$ values[2] in $\mathbb{Z}_{(p-1)}^*$. The initial value of the sequence is set to $S_0 = g^r \bmod p$.

Let $\{S_{i_k>0}\}$ denote the sequence elements. The reverse function distributed to the nodes is defined as:

$$h_{(0,n_k)}^k(x) = x^{(a_{i_1} a_{i_2} a_{i_3} \ \ a_{i_{n_k}})^{-1}} \bmod p$$

[2]Naturaly, we should avoid the unlikely case that $\prod_k a_{i_k} = 1$ on the path from the root to the i^{th} leaf

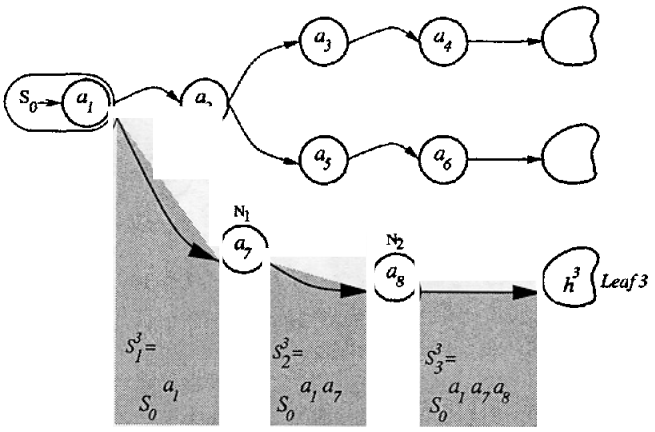

Figure 4: A discrete log tree.

The function f performed in each node is defined as $f(x, a_{i_k}) = x^{a_{i_k}} \bmod p$.

6.2 Key Distribution

The source wishing to distribute a key K sends the following initial data to its children in the tree:

$$S_1 = (S_0)^{a_1} \bmod p$$

$$T = K \oplus S_0$$

The intermediate elements in the tree perform f on their input S_{i_k-1} and send S_{i_k} to their children, along with T, where:

$$S_{i_k} = f(S_{(i_k-1)}, a_{i_k}) = \left(S_{(i_k-1)}\right)^{a_{i_k}} \bmod p$$

An example of this scheme is illustrated on the path from the root to the leaf 3 of the tree on figure 4:

- The source send $S_1^3 = (S_0)^{a_1} mod p$ and $T = K \oplus S_0$ to its children.
- N_1 receives $(S_1^3.T)$ and sends $S_2^3 = (S_0)^{a_1 a_7} \bmod p$ and $T = K \oplus S_0$ to its children.
- N_2 receives (S_2^3, T) and sends $S_3^3 = (S_0)^{a_1 a_7 a_8} \bmod p$ and $T = K \oplus S_0$ to leaf 3.

6.2.1 Decryption

The decryption process is straitforward, the reverse function h is simply applied to the received value, and the result is used to extract K from T:

$$h_{(0,i_k)}(S_{i_k}) = S_0 \bmod p$$

$$K = T \oplus S_0$$

Recalling the previous example, where $h^3(x) = h^3_{(0,3)}(x) = x^{\frac{1}{a_1 a_7 a_8}} \bmod p$, the value of K is computed simply:

$$K = T \oplus S_0$$

where

$$S_0 = h^3\left(S_3^3\right) = \left((S_0)^{a_1 a_7 a_8}\right)^{\frac{1}{a_1 a_7 a_8}} \bmod p$$

6.2.2 The Next Key

The next key to be sent $\widetilde{K}$ only requires $S_0 = g^r \bmod p$ to be changed to a new $\widetilde{S_0} = g^{\widetilde{r}} \bmod p$ in $\mathbb{Z}_p^*$. The initial value of the sequence is changed for every message. It should be noted here that if g is a generator of $\mathbb{Z}_p^*$ then g^r and $g^{\widetilde{r}}$ are also a generators of the cyclic group because r and $\widetilde{r}$ are invertible [7] in $\mathbb{Z}^*_{(p-1)}$. Transitively, this means that all elements in the created RPS_f sequence are generators. This assures a constant strength of the transformations in the RPS_f which is a sequence of generators of $\mathbb{Z}_p^*$.

6.3 Node Compromise and Member Collusion

Many of the requirements established in section 3.2 are naturally fullfilled by implementing the framework as described above. However member collusion and node compromise need to be considered on a per-algorithm basis.

6.3.1 Node Compromise

The previously described sequence is clearly a SPS_f because the reversing functions can be computed with the knowledge of the secret parameters in the nodes. Hence, compromise of the nodes offers some potential for unauthorized membership extension as described in 4.4.

Figure 5 will illustrate the node compromise scenario. The hypothesis here will be that a malicious member E of leaf 3 wishes to maintain membership in the group using the information of the terminal node N_2 he has compromised.

In a normal scenario where node compromise is not taken into account, in a leaf consisting of members $\{A, B, C, E\}$, when E leaves, the following actions take place:

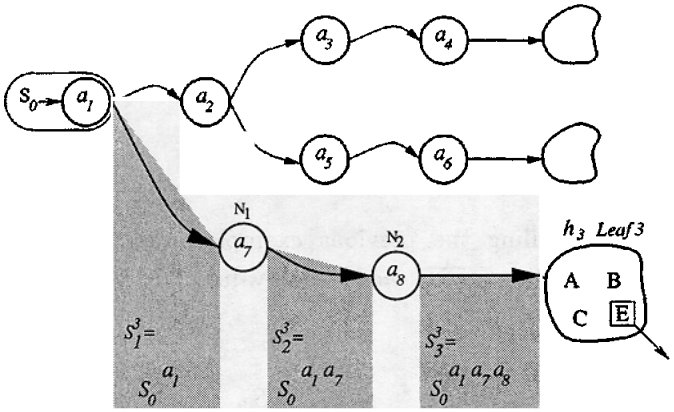

Figure 5: Node Compromise

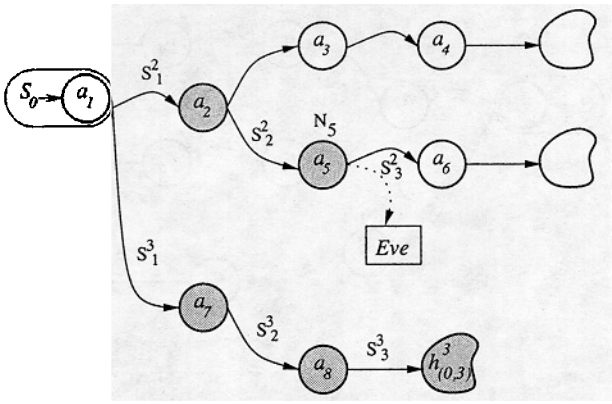

Figure 6: Multiple node compromise attack.

- In N_2, a_8 is changed to $\widetilde{a}_8$.
- The newly computed reverse function $\widetilde{h}^3_{(0,3)}$ is sent to $\{A, B, C\}$ but not E.

Once the leave procedure is complete, E cannot access further keys distributed to $\{A, B, C\}$ it cannot derive $\widetilde{h}^3_{(0,3)}$ from $h^3_{(0,3)}$.

However, in the case of node compromise, if E controls the last node, he can monitor the change from a_8 to $\widetilde{a}_8$. E can then derive $\widetilde{h}^3_{(0,3)}$ from $h^3_{(0,3)}$, because if $h^3_{(0,3)}(x) = x^{(a_0 a_1 a_7 a_8)^{-1}} \ mod\ p$ then:

$$\widetilde{h}^3_{(0,3)}(x) = x^{\frac{1}{a_0 a_1 a_7 a_8} \times \frac{a_8}{\widetilde{a}_8}} \ mod\ p$$

In summary, even if E doesn't receive the new reversing function, he will be able to compute it and thus access the keys distributed subsequently to the leave operation.

This attack can be extended to allow a malicious user to derive a reversing function from another one even if the reverse function comes from another leaf. It requires the attacker to compromise nearly all the nodes on the graph between him and the compromised member.

Figure 6 will serve as an example where user E -not a member of the group- listens to traffic coming out of N_5. The malicious user is assumed to know the following node parameters $\{a_2, a_5, a_7, a_8\}$ as well as $h^3_{(0,3)}$ from a compromised member in leaf 3. With these conditions together, E can compute a new local reverse function $h^2_{(0,3)}$ from $h^3_{(0,3)}$ thus breaking the clustering appeal of the model:

$$h^3_{(0,3)}(x) = x^{\frac{1}{a_1 a_7 a_8}} \ mod\ p$$

which allows to compute:

$$h^2_{(0,3)}(x) = x^{\frac{1}{a_1 a_2 a_5}} \ mod\ p = \left(h^3_{(0,3)}(x)\right)^{\frac{a_7 a_8}{a_2 a_5}} \ mod\ p$$

This second attack assumes that the nodes are easy to compromise, and the first one makes strong assumptions about the compromise power of the attacker. While these attacks on SPS_f's might be considered hard to implement in some cases, the possibility itself pushed us to study a second and stronger construction based on APS_f's, as described in section 7.

6.3.2 Member Collusion

A set of colluding members could compare their reversing functions to try to extract additional information. Distributing a reversing function of the form $h(x) = x^{y_i} \ mod\ p$ actually consists of distributing the exponent $y_i = (a_{i_1} a_{i_2} ... a_{i_n})^{-1} \ mod\ (p-1)$. A possible collusion attack would aim at deriving some individual sequence values a_i based on the knowledge of $\{y_1, y_2, ..., y_n\}$ by the colluding members. A trivial scenario in which the collusion attack can succeed occurs if at least one sequence is included in another. This can easily be prevented if none of the terminal nodes for one sequence is used as an intermediate node for another sequence.

7 Key Distribution using RSA.

Extending RSA to use multiple keys as in [5] allows the creation of an APS_f scheme. As in RSA, let $n = pq$ where p and q are carefully chosen large primes. The node function is defined as $f(x, a) = x^a \ mod\ n$.

7.1 Setup

The setup is even simpler here than in the discrete log case. Each node in the tree is assigned a value $a_{i>1}$ and the root uses a_1 where $\gcd(a_{i \geq 0}, \varphi(n)) = 1$. This assures that the product A of any subset of these a_i values also verifies $\gcd(A, \varphi(n)) = 1$. The multiplicative inverse B of A defined as $AB \equiv 1 \ (mod\ \varphi(n))$ can be computed using the Euclidean algorithm.

Let $\{a_{i_k>0}\}$ denote the set of parameters used in the nodes between the source and leaf k, plus $a_{i_1} = a_1$ in the root. The reversing function distributed to the nodes is defined as:

$$h^k_{(0,n_k)}(x) = x^{D_k} \ mod \ n$$

where,

$$(a_{i_1}.a_{i_2}...a_{n_k}).D_k \equiv 1 \ (mod \ \varphi(n))$$

Like the basic RSA algorithm, the security of this scheme relies on the difficulty of factoring n, that is, computing D_k requires the knowledge of $\varphi(n)$ which currently seems to be only derivable from the factors of $n = pq$.

7.2 Key Distribution

The source wishing to distribute a key K, sends the following value to its children in the tree:

$$S_1 = K^{a_1} = (S_0)^{a_1} \ (mod \ n)$$

Each node N_i in the secure multicast tree processes the S_{i-1} value received from its parent node and sends S_i to its children nodes where:

$$S_i = f(S_{i-1}, a_i) = (S_{i-1})^{a_i} \ (mod \ n)$$

Recalling figure 5 while assuming an RSA like APS_f sets the following scenario on the path from the root to leaf 3 of the tree:

- The root send $S^3_1 = (S_0)^{a_1} \ mod \ n$ to its children.
- N_1 receives S^3_1 and sends $S^3_2 = (S_0)^{a_1 a_7} \ mod \ n$ to its children.
- N_2 receives S^3_2 and sends $S^3_3 = (S_0)^{a_1 a_7 a_8} \ mod \ n$ to leaf 3.

7.2.1 Decryption

The decryption process is also simpler than in the discrete log case. The decryption function h is applied to the received value in the leaf to recover K. For example, on figure 6:

$$K = S_0 = h^3_{(0,3)}(S^3_3) = ((S_0)^{a_1 a_7 a_8})^{D_3} \ mod \ n$$

assuming

$$a_1 a_7 a_8.D_3 \equiv 1 \ (mod \ \varphi(n))$$

7.2.2 The Next Key.

Sending a new key $\widetilde{K}$ only requires S_0 to be changed in the preceding description. Nothing else needs to be done.

7.3 Node Compromise and Member Collusion

The node compromise attack previously described in section 6 regarding the discrete log case does not apply here essentially because the RSA based sequences are asymmetric: to compute a reversing function $h_{(i,j)}$, the knowledge of the intermidiate parameters $\{a_{i+1}...a_j\}$ wouldn't be sufficient as $\varphi(n)$ is also required. However the node compromise attack based on membership extension in section 4.4 is still possible with the RSA based scheme.

Possible collusion scenarios do not lend themselves to the leakage of any secret information or capacity.

8 Related Work

Some other papers have presented scheme that address some of the requirements hightlighted in section 3.2. However, only [12] and [6] seem to address both scalability and security. Hence we will focus our comparison on those two schemes, which differ from ours in mainly two areas: containment and trust. Moreover we will look at the particular implication of using our scheme for key distribution.

Trust Though our solution uses intermediate components, it has a major difference with [6]: our framework does not put any trust in the intermediate components, whereas in [6] each intermediate component has access to the multicast data. This problem does not appear in [12] since no intermediate elements are involved. Hence even though we use intermediate elements, our scheme is equivalent to [12] in terms of trust.

Containment In terms of containment, our scheme is equivalent to [6], where each subgroup uses a different key to access the multicast data. On the other hand [12] does not address containment issues even though it uses a tree structure. In that scheme, the keys held by any user can be used to access the multicast group anywhere and all users are equivalently trusted with the security parameters of the group.

Key distribution Though we offer higher security in terms of trust and containment, this has a cost. Indeed, [6] and [12] have a clear advantage over our scheme in terms of performance. This has lead us to consider our scheme for key distribution and not bulk data encryption. In that respect the framework is used to distribute

a short term data encryption key k. As this short term key is common to all recipients, it may look as our scheme looses its containment advantage over [12]. However, the short term key can be frequently updated and its disclosure does not provide a means of long term group access to intruders. This is because in our scheme the group membership is represented by the long term reversing functions that are different in each leaf of the multicast tree as opposed to the shared secret group membership key of [12].

9 Conclusion

This paper has presented a framework designed to support data confidentiality in a large dynamic multicast group. The framework meets a set of requirements wider than the previous work. While covering scalability, the new concept of containment was introduced as we believe the latter is a key requirement in very large groups.

The introduction of Reverse Parametric Sequences, or RPS_f, permits a formal but yet practical description of the framework elements, with a voluntary distinction between symmetric and asymmetric behaviors. The mapping of these sequences over a multicast tree is the core mechanism that allows this framework to meet the previously described requirements.

Next, two key distribution schemes have been exposed as implementations of the framework. Further detailed studies of the those two schemes, that would each deserve a complete paper, will be necessary before a concrete implementation. Nevertheless these two schemes have served as a proof of concept for the framework and they have allowed us to discuss the implication of various node compromise scenarios, as the possibility of node compromise cannot be neglected in a large multicast network.

The next major step would be the design of efficient functions that could be used to build $RPS_f's$ that operate on bulk data, in order to fully capitalize on this framework. Beyond just multicast, such functions will have applications in many group security problems.

References

[1] Tony Ballardie, "Scalable Multicast Key Distribution", RFC 1949, may 1996.

[2] Tony Ballardie and Jon Crowcroft, "Multicast-Specific Security Threats and Counter-Measures", The Internet Soc. Symposium on Network and Distributed System Security, February 16-17, 1995, San Diego, California.

[3] Jan Camenish and Markus Stadler, "Efficient Group Signature Schemes for Large Groups", Advances in Cryptology - CRYPTO'97, 1997.

[4] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", Advances in Crytology - CRYPTO'84, Santa Barbara, California, USA, 1984.

[5] Lein Harn and Thomas Kiesler, "Authenticated Group Key Distribution Scheme For A Large Distributed Network", Symposium on Security and Privacy, 1989.

[6] Suivo Mitra, "Iolus: A Framework for Scalable Secure Multicasting", In Proceedings of the ACM SIGCOMM'97, September 14-18, 1997, Cannes, France.

[7] Neal Koblitz, "A course in Number Theory and Cryptography", Springer-Verlag, 1994.

[8] R. L. Rivest, A. Shamir, L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21(2):120-126, 1978.

[9] Michael Steiner, Gene Tsudik, Michael Waidner, "Diffie-Hellman Key Distribution Extended to Group Communication", in Proceedings of the 3rd ACM Conference on Communications Security, March 14-16, 1996, New Delhi, India.

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

[11] Debby M. Wallner, Eric J. Harder, Ryan C. Agee, "Key Management for Multicast: Issues and Architectures", Internet draft, Network working group, september 1998.

[12] C. K. Wong, M. Gouda, S. S. Lam, "Secure Group Communications Using Key Graphs", Technical Report TR 97-23, University of Texas at Austin, July 1997.