DEA Training period report.
DEA RSD - Juin 1998

Scalable key distribution for secure multicast.

Alain pannetrat - DEA RSD
Ecole Superieur des Sciences Informatiques (ESSI)

Supervisor : Pr. Refik Molva
Eurecom

Abstract

This report presents and analyzes possible solutions for key distribution in a multicast group. Unlike, two-party security that is a well studied field, multiparty communications offer some interesting problems open for research. For key distribution in a multicast group, the biggest problem we have holds in one word : scalability.

Scalable multicast protocols call for a security scheme that avoids the sharing of secret keying material among the recipients. In the proposed model the source transmits a unique message on the multicast tree but each recipient or sub-group of recipients uses different keying material to get accross the protected multicast channel.

We suggest several protocols based on extensions of classical cryptographic functions. The first one is a simple XOR scheme and the two other schemes are based respectively on ElGamal and RSA.

Résumé

Ce rapport présente et étudie des solutions possibles pour la distribtution de clé dans un groupe multicast. En effet, si on maîtrise bien la sécurité des communications biparties, les communications multiparties offre un certain nombre de problèmes ouverts à la recherche. Pour la distribution de clé dans un groupe multicast, le problème principal tiens en un mot : scalabilité.

Les protocoles multicasts offrant cette scalabilité nécessitent un shéma de sécurité où l'on évite de partager une unique clé entre tous les membres du groupe. Dans le modèle proposé la source transmet un message unique dans l'arbre multicast, mais chaque récepteur ou chaque sous groupe de récepteurs utilise une clé différente pour accéder au flot multicast.

Nous suggérons un certain nombre de protocoles basés sur des extensions de fonctions cryptographiques classiques. Le premier est basé sur une simple opération XOR, les deux autres sur ElGamal et RSA.

Contents

Acknowledgments

I would particularly like to thank Pr. Refik Molva for giving me the chance to work in computer security and for always keeping his office door open to me. I would like to thank Sergio for taking the time to share some ideas and letting me have access to the "cookbook" [1]. My thoughts also go the cool german speaking guys in the lab, the cheerfull doctorants and all the staff at Eurecom.

Eurecom's research is partially supported by its industrial members: Ascom, Cegetel, France Telecom, Hitachi, IBM France, Motorola, Swisscom, Texas Instruments, and Thomson CSF.

Introduction

As "groupware", "multicast" or other group oriented protocols get more popular, there is a growing need to implement security in a multiparty context. This is particularly true in open environments such as the Internet. Multiparty security may involve confidentiality, integrity, authentication, non-repudiation and also new specific problems. The Corporate Communication department at Eurecom is naturally showing a great interest in that area which will probably be the opportunity of great developments in modern open networks.

The original subject of this internship started as an open theme : "multiparty security". It was norrowed down to multicast key distribution aspects after some interesting ideas seemed to emerge through discussions and graffiti on the drawing board. Nevertheless partial work also arose in the more generic field of group security as shown in Appendix.

This report is divided in three parts.

The first part reviews multiparty security in general, giving some ideas of how different it can be from two-party security. Focus is put on group security and some specific problems related to key distribution in multicast groups are highlighted. Scalability issues are identified as the central problem.

The second part concentrates on the core problem statement, giving the objectives that should be fullfilled to do key distribution in a multicast group. A partitioning model designed to cope with the scalability issues is presented. It is highly correlated to the multicast routing tree structure.

Finally, in the third part, solutions are presented with some analyses. First a XOR pad scheme is used. Its natural weakness motivates the study of one way functions based on simple extensions to classical public key cryptosystems such as ElGamal and RSA. The ElGamal scheme has the added feature of source authentication while the RSA scheme offers a higher degree of security.

Part I
Background

1 Mutiparty security

Intuitively one might think that most group security protocols could just be extensions of well known two-party protocols. This is not always true. First of all, in the two-party case it's easy to define what a session is, at the security level. It starts when the two end users have setup their security parameters to communicate with each other and ends when one of the two participants exits from the session. In the multiparty case, things are different, the definition of a session depends much more on the context. For example, a protocol may require a fixed number of participants or a dynamic number of participants. In these two different cases, the meaning of a session is different.

Some protocols must be redefined in the multiparty case. A good example is group signatures. If someone signs a document electronically on behalf of a group, should a trusted party be able to identify the signer in case of a problem our should the protocol be absolutely anonymous even to the group members ?

Hence adapting, creating and evaluating protocols for the multiparty case offers plenty of problems to explore.

2 Multicast groups versus ordinary groups

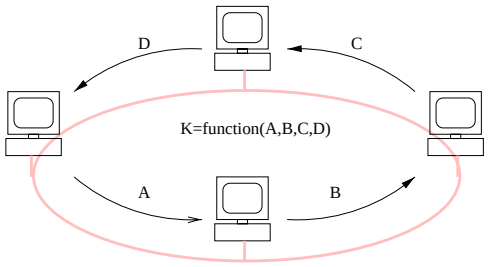
Multicast is now clearly in the mainstream focus. Popular computer magazines feature articles about it and as it gets a wider audience, need for security will probably arise. In the extreme case we can imagine a multicast pay-per-view TV, which will require the multicast stream to be protected from users that are not registered by the broadcaster. Ideally we would like to provide confidentiality services for multicast groups. There are protocols to achieve this in groups but they do not appear to be adaptable to the multicast case.

2.1 Classical group security

Group security schemes usually come in two flavors : distributed or centralized. In the distributed case, all users exchange messages to setup the parameters needed to perform the security objectives. Authority-managed groups involve a trusted host that manages the security of the group.

To offer confidentiality, the goal of such procedures is to do key distribution or key agreement. This often means that all the users that want to do secure group communications agree on the same key or share some common piece of information.

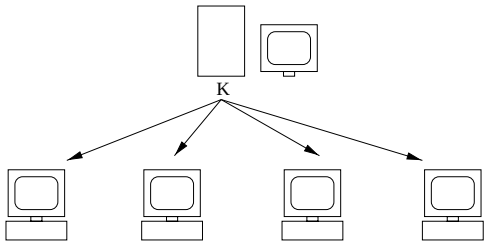
Key agreement. In the distributed case, members of a group generally arrange in a "virtual ring" and pass around information needed to create a common session key. A good example of key agreement is the extention of the Diffie-Hellman scheme to groups described in [10] and [11].



(a) Distributed key agreement.

Adding or removing a member often means a new computation round.

Key distribution. In an authority centered model, a trusted host distributes session keys to members wishing to communicate together.



(b) Authority managed key distribution.

Here, if adding a member is not too much of a problem, removing one still means redistributing a new key to all remaining members.

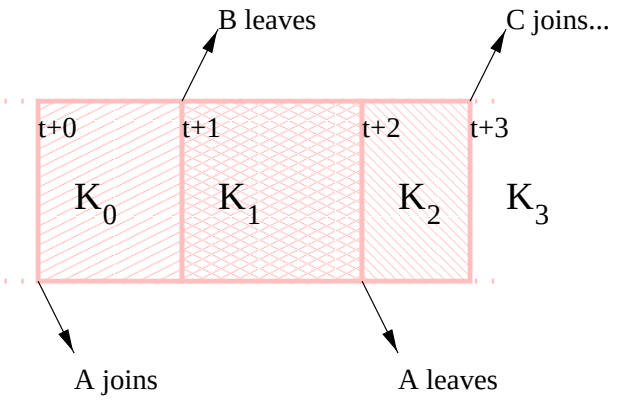
Related work. Appendix A describes a method to create a group public key system. As an aside to our work we have shown how to transform the original authority-managed scheme into a completely distributed scheme.

2.2 The characteristics of a multicast group

A good source to apprehend the specific requirements of multicast security is in [7]. We will recall some of the concepts defined there with a few remarks while focussing on key distribution. Two other good ressources are [13] and [12].

First of all, multicast in inherently less secure than unicast, because there is no control on who can subscribes to a group. Interception of traffic is made a lot easier. Access control has been proposed in [8] extending IGMP, but this will clearly not provide enough security in a large group, specially in terms of confidentiality. On the other hand access control might partially prevent denial of service, by stopping or at least reducing unwanted traffic from polluting a multicast datastream.

The second characteristic of a multicast group is its dynamism. As we previously stated, in the unicast case, a session is clearly defined. The key used between two users is generally the same from the beginning to the end of the session. The situation is completely different in a multicast group, if users are expected to join and leave the group at any time. In fact we can view JOIN and LEAVE operations as a time slicing procedure of a secure multicast group.



(c) Key change enduced time slicing.

If a user leaves a group, he should not have access to future communications. Therefore, the key should change upon departure of a user. This is the "one affects all" core problem of multicast key distribution. This also holds most of the time when a user joins the group : he should not have access to past data. Hence the keying material should change on every JOIN/LEAVE operation.

The third obvious characteristic is that the size of the multicast packets should not grow with the size of the multicast group. In fact this means that the security payload added to the multicast data should be independent of the group size. The example in Appendix A is clearly not a candidate. We cannot simply encrypt the time slice session key with each member's public key in the group or have a key that grows with member addition.

Trying to satisfy the requirements of a multicast group key distribution clearly bumps on scalability issues. Using traditional group approaches is not possible. Having the source send a new key to every user each time a user joins or leaves will be to demanding on network and computing ressources as the group increases in size. Instead if we choose a distributed approach, every user relies on some other users. If one user "crashes" in the middle of the key change every one is affected. Moreover, the key agreement procedure will have a time complexity in an order of the group size.

Finally, one should note that the wider the group gets, the higher the chances of key exposure. It would be interesting to lower the power of the keying material a user has. For example, if the key changes very often, it might prove harder for an attacker to successfully attack a host several times to keep up with the keying material. But again this puts scalability issues back on the drawing board.

Part II
Problem Statement

For simplicity, we will restrict the problem to a multicast group with one unique source. The source will control the key distribution process and the membership of the participants. The source's functionality can probably be distributed but we will refer to the source as if it was a single entity.

3 Objectives

This section describes the objectives of key distribution in a multicast context.

Security overhead : The security overhead added to multicast packets should not be related to the number of effective members in the group.

Partitioning : Users should be grouped in independent sets each using a different key. The extreme case being *single user* sets. The idea is that a key used in a set should not be usable outside that set therefore giving the keying material lower power.

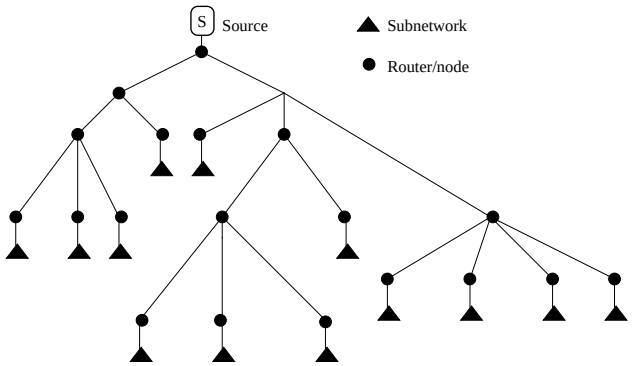
The JOIN/LEAVE operations should not bear the "one affects all" scalability issue described in section 2.2. Partitionning effectively reduces the problem to a small set of users.

Collusion proof : Users that collude together across different sets should not be able to discover more information than each other's secrets, neither should they be able to maintain membership after a LEAVE. This may be the hardest property to prove.

Extensibility : Ideally we would like our system to provide some extra feature, for example source authentication.

4 Model

To try to achieve the previously described goals we will consider the structure of the multicast network. Indeed such a structure can be viewed as a virtual tree. The source is the root, the nodes are routers and the leafs are networks with a real or simulated multicast capability.



(d) The tree model.

The users in the same leaf - or subnet - will be considered part of the same set : they will share the same key. This seems to be logical since the source has no power over what happens in a local area network.

Part III
Solution

The solutions we present are based on a quite simple idea : the nodes in the multicast tree participate in the encryption process. If each node operates on the encryption process in a different way, the keys used by the users in the leafs of the tree will be different from one another.

We will present 3 solutions. The first one, based on the XOR operation is more a study case than a real secure key distribution system. Nevertheless it demonstrate some of the basic principles used in all the solutions we present. The two other solutions are based on multiple key extentions to classical public key cryptosystems. We will actually present the basic mathematical principles needed to fully understand the way these two systems work.

We will assume throughout the discussion that provisions are made to secure the unicast traffic described in the key distribution systems below. We also assume that the security messaging used between the different agents of the systems is done in a reliable way.

5 The XOR tree.

The first idea was to use the exclusive or operation, denoted "$\oplus$". The XOR function offers perfect secrecy as follows : if $C = A \oplus B$, then if A or B is random, the result C is also random. The XOR has also the appealing property that it's really fast and easy to implement. Unfortunately, as one can probably guess, it's not secure in respect to the objectives we established in our model.

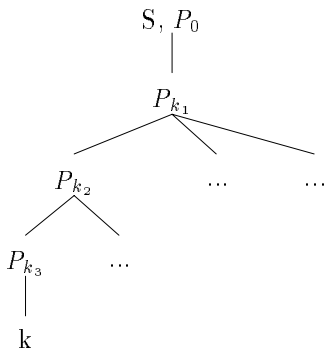
5.1 Description

Before focussing on the setup phase, we will suppose the elements of our network are setup correctly and we will describe how the system works.

Suppose the source S wants to send a message M to q selected terminal networks in our broadcast tree. The tree has n routers -or nodes- which are assigned a piece $P_{1 \leq i \leq n}$. The source itself chooses the piece P_0. These pieces should have a bit size greater or equal to the expected messages like M. The keys the end users get are the XORed together values of the pieces used in nodes along the path from the source to themselves. This always includes P_0. In other words, if $P_0, P_{k_1}, P_{k_2}, ..., P_{k_r}$ represent the $r+1$pieces used along the path from the source to subnetwork k, the key D_k used will be:

$$D_k = P_0 \oplus P_{k_1} \oplus P_{k_2} \oplus ... \oplus P_{k_r}$$

We can illustrate this simply:



On this drawing, $D_k = P_0 \oplus P_{k_1} \oplus P_{k_2} \oplus P_{k_3}$.

The way the system works is straightforward. The source send $M_0 = M \oplus P_0$. Each router along the path uses it's piece to XOR it over the message:

$$M_{j+1} = M_j \oplus P_{k_j} = M \oplus P_0 \oplus P_{k_1} \oplus P_{k_2} \oplus ... \oplus P_{k_{(j-2)}} \oplus P_{k_{(j-1)}}$$

In the end the final user recovers the data by computing $M = M_r \oplus D_k$, because what we essentially have is $M_r = M \oplus D_k$.

5.2 Setup phase.

The setup phase is simple here. Each time a node wants to attach itself to the secured broadcast tree, it chooses a random value which is the piece P_i described above. The end user who wishes to join the group goes through a tree stage dialogue with the source :

1. Source $\xrightarrow[multicast]{[M]_?}$ user : once the user has subscribed to the group he starts receiving data $[M]_? = M \oplus P_0 \oplus ...$, but he doesn't have a key to decrypt it.

2. User $\xrightarrow[unicast]{JoinRequest}$ source : the user asks for a key D_k, providing the necessary information and credentials. He sends back the encrypted message $[M]_?$ he previously received.

3. Source $\xrightarrow[unicast]{JoinAccept}$ user : the user gets $D_k = [M]_? \oplus M$ that the source computed from the previous message. This key should be sent in a secure way from the source to the user.

After this, the end user can decrypt the messages that come from the source. This setup process has the great advantage that the source does not have to be aware of the exact topology of the network. It does not even need to know what value each node or router has chosen.

The source still has to maintain a list of all users connected to the group. This is a logical assumption if we want to achieve confidentiality. Each user should be identified at least by a network address. As we will see it's also a good idea to keep track of what subnetwork or in other words in what leaf of the tree the user is. Note that all users that identify themselves to the source from the same leaf k should share the same D_k.

5.3 Discussion

5.3.1 Group dynamics.

One of our objectives was that the keys should change upon JOIN/LEAVE operations. When a user joins or leaves a subnet, the source can easily send a message asking the gateway router of the terminal network to change the key it uses. Since it maintains a database entry describing what users are attached to that gateway, it can easily send the new key to the remaining members of the group.

If we want to be able to change the pieces used in a node at a higher level in the tree, we will need to have a precise idea of the routing tree, to detect which users need a key update. It would probably not be acceptable to let the users detect themselves that they are not receiving data properly.

5.3.2 Security overhead

One nice aspect of the XOR function -in fact its greatest advantage- is that its cost is very low. Resources assigned to security functions in the source, routers and terminal networks are small.

5.3.3 Confidentiality

Direct observation of traffic by an external observer does not leak any information about the message if the XOR pad is used only once. If it is used many times, some form of cryptanalysis could damage the scheme. To counter this we have to make sure that the message is random. This is generally true if the message is itself a random symmetric cipher key used to encrypt data. Instead of directly distributing data, we can use the XOR scheme to distribute the DES key used to encrypt data. Such encrypted data doesn't even need to travel in the same multicast stream.

On the other hand, observation of traffic flowing in and out of a router allows someone to find out the piece used in the router. This allows someone to monitor key change in a router and if he already is a member he can easily maintain membership after a key change :

1. Suppose a member observes traffic in a router and collects the piece P_{k_r} used in the router. Then from D_k he can compute $P_0 \oplus P_{k_1} \oplus P_{k_2} \oplus ... \oplus P_{k_{(r-1)}} = D_k \oplus P_{k_r}$.

2. If the source wants the member to leave, it asks for a piece change in the router which the member can deduce by observation. P_{k_r} becomes $P\prime_{k_r}$ and the member can compute $D\prime_k = (P_0 \oplus P_{k_1} \oplus ... \oplus P_{k_{(r-1)}}) \oplus P\prime_{k_r}$, thus maintaining membership.

5.3.4 Partitioning and collusion

The keys used by each subset even though different from each other, can be used to forge keys anywhere in the multicast tree. To be precise, anyone that steals a (M, D_k) pair can find the key D_l used in his network.

1. The user receives $M \oplus P_0 \oplus P_{l_1} \oplus ... \oplus P_{l_s}$ which he can't understand but he stores it and steals the pair (M, D_k) from another network.

2. He computes $D_l = (M \oplus P_0 \oplus P_{l_1} \oplus ... \oplus P_{l_s}) \oplus M$, then he can maintain membership if needed using the technique described in the previous paragraph.

Thus partitioning offers no security here. One can easily see that a user can help anyone to join the group. The key generation process is transparent to members.

5.4 Conclusion for the XOR scheme.

Even though it's not secure, the XOR tree has given a general idea of how we can achieve our objectives. This leads us to consider functions that are more complicated than XOR. In fact we would like to use functions that are not reversible.

6 Mathematical considerations.

Before trying to move to a more complex ciphering process, we will look at some of the mathematics commonly used in cryptography.

6.1 Modular arithmetics

This is not meant to be exhaustive, for more details refer for example to [6]. We quickly presents some fundamental properties we will use.

6.1.1 Exponentiation.

The Euler φ function. Let n be a positive integer, then $\varphi(n)$ is the number of nonnegative integers less than n which are prime to n.

Moreover, the Euler φ function has the following two properties :

1. If $a > 0$ and $b > 0$ are relatively prime to each other then $\varphi(ab) = \varphi(a)\varphi(b)$.
2. If $n = p^a$, where p is prime, then $\varphi(n) = \varphi(p^a) = p^a - p^{a-1}$.

This allows us to calculate $\varphi(n)$ for any integer n, as long as we know how to factor n.

Residue classes. Most algorithms in public key cryptography, work in a set S of integers modulus n, usually noted $\mathbb{Z}/n\mathbb{Z}$ or $\mathbb{Z}_n$. This set forms a commutative ring with operations such as addition and multiplication, but only elements that are relatively prime to n have a multiplicative inverse. If b is an element of S, we naturally define modular exponentiation as :

$$b^i \, mod \, n = 1 \times \underbrace{b \times b \times ... \times b}_{i \, times, \, i \geq 0} \, mod \, n$$

The exponents express themselves modulus $\varphi(n)$. In other words, if we work in $\mathbb{Z}_n$ then the exponents take their values in $\mathbb{Z}_{\varphi(n)}$.

Generators. Let n be a positive integer. Let the set $< g >= \{g^i \, mod \, n, \, i \in \mathbb{Z}\}$ where $g \in \mathbb{Z}_n$, if $< g >= \mathbb{Z}_n^*$ then g is called a generator of $\mathbb{Z}_n$.

If $n = p^q$ where p is prime, then $\mathbb{Z}_n$ is a field, and in that case, there exists at least one generator g of $\mathbb{Z}_n$. In general we will work in $\mathbb{Z}_p$, where p is prime.

Generators are interesting when they serve as the base of an exponentiation because the number of possible values is as big as the field in which we are working. If the exponent serves as a key in a cryptosystem, the larger the keyspace the better it is.

6.1.2 Discrete logarithms.

Let n be a positive integer - usually prime - calculating the following expression is easy :

$$y = b^i \, mod \, n$$

In fact, it's usually considered that the time complexity of this is something like $O(log^3 n)$. The inverse problem, is conjectured to be a hard problem. Knowing y and b, it is considered computationally hard to find i.

This problem also called the Diffie-Hellman assumption serves as a base for the ElGamal cryptosystem and several others.

ElGamal. Let g be a generator of the finite field $\mathbb{Z}_p$, where p is a large prime. The p and g values are publicly known to everyone. Bob chooses a random value a to be his private key, and publishes his public key E :

$$E = g^a \, mod \, p$$

If Alice wants to send a message m to Bob she chooses a random value k and computes :

$$c_1 = mE^k \, mod \, p = mg^{ak} \, mod \, p$$

$$c_2 = g^k \, mod \, p$$

To recover the message Bob uses his secret a :

$$m = \frac{c_1}{(c_2)^a} \, mod \, p$$

We will exploit this scheme by using a greater number of exponents.

6.1.3 Composite moduli

This time, let's pick n as the product of two large primes p and q. Knowing those two primes it's easy to compute n. On the other hand factoring n to p and q is conjectured to be a hard problem.

If n is chosen as described above, exponentiations will take place in $\mathbb{Z}_{\varphi(n)}$. Suppose we choose a positive integer e which is prime to $\varphi(n)$. In other words, we choose e so it has a multiplicative inverse in $\mathbb{Z}^*_{\varphi(n)}$. This is the foundation of the RSA public key cryptosystem. Bob can compute an integer d where $d.e \equiv 1 \, mod \, \varphi(n)$. He then publishes his public key :

$$E = (e, n)$$

If Alice wants to send a message m to to Bob she computes :

$$c \equiv m^e \, mod \, n$$

Bob can recover m using his private key d :

$$m \equiv c^d \equiv m^{(e.d)} \, mod \, n$$

We will use this scheme by extending it to a number of exponents greater than two.

6.2 Prime fields of interest

Suppose we want to do several exponentiation like the ones described in 6.1.1. Hence we would like to build a sequence G defined as : $G_{i+1} = G_i^{x_i} \, mod \, n$. But we want to keep the order of every term in the sequence high. Ideally, we would like them to be all generators of $\mathbb{Z}_n$.

To make things clearer, we will take a look at $\mathbb{Z}_{11}$. Let $A \in M_{10}(\mathbb{Z}_{11}^*)$ be the matrix defined as : $A[i,k] = i^k \ mod \ 11$:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 5 & 10 & 9 & 7 & 3 & 6 & 1 \\ 3 & 9 & 5 & 4 & 1 & 3 & 9 & 5 & 4 & 1 \\ 4 & 5 & 9 & 3 & 1 & 4 & 5 & 9 & 3 & 1 \\ 5 & 3 & 4 & 9 & 1 & 5 & 3 & 4 & 9 & 1 \\ 6 & 3 & 7 & 9 & 10 & 5 & 8 & 4 & 2 & 1 \\ 7 & 5 & 2 & 3 & 10 & 4 & 6 & 9 & 8 & 1 \\ 8 & 9 & 6 & 4 & 10 & 3 & 2 & 5 & 7 & 1 \\ 9 & 4 & 3 & 5 & 1 & 9 & 4 & 3 & 5 & 1 \\ 10 & 1 & 10 & 1 & 10 & 1 & 10 & 1 & 10 & 10 \end{bmatrix}$$

This matrix allows us to follow the sequence G through it's different values. For example we could have $G_0 = 2$, $G_1 = 2^3 = 8$, $G_2 = 8^5 = 10$, $G_3 = 10^2 = 1$ and $G_4 = G_5 = ... = G_k = 1$. It is understandable that a value such as 1 or even 10 should be avoided to do exponentiation sequences in $\mathbb{Z}_{11}$. In order to keep things secure, the number of possible values G_{i+1} we can reach from G_i has to remain big enough so the discreet log remains difficult to compute. Stationary sequences are thus not interesting.

If we take a look at A we can see that the positions in which 10 and 1 appear are not random. Intuitively they must be a relation between this distribution and the numbers we are using.

Hence this is a motivation to look more carefully at some prime fields.

6.2.1 Basic properties

Let's first state two classical properties in finite fields [6].

Property 6.2.1.1 The elements of $\mathbb{Z}_m$ which have multiplicative inverses are those which are relatively prime to m, i.e. the numbers a for which $\gcd(a, m) = 1$.

Property 6.2.1.2 If g is a generator of the finite field $\mathbb{F}_m$, then g^i is also a generator if and only if $\gcd(i, m-1) = 1$. In particular, there are a total of $\varphi(m-1)$ generators of $\mathbb{F}_m^*$.

NOTE : Finite fields have a characteristic that is always the power of a prime. In other words, if $\mathbb{F}_m$ is a field then $m = p^q$ where $p \geq 0$ is prime. Here we will focus of fields $\mathbb{F}_m$ where $m = p$ is prime, but it might be interesting to extend the following remarks to any fields.

6.2.2 Germain Prime fields

We call p a strong prime if $p = 2q + 1$, where q is also an odd prime. Strong primes are also called Germain primes, from the mathematician Sophie Germain.

Property 6.2.1.2 enables us to compute the number of generators of a field $\mathbb{F}_p$ where p is a strong prime :

$$\varphi(p-1) = \varphi(2q) = q - 1 = \frac{p-3}{2}$$

More generally, if $p = 2^t q + 1$, where q is an odd prime and $t \geq 1$, we can compute the number of generators in a field $\mathbb{F}_p$ the same way :

$$\varphi(p-1) = \varphi(2^t) = 2^{(t-1)}(q-1) = \frac{p-1-2^t}{2}$$

Thus, the lower t is, the greater the number of generators of $\mathbb{F}_p$ will be.

6.2.3 Fermat primes.

A Fermat is a prime of the form $2^{(2^r)}+1$. The demonstration is omitted here, but applying the Miller-Rabin primality test to $p = 2^k + 1$ shows us that p can't be prime unless $k = 2^r$. In other words, if $p = 2^k + 1$ is prime, it's a Fermat prime. These numbers seem to be rare.

Using property 6.2.1.2, we compute the number of generators of the field $\mathbb{F}_p$, where $p = 2^k + 1$:

$$\varphi(p-1) = 2^{k-1} = \frac{p-1}{2}$$

This is even better than with strong primes. Unfortunately, such prime numbers have certain weaknesses that moderate their use in cryptography. This is why we will use strong primes.

6.2.4 Other primes.

We took a look at prime numbers of the form $2^k q^r + 1$, where $(k, r) \in \{\mathbb{Z}^2 - (0,0)\}$. Any other prime number p can be written as :

$$p = 2^{t_0} q_1^{t_1} q_2^{t_2} ... q_n^{t_n} + 1 = 2^{t_0} \prod_{i=1}^{n \geq 2} q^{t_i}$$

where $q_1, q_2 ... q_n$ are the n odd prime factors of $\frac{p-1}{2^{t_0}}$, and $1 \leq t_{0 \leq i \leq n}$. Here we assume $n \geq 2$ since the case where $n = 0, 1$ have been studied previously.

If we evaluate $\varphi(p-1)$ we have :

$$\begin{aligned} \varphi(p-1) &= \varphi(2^{t_0}) \textstyle\prod_{i=1}^{n \geq 2} \varphi(q_i^{t_i}) \\ &= 2^{(t_0-1)} \textstyle\prod_{i=1}^{n \geq 2} (q_1^{t_1} - q_1^{(t_1-1)}) \end{aligned}$$

Since $1 \leq t_{0 \leq i \leq n}$, we have $1 \leq q_i^{(t_i-1)}$, thus we can write :

$$\varphi(p-1) \leq 2^{(t_0-1)} \prod_{i=1}^{n \geq 2} (q_i^{t_i} - 1)$$

And in turn we can write :

$$\varphi(p-1) < 2^{(t_0-1)} \left(\prod_{i=1}^{n \geq 2} q_i^{t_i} - 1 \right) \leq 2^{(t_0-1)} \prod_{i=1}^{n \geq 2} q_i^{t_i} - 1$$

Which enables us to compare with p :

$$\varphi(p-1) < \frac{p-1}{2} - 1$$

Which brings us to a comparison with strong primes :

$$\varphi(p-1) < \frac{p-3}{2}$$

Combining this result with the previous ones allows us to make further conclusions.

Lemma 6.2.4.1 If p is an odd prime number, the finite field $\mathbb{F}_p$ has at most $\frac{p-1}{2}$generators. Further more, this limit is reached if and only if p is a Fermat prime.

Lemma 6.2.4.2 If p is a prime number and if $p-1$ has at least one odd factor, then the finite field $\mathbb{F}_p$ has at most $\frac{p-3}{2}$generators. Further more this limit is reached if and only if $p = 2q+1$, where q is an odd prime number.

6.3 Generator sequences.

We will now use the preceding results to build a sequence of generators in a field $\mathbb{Z}_p$ where p is a Germain prime.

6.3.1 The exponentiation subgroup.

Let H be a set defined as $H = \{c \in \mathbb{Z}_{2q}, \gcd(c, 2q) = 1\}$. The elements of H are the elements $c = 2i+1$, where $i \in \{0..q-1\}$ that are different from q (i.e. $i \neq \frac{q-1}{2}$). In fact H has $q - 1 = \frac{p-3}{2}$ distinct elements since $\varphi(2q) = \varphi(2)\varphi(q) = q-1$.

Proposition 6.3.1.1 H is a subgroup of $\mathbb{Z}^*_{2q}$.

Proof. We have $1 \in H$ and from property 6.2.1.1, we know that if $a \in H$, $a^{-1} \in H$. Now suppose $a, b \in H$. Then if $ab \equiv q \, mod \, 2q$, we can write $a = qb^{-1} mod \, 2q$ (or $b = qa^{-1}$) which contradicts the fact that $\gcd(a, 2q) = 1$. Thus if $a, b \in H$, we have $\gcd(ab, 2q) \neq q$. Similarly, if we have $ab \equiv 2i \, mod \, 2q$, we have the same type of contradiction, thus $\gcd(ab, 2q) \neq 2$. This leads to $\gcd(ab, 2q) = 1 : ab \in H$.

6.3.2 Consequences in cryptography.

Let g be a generator of $\mathbb{Z}^*_p$ and $(r_k)_{0 \leq k \leq n}$ any sequence of numbers from H. A sequence S defined as :

$$\begin{cases} S_0 = g \, mod \, p \\ S_k = (S_{k-1})^{r_{k-1}} \, mod \, p \end{cases}$$

forms a set of generators of $\mathbb{Z}^*_p$. This is just a consequence of property 6.2.1.2. And since H has a group structure, if $j > i$ we have :

$$S_i = (S_j)^{\frac{1}{r_i r_{i+1} \cdots r_{j-2} r_{j-1}}} \, mod \, p$$

and in particular:

$$g = (S_j)^{\frac{1}{r_0 r_1 \cdots r_{j-1}}} \, mod \, p$$

This construction involving strong primes is interesting at the cryptographic level for several reasons, relating to the Diffie-Hellman assumption[1] :

[1] See 6.1.1.

- Finding the discreet log of S_i to the base S_{i-1} in $\mathbb{Z}_p^*$ involves $q-2$ possible values if we don't consider "1" as a candidate. This is similar in terms of complexity to finding the discreet log in $\mathbb{Z}_q^*$.
- If n bits are required to represent the values of $\mathbb{Z}_p^*$, the number of bits needed to represent the exponents (i.e. the log key space) is $n-1$.
- If p is large, then *p-1* has a large factor q which makes the discreet log problem more secure (relatively to classical discreet log algorithms).
- All the exponents have a multiplicative inverse.

The strength of the construction of the sequence S in the field $\mathbb{Z}_p^*$ is directly correlated to $\varphi(p-1)$ which is the number of generators of $\mathbb{Z}_p^*$.

6.3.3 Using other primes.

Ideally, we should work in $\mathbb{Z}_p^*$ where p is a Fermat prime to maximize the amount of generators, but since $p-1$ is a power of 2, it will allow someone to find the discreet logarithm in $\mathbb{Z}_p^*$ using the Silver-Pollig-Hellman algorithm for example.

Prime fields $\mathbb{Z}_p^*$, where p is neither a strong or a Fermat prime, are less interresting. This is a direct consequence of lemma 6.2.4.1 and 6.2.4.2.

7 The DH tree.

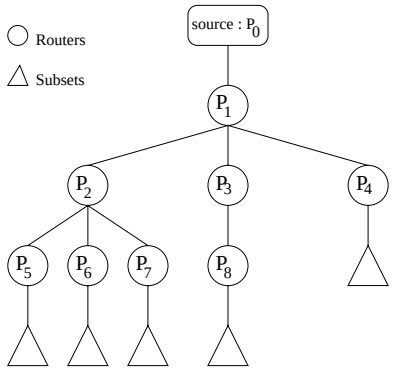
Now that we have enough information to build more complicated systems, we will first look at "Diffie-Hellman" trees or "DH tree".

Each node will perform an exponentiation in well chosen prime field $\mathbb{Z}_p$. The corresponding group H as described in 6.3.1 will simply be denoted H_p in the following sections.

7.1 Description

7.1.1 The simple case

As in the XOR tree, we will suppose that there are q terminal network, in a tree with n nodes. The source wants to send a message M to the group to achieve confidentiality services.



(e) The DH tree setup...

The source assigns to each router a piece $P_i \in H_p$ chosen at random and picks a value $P_0 \in H_p$ for itself. The source has a precise idea of the tree construction, so it knows the values assigned to each node that is on the way from itself down to the terminal set of users in a common subnetwork. If the users in the terminal network k are separated from the source by r nodes using the pieces P_{k_1}, P_{k_2}, ... , P_{k_r}, we compute the decryption key D_k used in the subnet :

$$D_k \equiv \frac{1}{P_0 \times P_{k_1} \times P_{k_2} \times ... \times P_{k_r}} mod\,(p-1)$$

Calculating the inverse is guaranteed by the fact that we have a group structure in H_p.

The source wishing to send M down to the group will choose a random value $v \in H_p$ and compute :

$$Y_0 \equiv g^{vP_0}\ mod\, p$$

$$Z \equiv M g^{v}\ mod\, p$$

Where g is a generator that can remain secret because it isn't needed outside the source. Every router along the path will perform an exponentiation on Y_i :

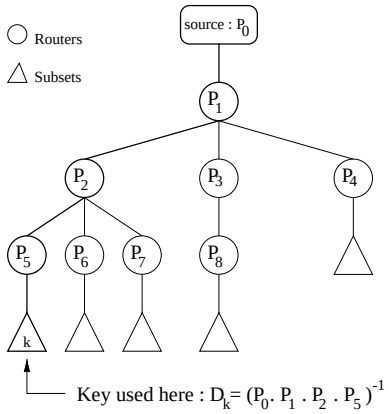
$$Y_{i+1} \equiv (Y_i)^{P_{k_i}}\ mod\, p$$

The final user will recover M using his key D_k :

$$Z' \equiv (Y_r)^{D_k} \equiv g^{v(P_0 . P_{k_1} . P_{k_2} ... P_{k_r})D_k} \equiv g^{v}\ mod\, p$$

$$M \equiv Z/Z'\ mod\, p$$

To take the same example as in the XOR tree we have :



(f) Key composition in the DH tree.

The key used in subnetwork k will simply be $D_k \equiv \frac{1}{P_0 P_1 P_2 P_5} mod\,(p-1)$.

7.1.2 Generalization

The previous case can be generalized, giving each router $s+1$ pieces instead of just one. For mathematical reasons $s+1$ must be an odd number to make sure[2] $D_k \in H_p$. Every router i gets a set $\{P_{i,0},\ P_{i,1},\ \ldots, P_{i,s}\}$ of elements of H_p. The source gets the corresponding $\{P_{0,0},\ P_{0,1},\ \ldots, P_{0,s}\}$ set. The decryption key D_k that is used by the users in the same terminal subnetwork k separated from the source by r nodes is (considering $P_{0,i} \equiv P_{k_0,i}$) :

$$D_k \equiv \left[\sum_{i=0}^{s}\left(\prod_{j=0}^{r} P_{k_j,i}\right)\right]^{-1} \ mod\ (p-1)$$

The source wishing to send a message M to the group will choose a random v and compute

$$\begin{aligned} Y_{0,0} &\equiv g^{v P_{0,0}} \ mod\, p \\ Y_{0,1} &\equiv g^{v P_{0,1}} \ mod\, p \\ &\vdots \\ Y_{0,s} &\equiv g^{v P_{0,s}} \ mod\, p \end{aligned}$$

$$Z \equiv M g^{v} \ mod\, p$$

Each router performs exponentiation on the set $\{Y_{i,j} \mid 0 \le j \le s\}$:

$$\begin{aligned} Y_{i+1,0} &\equiv (Y_{i,0})^{P_{k_i,0}} \ mod\, p \\ Y_{i+1,1} &\equiv (Y_{i,1})^{P_{k_i,1}} \ mod\, p \\ &\vdots \\ Y_{i+1,s} &\equiv (Y_{i,s})^{P_{k_i,s}} \ mod\, p \end{aligned}$$

The end user will recover M by computing :

$$Z' \equiv \left(\prod_{j=0}^{s} Y_{r,j}\right)^{D_k} \ mod\, p$$

$$M = Z/Z'$$

The trade-off of more exponents per node is the need for more ressources. But more exponents gives a higher degree of security, as we will see.

7.2 Meeting the objectives.

7.2.1 Group dynamics

This scheme adapts to group dynamics in a straitforward way. Each time a new subnet is connected to the multicast group, the source makes sure that all the nodes along the path have pieces and it sends the new user(s) the appropriate key.

[2] To be more precise, in H_p where $p = 2q+1$, the only odd number that is not in H_p is q.

A requirement was that upon the JOIN and LEAVE of a member, the key in the group must change locally. This simply means that when a JOIN/LEAVE operation is performed, the piece in the gateway router of the subnetwork has to be changed. Once that is done, the source can send the new key to the members of the subset that are allowed to stay in the group, using for example their individual public keys to encrypt the key D_k.

Therefore, the JOIN/LEAVE operation avoids the "one affects all" syndrome, limiting the affect to the subset of destinations which are in the same subnetwork.

7.2.2 Confidentiality

Someone observing traffic needs to find g^v to discover the content of the message. Since v is secret, this is impossible. In fact, since g does not need to be public the problem seems somewhat harder than cracking the ElGamal cryptosystem.

7.2.3 Securtity overhead

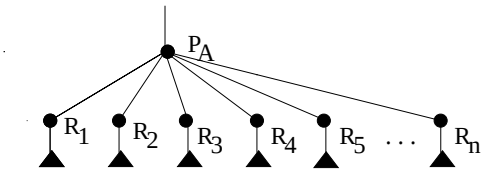
The amount of data multicast by the source is clearly independent from the number of users in the group. On the other hand the unicast payload is proportional to the JOIN/LEAVE frequency. This scheme does not cope well with massive JOIN or massive LEAVE. This is not be a problem for application which tolerate some delay in JOIN and LEAVE operations.

7.2.4 Partitioning

Each terminal network is separated by a different set of nodes. Since the key used is the product of random numbers, the chances that two keys are the same is very low. This means that if a user in a terminal subgroup k gets his key D_k stolen, it will be useless outside the subgroup.

7.2.5 Collusion

Users who collude together across subnets cannot infer any valuable information. In fact in the simplest case, if n users are behind n different gateways with a common ancestor in the router tree hierarchy :



(g) A simple case : routers with a common ancestor

In which case the n users hold the $X_{1 \leq i \leq n}$ following values which are obtained by calculating the inverse of the user keys :

$$X_1 = P_A P_1$$
$$X_2 = P_A P_2$$
$$\vdots$$
$$X_n = P_A P_n$$

In other words, we have n equations with $n+1$ unknowns. Since the values of P_i are independent there is no way of giving another equation that would link two values P_i and P_j where $i \neq j$. Therefore collusion between users is apparently fruitless.

On the other hand a router attack can prove more interesting (see 7.3 for more details).

7.2.6 Extensibility

We can extend our protocol to add source authentication. The users will be able to verify that the source is the originator of the multicast encrypted message M. This is done by proving knowledge of discreet logs in a way vaguely similar as described in [9].

Suppose the source publishes $g^s \ mod\, p$ where s is a secret value from H_p. The source give the end users the following key in a very similar way as before :

$$D_k \equiv \frac{1}{P_0 \times P_{k_1} \times P_{k_2} \times ... \times P_{k_r} + s} mod\ (p-1)$$

Someone receiving D_k has apparently no way to extract s from it. The source computes $w \in H_p$ a publicly known function of $hash(M)$, for example $w = 2 \times hash(M) + 1 \, mod\, (p-1)$, and sends :

$$W = g^{ws} \ mod\, p$$

$$Y_0 \equiv g^{wP_0} \ mod\, p$$

$$Z \equiv M g^w \ mod\, p$$

The Y_i follow a similar destiny as described before. The final user recovers M by computing :

$$Z\prime \equiv (W \times Y_r)^{D_k} \equiv g^{w(P_0 P_{k_1} P_{k_2} ... P_{k_r} + s) D_k} \ mod\, p$$

$$M \equiv \frac{Z}{Z\prime} \ mod\, p$$

The receiver can verify that the source of the M is the same that distributed D_k by computing w and verifying that :

$$W \equiv (g^s)^w \ mod\, p$$

Of course, authentication can be done using DSS or ElGamal native signatures. Another alternative could be to use signature proofs of knowledge as described in [3]. But those solutions will require an extra cost in terms of computing ressources.

7.3 Attacks

As we have seen, stealing a user's key has a limited impact. The stolen key will only be valid in a little portion of the multicast tree. But there still is another possible attack that can allow a member to remain in the group after a LEAVE operation. To achieve such an attack, a member will need to take control of the router to monitor the values selected by the source for exponentiation in the router. Suppose the following scenario in a subset of user $\{A, B, C\}$:

1. A malicious user A takes control the router. Therefore if his user key is D_A he knows $D_A = (PR)^{-1} mod\ (p-1)$ where R is the exponent used in the router.
2. The source decides A should not be a member of the group anymore, therefore it changes R to $R\prime$ in the router and sends users B and C the new key : $D\prime_B = D\prime_C = (PR\prime)^{-1} mod\ (p-1)$.
3. The user A can forge his own $D\prime_A = (PR)^{-1} \times R \times (R\prime)^{-1} mod\ (p-1)$

This attack can be made harder by using several exponents in the router as described in 7.1.2. If a router does m exponentiations, using m different exponents it will take m interceptions of router key change to discover enough information to allow a user to maintain membership after the $(m+1)^{th}$ router key change. This is true provided no parent router in the hierarchy changes its key in the mean time.

To illustrate this we will take a triple exponentiation as a working example. We will omit the moduli here since they are clearly "$mod\ (p-1)$". We suppose A a malicious user uses the key $D_{A,1} = (P_1 R_{1,1} + P_2 R_{1,2} + P_3 R_{1,3})^{-1}$ where the P_i values come from the parent router in the hierarchy and the $R_{j,i}$ are the j^{th} set of exponents used in the router. Well if A views 3 different key changes, he has to solve the following system of three unknowns (P_1, P_2, P_3) :

$$
\begin{aligned}
D_{A,1} &= (P_1 R_{1,1} + P_2 R_{1,2} + P_3 R_{1,3})^{-1} \\
D_{A,2} &= (P_1 R_{2,1} + P_2 R_{2,2} + P_3 R_{2,3})^{-1} \\
D_{A,3} &= (P_1 R_{3,1} + P_2 R_{3,2} + P_3 R_{3,3})^{-1}
\end{aligned}
$$

Solving such a system is not difficult. The result will be omitted here since it's a rather complex expression. Nevertheless now that A knows $\{P_1, P_2, P_3\}$ he can forge $D_{A,4} = (P_1 R_{4,1} + P_2 R_{4,2} + P_3 R_{4,3})$ intercepting $\{R_{4,1}, R_{4,2}, R_{4,3}\}$.

All in all this shows that besides the source of course the routers are the weak point of the system in regard of membership enforcement. Correcting this is the base of the RSA tree idea.

8 The RSA tree.

The RSA tree is based on the same basic mathematical structures used in RSA. In fact a similar idea is used in [5] to manage large groups.

8.1 Description

Let n be the product of two large primes p and q. Let G_n the group of elements of $\mathbb{Z}_{\varphi(n)}$ that are relatively prime to $\varphi(n) = [(p-1)(q-1)]$, i.e. the elements of $\mathbb{Z}_{\varphi(n)}$ that have an inverse. Now suppose a similar setting as in the DH tree except for a change in the mathematical structures. We take P_0 in the source and all the P_i values in the routers as elements of G_n. The D_k key distributed to users in terminal networks is calculated using the same writing conventions as in the DH tree :

$$D_k \equiv \frac{1}{P_0 P_{k_1} P_{k_2} ... P_{k_r}} \, mod \, \varphi(n)$$

The source willing to send a message M has even less work to do than in the DH tree. It simply sends :

$$Y_0 \equiv M^{P_0} \, mod \, n$$

Routers along the path down to the user subset k use their pieces to participate in the encryption :

$$Y_{i+1} \equiv (Y_i)^{P_{k_i}} \, mod \, n$$

The final user recovers M by computing :

$$M \equiv (Y_r)^{D_k} \, mod \, n$$

8.2 Objectives

This scheme which is quite similar to the DH scheme fullfills the same objectives. Moreover it is stronger than the previous scheme in respect to the router attacks we described. Even if a malicious user takes control of the exponentiation values used in the routers, he will not be able to make calculations with them because he cannot find $\varphi(n)$. Thus he has no idea of the mathematical structure used for exponents.

Even better : there is no reason why the source could not send the pieces belonging to the routers in clear text over the network, as long as the original P_0 remains secret.

8.3 Attacks

Giving many ciphertexts of the same message might make some form of cryptanalysis easier. It is not clear if several ciphertexts of M could allow an easier way to factor n thus breaking the encryption. Note however that the common modulus attack on RSA described for example in [1] does not apply here since P_0 is secret.

9 Future work.

9.1 Implementation issues

We have concentrated on the theoretical aspects of the DH and RSA trees but some aspects will probably need some work before an implementation can be done.

Some might not like the idea of putting the functionalities we have described in routers. Because of the complexity of some of the operations, it's true this might not be appropriate. It would probably mean that all the multicast routers should be changed. But one should note that the real strategic points in our multicast tree is the source and the gateway routers of the terminal networks. It can be considered realistic that special multicast gateway routers will be build and used for companies that need security. After all some gateway routers have firewall functions included in them.

One should also note that while we used the terms *router* and *node* interchangerably, the nodes don't have to be in fact routers. We could use what the author of [7] call agents. The source would broadcast to a specified group to which agents only would respond. These agents would apply the transformations we described in our tree models and rebroadcast the data to another multicast address. This can go on recursively to finally form a tree structure similar to what we have done with routers. This would have the added advantage to allow a more distributed control and also allow the agents to take care of reliability issues needed in our secure multicast key distribution.

Another issue is the added delay of the security operations. The exponentiation needed in RSA and DH are slow compared to most operation that would be expected to be done on network data. They are considered to be of order $O(n^3)$ bit operations relatively to the length n in bits of the keys. There are two things to consider that makes this less of a problem.

First we don't need to perform our operations on every message. Actually the source can encrypt the data it sends with a performant bloc cipher such as DES or IDEA and distribute the key with the algorithms we have described. This can be seen as some kind of indirection. Moreover in some cases, the data sent does not need to be completely encrypted. For example some satelite TV broadcastings only encrypt the horizontal synchronization descriptors in the picture. Hence the bandwith used by such satellite video traffic is wide but the decryption needed remains reasonable.

Secondly the key distribution process can often be done ahead in time. A user that is going to join a group can receive his key minutes before it becomes valid. Everything depends on the time slicing policy of the source.

Besides one should note that the key distribution data and the effective data itself don't need to use the same multicast group address if we use indirection.

9.2 Possible directions

Tamperproof hardware Making the routers more secure in the DH tree algorithm might be better achieved using tamperproof hardware. The idea would be to protect the exponent and exponentiation operation from external evesdropping. This could be done by hiding the random generator of the exponent and the exponentiation in some form of tamperproof hardware.

Super large multicast groups Another problem is related to really large groups similar to groups you can expect doing multicast pay TV. In all the previous discussions, the assumption was that the user had a minimum interest in keeping the

information he receives secret. If that isn't the case a member can quite simply rebroadcast the data to other users without encryption.

To counter this, tamper proofhardware might provide some kind of protection. But in the end it might prove impossible to stop "rebroadcasting". It will be interesting to look at way to detect this type of behavior on open networks, the same way TV broadcasting authorities detect fraudulent TV broadcasters.

Conclusion

This report has shown three algorithms to do key distribution in a multicast tree. The first one is based on the XOR binary operation while the two others are based on two classical public key algorithms that have been extended to use multiple pieces. The key idea was to let the routers -or more generally the nodes in the network- contribute to the encryption. This allows a partitioning of all the members of a multicast group and in turn it also allows to give a possible solution to the key change scalability issue.

The XOR scheme demonstrated the basic principles that can be used to achieve key distribution in a multicast context but also suffered from some clear weaknesses.

The second scheme, based on ElGamal has the added possibility to do source authentication. It partially looses it's security if the gateway nodes are corrupted. In such a case, members can maintain membership fraudulently.

The third scheme, based on RSA allows less trust in the participating nodes of the tree model, because the exponentiation values don't need to be hidden from the public. On the other hand, the mathematic structures should probably go through a deeper study to make sure the RSA model doesn't become weaker with so many users working with the same modulus.

The schemes described may also be usefull in ordinary groups with a large number of users and membership dynamics.

Part IV
Appendix

A Shared polynomials

This section, though not part of the core subject of this report came as an aside from the work done on group communications.

In an unpublished paper [2], Yi Mu, Vijay Varadharan and Khan Quac Nguyen build an interesting group public key scheme based on exponentiated polynomials. Basically, users in a group each choose a secret number and send it to a trusted host. This host then computes a polynomial with those secrets as roots. This polynomial is then used to create a group public key, which can be used for encryption. The mathematical construction behind this scheme will be exposed first, then we will demonstate a way to extend the scheme to distribute the construction of the public key. This new scheme as the main advantage of eliminating the need to send the secret keys over the network to a trusted host or any other host.

A.1 A group public key cryptosystem

A.1.1 The setup phase

Let p be a carefully chosen prime number and g a publicly known generator of $\mathbb{Z}_p$. Each member i of the group of n users chooses a number $r_i \in \mathbb{Z}_p$ and send it to a trusted host. The host then computes :

$$P(x) = z \prod_{i=0}^{n-1} (x - r_i) = \sum_{i=0}^{n} p_i x^i \quad (mod\, p)$$

Where z is a constant that is thrown away after the computation. The host then computes $\beta_i = g^{p_i} \, mod\, p$. The set $K = \{\beta_i,\, i \in 1 \ldots n\}$is the group public key.

This key has the specific property that for any r_j root of the polynomial, we have :

$$\prod_{i=0}^{n} \beta_i^{(r_j)^i} \equiv g^{P(x)} \equiv 1 \, mod\, p$$

A.1.2 Encryption and decryption

If Alice wishes to encrypt a message m to send it to the members of the group, she chooses a random value k and computes :

$$c_1 = \left\{ g^k \beta_0^k,\, \beta_1^k,\, \ldots,\, \beta_n^k \right\} \quad (mod\, p)$$

$$c_2 = m g^k \quad (mod\, p)$$

The ciphertext is the pair $\{c_1,\, c_2\}$. To decrypt the message any group member can use his root r_j and compute :

$$
\begin{aligned}
c_1^{'} &= g^k \beta_0^k \prod_{i=1}^{n} \beta_i^{k(r_j)^i} \\
&= g^k \left(\prod_{i=0}^{n} \beta_i^{k(r_j)^i} \right) \\
&= g^k \left(\prod_{i=0}^{n} g^{k p_i (r_j)^i} \right) \\
&= g^k g^{k P(r_j)} = g^k \; mod \, p \\
m &= c_2 / c_1^{'} = \left(m g^k \right) / g^k \; mod \, p
\end{aligned}
$$

This scheme enables a group of users to share a public key, while still having a different decryption key for each user.

It is also probably possible to do group signatures using this public key. As in most group public key systems this scheme requires the users to send their secret to a trusted group manager. This can be a drawback, since in might not be desirable to let the secret leave the host that uses it. In the next section, we will show a simple way to compute the public key in a distributed way without revealing any secrets from the hosts.

A.2 A distributed approach

A.2.1 Constructing a shared polynomial.

First, let's forget about exponentiation and take a general look at polynomials.

Adding a root to a polynomial in $\mathbb{Z}[x]$. Let $P(x)$ be a polynomial with k roots in $\mathbb{Z}$. We can write :

$$
P(x) = z \prod_{i=0}^{k-1} (x - a_i) = \sum_{i=0}^{k} p_i x^i
$$

Where z is a constant, the a_i values are roots of the polynomial and the p_i values are the coefficients of the expanded expression of the polynomial. Now, suppose we want to create a polynomial $Q(x)$ from $P(x)$, with the k previous roots, plus a new one we will call b. We have $Q(x) = P(x)(x - b)$. Thus, from the expanded form of $P(x)$ we can compute $Q(x)$:

$$
\begin{aligned}
Q(x) &= q_0 x^0 + q_1 x^1 + ... + q_{k+1} x^{k+1} \\
&= (x - b) \left(p_0 x^o + p_1 x^1 + ... + p_k x^k \right) \\
&= -b p_0 x^0 - b p_1 x^1 - ... - b p_k x^k + p_0 x^1 + p_1 x^2 + ... + p_k x^{k+1}
\end{aligned}
$$

So if we only represent $Q(x)$ by its coefficients in the expanded form, one can verify that :

$$
\begin{aligned}
q_0 &= -b p_0 \\
q_i &= -b p_i + p_{i-1} \quad , for \; 1 \leq i \leq k \\
q_{k+1} &= p_k
\end{aligned}
$$

Knowing the coefficients of a polynomial $P(x)$ of degree k in $\mathbb{Z}$, allows someone to easily create a new polynomial of degree $k + 1$ with the same roots as $P(x)$ plus a new one that can be chosen at random. This scheme can be extended to polynomials with coefficients in an exponentiated field.

Extension to a exponentiated field. Let m be a prime number and g a generator of the cyclic group of order[3] $m-1$. We can create a represention of $g^{P(x)}$ where $P(x)$ is defined similarly as above, except for the fact that $P(x)$ is in $\mathbb{Z}/(m-1)\mathbb{Z}[x]$. Using the coefficients of $P(x)$, we define :

$$\beta_i = g^{p_i} \ (mod\, m)$$

In such a case $g^{P(x)}$ will be represented in $\mathbb{Z}_m$ by the set $B_k = \{\beta_0, \beta_1, \ ..., \beta_k\}$. If r is a root of $P(x)$ then we have :

$$\beta_0^{r^0} \times \beta_1^{r^1} \times ... \times \beta_k^{r^k} = g^{p_0 r^0} \times g^{p_1 r^1} \times ... \times g^{p_k r^k} \equiv g^{P(r)} mod\, m \equiv 1 \, mod\, m$$

Now, suppose we want to create a new set $B_{k+1} = \{\delta_0, \delta_1, \ ..., \delta_{k+1}\}$ representing $g^{Q(x)}$ where $Q(x) = P(x)(x-b)$ in $\mathbb{Z}_{m-1}[x]$. From above we have :

$$\begin{aligned}
&\delta_0 = g^{-bp_0} = (\beta_0)^{-b} \\
&\delta_i = g^{-bp_i + p_{i-1}} = (\beta_i)^{-b} \times \beta_{i-1}, \ for\ 1 \leq i \leq k \\
&\delta_{k+1} = g^{p_k} = \beta_k
\end{aligned}$$

So, adding a root b to a system with k coefficients requires the computation in $\mathbb{Z}_m$ of $k+1$ exponentiations and k products.

A.2.2 A group public key agreement.

This can be used as the base of a group public key agreement. An "initiator" chooses a polynomial of degree 1 of the form $z(x-a)$, where a is the root of the polynomial - the initiator's private key - and z a random value that is thrown away after the computation. The initiator then computes $\{\beta_0, \beta_1\}$where $\beta_0 = g^a \ (mod\, m)$ and $\beta_1 = g^z \ (mod\ m)$, then it sends them to the next member of the group. In turn the next member computes a set of 3 values and passes it along. This goes on until the last member has done his computation. After that, the public key can be published and the group members can eventually verify individually that they are in the group.

As one might guess, some questions have been voluntarily eluded here, for time and space reasons. The setup phase described above needs to be secured in a way or another. This is a similar problem as in the Diffie-Hellman key agreement procedure. Another problem is key certification : since it's so easy to add a user to a group, provisions should be made for key certification...

Drawbacks. There are a few drawbacks in this scheme. The first one is that if we use it for signatures, there is no central authority to verify the signature in case of a disagreement. More precisely, there is no practical way of finding who signed a document on behalf of the group. A hybrid scheme could probably be used as suggested in [4] : we would share a secret with a authority that would be capable of "opening" the signature in case of a problem and the hosts would also keep an individual secret, possibly similar to the polynomial roots described above.

The second problem is that removing a user is not simple. Mathematically, it's possible but it requires cooperation from the user who leaves the group. So the only safe way to do it is to recompute the public key from the start, excluding the leaving user from the new key agreement.

The third and biggest problem is scalability. As we can see, the computation and the key size of the group is proportional to the group size. This is often the case

[3] this can probably be generalized to a non prime m with little modification

in group public key schemes but ideally we would like to have fixed size keys and computation amounts. For example, Camermish[3] has proposed such a scheme, but it requires a very strong hash function.

References

[1] Bruce Schneier, "Applied Cryptography", ???

[2] Yi Mu, Vijay Varadharajan, and Khan Quac Nguyen, "Combined Signature and Encryption Schemes for groups. Unpublished 1998.

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

[4] L. Chen, T. P. Pedersen, "New Group signature schemes".

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

[6] Neal Koblitz, "A Course in Number Theory and Cryptography". Second Edition, Springer-Verlag 1994.

[7] Suvo Mittra, "Iolus: A framework for scaleable Secure Multicasting". SIGCOMM'97.

[8] Ballardie, "scalable multicast key distribution", RFC 1949.

[9] Yuliang Zheng, "Signcryption and Its Applications in Efficient Public Key Solutions". Proceedings of 1997 Information Security Workshop (ISW'97).

[10] Michael Steiner, Gene Tsudik, Michael Waidner, "Diffie-Hellman Key Distribution Extended to Group Communication", proceedings ACM conference communication and Computer Security 1996.

[11] Michael Steiner, Gene Tsudik, Michael Waidner, "CLIQUES: A new approach to group key agreement", Reserach report.

[12] Stuart G. Stubblebine, "Security Services for Multimedia Conferencing", 16th Nationnal Computer Security Conference, Baltimore 1993.

[13] Tony Ballardie, Jon Crowcroft, "Multicast security threats and Counter-Measures", Proceedings Internet Soc. Symposium on Networks and Ditributed System Security, San Diego 1995.