

# A Reliable Key Authentication Schema for Secure Multicast Communications \*

Roberto Di Pietro, Antonio Durante , and Luigi V. Mancini

Dipartimento di Informatica, Università di Roma "La Sapienza",

Via Salaria 113, 00198-Roma, Italy.

{dipietro, durante, mancini}@dsi.uniroma1.it

## Abstract

*This paper analyzes the Logical Key Hierarchy (LKH) secure multicast protocol focusing on the reliability of the re-keying authentication process. We show that the key management in the LKH model is subject to some attacks. In particular, these attacks can be performed by entities external to the multicast group, as well as from internal users of the multicast group. The spectrum of these attacks is spread from the Denial of Service (DoS) to the session hijack, that is the attacker is able to have legitimate users to commit on a session key that is provided by the attacker. The contributions of this paper are: (1) the definition of the threats the LKH key management is subject to; and (2) a reliable key authentication scheme that solves the weaknesses previously identified. This objective is achieved without resorting to public key signatures.*

**keywords:** *Secure multicast communications, logical key hierarchy, group key management, key distribution, re-keying, confidentiality, authenticity, security.*

## 1. Introduction

Many emerging applications (for instance pay per view TV, stock option bid), are based upon a group communication model. In particular, they require message delivery from one or more authorized senders to a large number of authorized receivers. This model is available on the Internet, where the multicast communication has been successfully implemented to provide an efficient, best effort delivery service to large groups of users [3]. The deployment of network applications requiring group communication will accelerate in coming years. Thus, security is an important concern to enable the adoption and diffusion of the multicast paradigm.

\*This work was partially funded by the WEB-MINDS project supported by the Italian MIUR under the FIRB program and by the EU IST-2001-34734 EYES project.

Cryptographic techniques should prevent unauthorized users from accessing the content of delivered messages. As a result, securing group communications, that is, providing confidentiality, authenticity, and integrity of messages delivered between group members, is a critical issue. Usually, message confidentiality is assured by using simple and efficient symmetric key encryption for group data encryption. Note that, although group communications using sophisticated cryptographic techniques are proposed in the literature [18], the use of symmetric key is motivated by the fact that asymmetric cryptography requires much more resources to decrypt messages [16], thus depleting the level of service perceived by the users.

Once the encryption algorithm has been chosen the solution to the authentication problem for unicast transmission is rather simple and well known (for instance, the use of Hash-based Message Authentication Codes (HMAC)). However, this solution seems inadequate for multicast communications. Indeed, on one hand simply employing a single key shared by the users and the center could not prevent the forgery of the packets by a group member. On the other hand, sending an HMAC encrypted with a secret key shared by a single user and the center would require to send a number of HMAC which is equal to the number of users. Therefore, we strive to devise a mechanism that, at least under some assumptions, allows authenticity of the re-keying packets, without incurring the overhead related to public key signature/verification.

Confidentiality, authenticity and availability are among the classical security requirements. Confidentiality implies that only authorized users should decrypt a multicast message, even though this message is broadcast over a geographical region. The confidentiality requirement can be translated in the context of secure multicast into the following four requirements on key distribution: (1) Non-group Confidentiality: users that were never part of the group should not have access to any key that can decrypt any multicast data sent to the group; (2) Forward Confidentiality: users deleted from the group at some time  $t$  do not have access to any key used to encrypt data after  $t$ , unless they are

authorized to join again the group; (3) Collusion Freedom: no subset of deleted users should be able to decrypt future group communication, even by sharing the keys they had before deletion; (4) Backward Confidentiality: a user added at time  $t$  should not have access to any key used to encrypt data before  $t$  while the user was not part of the group. In this paper, authenticity means that when a user receives a message it is assured about the identity of the sender. The authenticity requirement can be translated in the context of secure multicast into two requirements on key and data distribution: (1) key authenticity: only the center can generate a session key; (2) data authenticity: the users can distinguish among the data sent by the center and the malicious data sent by an attacker. The availability requirement consists in the protocol capacity to detect and resist to some Denial of Service (DoS) attacks.

This paper focuses on authenticity. We first highlight the threats the re-keying procedure in the Logical Key Hierarchy (LKH) model is subject to. Then, under the threat model assumed, we show some of the possible attacks on the re-keying procedure. Our main contribution is to provide a reliable re-keying protocol that is resilient to the exposed attacks. Moreover, we sketch how to adopt such a protocol to assure the authenticity of multicast keys and payload data. It is worth noting that the proposed protocol enhances the reliability of re-keying, while introducing very limited overhead. In Section 2, we introduce the LKH model that will be referred throughout the rest of the paper. Section 3 introduces the threat model that will be employed to discuss the security of the LKH model. In Section 4, the attacks to which the LKH model is subject to are detailed. In Section 5 we first provide a mechanism to neutralize the exposed attacks, that is to provide reliable re-keying, and then we show how to adopt such a mechanism to provide reliable authentication of multicast data. Section 6 reviews the current work in the area, while in Section 7 some concluding remarks are exposed.

## 2. The LKH Model

We assume the LKH model described in [21], also called *key graph model*. In LKH model, there is a multicast group  $\mathcal{M} = \{u_1, \dots, u_n\}$ , which is a dynamically changing subset of all possible users. This subset can change according to the eviction of a user from the group, or according to a new user joining the group. We assume there is a super user, called the center, which can send a message to the multicast group that can be received by all members of  $\mathcal{M}$ . The message is sent over an insecure channel, therefore the same message can be received by other entities not belonging to  $\mathcal{M}$ . To enforce the confidentiality of data, we can assume available to the users in  $\mathcal{M}$  and to the center a cryptographic module based on symmetric key cryptography. In the fol-

lowing, when the center needs to send a message  $m$  to  $\mathcal{M}$ , the center will compute  $m' = E_k(m)$  and then will broadcast to the group  $\mathcal{M}$  the message. The key  $k$ , shared by all users and the center is the session key. A join occurs when a new user  $u_j$  is added to  $\mathcal{M}$ , while a deletion occurs when a user  $u_e$  is evicted from  $\mathcal{M}$ .

The *key graph* model builds up a tree of auxiliary keys (in the following *key graph tree*), whose leaves are the private key of the users in  $\mathcal{M}$ . Each user has to store the keys that are on the leaf-root path. Within this framework, an excellent trade-off between the *per user* required storage, and efficiency in the number of message required to perform re-keying is achieved.

Referring to this model, we will adopt the following terminology: (1) the siblings of a user  $u_i$  in the key graph is the set of users that share with the user  $u_i$  the same parent; (2) we will assume, unless otherwise specified, that the number of the users in the key graph is exactly  $n = |\mathcal{M}| = b^d$ , where  $b$  is the arity of the key graph and  $d$  is the depth of the tree. This assumption (that is, the key graph is perfectly balanced) does not affect the generality of our findings, but will simplify the presentation; (3) the root of the key graph (that is the session key) is located at level 0, while its leaves are at level  $d$ ; (4) the user to be excluded from the multicast group will be denoted by  $u_e$ ; (5) the total number of keys, excluded the leaves, are  $n - 1$ ; these keys are called auxiliary keys. Let  $k_{i,j_i}$  a generic auxiliary key, where  $i$  is the position in the key graph (for instance, in Figure 1 we have  $i = 0..6$ ), and  $j_i$  counts the number of time the  $i^{th}$  auxiliary key has been changed; (6) we denote with  $k_{c,u_z}$  the private key the center shares with the user  $u_z$ ; (7) note that each user  $u_i$  stores only  $d + 1$  keys: the  $d$  keys along the leaf-root path plus the private key.

In Figure 1 an example of such a key tree is reported.

### 2.1. Confidentiality and authenticity in the LKH model

When the session key has to be changed due to the occurrence of an eviction, all users need to change the auxiliary keys shared with evicted user  $u_e$ . Indeed, employing one of these keys to encrypt any message to be sent would result in a violation of the confidentiality, since the encryption key is hold by  $u_e$ , which can correctly decrypt the message.

When a join occurs, the newly joined user  $u_j$  receives the appropriate sequence of auxiliary keys from the center. The center encrypts such messages *via* the private key of  $u_j$ , which is known only by the center and  $u_j$  himself. In this way  $u_j$  receives, along with the appropriate sequence of auxiliary keys, also the session key required to decrypt the payload messages. In the rest of the paper we refer to the LKH re-keying protocol.

A protocol step is specified using the standard notation

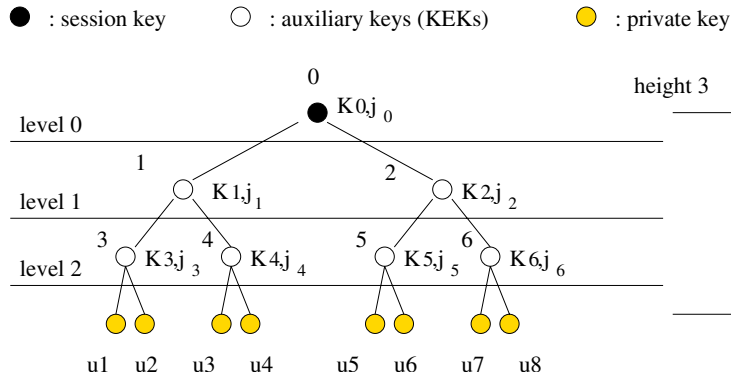


Figure 1. Notation and terminology employed in the following.

Table 1. An instance of the LKH protocol

Message 1.	$c \rightarrow u_2 : [k_{3,j_3+1}]_{k_{c,u_2}}$
Message 2.	$c \rightarrow u_2 : [k_{1,j_1+1}]_{k_{3,j_3+1}}$
Message 3.	$c \rightarrow u_3, u_4 : [k_{1,j_1+1}]_{k_{4,j_4}}$
Message 4.	$c \rightarrow u_2, u_3, u_4 : [k_{0,j_0+1}]_{k_{1,j_1+1}}$
Message 5.	$c \rightarrow u_5, u_6, u_7, u_8 : [k_{0,j_0+1}]_{k_{2,j_2}}$

$a \rightarrow b : m$  where  $a, b$  are the principals names and  $m$  is the exchanged message. For LKH we have  $c, u_i$ , that is the center and the  $i^{th}$  user. We extend the standard notation to express multicast messages as follows:  $c \rightarrow u_1..u_n : m$  means the center sends the message  $m$  to the user group  $u_1..u_n$ . In the rest of this paper we focus only on the key transmitted during a re-keying procedure, and we assume that the format of the message  $m$  is  $[k_{i,j_i+1}]_{k_{z,j_z}}$ , where the new key  $k_{i,j_i+1}$  is encrypted using the key  $k_{z,j_z}$ . Table 1 describes an instance of the basic LKH re-keying protocol. In particular, we consider the eviction of the user  $u_1$  in the LKH tree depicted in Figure 1. We assume that the re-keying invocation command does not appear in the protocol instance.

Intuitively the LKH protocol instance assures only confidentiality; to assure also authenticity, in [21] it has been proposed a key authentication schema that uses one signature and  $\log n$  hashes to authenticate the set of re-keying messages. For instance, the eviction of the user  $u_1$  requires one signature and three hashes. Hence, this authentication procedure requires an high computational and transmission overhead.

### 3. The Threat Model

To analyze a cryptographic protocol, it is necessary to study its behavior in the presence of a malicious user, called *attacker*, which can perform the following actions: (1) intercept and learn any sent message; (2) introduce into the system new messages forged using the information available. This definition of the attacker is based on the Dolev-Yao model [5]. We assume that the encryption algorithms used are cryptographically secure.

Here we give an informal description of the threat model used. In a multicast protocol the attacker can act as: (1) *external*: the attacker does not belong to the multicast group; or (2) *internal*: the attacker belongs to the multicast group. There is a transition state from internal to external (and vice-versa); this happens when the attacker is *evicted* from or *joins* to the multicast group. These three kinds of attackers have a different set of initial knowledge that changes during the attacks they perform. Let it be respectively  $K_i, K_z, K_e$  the set of initial knowledge belonging to the internal, evicted and external attacker. Intuitively the following relation  $K_e \subseteq K_z \subseteq K_i$  is always true during the attack. The attacker *power* can be characterized as a function of the knowledge it acquires during the session attack starting from a set of initial knowledge. We can consider that the number of *valid* messages the attacker can compose, to perform an attack, is proportional to its knowledge set. More precisely, all the attacks an external attacker can perform can be also performed by an evicted and by an internal attacker. The attacks performed by an evicted can also be performed by an internal attacker.

### 4. Possible attacks to the LKH model

In this section, we present a few attacks that violate some of the LKH security properties described in Section 1. The attacks can be performed by an attacker with the Dolev-Yao features. The attacks will be exposed in increasing order

**Table 2. The session disruption attack**

time	attacker	user(s)
$t_1$	$e :: \text{generate noise}$	
$t_2$	$e(c) \rightarrow u_1 : \text{noise}$	
...		
$t_i$		$u_1 \leftarrow e(c) : \text{noise}$
$t_{i+1}$		$u_1 :: k_{0,j_0+1} = E_{k_{1,j_1}}^{-1}(\text{noise})$

**Table 3. The replay attack**

time	attacker	user(s)
$t_1$	$i :: \text{store a re-keying session}$	
$t_2$	$i(c) \rightarrow u_t : [k_{0,j_0}]_{k_{2,j_2}}$	
...		
$t_i$		$u_t \leftarrow i(c) : [k_{0,j_0}]_{k_{2,j_2}}$
$t_{i+1}$		$u_t :: k_{0,j_0+x} = E_{k_{2,j_2}}^{-1}([k_{0,j_0}]_{k_{2,j_2}})$

with respect to the attacker power, that is proportional to the information the attacker is able to learn, as seen in Section 3. We will see that the attack effects are directly proportional to the attacker power.

We summarize the notation we use in the following for the description of the attacks:

- $k_{i,j_i}$  is the key associated to the  $i^{th}$  position in the key graph. The index  $j_i$  counts the number of time the key has been updated;
- $k_b$ , a bogus session key;
- $i(c) \rightarrow u : m$ , the attacker  $i$  speaking for center  $c$  sends the message  $m$  to the user  $u$ ;
- $u \leftarrow i(c) : m$ , the user  $u$  receives a message  $m$ , identifying the center  $c$  as the sender;
- $[k_{i,j_i+1}]_{k_{z,j_z}}$ , the key  $k_{i,j_i+1}$  is encrypted using the key  $k_{z,j_z}$ ;
- $E_{k_{i,j_i}}^{-1}(M)$ , the function  $E^{-1}$  decrypts  $M$  using the key  $k_{i,j_i}$ .

#### 4.1. Session Disruption

A session disruption occurs when an attacker feeds the users with *noise* during a re-keying session. The attacker knows when a re-keying procedure starts so it substitutes, in the re-keying messages, the  $k_{i,j_i+1}$  keys with *noise*. The users starting from the *noise*, will recover a bogus key  $k_b$  applying on such a *noise* the decryption procedure using a current and legal  $k_{z,j_z}$  key. The effect of this attack consist in a kind of DoS: the attacked users will not be able to correctly decrypt any further message sent by the center.

To perform this attack it is not necessary the attacker belongs to the multicast group, it can act as an external. We can summarize the attack steps:

1. generate the noise and forge bogus initialization/re-keying messages;
2. send the bogus messages to the targets users;

A possible instance of this attack is reported in Table 2 where  $e$  is the external attacker and  $u_1$  the target user. When the user  $u_1$  retrieves the new key using the key  $k_{z,j_z}$  (in this example  $z = 1$ ),  $u_1$  will obtain a  $k_b$ . Indeed, user  $u_1$  will believe that  $k_b$  is a valid session key  $k_{0,j_0+1}$ . Hence  $u_1$  will try to use  $k_b$  to decrypt the center's future data messages. Unfortunately  $k_b$  is a bogus session key and  $u_1$  cannot access to the group communication any longer.

This attack may succeed, since: (1) the values of the encryption keys are supposed random, so they cannot be verified; (2) the re-keying command invocation is supposed to be in plain text, so it can be forged by the attacker.

Note that if the attacker is an internal, even encrypting the re-keying command the center cannot prevent the attacker to succeeded, while if the attacker is an external encrypting the re-keying command requires the attacker to exploit the interleaving of the re-keying protocol messages.

#### 4.2. Replay Attack

A replay attack can occur when the attacker is able to feed the target users with old keys  $k_{i,j_i}$ . The target users will recognize as *valid* any message encrypted with the attacker old keys discarding the data messages sent by the center and encrypted with the current legal keys. This attack can be performed by an internal or by an evicted user.

The aim of this attack is to inject messages that are meaningful for the target users, using old session keys.

The requirements to perform this attack is that the attacker has stored the messages of an old re-keying session. The attacker steps are the following:

1. stores the re-keying communication between the center and the target users;
2. replays the messages it stored previously;

Note that it is not need to cover the target users.

An instance of a replay attack to the session key is reported in Table 3.

With reference to Figure 1, let  $i$  be the attacker acting as an internal and  $u_t = \{u_5, u_6, u_7, u_8\}$  the set of target users. First, the attacker stores the messages originated as a consequence of a previous re-keying session for the eviction/join of a user belonging to the left sub-tree. Then, at time  $t_2$ , the attacker replays to the target users the root re-keying message ( e.g. Message 5 in Table 1) we use the  $k_{2,j_2}$  key because the target users belong to the right sub-tree. All the users in  $u_t$  will assume the old session key  $k_{0,j_0}$  as the new one,  $k_{0,j_0+x}$ . Where  $j_0 + x$  is the actual session key counter. Since the attacker  $i$  is an internal user, the attacker knows the old session key  $k_{0,j_0}$  and can start feeding the target users with malicious data messages. Note that the target set can be any of the users in the multicast group; it is sufficient for the attacker to store a session of re-keying message which had the target user among its final recipients.

### 4.3. Session hijack

This attack can occur when the attacker can feed the users of the sub-tree it belongs to with bogus keys  $k_b$  it generates. As a consequence of this attack, the target users will only be able to correctly decrypt the messages sent by the attacker, encrypted with the bogus keys  $k_b$ . The *valid* messages coming from the center will not be recognized. Note that this kind of attack can only be performed by an internal user.

In order to perform this attack, the attacker has to belong to the same sub-tree to which the target users belong to. We sketch the attack steps:

1. forge the bogus keys;
2. substitute during a re-keying protocol the center session key.

The attacker belongs to the multicast group so it knows the valid sub-tree keys it has to use to negotiate a new  $k_{0,j_0+1}$  center session key. An instance of this type of attack follows.

According to Figure 2 let  $u_5$  be the internal attacker and  $u_t = \{u_6, u_7, u_8\}$  the target users belonging to the attacker

sub-tree. The attacker first generates a new key  $k_b$ , then it uses the *valid* key sub-tree, namely  $k_{2,j_2}$ , to negotiate  $k_b$ . When the users in  $u_t$  accept  $k_b$  as a valid session key then in the future communication they will accept as valid only the attacker traffic encrypted with  $k_b$  discarding the traffic data sent by the center. The details of the messages exchanged are in Table 4.

### 4.4. Delayed disclosure

The attacker can perform this attack when it is able to intercept and delay an entire re-keying session for a set of target users. In this case, the target users will continue to use the old session key. Supposing the attacker acts as an external the effect of the attack consists in a DoS, that is the target users will continue to use the old session keys, hence discarding all the subsequent center communications. If the attacker acts as an internal, it can use the old keys to feed the users with malicious data. Moreover, the attacker can perform a re-keying of the target users by a replay attack (it can make the target users to accept the re-keying messages previously intercepted as new re-keying session) and then continue feeding the target users with malicious data.

Note that the key authenticity is not violated, since the key the attacker will employ were actually generated by the center.

## 5. A Reliable Authentication Schema

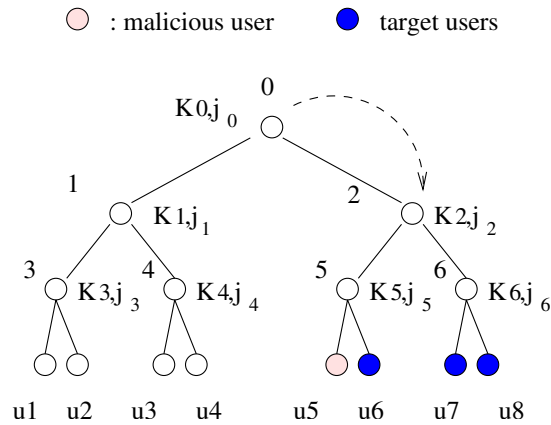
We propose a new key authentication schema to fix the flaws exposed in the previous section. This scheme is an extension of the authentication scheme originally proposed by Lamport [6, 8].

Let  $H$  be a one-way hash function [2]. The center generates a chain of keys for each key in the graph, for instance for the key  $k_{i,j_i}$  we have  $k_{i,1}, \dots, k_{i,n}$  where  $H(k_{i,j_i+1}) = k_{i,j_i}$ . The keys will be deployed first assigning  $k_{i,1}$ . For instance, when the  $x^{th}$  re-keying is performed, the session key that will be eventually delivered is the  $k_{0,x}$  in the chain originated by  $k_{0,n}$ . Hence, the user can easily verify if  $H(k_{0,x}) = k_{0,x-1}$ . If the match fails, the new received key  $k_{0,x}$  is not authenticated and is discarded, otherwise it is accepted and the old key  $k_{0,x-1}$  is discarded. Note that the same procedure is applied for each of the logical keys in the  $u_e$ -root path.

Assume that the session keys delivered by the center are originated as above exposed. In accordance with the basic LKH model [21] we assume that the center sends the first leaf-root path to the users *via* a secure channel. Note that, under the scheme exposed above, each user still needs to store only the  $d$  keys that are along the leaf-root path. In the following, we will discuss how this authentication schema

**Table 4. Session hijack attack**

time	attacker	user(s)
$t_1$	$i :: generate\ k_b$	
$t_2$	$i(c) \rightarrow u_t : [k_b]_{k_{2,j_2}}$	
...		
$t_i$		$u_t \leftarrow i(c) : [k_b]_{k_{2,j_2}}$
$t_{i+1}$		$u_t :: k_{0,j_0+1} = E_{k_{2,j_2}}^{-1}([k_b]_{k_{2,j_2}})$



**Figure 2. Session hijack attack: relationship in the key graph.**

**Table 5. Avoiding the session disruption attack**

time	attacker	user(s)
$t_1$	$e :: generate\ noise$	
$t_2$	$e(c) \rightarrow u_1 : noise$	
...		
$t_i$		$u_1 \leftarrow e(c) : noise$
$t_{i+1}$		$u_1 :: k_{0,j_0+1} = E_{k_{1,j_1}}^{-1}(noise)$
$t_{i+2}$		$u_1 :: H(k_{0,j_0+1}) \llcorner \llcorner k_{0,j_0}$

prevents the exposed attacks and how such a schema can be extended to provide multicast data authentication.

### 5.1. Reliable Key Authentication

Adopting the above proposed schema we avoid the session disruption attack. Indeed, the users can recognize when they receive a bogus session key. This is because the format of the keys is verifiable due to the relation introduced in the key generation process. In Table 5 at time  $t_{i+2}$ , user  $u_1$  retrieves a bogus key  $k_b$  thinking to be  $k_{0,j_0+1}$  so it is able to check if the key received  $k_b$  satisfies the relation  $H(k_{i,j_i+1}) = k_{i,j_i}$ .

If the attacker tries to replay old session keys, the users are able to check that they have received an old session key, see line  $t_{i+2}$  in Table 6. The users  $u_t$  retrieve the key  $k_{0,j_0+x}$ , but since the relation  $H(k_{0,j_0+x}) = k_{0,j_0}$  is not satisfied, the session key is discarded.

Our key authentication schema prevents the session hijack attack because the attacker is not able to create a new session key respecting the relation  $H(k_{i,j_i+1}) = k_{i,j_i}$ . In Table 7 at time  $t_{i+2}$  the users will detect the session hijack attack, they retrieve the bogus session key  $k_b$  they suppose to be  $k_{0,j_0+1}$  and check the hash relation, that is  $H(k_{0,j_0+1}) \neq k_b$ .

However, the schema proposed so far does not prevent the *delayed disclosure* attack. A possible solution to this attack consists in performing periodic synchronized re-keying sessions, that is every  $t$  time unit the center performs a re-keying session. The implementation of this schema, requires that the center and the users are loosely synchronized. This means that the clock drift between the center and any of the users cannot exceed a value of  $\delta$  time unit. In case the users do not receive a re-keying signal within  $t$  time unit, an attack to the session key is detected.

Finally, note that the length of each chain of keys is limited. Hence, the problem to periodically renew the chain while assuring authentication could arise. In the following, we highlight two techniques to solve this issue. The first is based on the observation that the computational cost of an hash is almost negligible. Hence, the center can generate a chain of keys of a length longer than the number of join/eviction that could occur during the operational period of the multicast group. The second solution is that, once the center is close to exhaust a chain of keys, for instance the chain originated by  $k_{0,1}$ , it generates a new chain of keys  $\bar{k}_{0,1} \dots \bar{k}_{0,n}$  such that  $H(\bar{k}_{0,i+1}) = \bar{k}_{0,i}$ , for  $i = 1 \dots n-1$ . Assume that the current session key is  $k_{0,n}$ , that is, the last useful key of the chain of keys. The center then sends the value  $E_{k_{0,n}}(\bar{k}_{0,1} || \text{Sig}_{PK_C}(\bar{k}_{0,1}))$ , that is the encryption of the key in the new chain of keys, together with the same value signed with the private key of the center. Every user can thus verify that the new chain of keys is originated by

the center. Note that with this solution, for a given chain of keys of length  $n$ , the center sends one signed message only every  $n$  keys.

### 5.2. A reliable data authentication schema

Once the attacker can feed a shared key to some users, hence violating the key authenticity, the data authenticity is violated as well. Indeed, the attacker could feed bogus data (encrypted with the bogus key) to the target users. However, note that data authenticity can be violated without violating the key authenticity. Indeed, the attacker can simple use the current session key to encrypt bogus data that will be sent to the target users.

Note that if the target user can check for some sort of relationship among the data sent by the center, it could recognize that some of the received data (the bogus data) do not respect this relationship. Hence, the target user could resort to some mechanism that could eventually confirm that it is subject to some sort of attack.

Let  $m_i$  a message containing the payload. When the center decides to start a data authentication procedure: (1) it first produces the xor of the next  $v$  messages; (2) it creates the value *sign* as the encryption with the future session key  $k_{0,j_0+1}$  of the xoring previously computed, that is:

$$sign = E_{k_{0,j_0+1}}(\bigoplus_{i=1}^v (m_i))$$

(3) it will send the payload messages, encrypted with the current key  $k_{0,j_0}$ . In this way, a user can check, when it receives the new session key  $k_{0,j_0+1}$ , if the  $v$  data messages received have been really sent by the center. The attacker cannot forge the value *sign* because it does not know the key  $k_{0,j_0+1}$  that is generated according to the key generation rule:  $H(k_{0,j_0+1}) = k_{0,j_0}$ . Moreover being the key  $k_{0,j_0+1}$  sent by the center after the value *sign* is received by users, an attacker cannot forge it.

Note that the proposed scheme works under the assumption that all the packets are received, otherwise the users could not compute the value *sign*. In particular, the assumption is valid for all the transmission protocols that assure packet delivery, as the TCP does. Moreover, the scheme can be easily extended to those protocol that do not assure delivery, like the UDP, resorting to reliable transmission techniques, as the ones in [7, 19, 22] and discussed in Section 6.

## 6. Related Work

In [10] the security requirements for a secure multicast protocol are formally defined. In particular, we have security requirements for: authenticity, freshness, secrecy. Some of these are the requirements the attacks we expose in

**Table 6. Avoiding the replay attack**

time	attacker	user(s)
$t_1$	$i :: \text{store a re-keying session}$	
$t_2$	$i(c) \rightarrow u_t : [k_{0,j_0}]_{k_{2,j_2}}$	
...		
$t_i$		$u_t \leftarrow i(c) : [k_{0,j_0}]_{k_{2,j_2}}$
$t_{i+1}$		$u_t :: k_{0,j_0+x} = E_{k_{2,j_2}}^{-1}([k_{0,j_0}]_{k_{2,j_2}})$
$t_{i+2}$		$u_t :: H(k_{0,j_0+x}) \langle \rangle k_{0,j_0+x-1}$

**Table 7. Avoiding the session hijack attack**

time	attacker	user(s)
$t_1$	$i :: \text{generate } k_b$	
$t_2$	$i(c) \rightarrow u_t : [k_b]_{k_{2,j_2}}$	
...		
$t_i$		$u_t \leftarrow i(c) : [k_b]_{k_{2,j_2}}$
$t_{i+1}$		$u_t :: k_{0,j_0+1} = E_{k_{2,j_2}}^{-1}([k_b]_{k_{2,j_2}})$
$t_{i+2}$		$u_t :: H(k_{0,j_0+1}) \langle \rangle k_{0,j_0}$

Section 4 violate. For instance, the replay and the delayed disclosure attack violate the *recency freshness* requirement: the key a user receives it is not current at some specified point in time according to the user's local clock; the session hijack violates the *authenticity* requirement: the key a user receives it is not generated by the center. The security requirements violated by the session disruption attack is the *availability* requirements defined in Section 1: the attacked users are excluded from the center communications.

LKH key authentication schema [21] avoids only the session hijack attack. However, this feature is achieved with  $\log n$  hash digest and one signature for each re-keying invocation.

In [12, 11] a data authentication schema has been proposed, it requires one hash for packet assuring individual packet authentication. We assume that our data authentication protocol can rely upon a lower level protocol assuring that the data packets are delivered.

The following works refer to extension of the basic LKH model, and all of them can benefit from the reliable key authentication schema proposed in our paper.

In [20] a protocol is proposed called *LKH+*. The performances achieved by *LKH+*, as for the join of new users, are better than those in [21]. Indeed the keys to be refreshed, instead of being encrypted with their respective children's key, are simply hashed. In this way, only the indexes of the refreshed keys need to be multicast. Note that an index size is smaller than the key size. *LKH+* is related to the protocol proposed in *rfc2627*.

The *OFT* protocol [9, 1] requires each re-keying message to carry just  $\log n$  keys instead of  $2 \log n$  as in the basic LKH model. In particular, each intermediate node is asso-

ciated with two cryptographic keys: a key  $K_x$  and a key  $K'_x = \mathcal{H}(K_x)$ , where  $\mathcal{H}$  is a one-way function. The key  $K'_x$  is *blinded* in the sense that it is computationally unfeasible, for an adversary, to derive  $K_x$  from  $K'_x$ . Each member knows the unblinded node keys on the path from its node to the root, and the blinded node keys that are siblings to its path to the root, and no other blinded or unblinded keys. When the key of a node changes, its blinded version is encrypted with the key of its sibling node. Thus, the re-keying message carries just  $\log n$  keys.

The *ELK* protocol [13] focuses on secure multicast for large groups. Such protocol addresses also reliability issues for secure multicast. The ELK protocol is similar to OFT, but ELK employs pseudo random functions to build and manipulate the keys in the tree. ELK is based on periodic re-keying, that is at given time interval the center refreshes the root key using the pseudo random functions and then uses it to update the whole key tree. One of the main advantages of ELK is that by deriving all keys, the protocol does not require any multicast message during a join operation. Indeed, all members can autonomously refresh their own keys. Deletion, as in OFT, requires that new keys are generated from the children's keys.

In [15], the use of the old keys to create or update the new keys saves the information to be transmitted to the users when a membership change occurs. Two operations are performed on local key by the users: (1) *refresh* which requires the user to receive an index from the center to generate a new local key; (2) *update* which requires the user to receive an index from the center to generate either the left or the right child key. Given that the users *refresh* or *update* their local key only on the base of the old key an extra message

from the center containing the index is required to enforce backward secrecy.

In [14], a new way to exploit the LKH model is introduced. In particular, the approach takes advantage of: (1) the set of information the users share and that can be used to locally generate the new keys; (2) the set of keys that the users logically share from a certain point onward in the LKH, and that allows the users to compute autonomously their path to the root employing a one-way hash functions. In this way, it has been shown how to reduce of the 50% the bandwidth required by the center for unicast communications to perform group set-up, and to deal with mass join. Moreover, this approach allows to save more than the 50% of the computations required by the center in mass evictions and requires less computations on the user device than the other solutions.

As for literature specifically addressing reliability in multicast communications, the protocol proposed by Yang et al. [22] uses proactive FEC in which parity packets are transmitted along with payload packets in each FEC block. An interesting approach that has been proposed by some studies [7, 19, 17] is to send the key updates in the same stream as data packets. The main advantage of this approach is that key updates are synchronized with the encrypted data, and a separate protocol is not needed for reliable key delivery. Finally, in [4] a methodology to establish the minimal key length that guarantees a specified level of confidentiality is proposed. Reducing the keys length reduces the communication cost, the computation cost, the total re-keying completion time, the storage required by the users, and enhances the reliability of re-keying communications. The proposed methodology scales with the number of users in the multicast group.

## 7. Concluding Remarks

In this paper we have: (1) showed a few of the possible attacks to authentication the re-keying procedure of the LKH model is exposed to; (2) provided a solution to the possible attacks exposed. Moreover, the solution devised requires only the application of an hash function, while current solutions rely on public key signatures; (3) sketched a solution that assures, with negligible overhead, data authenticity. The contribution provided by this paper is a general scheme that enhances the reliability of security in multicast communications as for the authenticity of both encryption keys and data. Finally, some open research problem in the area, namely the relationship between timing issue and authentication in symmetric key based security solutions, have been introduced.

## References

- [1] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization. Internet draft, IETF, June 2002.
- [2] S. Bkhtiari, R. Safavi-Nini, and J. Pieprzyk. Cryptographic hash functions: A survey. technical report, University of Wallongon, July 1995.
- [3] S. E. Deering. Multicast routing in internetworks and extended LANs. *Computer Communication Review*, 18(4), 1988. ACM SIGCOMM '88 Symposium: Communications Architectures and Protocols.
- [4] R. Di Pietro, L. V. Mancini, and A. Mei. A time driven methodology for keys dimensioning in secure multicast communications. In *Proceedings of the 18<sup>th</sup> International Conference on Information Security*, pages 121–132. Kluwer, 26-28 May 2003, Athens-Greece.
- [5] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
- [6] N. M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 151–157, 1994.
- [7] M. Kandansky, D. Chiu, J. Wesley, and J. Provino. Tree-based reliable multicast (tram). IETF Internet Draft, 2000.
- [8] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.
- [9] D. A. McGrew and A. T. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical Report 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, May 1998.
- [10] C. Meadows and P. Syverson. Formalizing gdoi group key management requirements in npatr1. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 235–244. ACM Press, 2001.
- [11] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS'01*, pages 35–46. Internet Society, February 2001.
- [12] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, May 2000.
- [13] A. Perrig, D. Song, and J. D. Tygar. Elk, a new protocol for efficient large-group key distribution. In *Proceedings of 2001 IEEE Symposium on Security & Privacy*, pages 247–262, 2001.
- [14] R. D. Pietro, L. V. Mancini, and S. Jajodia. Efficient and secure keys management for wireless mobile communications. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 66–73. ACM Press, 2002.
- [15] S. Rafaei, L. Mathy, and D. Hutchison. EHBT: an efficient protocol for group key management. *Lecture Notes in Computer Science*, 2233:159–171, 2001.
- [16] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, Inc., 2nd edition, 1996. ISBN 0-471-12845-7.

- [17] S. Setia, S. Zhu, and S. Jajodia. A comparative performance analysis of reliable group rekey transport protocols for secure multicast. In *Proc. of the 22nd International Symposium on Computer Modeling, Measurement and Evaluation*, Rome-Italy, 23-27 September, 2002.
- [18] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A protocol suite for key agreement in dynamic groups. In *Proceedings. 18th IEEE International Conference on Distributed Computing Systems*, 1998.
- [19] W. Trappe, J. Song, R. Poovendran, and K. Liu. Key distribution for secure multimedia multicasts via data embedding. In *Proc. of IEEE ICASSP 2001*, pages 1449–1452, 2001.
- [20] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The versa-key framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, Sept. 1999.
- [21] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transaction on Networking*, 8(1), 2000.
- [22] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam. Reliable group rekeying: a performance analysis. In *Proc. of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 27–38. ACM Press, 2001.