Multi-Layers Balanced LKH

Wee Hock Desmond Ng, Zhili Sun
Centre for Communication Systems Research
University of Surrey
Guildford, Surrey, GU2 7XH, United Kingdom
{W.Ng | Z.Sun} @surrey.ac.uk

***Abstract*—Secure multicast communication is important for applications such as pay-per-view distribution. LKH has been proposed to distribute a shared secret key in a way that scales efficiently for groups with many members. However, the efficiency of LKH depends critically on whether the key tree remains balanced. For applications such as video streaming or online teaching, several multicast sessions might be related in one way or another. In this paper, we consider the balance of the key tree and treat related multicast sessions as a whole. Our approach shows significant improvement over traditional method and trades off additional rekeying costs for similar computation power at members' side compared to existing related approach. Two optimisations are also proposed to further enhance the efficiency of our algorithm.**

Keywords- secure multicast, group key management, secure group communication

I. INTRODUCTION

IP multicast [1] enables efficient group communication by allowing the sender to transmit a single copy of data, with network elements such as routers and switches making copies as necessary for the receivers. This solves the scalability issues at the sender side and allows better utilisation of network resources such as bandwidth and buffer space.

In order for IP multicast to scale to virtually any group size, it relies on a single group address to identify the set of recipients rather than explicitly listing them. However, this anonymous receiver model prevents the content providers from charging the members. The only way to provide controlled access to data is to encrypt the multicast data and distribute the encryption key to the members. If the membership is dynamic, this shared encryption key has to be updated and redistributed to all authorised members securely every time there is a change in the group membership in order to provide forward and backward secrecy. Forward secrecy means a departing member cannot obtain information about future group communication, and backward secrecy means that a joining member cannot obtain information about past group communication. Changing of keys, also known as rekeying, is necessary even when there is no change in membership to prevent the key from being compromised after long period of usage. A number of scalable approaches [2, 3, 4, 5, 6, 7] have been proposed, and one in particular, logical key hierarchy (LKH) [2, 3] is analysed in this paper. The trusted entity, which is responsible for distributing the key to the members, is known as group controller (GC).

In LKH, there are two types of keys: Traffic Encryption Key (TEK) and Key Encryption Key (KEK). In short, TEK is the group key, which is used to encrypt the multicast data while KEK is used to provide scalable rekeying. The efficiency of LKH depends critically on whether the key tree remains balanced. A key tree is considered balanced if the distance from the root to any two leaf node differs by not more than one [8, 9]. If the key tree becomes unbalanced, the distance from the root to a leaf node can become as high as N, where N is the number of members. In other words, some group members might require up to N decryptions if any of its siblings departs from the multicast group.

Within LKH, two types of groups, data group (DG) and service group (SG), have been defined in [10, 11]. A DG is a set of members who receives the same single data stream. The information distributed in each DG is encrypted with the TEK. A SG is a set of members who entitles the same privileges and receives the exactly same set of data stream. This is because in existing group applications such as video streaming and online teaching, some members might subscribe to several similar DGs at the same time. Figure 1 illustrates a multicast video encoded in cumulative layers for heterogeneous receivers. Each receiver subscribes to a subset of layers in such a way that the total capacity of the subscribed layers does not exceed the receiver's capacity. In this illustration, BL, EL1 and EL2 are the DGs and SG_{BL} are members in DG_{BL} excluding those in DG_{EL1} and DG_{EL2}. For SG_{EL1}, it consists of the members in DG_{EL1} excluding those in DG_{EL2} and SG_{EL2} are members in DG_{EL2}.

If the above multicast sessions are considered using traditional method, each DG will be considered separately, which leads to inefficiency in term of key storage at both GC and members side [10, 11]. In addition, the rekeying costs are higher [11, 12]. The term rekeying cost refers to the total number of keys to be unicast or multicast out to the members when the GC spawns a rekey.

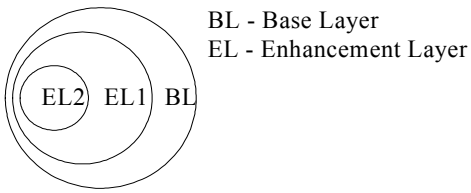

Figure 1. Multicast video encoded in cumulative layers

In this paper, we consider all related multicast sessions as a whole to minimise key storage and rekeying costs needed by the GC and members. In addition, we also take the balance of key tree into consideration to allow similar key storage among members and preserve the rekeying scalability of LKH; this means each member needs at most $\log_k N$ decryptions when any of its siblings departs rather than N decryptions (worst case) in an unbalanced LKH. The rest of this paper is organised as follows. In section II, we provide the background material of LKH and explain how an unbalanced key tree affects the GC and members in term of key storage and rekeying costs. Related works are examined in this section too. In section III, we discuss how our Multi-Layers Balanced LKH (MLB-LKH) constructs the key tree for the related multicast sessions. The simulation results of our proposed algorithm with traditional and existing approaches are presented in section IV, followed by the conclusion in section V.

II. RELATED WORK

In LKH, the GC maintains a tree of keys, where the internal nodes of the tree hold the KEKs and the leaf nodes correspond to the group members. Each leaf node holds an individual key associated with that one member. Each member receives and maintains a copy of the individual key associated with its leaf node and the KEKs corresponding to each ancestor node in the path from its parent node to the root. All group members share the key held by the root of the tree, also known as TEK, as shown in Figure 2.

For a balanced key tree with outdegree, k, each member stores $\log_k N + 1$ keys while the GC stores all $(kN - 1)/(k - 1)$ keys. For example, in Figure 2, member U1 knows K1, K2, K5 and member U7 knows K1, K4 and K11. In this example, K1 is the TEK, which is used to encrypt the multicast data, K2 to K4 are the KEKs for rekeying purposes and K5 to K13 are the individual keys associated with the group members on the leaf nodes.

When a member is removed from the group, the GC must change all the keys in the path from this member's leaf node to the root to achieve forward secrecy. All the other members that remain in the group must update their keys accordingly, namely change the keys in the intersection between the path from their leaf nodes to the root and the path from the removed member's leaf node to the root. In particular, this means that every remaining member will learn the new TEK. When the key tree is balanced, the rekeying cost is $k \log_k N - 1$ keys. For example, suppose member U9 in Figure 2 is departing, all the keys he stores (K1, K4), except for his individual key, must be changed. The GC first encrypts the new K4, K4', with K11 and K12 for member U7 and U8 respectively. Finally, it encrypts K1' with the respective TEKs for all the group members.

If backward secrecy is required, then a join operation is similar to a remove operation in which the keys that the joining member receives must be different from the keys previously used in the group. The rekeying cost is $2 \log_k N$ keys when the key tree is balanced. Suppose member U9 is joining the group, the GC first encrypts K4' for member U7 and U8. Then it encrypts K1' with K1 for member U1 and U8. Finally, it encrypts K4' and K1' with K13 for member U9.

The efficiency of LKH depends critically on whether the key tree remains balanced. A key tree is considered balanced if the distance from the root to any two leaf node differs by not more than one. For a balanced key tree with N leaf nodes, the height from the root to any leaf node is $\log_k N$. However if the key tree becomes unbalanced, the distance from the root to a leaf node can become as high as N. Figure 3 shows an unbalanced key tree. First of all, we can see that key storage among members varies from 3 to 5 rather than 4 in a balanced LKH. Secondly, U1 or U2 needs 4 decryptions if any of its siblings departs rather than $\log_2 7$ decryptions in a balanced LKH and lastly, the rekeying cost for an unbalanced LKH is 7 keys rather than $2 \log_2 7$ keys in a balanced LKH when U1 or U2 departs since K1, K2, K4 and K8 need to be changed. In this example, the difference between balanced and unbalanced LKH varies slightly as the group size is small. In cases where the group consists over several thousands or millions of members such as pay-per-view, this effect can be very obvious.

Centralised multi-group key management scheme has been proposed in [10, 11] which considers related multicast sessions as a whole. There are three steps in the scheme. First, a subtree, known as SG-subtree, is constructed for each SG with the leaf nodes being the users of that particular SG. Next, a subtree, known as DG-subtree, is constructed for each DG. Finally, the leaf nodes of DG-subtrees and roots of SG-subtrees are connected to generate the key tree. Our approach differs from theirs as we are able to trade-off additional rekeying costs in order to obtain a balanced key tree. This is because we try to preserve the scalability of LKH by spreading the computation power equally among the members. Furthermore, if we properly place the members in the key tree, these additional rekeying costs can be considered as quite insignificant.

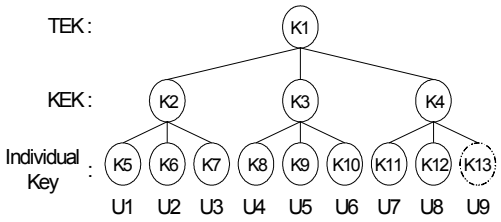

Figure 2. Logical key tree

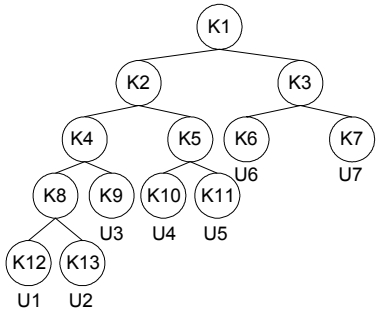

Figure 3. Unbalanced key tree

III. MULTI-LAYERS BALANCED LKH (MLB-LKH)

In this section, we discuss how we can minimise the key storage and rekeying costs for several related multicast sessions within LKH. Since the efficiency of LKH depends on the balance of the key tree, we trade-off additional rekeying messages for a balanced key tree. Figure 4 shows how the video encoded in cumulative layers in Figure 1 can be mapped onto our MLB-LKH. Each member not only receives the KEKs on its path to root but the TEKs that are used to encrypt the respective layers as well. It can be seen that the GC needs to multicast the TEK of EL1 twice.

The placement of the members in MLB-LKH is important since proper placement can minimise the number of rekeying messages needed by the members. As shown in Figure 4, all SGs form at least one key tree of its own and related SGs are placed side by side. As the multicast sessions might be dynamic, the GC must place the joining members according to their SGs. The worst case rekeying cost is $2 \log_k N + m$ if a joining member subscribes to the DG on the highest layer, where m is the highest layer in DG. Similarly for a depart event, the worst case rekeying cost is $k \log_k N + m$ if the departing member departs from the DG on the highest layer.

Switching between SGs is normal when a member requires additional service or does not require that service anymore. The only way is to treat that member as a departing member in the old SG and a joining member in its desired SG. For example, a member might wish to subscribe for a higher quality multicast video due to the ample amount of bandwidth. The worst case rekeying cost happens when the two SGs have no interception other than the root as shown in Figure 5(a). The rekeying cost is $(k + 2)(\log_k N - 1) - 1$. Suppose there is an interception along the path as shown in Figure 5(b), the best rekeying cost is $(k + 2)(\log_k N - l - 1) - 1$.

Some multicast applications such as military communications not only require the group key to be changed immediately after each membership changes, it may even be necessary to stop data flow while such groups are being rekeyed [13]. Therefore, in order to reduce the latency needed by members who subscribe more layers than others, one method is to perform pseudo random function (*PRF*) on TEK in the highest affected layer to generate TEK in the lower layer. Assume the highest affected SG is m, the members in SGm need to perform m *PRF*, $F^m(\text{TEK}_m) = \text{TEK}_0$, to obtain all the necessary TEKs for the multicast sessions as shown in Figure 6. As for the required KEKs, it can take its time to decrypt the necessary rekeying messages while receiving the multicast data at the same time. For example in Figure 4, one member in SG_{EL1} is departing; both TEK_{BL} and TEK_{EL1} need to be changed. The members in SG_{EL1} just need to receive the new TEK_{EL1}, TEK_{EL1}', and perform *PRF* on it to get TEK_{BL}'.

IV. SIMULATIONS AND PERFORMANCE COMPARISON

For simulation purpose, we adopted the scenario where the multicast video is encoded with 4 cumulative layers. We use a binary key tree with members ranging from 0 to 8096 and the members are dividing into 2/16, 6/16, 7/16 and 1/16 for each layer respectively.

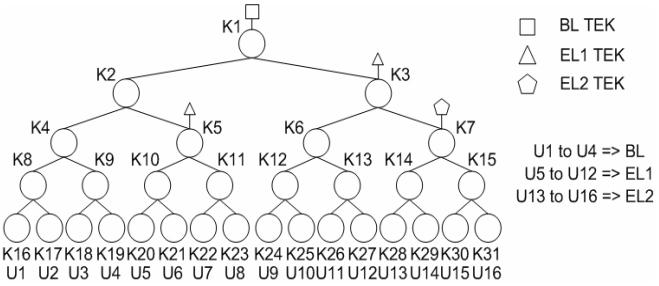

Figure 4. Mapping of video encoding in cumulative layers on MLB-LKH

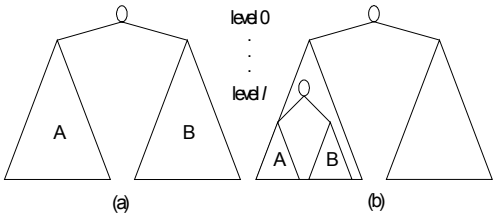

Figure 5. (a) Worst case and (b) Best Case of switching between SGs

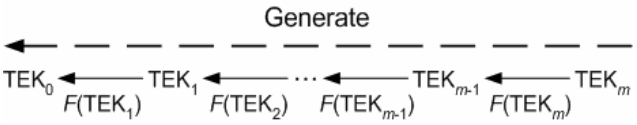

Figure 6. Generation of TEK using one-way chain

Figure 7 shows the key storage for the GC and group members. The GC's storage in LKH grows at the faster rate compared to MLB-LKH. This is because in LKH, each multicast session is considered separately and this causes a lot of unnecessary keys to be stored by the GC. Although there are several work on minimisation of key at the GC side, which can minimise these effects [14, 15], the members still require a significant amount of storage if each multicast session is considered separately; the more services the members subscribe to, the more redundant keys the members need to store. As for MLB-LKH, the difference in key storage between the members is at most m. From Figure 7(b), we can see that in LKH, the members in the highest layer store thrice the number of keys than the members in MLB-LKH.

For both approaches, once the number of members exceeds 1000, the members' storage grows at a much slower rate. This is because the total number of members that can be accommodated double each time for every increment in height. However, the number of members in the key tree has significant effects on the GC side since it needs to store the individual key of all group members as well as KEKs.

In Figure 8 and 9, we look at the effects of individual join and depart event in LKH and MLB-LKH. When the affected member is in layer 2 or higher, the rekeying cost for LKH is at least twice compared to MLB-LKH. Similarly as before, the higher the affected layer, the higher the rekeying costs. For both algorithms, the rekeying cost for joining member is higher

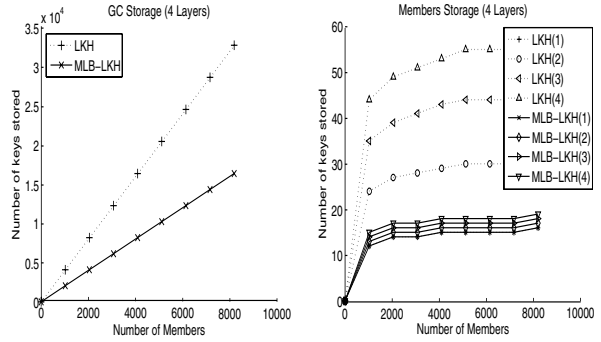

Figure 7. (a) GC storage and (b) members storage for LKH and MLB-LKH

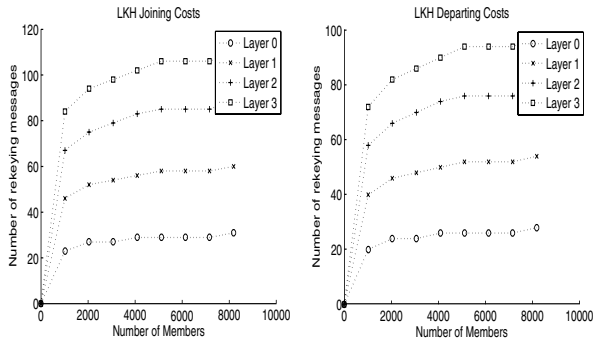

Figure 8. LKH rekeying costs - (a) joining and (b) departing

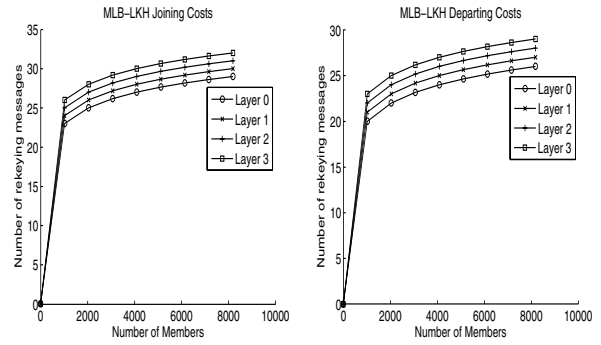

Figure 9. MLB-LKH rekeying costs - (a) joining and (b) departing

compared to departing member because we started with a completely balanced LKH for all layers. In this example, the rekeying cost for joining and departing member in each layer for LKH and MLB-LKH is $2 \log_2 N_0 + 2$ and $2 \log_2 N_0 - 1$ respectively, where N_0 is the number of members in the key tree.

As it is common for members to subscribe additional services or unsubscribe unwanted services, Figure 10 and 11 investigate all possibilities of switching for both algorithms. Usually, the rekeying cost for LKH is better compared to MLB-LKH when the group member switches by one layer regardless of whether that member is joining or departing. This is because only one key tree in LKH needs to be rekeyed for such changes whereas two SGs in MLB-LKH are affected. As

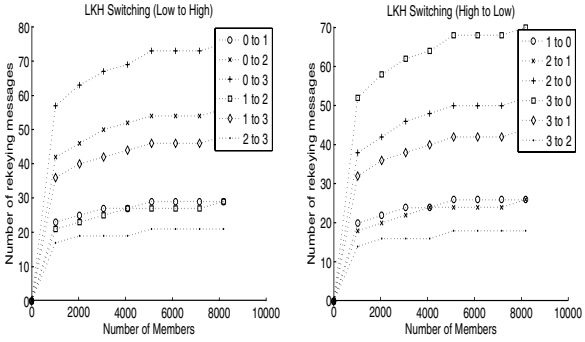

Figure 10. LKH switching - (a) Low to high and (b) High to low

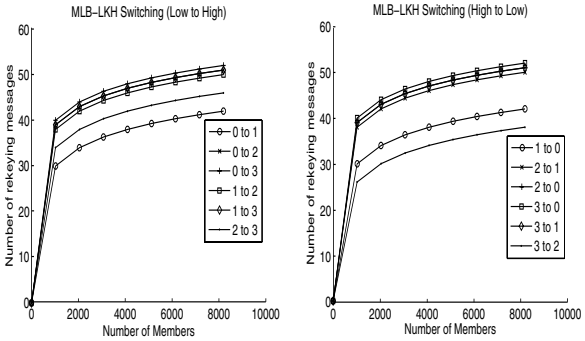

Figure 11. MLB-LKH switching - (a) Low to high and (b) High to low

for other switching possibilities, MLB-LKH tends to outperform LKH when two or more key trees in LKH are affected.

There is another optimisation that can be used to reduce the rekeying costs and latency when a member, $U_{x \to y}$, switches from SGx to SGy, where keys in SGx are a proper subset of keys in SGy. Rather than treating $U_{x \to y}$ as a departing member in SGx and a joining member in SGy as before, the GC can multicast the new TEK of SGy, TEK'$_{SGy}$, encrypted with the current TEK of SGy, TEK$_{SGy}$, to the existing members in SGy and unicast TEK'$_{SGy}$ to $U_{x \to y}$ encrypted with its individual key. This is because all the other TEKs and KEKs are unaffected. However, this optimisation requires the GC to keep records of the members who perform such switching. The actual switching of $U_{x \to y}$ takes place when there is a join, depart or other form of switching events. A point to be noted is that TEK'$_{SGy}$ does not need to be able to generate the TEK below it using *PRF* because all the other keys still remain the same.

Although centralised multi-group key management scheme considers related multicast sessions as a whole, our approach differs from theirs by trading off rekeying costs for similar key storage and number of decryptions among members. Suppose there are a lot of related multicast layers and if the higher layer is very heavily populated compared to the lower layer, the number of decryptions that are needed can be quite significant for centralised multi-group key management scheme. Reducing the number of decryptions might reduce the waiting latency as data flow might stop until all the remaining members get the keys.

For the simulation, we adopted the scenario where the video is encoded with 5 cumulative layers. Binary key tree with three different group sizes are used. The splitting of the members are 0.05%, 0.1%, 0.1%, 0.3% and 0.45% for each layer respectively.

From Figure 12, we can see the difference between the lowest and highest layer for centralised multi-group key management is around 10 keys. This means that every member that is joining or departing the highest layer will need 10 additional decryptions compared to the member that is joining or departing in the lowest layer. Furthermore, the rekeying cost will be higher as explained in section II. As for MLB-LKH, the difference between the lowest and highest layer is just the number of layers in the multicast session.

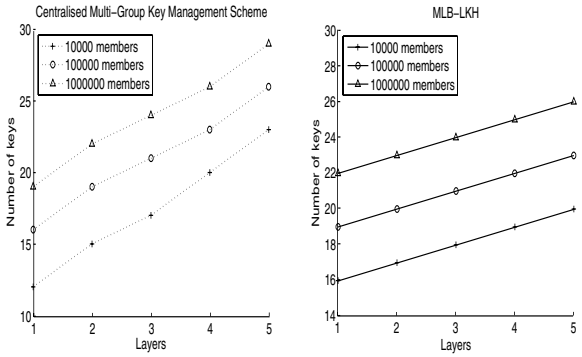

Figure 12. Number of key held by members at different layers: (a) Centralised multi-group key management scheme and (b) MLB-LKH

V. CONCLUSION

In this paper, we have discussed how LKH is used to secure multicast communication. First, we described how LKH can be used to secure a multicast session for a group of members. From there, we extended this idea to secure several related multicast sessions. In addition, we have also taken the balance of LKH into consideration.

If related multicast sessions are considered as a whole, the total number of keys needed in the key tree and the rekeying costs are greatly reduced. For our simulation using video encoded with 4 cumulative layers, the key storage at the GC and members' side is reduced by at least half. As for the individual join and depart rekeying costs, the higher the affected layer, the higher the rekeying costs. In the case of MLB-LKH, the difference in rekeying costs for a join or depart event is at most *m*. However, we observe that when a member switches by one layer, regardless of whether that member is joining or departing the group, the rekeying cost is generally lower for LKH due to the fact that only that layer key tree needs to be rekeyed whereas two SGs in MLB-LKH are affected. For other switching possibilities, MLB-LKH tends to be better than LKH. However, MLB-LKH requires proper placement of members in the key tree in order to minimise the rekeying costs. Most importantly, the members must be grouped according to their SGs. When compared to centralised multi-group key management scheme, we trade-off rekeying costs for a balanced key. This not only allows the member to have similar key storage but similar number of decryptions whenever there is change in group membership.

Two optimisation techniques are proposed to further enhance our algorithm. Since it might be necessary for some multicast applications to stop data flow during rekeying, one method to reduce latency is to perform *PRF* on the TEK in highest affected layer to get the TEK in the lower layer. As for KEKs, the members can take its time to decrypt the rekeying messages while receiving the multicast data. Another optimisation is when a member switches between layers, where keys in the old layer are a proper subset of the keys in the new layer. The GC just needs to generate another TEK for the new layer and send it to the existing members in that layer and the joining member.

REFERENCES

[1] S. .E. Deering, "Host Extensions for IP Multicasting", RFC 1112, IETF, Aug 1989.

[2] D.M. Wallner and E.J. Harder and R.C. Agee, "Key management for multicast issues and architectures", RFC 2627, Jun. 1999.

[3] C. Wong, M. Gouda and S. Lam, "Secure group communication using key graphs", IEEE/ACM Trans. Networking, Vol. 8. pp. 12-23, Feb. 2000.

[4] S. Mittra, "Iolus: A framework for scalable secure multicasting", In Proc. ACM SIGCOMM, Vol. 27, pp. 277-288, Sept. 1997.

[5] D. . Balenson, D. McGrew and A. Sherman, "Key Management for large dynamic groups: One-way function trees and amortised initialisation", Internet Draft, *draft-irtf-smug-groupkeymgmt-oft-00.txt*. Aug. 2000.

[6] M. Valdvogel, G. Caronni, D. Sun, N. Weiler and B. Plattner, "The versakey frameworks: versatile group key management", IEEE JSAC (Special Issue on Middleware), Vol. 17, No. 9, pp. 1614-1631, Sept 1999.

[7] I.Chang, R. Engel, D. Kandlur, D. Pendarakis, D. Saha, "Key management for secure Internet multicast using boolean function minimization techniques. IEEE INFOCOM, Mar. 1999

[8] M.J. Moyer, J.R. Rao, P. Rohatgi, "Maintaining balanced key trees for secure multicast", Internet Draft, *draft-irtf-smug-key-tree-balance-00.txt*, Jun. 1999.

[9] J. Pegueroles, F. Rico-Novella, "Balanced Batch LKH: New proposal, implementation and performance evalution", IEEE Symposium on Computers and Communications (ISCC), Jun. 2003.

[10] Y. Sun, K.J. Ray Liu, "Multi-layer management for secure multimedia multicast communications", IEEE Iternational Conference on Multimedia and Expo (ICME), Vol. 2, pp. 205-208, Jul. 2003.

[11] Y. Sun, K.J. Ray Liu, "Scalable hierarchical access control in secure group communications", IEEE INFOCOM, Hong Kong, Mar. 2004.

[12] M. P. Howarth, S. Iyengar, Z. Sun, H. Cruickshank, "Dynamics of key management in secure satellite multicast", IEEE JSAC, Feb 2004.

[13] B. DeCleene, L. Dondeti, S. Griffin, T. Hardjono, D. Kiwior, J. Kurose, D. Towsley, S. Vasudevan, C. Zhang, "Secure group communication for wireless networks", In Proc. MILCOM, Oct. 2001.

[14] Y. Tseng, "A scalable key management scheme with minimizing key storage for secure group communications", International Journal of Network Management, Nov. 2003.

[15] J. Pegueroles, J. Hernandez-Serrano, F. Rico-Novella, M. Soriano, "Adapting GDOI for balanced batch-LKH", Internet Draft, *draft-irtf-gsec-gdoi-batch-lkh-00.txt*, Jun. 2003.