Chinese Remainder Theorem Based Group Key Management

Xinliang Zheng Chin-Tser Huang Manton Matthews
Dept. of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
{zheng2, huangct, matthews}@cse.sc.edu

ABSTRACT

In this paper, we present two new centralized group key management protocols based on the Chinese Remainder Theorem (CRT). By shifting more computing load onto the key server we optimize the number of re-key broadcast messages, user-side key computation, and number of key storages. The first protocol is the base Chinese Remaindering Group Key (CRGK) protocol, which with a group of n users requires the key server to do O(n) XORs, additions, multiplications, and Extended Euclidean Algorithm computations and broadcast 1 re-key message; each individual user is required to do only 1 modulo arithmetic and 1 XOR operation for each group key update. The second protocol is the Fast Chinese Remaindering Group Key (FCRGK) protocol, which only requires the key server to do O(n) XORs, additions, and multiplications most of the times with no change to the number of re-key messages and user computation per group key update. For both protocols each user only needs to store 2 keys all the time. One special attraction for our FCRGK protocol is that it allows most of the re-keying computation to be done preemptively, which means when a user-join or user-leave event happens the response time for the key server to send out the new group key can be very short.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection.

General Terms

Design, Management, Security, Performance.

Keywords

Chinese Remainder Theorem (CRT), Congruence System, Group Key Management, Chinese Remaindering Group Key (CRGK) Protocol, Fast Chinese Remaindering Group Key (FCRGK) Protocol.

ACMSE 2007, March 23-24, 2007, Winston-Salem, North Carolina, USA
©Copyright 2007 ACM 978-1-59593-629-5/07/0003...$5.00

1. INTRODUCTION

As group-oriented applications become increasingly popular, the need for confidentiality of group communications also grows. While there are many mature secure protocols for peer-to-peer communication, the scenario for group communication with dynamically changing members is very different. Efficient agreement on a new group key after user join or leave is crucial to group communication confidentiality. During the past decade a variety of group key management protocols have been proposed. Among them are a set of efficient and scalable centralized group key management protocols [6], [16], [19], [22], [23] based on certain hierarchical structure require about O(log n) of keys to be received, decrypted or computed, and stored by each individual group user for a group of n users. While this is already an improvement compared to previous schemes, it may still represent a large overhead for group users with limited capacity.

In this paper, we introduce two new centralized group key protocols based on the CRT. By shifting more computing load onto the key server, we optimize the number of re-key broadcast message, user-side key computation and number of key storages. Our protocols require the key server to broadcast 1 re-key message and each group user to compute only 1 modulo arithmetic and 1 XOR operation for each key update and store only 2 keys all the time. While our protocols require more computation power from the key server, it does not need to maintain any complex hierarchical structure. With the tremendous advantage on re-key broadcasting message number, user key computation, user key storage, and the relatively simple nature compared to other protocols, we consider our protocols are well worth exploring.

While this paper deals with group key management for dynamic group, our protocols are based on the assumption that certain authentication protocol involving two parties is needed before the key server grants group access to each user. Since two-party authentication protocols are well studied [20], it is not included in the scope of this paper.

The rest of this paper is organized as follows. Section 2 provides a brief review of related work. Section 3 describes our protocols in detail. Security analysis and performance evaluation are given in Section 4 and 5 respectively. Section 6 summarizes our conclusions.

2. RELATED WORK

Group key management protocols can be largely classified into three groups [17]: centralized, with one center managing the whole group; decentralized, with group and subgroup controllers managing the group; distributed, with no group center and each group member can contribute to the group management.

Ballardie [2] proposed a multicast key distribution protocol, which can be classified as decentralized protocol. This protocol requires certain support mechanism to be integrated into version 3 of IGMP and it provides no forward secrecy. Mittra [15] proposed another typical decentralized protocol: Iolus. This protocol decentralizes the group control to each subgroup controller (Group Security Agent). Since it lacks a general group key, the real multicast data need to be relayed (decrypted and re-encrypted) by each subgroup controller which impedes the performance of real data multicasting.

Steiner et al. [21], Kim et al. [14] proposed distributed group key protocols based on Group Diffie Hellman methods for small dynamic peer groups. Rodeh et al. [18] proposed a distributed logical key hierarchy protocol using AVL trees. Those protocols require many rounds of messages to update a new group key. Their contributory nature may only attract applications involving small group of peer users.

The class of centralized group key protocols is the most widely explored group key protocols among the three. Harney and Muckenhirn [12], [13] proposed a group key management protocol by extending two-party shared key establishing scheme into group case. It requires O(n) encryptions to update a group key when user joins or leaves if backward and forward secrecy are required. A set of scalable hierarchical structure based group key protocols [6], [16], [19], [22], [23] have been proposed. In general those protocols requires the key server to store about 2n keys and update O(log n) keys each time re-keying is needed, and each user stores O(log n) keys (or secret information) and performs O(log n) decryptions or some type of computation per group key update. Eltoweissy et al. [10] proposed a protocol based on Exclusion Basis Systems, a combinatorial formulation of the group key management problem, which allows protocol users to trade-off between number of keys needed to be stored and the number of messages needed to be transmitted for each key update with no collusion solution provided. Fiat and Naor [11] take the information theoretic approach and propose k-resistant protocol, i.e. coalitions of up to k users are secured, with each user storing O(k log k log n) keys and the server broadcasting $O(k^2 \log^2 k \log n)$ messages per re-keying. Chiou et al. [9] proposed a secure broadcasting protocol also based on CRT, however its application of CRT is different from our approach. Their protocol requires O(n) encryptions for each real data broadcast while ours only needs 1 encryption.

3. CHINESE REMAINDERING GROUP KEY PROTOCOL

In this section, we introduce our new group key management protocol based on the Chinese Remainder Theorem (CRT). First, we will give a brief review on CRT; then we will present our base Chinese Remaindering Group Key (CRGK) protocol in detail by describing the protocol behavior in subsections: group initialization, member join, member leave, mass join, mass leave, and key refresh; finally, we will introduce our Fast CRGK (FCRGK) protocol following the similar structure except that a new subsection about group expansion operation is added.

3.1 Chinese Remainder Theorem

Let $u_1, \ldots, u_m$ be m pairwise relatively prime positive integers, and let $k_1, \ldots, k_m$ be m arbitrary integers. Then CRT states that the congruence system

$$\begin{aligned} X &\equiv k_1 \pmod{u_1} \\ &\vdots \\ X &\equiv k_m \pmod{u_m} \end{aligned}$$

has a unique solution modulo $u_1 \ldots u_m$.

To compute the unique solution X we can do

$$\begin{aligned} X &= \sum_{i=1}^{m} k_i M_i M_i' \pmod{M}, \text{ where} \\ M &= u_1 \cdots u_m, \\ M_i &= M / u_i, \\ M_i' &\text{ is the multiplicative inverse of} \\ M_i &\bmod u_i, \text{ i.e. } M_i M_i' \equiv 1 \pmod{u_i}. \end{aligned}$$

Since M_i is relatively prime to u_i there must exist a unique multiplicative inverse mod u_i. Then the above computation of the unique solution X is well defined. Efficient computation of the multiplicative inverse can be carried out using Extended Euclidean Algorithm which is out of the scope of this paper.

3.2 The Base CRGK Protocol

As we stated in the introduction, this paper is dealing with group key management with a key server. Authentication of group users is assumed and is not part of our protocol. After the authentication of a user, if the user is granted access to the group communication then the key server will choose a private key and send it to the user using some secured channel, for instance using public key system. For the purpose of our protocol we impose another constraint on the private key, that is, the key server will pick this private key from a pool of pairwise relatively prime positive integers and the size of this private key should be much larger than the size of group keys we need to generate. Let n be the number of group users joining the group communication. We will construct the first group key in group initialization.

3.2.1 Group Initialization

After the key server communicates a private key, picked randomly from a pairwise relatively prime integer pool, to each initial group user securely, the key server will pick an initial group key K randomly and build the following congruence system for this group with n initial users.

$$\begin{aligned}
X &\equiv k_1 \pmod{u_1} \\
&\vdots \\
X &\equiv k_n \pmod{u_n}, \text{ where}
\end{aligned}$$

n is the size of the current group,
k_i is the value of the corresponding bits of $K \oplus u_1$, for all $i \in \{1,2,\cdots,n\}$,
K is the initial group key,
$u_1,\cdots,u_n$ are the private keys picked by the key server for each group user in the initial group respectively, and they are pairwise relatively prime positive integers.

Obviously, the above congruence system meets the requirement of a CRT congruence system. Therefore the key server can compute the unique solution X for the above congruence system. After the X value is computed the key server can simply broadcast this X value to all users in plaintext. Any of the initial n group user can compute the group key K by simply do 1 modulo and 1 XOR operations (K is equal to the corresponding bits of (X mod u_i) XOR u_i.). After these simple computations the n group users now share the same group key K and any outside user can not compute the shared group key K without any of the secret information u_1, …, u_n.

3.2.2 *Member Join*

After the initial group has been set up, if a new user wants to join the group then it will first go through the same authentication process as other group users do. If the joining new user is granted the access to the group communication, it will be assigned a new private key u_{new}. This key is picked by the key server from the pool of pairwise relatively prime positive integers just like u_1, …, u_n. Now the key server will merge the new user's private key into the initial congruence system. Then a new group key K′ is chosen following the same requirements as described in group initialization. Then again the key server will compute the new X value based on the updated congruence system.

$$\begin{aligned}
X &\equiv k_1 \pmod{u_1} \\
&\vdots \\
X &\equiv k_n \pmod{u_n} \\
X &\equiv k_{new} \pmod{u_{new}}, \text{ where}
\end{aligned}$$

k_i is the value of the correspond ing bits of $K' \oplus u_1$, for all $i \in \{1,2, \cdots, n, new\}$,
K is the initial group key,
u_{new} is the private keys picked by the key server for the new joining group user and it is pairwise relatively prime to $u_1,\cdots,u_n$.

After receiving the new broadcast X value each user can compute the new group key K' easily by doing 1 modulo and 1 XOR operations as in group initialization.

3.2.3 *Member Leave*

Group key updating when a member leaves usually requires more efforts in most other group key management protocol since we can not use the old group key to encrypt the new group key. However in our protocol group key updating for member leave are also very simple. What we need to do is just take a leaving user's private key out of the congruence system, pick a new group key, compute the new X value, and broadcast to each user. Note that to prevent future misuse this key should be taken out of the pool of pairwise relatively prime positive integers.

$$\begin{aligned}
X &\equiv k_1 \pmod{u_1} \\
&\vdots \\
&\cancel{X \equiv k_i \pmod{u_i}} \\
&\vdots \\
X &\equiv k_n \pmod{u_n}, \text{ where}
\end{aligned}$$

u_i leaves the group.

Again, after receiving the X value each user only needs to do 1 modulo arithmetic and 1 XOR operation to get the new group key.

3.2.4 *Mass Join*

Mass join scenario is very similar to the single member join case. The only difference in this case is that it will have a set of new private keys, one for each new joining member respectively, being added into the congruence system. New group key is computed and distributed the same way as in the single member join case.

3.2.5 *Mass Leave*

The behavior of a mass leave is very similar to the single member leave case. The difference is that the key server needs to take more than one user private keys out of the congruence system. New group key is computed and distributed the same way as in the single member leave case.

3.2.6 *Key Refresh*

Key refresh is also very simple in our protocol. The only thing the key server needs to do is to pick a new group key and compute the new X value and broadcast it to each group user. If the intermediate results of the previous key updating computation are saved, such as the values of $M_i M_i'$ for all i in [1, …, n], then the key refreshing computation will be much faster. For group users the process of getting the new group key is the same as before, namely 1 modulo arithmetic and 1 XOR operation.

3.3 The Fast CRGK (FCRGK) Protocol

For our base CRGK protocol to set up or update a group key it is very efficient from the perspective of number of communication message, user computation effort, and user storage requirement, since it needs 1 plaintext broadcast message, 1 modulo arithmetic and 1 XOR operation, and 2 key storage spaces for each user. However for the key server side, except for the key refresh scenario, it still requires O(n) of XOR, multiplication, addition, multiplicative inverse computations. In this section, we present the Fast CRGK (FCRGK) protocol which performs much faster than our base CRGK protocol. Even more, with the FCRGK protocol the key server can do most of the computation long before members join or leave the group, so that when the joining or leaving occurs the key updating message can be sent out by the key server immediately with very little computation and latency. To gain this performance improvement on most of the member join or leave events the key server needs to perform an additional group expansion operation when the number of group member change total reaches certain multiple of the starting group size. The group expansion operation is expensive in the sense that it is about the same scale of a group initialization, but it is designed to be carried out very infrequently. Also, the key server needs more storage space to execute the FCRGK protocol when compared to the base CRGK protocol.

Similar to our base CRGK protocol, the FCRGK protocol also assumes each group user will be authenticated through some method and the key server will assign each of them a private key from a pool of pairwise relatively prime positive integers. The following is the detailed description of the FCRGK protocol with a variable n number of group users.

3.3.1 Group Initialization

Group initialization for FCRGK is largely similar to the CRGK case. The difference will be the size of the congruence system. In the CRGK protocol the size of the congruence system is the same as the initial group size n, while in the FCRGK protocol we will construct a congruence system larger than the initial group size n. Assume m is the size of the initial congruence system we built. We will pick that $\mathbf{m} = \mathbf{nd}$ for some constant d value, which is dependent on how fast group members change (joining and leaving). For instance for a slowly changing group we can choose $\mathbf{d} = 2$, and for a fast changing one we can choose $\mathbf{d} = 5$. The actual value of d can be adjusted by the key server based on the group dynamics.

$$\begin{aligned}
&X \equiv k_1 \pmod{u_1}\\
&\quad\vdots\\
&X \equiv k_n \pmod{u_n}\\
&X \equiv k_{n+1} \pmod{u_{n+1}}\\
&\quad\vdots\\
&X \equiv k_m \pmod{u_m}\text{, where}\\
&\qquad n \text{ is the size of the initial group,}\\
&\qquad m = nd,\ \text{for some value d, which is}\\
&\qquad \text{determined by the group dynamics,}\\
&\qquad k_i \text{ is the value of the corresponding}\\
&\qquad \text{bits of } K \oplus u_1, \text{for all } i \in \{1,2,\cdots,n\},\\
&\qquad K \text{ is the initial group key,}\\
&\qquad k_j \neq K, \text{is some randomly picked value,}\\
&\qquad \text{for all } j \in \{n+1, n+2, \cdots, m-1\},\\
&\qquad u_1, \cdots, u_n \text{ are the private keys}\\
&\qquad \text{picked by the key server for each group}\\
&\qquad \text{user in the initial group respectively,}\\
&\qquad u_{n+1}, \cdots, u_m \text{ are picked from the pool of}\\
&\qquad \text{pairwise relatively prime positive integers}\\
&\qquad \text{and serve as the reserve for the private keys}\\
&\qquad \text{of future joining users.}
\end{aligned}$$

For the FCRGK protocol we require that during the group initialization and the group expansion operations in the future, the intermediate calculation results $M_i M_i'$ for all i in [1, …, m] and $k_i M_i M_i'$ for all i in [n+1, …, m] need to be saved by the key server to reduce the future computation cost. Also, if the key server has additional computation power and storage space, it can pick a new future group key, pre-compute the new X value, and save the intermediate results including the X value for preparing a quicker response to a future group member change.

3.3.2 Member Join

When new member joins the group, in our FCRGK protocol the key server just picks u_{n+1} from the current congruence system and gives it to the new group user as its private key. If the key server has a pre-computed X value based on the pre-picked new group key, then the X′ value, which is the real new group key updating message, can be easily calculated in one step as follows:

$$\begin{aligned}
&\mathbf{X}' = \mathbf{X} - \mathbf{k}_{\mathbf{n}+1}\mathbf{M}_{\mathbf{n}+1}\mathbf{M}'_{\mathbf{n}+1} + \mathbf{k}'_{\mathbf{n}+1}\mathbf{M}_{\mathbf{n}+1}\mathbf{M}'_{\mathbf{n}+1} \pmod{\mathbf{M}},\\
&\textbf{where}\\
&\qquad X' \text{ will be the new key updating message,}\\
&\qquad X \text{ is the pre-computed value based on}\\
&\qquad \textbf{a} \text{ new group key } K',\\
&\qquad \mathbf{k}_{\mathbf{n}+1} \text{ is the old randomly picked value,}\\
&\qquad \mathbf{k}'_{\mathbf{n}+1} \text{ is the value of the corresponding bits of } K' \oplus u_{n+1}.
\end{aligned}$$

If there is no such pre-computed X value saved by the key server, then the key server will simply pick a new group key and use the intermediate results saved from the last group initialization or group expansion operation. Those intermediate calculation results can be used here but not in the CRGK protocol because for the FCRGK protocol the foundation of the congruence system, i.e. u_1, …, u_m, is not changed after a new member join.

3.3.3 Member Leave

After a user i left the group, to update the group key the key server will move u_i from the range 1 ~ n to the range n+1 ~ m and a new random value k_i (as long as k_i is not equal to the new group key value) is picked to be associated with u_i for new X value computation, by which the foundation of the congruence system is kept stable so that the intermediate calculation results saved before can be used again. Obviously the u_i value will be marked as used and cannot be assigned as a private key for any future new joining user.

3.3.4 Group Expansion

After group initialization since the foundation of the congruence system is kept the same the original secret reserve $u_{n+1}, \ldots, u_m$ will gradually be used up after many group member changes. Group expansion operation will expand the size of the current congruence system associated with the group by a factor d, i.e. make $\mathbf{m} = \mathbf{md}$ .Group expansion is a new operation specific to the FCRGK protocol and group expansion should be carried out very infrequently. If the key server has to carry out group expansion operation frequently, then it is an indication that the value of d should be increased.

Let $\mathbf{r} = \mathbf{m} - \mathbf{n} - \mathbf{tl}$, where tl is the total number of leaves since the last group expansion (or group initialization if no group expansion has been executed yet), then group expansion operation is carried out each time the value r reaches some predefined threshold. The value of the threshold should be determined by group dynamics and the key server computing power. In other words, based on the current group dynamics the time duration needed for the key server to do a group expansion computation is the key factor of determining the threshold value. The longer the time needed, the bigger the threshold value should be.

The duty of group expansion includes kicking out all the private key values used by group members that have left the group, adding new secret u values to make up the new size m of the congruence system, picking the new group key, calculating the new X value, and saving all the intermediate results like in the group initialization operation. The new X value is then broadcast to each user to compute the new group key.

3.3.5 Mass Join

Mass join is similar to the single member join case and the only difference will be a set of u's chosen from $u_{n+1}, \ldots, u_m$ to be used as the private keys for each new user respectively.

3.3.6 Mass Leave

The behavior of the mass leave case is similar to the single member leave case except that a set of u's are moved from the range 1 ~ n to the range n+1 ~ m.

3.3.7 Key Refresh

Key refresh could be exactly the same as in the base CRGK protocol as long as the intermediate results are used to compute the new group key, since in both cases the foundation of the congruence system is not changed. If pre-computed X value exists then when the time for key refreshing comes, the X value can simply be broadcast.

4. SECURITY ANALYSIS

The security of our protocols is based on the assumption that each current group user will keep its private key ($u_1, \ldots, u_n$ respectively) secret, and the key server will keep a set of secret information (called *Server Secret Set*), the private key reserve for the FCRGK protocol, a subset of the set $u_{n+1}, \ldots, u_m$. Moreover, the set of u values are randomly picked from an unlimited large pool of pairwise relatively prime positive integers, hence knowing one number gains little knowledge about the others.

4.1 Forward Secrecy

Forward Secrecy is about preventing leaving users to continue accessing future group communications. For the base CRGK protocol each leaving user's private key is kicked out of the new congruence system, therefore its private key no longer contributes to new X value calculation which means it can no longer compute the new group key like other current group users do. For the FCRGK protocol, even though each leaving user's private key is still kept inside the congruence system for speeding up server computation, it can only compute the k_i value (see section 3.3.3) which is randomly picked and not related to the new group key in any way at all.

4.2 Backward Secrecy

Backward Secrecy is about preventing joining users from accessing previous group communications. In our protocols each new group key is an arbitrary picked value with no relation to any old group key. Each new joining user's ability of computing the new group key will not gain knowledge about the previous group key.

4.3 Collusion Attack

Collusion attack on group key protocol is about a set of previous group users working together to try to gain access to the secret group key. For the base CRGK protocol, since the foundation of the congruence system is changed after each group member change, any number of previous users' collusion will not gain any significant more information about the congruence system as long as the pairwise relatively prime integers are large enough. For the FCRGK protocol, since the congruence system remains stable with certain number of member changes, collusion of previous users may gain some information about the congruence system, for instance leaving users know that the u values they held as private keys are still used in the congruence system before the next group expansion operation, but again without server secret set they still can not compute the secret group key.

5. PERFORMANCE EVALUATION

Our protocols focus on the optimization of user computation, number of stored keys, and number of the re-key message with loading certain amount of computation on the key server. On the other hand comparing to other hierarchical structure based group key management protocol, our protocol's simple nature, only requiring one two-dimensional table structure, keeps a minimal structure management load on the key server.

5.1 The Base CRGK Protocol

For each key update request our base CRGK protocol requires the key server to do O(n) XOR, addition, multiplication, and Extended Euclidean Algorithm computation and broadcast 1 re-key message, while each user only need to do 1 modulo arithmetic, 1 XOR operation, and store 2 keys all the time, one of which is the private key and the other is the group key.

5.2 The FCRGK Protocol

With the very infrequent group expansion operations the FCRGK protocol requires the key server to do O(n) XOR operation, addition and multiplication arithmetic, and still keeps re-key message number, user computation and key storage space minimal.

One very attractive quality of the FCRGK protocol is that by maintaining a stable congruence system most of the re-key computation can be done beforehand. That is, the key server can use its free time to get future group keys almost ready. When a key update is needed, no matter it is because of user join, user leave or simple key refreshing, the key server only needs to do O(1) operation to respond to that request. To the best of our knowledge, we see no other protocols allow such key pre-computation.

6. CONCLUSIONS

In this paper, we present two centralized simple and efficient CRT based group key management protocols for small to medium size dynamically changing groups. The simple nature of our protocols together with minimal re-key messages, minimal requirements on user computation and key storage space make them suitable for a variety of secure group communication. Even though we evaluate our protocol performance using some O-notations, the unit operation of our protocols (mainly XOR, addition, multiplication, and modulo arithmetic) is different from most of other protocols (mainly encryption, decryption, and hashing) which can cause real performance results to be deviated. Our future work will involve providing optimized implementation of our protocols to evaluate their real time performance and the comparison with other protocols.

7. REFERENCES

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the Performance of Group Key Agreement Protocols", ACM Transactions on Information and System Security, vol. 7, no. 3, Aug. 2004.

[2] A. Ballardie, "Scalable Multicast Key Distribution", RFC 1949, May 1996.

[3] C. Blundo and A. Cresti, "Space Requirements for Broadcast Encryption", Proceedings of Advances in Cryptology - Eurocrypt '94, May 1994.

[4] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences", Proceedings of Advances in Cryptology - Crypto '92, Aug. 1992.

[5] M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System", Proceedings of Advances in Cryptology — EUROCRYPT'94, May 1994.

[6] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions", Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies - INFOCOM 1999, Mar. 1999.

[7] R. Canetti, T. Malkin, and K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption", Proceedings of Advances in Cryptology - Eurocrypt '99, May 1999.

[8] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques", Proceedings of IEEE Infocom'99, Mar. 1999.

[9] G.-H. Chiou and W.-T. Chen, "Secure broadcasting using the secure lock", IEEE Transactions on Software Engineering, vol. 15, no. 8, pp. 929-934, Aug. 1989.

[10] M. Eltoweissy, M. H. Heydari, L. Morales, and I. H. Sudborough, "Combinatorial Optimization of Group Key Management", Journal of Network and Systems Management, vol. 12, no. 1, Mar. 2004.

[11] A. Fiat and M. Naor, "Broadcast Encryption", Proceedings of Advances in Cryptology - Crypto '93, Aug. 1993.

[12] H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, Jul. 1997.

[13] H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, Jul. 1997.

[14] Y. Kim, A. Perrig, and G. Tsudik, "Tree-Based Group Key Agreement", ACM Transactions on Information and System Security, vol. 7, no. 1, pp. 60-96, Feb. 2004.

[15] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", Proceedings of the ACM SIGCOMM '97, Sept. 1997.

[16] A. Perrig, D. Song, and J. D. Tygar, "ELK, a New Protocol for Efficient Large-Group Key Distribution", Proceedings of IEEE Symposium on Security and Privacy (S&P), May 2001.

[17] S. Rafaeri and D. Hutchison, "A Survey of Key Management for Secure Group Communication", ACM Computing Surveys, vol. 35, no. 3, pp. 309-329, Sept. 2003.

[18] O. Rodeh, K. P. Birman, and D. Dolev, "Using AVL Trees for Fault Tolerant Group Key Management", International Journal of Information Security, vol. 1, no. 2, Feb. 2002.

[19] A. T. Sherman and D. A. McGrew, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees", IEEE Transactions on Software Engineering, vol. 29, no. 5, May 2003.

[20] R. E. Smith, “Authentication, From Passwords to Public Keys”, Addison-Wesley, 2001.

[21] M. Steiner, G. Tsudik, and M. Waidner, "Key Agreement in Dynamic Peer Groups", IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 8, Aug. 2000.

[22] D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, Jun. 1999.

[23] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs", IEEE/ACM Transactions on Networking, vol. 8, no. 1, pp. 16-30, Feb. 2000.