A Scalable Group Key Management Protocol

Song, Ronggong; Korba, Larry; Yee, George

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

*A Scalable Group Key Management Protocol **

Song, R., Korba, L., Yee, G.
2008

* published in the Journal of IEEE Communications Letters, Volume 12. 2008. NRC 50355.

Canada

A Scalable Group Key Management Protocol

Ronggong Song, *Senior Member, IEEE,* Larry Korba, George O.M. Yee, *Senior Member, IEEE*

***Abstract*—Group key management brings challenges on scalability for multicast security. In this paper, we propose a new group key management protocol and demonstrate that it has better scalability when compared with other important centralized protocols.**

***Index Terms*—group key management, multicast security, scalability.**

I. Introduction

Multicast applications have grown and greatly influenced our life along with the growth of the Internet. Examples of such applications include video conferencing, interactive group games, TV over Internet, e-learning, and public stock quote broadcasting. As an important and mandatory building block for multicast applications, multicast security has been extensively researched in the past decades for protecting multicast communications. The research on multicast security addresses authentication, confidentiality, and access control, among other areas, where group key management is a key component. However, scalability is still a hard problem and a sizable challenge for group key management technologies.

The latest and more efficient centralized group key management protocols are the Local Key Hierarchy (LKH) protocols presented by Wong et al. [1] and Wallner et al. [2]. They reduce the re-key messages and encryption operations from $O(n)$ to $O(\log n)$ when compared to the Group Key Management Protocol (GKMP) [3, 4] and Secure Lock [5], where n is the number of group members. However, they are still vulnerable to scalability issues when the group size goes up to millions of members and the re-key messages require strong security protection such as signature.

In this paper, we propose a new group key management protocol (SGKMP) based on the Chinese Remainder Theorem and a hierarchical graph in which each node contains a key and a modulus. The new protocol reduces the hashing operations from $O(\log n)$ to 1 when compared to LKH, and the length of the lock from $O(n)$ to $O(\log n)$ when compared to the Secure Lock, by using a hierarchy modulus graph, which makes the length of the secure lock more scalable. We demonstrate that the new protocol has better scalability through a detailed comparison and performance testing.

The remainder of the paper is organized as follows. Section II presents our new SGKMP protocol. Section III presents our proposed scalability metrics and compares the new protocol with others. Section IV shows the testing performance of the new protocol. Section V gives our conclusions.

Manuscript received.

R. Song is with the Institute for Information Technology, National Research Council Canada, Ottawa, Ontario K1A 0R6, Canada (phone: 613-990-6869; fax: 613-952-7151; e-mail: ronggong.song@nrc-cnrc.gc.ca).

L. Korba is with Institute for Information Technology, National Research Council Canada, Ottawa, Ontario K1A 0R6, Canada (e-mail: larry.korba@nrc-cnrc.gc.ca).

G. Yee is with Institute for Information Technology, National Research Council Canada, Ottawa, Ontario K1A 0R6, Canada (e-mail: george.yee@nrc-cnrc.gc.ca).

II. A Scalable Group Key Management Protocol

Our new scalable group key management protocol is based on the following: the Chinese Remainder Theorem and a hierarchical graph in which each node contains a key and a modulus. The protocol is designed to minimize re-key messages, bandwidth usage, encryption, and signature operations.

Chinese Remainder Theorem: Let $m_1, m_2, \ldots m_n$ be n positive integers where they are pairwise relatively prime (i.e. $gcd(m_i, m_j)=1$ for $i \neq j$, $1 \leq i, j \leq n$), $R_1, R_2, \ldots R_n$ be any positive integers, and $M = m_1 \cdot m_2 \cdot \ldots \cdot m_n$. Then the set of linear congruous equations $X \equiv R_1 \bmod m_1, \ldots, X \equiv R_n \bmod m_n$ have a unique solution as: $X = \sum_{i=1}^{n} R_i M_i y_i \bmod M$ where $M_i = M/m_i$ and $y_i = M_i^{-1} \bmod m_i$.

In the new protocol, the keys and moduli are constructed as a tree and maintained by the key server. The tree graph is similar to the tree graph in the LKH protocol but each node of the tree in the new protocol is assigned two values: a key and a modulus. Figure 1 depicts the key and modulus graph, where *TEK* is a traffic encryption key, k_{ij} is a key encryption key, and m_{ij} is a modulus.

Moduli Maintenance: The key server needs to store $2\log_2 n$ moduli and each member needs to store $\log_2 n$ moduli but they do not need to keep the moduli secret. The sibling nodes in the tree graph are assigned with two different moduli (i.e., m_{i1} and m_{i2} where i is the depth of the tree) and the nodes in the different level of the tree are assigned with the different moduli but each a pair of siblings at the same tree depth are assigned with the same two moduli under the different parents (see Figure 1). This means there are only $2\log_2 n$ different moduli in the tree graph, i.e. m_{ij} ($1 \leq i \leq \log_2 n$, $j=1,2$) where i is the depth of the node in the tree, and the nodes (except the root) on a path from a leaf to the root and its direct children exactly cover all moduli. For instance, in Figure 1, for a path from u_1 to the root, the moduli on the path include m_{11}, m_{21}, and m_{31}, and the moduli on its direct children include m_{12}, m_{22}, and m_{32}. In addition, all different moduli in the tree graph should be pair wise relatively prime (i.e., $\gcd(m_{ij}, m_{st})=1$ for

$i \neq s$ or $j \neq t$), and each modulus should be bigger than the key encryption value, i.e., $m_{ij} > E_{k_{il}}(k_{st})$ where m_{ij} and k_{il} belong to the same node and k_{st} belongs to its parent node.

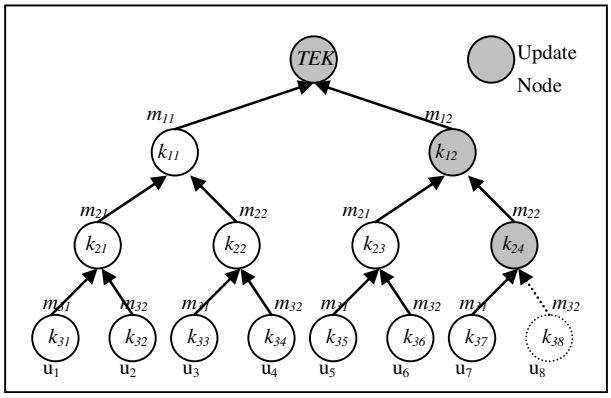

Fig. 1. A tree graph with nodes containing key and modulus.

Key Maintenance: The key server needs to store *2n-1* keys, i.e., *TEK* and k_{ij} ($1 \le i \le \log_2 n$, $1 \le j \le 2^i$) where i is the depth of the node in the tree and j is the ordinal number of the node in the ith depth of the tree, and each member needs to store $\log_2 n + 1$ keys. The key server shares the keys with each member on the path from its leaf to the root. The keys on its path from the leaf to the root need to be updated in the protocol when a member joins or leaves the group but all moduli must be kept fixed.

To update the keys on the tree graph, the key server generates a new key for each update node and encrypts it with its children keys on its path from the leaf to the root. For instance, the key server needs to generate new keys $\{TEK', k'_{il}\}$ to update $\{TEK, k_{il}\}$ for the arrival of member u_d (its leaf key is k_{wd}, $w = \log_2 n$) to the group, where $1 \le i \le \log_2 n - 1$ and $l = \left\lceil \frac{d}{2^{(\log_2 n)-i}} \right\rceil$ which is the upper-limit integer of $\frac{d}{2^{(\log_2 n)-i}}$, and encrypts the updated keys using the following formula:

$$K_{st} = \begin{cases} E_{k_{st}}(k'_{il}) \text{ if } i = (\log_2 n) - 1, \text{where } s = (\log_2 n) \text{ and } t = 2l-1 \text{ or } 2l \\ E_{k_{st}}(k'_{il}) \text{ if } 1 \le i < (\log_2 n) - 1 \text{ and } t \ne e, \\ \quad \text{where } s = i+1, \text{and } t = 2l \text{ if } e = 2l-1, \text{otherwise } t = 2l-1 \\ E_{k'_{st}}(k'_{il}) \text{ if } 1 \le i < (\log_2 n) - 1 \text{ and } t = e, \text{where } s = i+1 \\ E_{k_{st}}(TEK') \text{ if } t \ne v, \text{where } s = 1, \text{and } t = 2 \text{ if } v = 1, \text{otherwise } t = 1 \\ E_{k'_{st}}(TEK') \text{ if } t = v, \text{where } s = 1 \end{cases}$$

where $e = \left\lceil \frac{d}{2^{(\log_2 n)-(i+1)}} \right\rceil$ and $v = \left\lceil \frac{d}{2^{(\log_2 n)-1}} \right\rceil$.

The key server then calculates a lock L as follows and multicasts the lock with the indices of keys (i.e., st in the following formula) to all valid members.

$$L = \sum_{s=1}^{\log_2 n} \sum_{t=z}^{z+1} K_{st} M_{sj} y_{sj} \mod M$$

where $j = \begin{cases} 1, \text{if } t \equiv 1 \bmod 2 \\ 2, \text{Otherwise} \end{cases}$

$$z = \begin{cases} \left\lceil \frac{d}{2^{(\log_2 n)-s}} \right\rceil, \text{if } \left\lceil \frac{d}{2^{(\log_2 n)-s}} \right\rceil \text{is an odd integer} \\ \left\lceil \frac{d}{2^{(\log_2 n)-s}} \right\rceil - 1, \text{Otherwise} \end{cases},$$

$M = \prod_{s=1}^{\log_2 n} \prod_{j=1}^{2} m_{sj}$, $M_{sj} = M/m_{sj}$, and $y_{sj} = M_{sj}^{-1} \mod m_{sj}$.

Each member decrypts the updated traffic encryption key and related key encryption keys based on their own moduli and keys.

For the departure of member u_d from the group, the process is as same as the above except calculating K_{wd} (i.e., $K_{wd} = 0$).

As an illustration, we give the following example for the re-key process in Figure 1, where the member u_8 requests to join the group. The key server generates new keys $\{TEK', k'_{12}, k'_{24}\}$ to update $\{TEK, k_{12}, k_{24}\}$ and does the following encryption.

$$K_{38} = E_{k_{38}}(k'_{24}),\ K_{37} = E_{k_{37}}(k'_{24}),\ K_{24} = E_{k'_{24}}(k'_{12}),$$

$$K_{23} = E_{k_{23}}(k'_{12}),\ K_{12} = E_{k'_{12}}(TEK'),\ K_{11} = E_{k_{11}}(TEK')$$

The key server then calculates a lock as

$$L = K_{38}M_{32}y_{32} + K_{37}M_{31}y_{31} + K_{24}M_{22}y_{22} + K_{23}M_{21}y_{21} + K_{12}M_{12}y_{12} + K_{11}M_{11}y_{11} \mod M,$$

where $M = m_{11} \cdot m_{12} \cdot m_{21} \cdot m_{22} \cdot m_{31} \cdot m_{31}$, $M_{ij} = M/m_{ij}$, $y_{ij} = M_{ij}^{-1} \mod m_{ij}$.

In the protocol, we can see that the key server uses the same modulus (M) and parameters (M_{ij}, y_{ij}) to calculate the lock for any re-key process but the key encryption value (i.e., K_{st}) for calculating the lock are changed based on the re-key requested by the different members. This means the key server can pre-calculate the modulus (M) and parameters (M_{ij}, y_{ij}) to be used for later re-key processing steps and only needs to calculate them once for a fixed tree graph.

III. Scalability of Group Key Management Protocols

In order to measure the scalability of group key management protocols more accurately, we propose the following scalability metrics: "computational complexity", "bandwidth usage", "storage", "number of re-key messages", and "level of processing difficulty". Computational complexity measures the processing time in the central key server. Bandwidth usage accounts for the size of total messages sent out by the key server for a re-key process. Storage measures the total size of keys maintained by the key server. The number of re-key messages is the number of such messages needed to be processed by the key server. The level of processing difficulty indicates applicability for small mobile devices. Table I gives a comparison of the new protocol with the GKMP, Secure Lock, and LKH protocols without signature protection. Table II gives a comparison with signature protection, where storage, number of re-key messages and level of processing difficulty are the same as in Table I. The signature technique for GKMP and LKH is based upon a single digital signature scheme proposed by Merkle [6], which has been the most efficient method so far for signing a set of messages destined to different receivers.

TABLE I
A COMPARISON OF GROUP KEY MANAGEMENT PROTOCOLS WITHOUT SIGNATURE

Scalability Metrics / Protocols	Computational Complexity		Bandwidth Usage		Storage	Number of Re-key Messages		Level of Processing Difficulty
	Join	Leave	J	L		J	L	
GKMP	4E	$2n$E	$4N$	$2nN$	$(n+2)N$	4	$2n$	Low
Secure Lock	nE+$(3n-1)$M +$(n-1)$A +nMR+1MD		nN		$(2n+1)N$	1		High
LKH	$2log_2n$E		$2Nlog_2n$		$(2n-1)N$	$2log_2n$		Low
SGKMP	$2log_2n$E+ $4log_2n$M +$(2log_2n-1)$A +1MD		$2Nlog_2n$		$(2n +2log_2n -1)N$	1		Low

TABLE II
A COMPARISON OF GROUP KEY MANAGEMENT PROTOCOLS WITH SIGNATURE

Scalability Metrics / Protocols	Computational Complexity		Bandwidth Usage	
	Join	Leave	Join	Leave
GKMP	4E+7H+1S	$2n$E+$(4n-1)$H +1S	$6N+1L$	$(2n+log_2 2n)N$ +$1L$
Secure Lock	nE+$(3n-1)$M+$(n-1)$A +nMR+1MD+1H+1S		$(n+1)N+1L$	
LKH	$2log_2n$E+$(4log_2n-1)$H+1S		$(2log_2n+2log_4n)N+1L$	
SGKMP	$2log_2n$E+$4log_2n$M+ $(2log_2n-1)$A+1MD+1H+1S		$(2log_2n+1)N+1L$	

Please note the following for Tables I and II:

- N is the length of the encrypted secret key (default is 128 bits for a symmetric cryptograph) or the length of a hash value (default is 128 bits)
- L is the length of the signature
- n is the number of members
- H is a hash operation
- E is a symmetric key encryption operation
- S is a signature operation
- A is an BigInteger addition operation
- M is a BigInteger multiplication operation
- MD is a BigInteger modulus operation
- MR is a BigInteger modulus reverse operation

From Tables I and II, we see that the LKH and SGKMP reduce the encryption operation and bandwidth usage from $O(n)$ to $O(log\ n)$ when compared to the Secure Lock and GKMP protocols, and the length of the lock from $O(n)$ to $O(log\ n)$ when compared to the Secure Lock. In addition, SGKMP has better performance when compared to the LKH protocols. The detailed processing time performance according to computational complexity is tested in the next section for both with and without the signature operation.

IV. PERFORMANCE OF THE NEW PROTOCOL

The performance of the SGKMP and LKH protocol were tested in order to compare their scalability. The testing was done on a PC (Intel Pentium 4 CPU 3.00GHz and 1 GB RAM). The software used for the testing was Java JDK 1.6. The main classes included Java BigInteger, security, and crypto. The encryption algorithm was AES with a 128 bit encryption key, and the signature algorithm was 512 bit RSA. The testing determined the processing time performance according to group size (see Figure 2 and 3) both with and without signature protection.

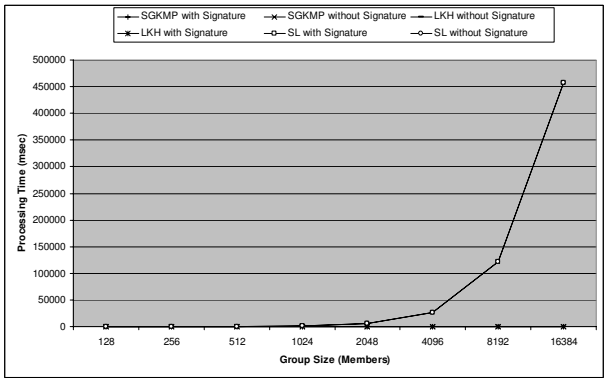

Fig. 2. Performance of the SGKMP, LKH, and SL protocols.

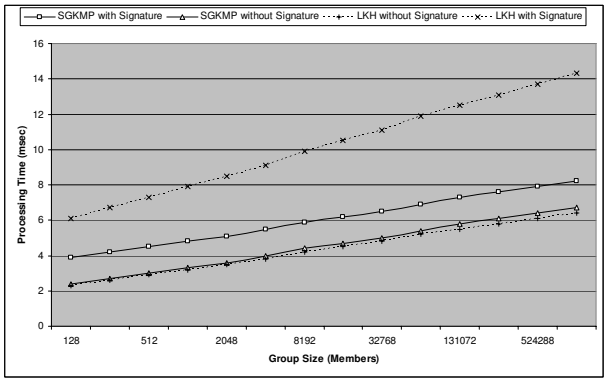

Fig. 3. Performance of the SGKMP and LKH protocols.

From the testing results, we can see that the Secure Lock has very poor scalability when compared to SGKMP and LKH (Figure 2), and the SGKMP has very good scalability for both with and without signature protection when compared to LKH (Figure 3). The processing time of SGKMP with signature for a re-key request corresponding to a group size of up to a million members is less than 10 milliseconds.

V. CONCLUSION

To improve the scalability of group key management, we propose a scalable group key management protocol and demonstrate that it has better scalability in terms of computational complexity (from testing) and bandwidth usage (from calculations in Tables I and II).

REFERENCES

[1] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. IEEE/ACM Transaction on Networking, Vol.8, No.1, pp.16-30. February 2000.

[2] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architecture. National Security Agency, RFC 2627, June 1999.

[3] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Architecture. RFC 2093, July 1997.

[4] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Specification. RFC 2094, July 1997.

[5] G. H. Chiou and W. T. Chen. Secure Broadcast Using Secure Lock. IEEE Transaction on Software Engineering, Vol.15, No.8, pp.929-934, August 1989.

[6] R. C. Merkle. A Certified Digital Signature. Proceedings of Advances in Cryptology – CRYPTO'89, pp.241-250, 1989.