Hierarchical Scalable Group Key Management Based on Chinese Remainder Theorem

Xinliang Zheng, Manton M. Matthews, Chin-Tser Huang

Dept. of Computer Science and Engineering
University of South Carolina

Abstract

In this paper, we present a new group key management protocol based on the Chinese Remainder Theorem (CRT), which is developed based on our previous paper [29]. By shifting more computing load onto the key server we optimized the number of re-key broadcast message, user-side key computation and number of key storages. Our protocol, Hierarchical Chinese Remaindering Group Key (HCRGK) Protocol, only requires the Key Server and each Group Controller to do $O(q)$ ($q$ is the average size of subgroups) XOR, addition and multiplication operation most of the times, and broadcast 1 re-key message, and requires each user to do only 1 XOR and 1 modulo arithmetic for each group key update and only need to store 4 keys all the time.

Keywords: Chinese Remainder Theorem (CRT), Congruence System, Group Key Management, Hierarchical Chinese Remaindering Group Key (HCRGK) Protocol, Key Server (KS), Group Controller (GC)

1. Introduction

As the popularity of group-oriented applications increases the need for security services to provide group communication confidentiality emerges. While there are many mature secure protocols for peer-to-peer communication, the scenario for group communication involving dynamically changing members is very different. To efficiently agree on a new group key after user addition or eviction is crucial to group communication confidentiality. During the past decade a variety of group key management protocols have been proposed. Among them a set of efficient and scalable group key management protocols [6], [19], [22], [27], [28] 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 a new group key protocol based on the Chinese Remainder Theorem, which is developed based on our previous paper [29]. By shifting more computing load onto the KS, we optimized the number of re-key broadcast messages, user-side key computations and number of key storages. Our protocol requires the KS and each GC to broadcast 1 re-key message and each group user to compute only 1
modulo and 1 XOR arithmetic for each key update and store only 4 keys all the time. While our protocols require more computation power from the KS and GCs, it only needs to maintain a relatively simple hierarchical structure. With the tremendous advantage on user key computation, user key storage, re-key broadcasting message number, we consider our protocol is well worth exploring.

While this paper deals with group key management for dynamic group, our protocol is based on the assumption that certain authentication protocol involving two parties is needed before the KS grants group access to each group user. Since two-party authentication protocols are well studied [23], 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 works. Section 3 describes our protocol 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 have attracted a lot of researches during the past decade. Many excellent scholar papers have been proposed to tackle different aspects of the problem.

Ballardie [2] proposed a multicast key distribution protocol. This protocol requires certain support mechanism to be integrated into version 3 of IGMP and it provides no forward secrecy. Mittra [17] proposed a 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. [25], Kim et al. [12] proposed distributed group key protocols based on Group Diffie-Hellman methods for small dynamic peer groups. Rodeh et al. [21] 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 [14], [15] 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 is added or evicted if backward and forward secrecy are required. A set of hierarchical structure based scalable group key protocols [6], [19], [22], [27], [28] have been proposed. In general those protocols requires the KS 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 user 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 counter 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(k2 log2k 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. Zou et al. [30] proposed a protocol for hierarchical access control for secure group communication, however, again, its application of CRT is different from our approach. Their protocol is mainly targeting inter-group communication. Most importantly both Chiou and Zou’s papers use the CRT with their congruence modulo bases publicized hence requires more encryptions and decryptions, while our protocol is not.

3. Hierarchical 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 overview of CRT; then we will present our Hierarchical Chinese Remaindering Group Key (HCRGK) protocol in detail by describing the protocol behavior in subsections.

3.1 Chinese Remainder Theorem

Suppose m₁, …, mₙ are pairwise relatively prime positive integers, and suppose r₁, …, rₙ are arbitrary integers. Then the system of n congruences

\[ X \equiv r₁ \pmod{m₁} \]
\[ X \equiv r₂ \pmod{m₂} \]
\[ \vdots \]
\[ X \equiv rₙ \pmod{mₙ} \]

has a unique solution modulo \( M = m₁ \cdots mₙ \), which is given by

\[ X = \sum_{i=1}^{n} rᵢ Mᵢ yᵢ \pmod{M} \]

where

\[ Mᵢ = M / mᵢ \]
\[ yᵢ = Mᵢ⁻¹ \pmod{mᵢ} \], for \( 1 \leq i \leq n \).

Since \( Mᵢ \) is relatively prime to \( mᵢ \) there must exist a unique multiplicative inverse \( yᵢ \) mod \( mᵢ \). 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 [12], [24], [26] which is out of the scope of this paper.
3.2 Group Initialization

Before a KS starts a new group it needs to do some preparation work. Firstly, the KS will collect a large set of pairwise relatively prime positive integers. We consider that constructing such a set of pairwise relatively prime positive integers are practically easy and we assume the KS can always provide more if needed. Secondly, each of the potential group users will be authenticated as we assumed in our introduction. After authentication every initial legitimate group user will receive two integers from the KS’s pairwise relatively prime positive integer pool, named ks and ksb respectively, through some secure channel. Thirdly, a subset of those legitimate group users, who is willing to act as a GC, having more computational power, and expecting to stay a long duration in the group, will be singled out as the set of potential GCs. Finally, a subset of the potential GC set will be picked as the current GCs. The rest of the potential GCs will be reserved for the future. Each current GC will receive a set of integers from the KS’s pairwise relatively prime pool. Particularly a subset of those integers has already been assigned as the ks’s for a subset of the group users and the rest of the integers are reserve for future new joining group users. Now, the KS will initialize the group with a hierarchical structure. Figure 1 illustrates a group structure containing one level of GCs.

Based on this group structure, we can now establish the first group key to be shared by all the group users. For each internal node (KS, GC1, GC2, ..., GCk) will build the following congruence system for its children respectively but not simultaneously.
\[ X \equiv r_1 \pmod{m_1} \]
\[ X \equiv r_2 \pmod{m_2} \]
\[ \vdots \]
\[ X \equiv r_p \pmod{m_p} \]
\[ X \equiv r_{p+1} \pmod{m_{p+1}} \]
\[ \vdots \]
\[ X \equiv r_q \pmod{m_q} \], where

\( p \) is the initial number of children of the internal node,
\( q \) is the maximal number of children that can be handled by the node,
\( m_i = ks_i \), for \( 1 \leq i \leq p \), \( ks_i \) is the \( ks \) for the \( i \)th children of the node,
\( m_i \) for \( p + 1 \leq i \leq q \) are the reserved pairwise relatively prime positive integers,
\( r_i = K \oplus m_i \), for \( 1 \leq i \leq p \), \( K \) is the group key.
\( r_i = r_{i+1} \) for \( p + 1 \leq i \leq q - 1 \) is a randomly picked integer.

The KS randomly generates a new group key \( K \) and computes the \( X \) value based on the constructed congruence system, then it broadcasts the \( X \) value to all its children (GC\(_1\), GC\(_2\), …, GC\(_k\)). Each GC can re-compute the \( K \) value easily.

\[ K = (X \mod ks) \oplus ks \]

After re-computed the group key \( K \) each GC can now compute the \( X \) value for its congruence system and then broadcast the \( X \) value to all its children. When each user received the corresponding \( X \) value it can re-compute the \( K \) value using the same equation but with its own \( ks \).

### 3.3 User Addition

After the initial group has been set up, if a new user needs to be added to the group then it will first go through the similar authentication process as other group users do. After authentication the KS will find a subgroup with more open slots and pick one of the reserved pairwise relatively prime positive integers from this subgroup, and pick another pairwise relatively prime from its own pool. Those two integers will serve as the \( ks \) and \( ksb \) for the new added group user and will be sent to the new user through some secure channel. The corresponding GC will then be notified of the addition of the new user and the \( ks \) value of the new user. The GC will then update the \( p \) value and corresponding variables of the congruence system in charging.

Now, the new group key can be established just like what we did in group initialization. The KS randomly picks a new group key, computes the new \( X \) value, and broadcasts it to
all its children. All the GCs will distribute the new group key to all their children in the same procedure.

Obviously, in the case of user addition there is one other way we can do. That is, the KS encrypts the new group key using the old group key and sends it to all the older group users, and sends the new group key to the new user using secure unicast. However, there is tradeoff here. If we distribute the new group key using the congruence system then each group user will only do one modulo and one XOR operation to get the new group key, while the other way will save computation on the KS and GCs but each end user will do a decryption to get the new key. One modulo and one XOR together is faster than a decryption. On the other hand no matter which way we choose to distribute the new group key for user addition we will add the new user into the hierarchical group structure like what we did, because it will make user eviction very easy to handle.

3.4 User Eviction

In our protocol group re-keying in the user eviction case has much more advantage over other protocols, because most of the times group re-keying after user eviction is very similar to user addition case in our design. After a user was evicted from the group the KS will notify the corresponding GC. The evicted user’s ks and ksb will be marked as revealed values. However ks is still kept in the congruence system managed by the evicted user’s parent node.

\[ X \equiv r_1 \pmod{m_1} \]
\[ X \equiv r_2 \pmod{m_2} \]
\[ \vdots \]
\[ X \equiv r_e \pmod{m_e} \]
\[ \Rightarrow \text{ for the evicted user} \]
\[ \vdots \]
\[ X \equiv r_p \pmod{m_p} \]
\[ X \equiv r_{p+1} \pmod{m_{p+1}} \]
\[ \vdots \]
\[ X \equiv r_q \pmod{m_q} \]

where

\( m_e \) is the ks value for the evicted user.

The reason for keeping an evicted user’s ks in the congruence system is that it saves a lot of computations for the corresponding GC. As long as \( m_1, \ldots, m_q \) are the same then the previous intermediate results for computing the \( X \) value can be reused, especially the GC doesn’t need to compute the multiplicative inverse for each \( M_i \) again. With the congruence system maintained as the same re-key after user eviction will be very similar to re-key after user addition. The KS picks a new group key, computes the \( X \) value, and
broadcasts it. Each GC receives the X value from the KS, extracts the new group key, computes the X value for its congruence system, and distributes the X value. The group users can extract the new group key from the received X value with one modulo and one XOR operation.

### 3.5 Congruence System Refreshing

For each subgroup in our protocol, after a number of user additions and user evictions, the reserved pairwise relatively prime positive integers for the congruence system will gradually be used up. At such time the GC can contact the KS to get a new set of pairwise relatively prime positive integers from the Server’s pool to refill its reserve.

\[
X \equiv r_1 \pmod{m_1} \\
\vdots \\
X \equiv r_i \pmod{m_i} \\
X \equiv r_{e_1} \pmod{m_{e_1}} \quad \Leftarrow \text{replace } m_{e_1} \text{ with new integer from server’s pool} \\
X \equiv r_{e_2} \pmod{m_{e_2}} \quad \Leftarrow \text{replace } m_{e_2} \text{ with new integer from server’s pool} \\
\vdots \\
X \equiv r_{e_v} \pmod{m_{e_v}} \quad \Leftarrow \text{replace } m_{e_v} \text{ with new integer from server’s pool}
\]

where

\[m_{e_i} \quad \text{for } 1 \leq i \leq v \text{ are the ks values for evicted users.}\]

During congruence system refreshing a subset of m values are changed, therefore the GC in charge need to re-compute all the multiplicative inverse values and other intermediate results. However congruence system refreshing should not happen frequently. If it happens quite frequently then the KS should increase the reserve for the subgroups. If most of the GCs have reached their computation limit then more subgroups should be added to the hierarchical structure.

### 3.6 Group Controller Addition

The first step of adding a new GC is similar to adding a new user to the subgroup. A new added user can take the role of the new GC or a current user picked from the GC reserve can to that. If it is a current user then it needs to be evicted from its current location and the KS will assign a new ks value. Then this user will be added into a new subgroup (it is the KS’s direct subgroup if there is only one level of GCs.). In such case no group re-key is needed. After this new GC establishes the current position, the KS will assign a set of new m values for this GC to build its own congruence system. Now, new group users can be added to the new subgroup.
3.7 Group Controller Eviction

Group controller eviction is a relatively more complicated operation in our protocol, but it should happen infrequently since the GCs are picked to be the long last group users. However, when it does happen, our protocol has been built in such a way it will minimize its impact. A new GC will be picked and the ksb values of the users in the GC evicted subgroup will be passed to the new GC at once taking the role as the ks values. The new GC will set up the congruence system just like in the case of GC addition. Group re-key will be carried out afterwards. After the new subgroup run smoothly with the new GC, the KS can now communicate with the users in the subgroup to replenish the used ksb values. This way the impact of GC eviction is minimized.

3.8 Hierarchical Expansion

Hierarchical expansion will happen when the group size is expanded to such a scale that the KS can no longer manage all the GCs as one subgroup efficiently. In such a case a new level of GCs will be added to the hierarchical structure.

Hierarchical expansion will be very similar to the GC eviction case. The difference is that in this case it is like the KS was evicted. However, since the KS is more likely have more computational power, the original set of GCs may not be handled by one new GC. It will need more new GCs and the original set of GCs managed by the KS will be split between the new added GCs. Hierarchical expansion should be extremely rare.
4. Security Analysis

The security of our protocol is based on the assumption that each current group user will keep its $k_s$ and $k_{sb}$ values secret, each GC will keep the $k_s$ values of its children secret. Moreover, the set of $m$ 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 except that the other numbers do not contain the same factors.

4.1 Forward Secrecy

Forward Secrecy is about preventing evicted users to continue accessing future group communications. In our protocol, since each user can gain little information about other users’ $k_s$ values, and after each GC eviction the $k_s$ values of its children are discarded, no evicted users or GC can obtain the $k_s$ values of the group users. Without this information, the modulo residue values or the new group key can not be computed given an $X$ value. For our protocol, even though each evicted user’s $k_s$ is still kept inside the congruence system for speeding up computation, it can only compute the randomly picked $r_e$ and which is not related to the new group key in any way at all.

4.2 Backward Secrecy

Backward Secrecy is about preventing new users from accessing previous group communications. In our protocol each new group key is an arbitrary picked value with no
relation to any old group key. Since previous user’s ks value is needed to compute an old group key given an old X value is available, without that each new added 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 new group key. For our protocol, if the pool of positive integers is only pairwise relatively primes then the collusion of previous users may gain some information about the ks values of current users. That is, those numbers do not contain the factors of the ks and ksb values of those colluding attackers. However, if the pool of positive integers is made up of primes then collusion gains no additional information.

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 KS and GCs. On the other hand comparing to other hierarchical structure based group key management protocol, our protocol’s hierarchical structure is simpler, which keeps a minimized structure management load on the KS. Most of other hierarchical structure based group key management protocol need to maintain a very low-numbered-ary tree, which means the height of the tree will increase tremendously when the group size getting larger, while, our protocol using a very high-numbered-ary tree, which means even with very large size group the height of the tree is very small. For instance if each GC is capable of handling 100 users and the key server can handle 100 users directly beside its chores then a tree with two layers of GCs can manage up to 1000,000 group users.

Except the very infrequent GC eviction and hierarchical expansion operations, most of the group re-key operations in the HCRGK protocol require the KS and each GC to do O(q) XOR, addition, and multiplication computation and broadcast 1 re-key message, while each user only need to do 1 modulo and 1 XOR operation, and store 4 keys all the time, one for secure symmetric communication between the KS and the user, one group key, and two relatively prime integers, where q is the size of the congruence system manage by the KS and each GC.

6. Conclusion

In this paper, we present a hierarchical structured CRT based group key management protocol which is very scalable for large size dynamically changing group. The scalability of our protocol together with relatively simple hierarchical structure, minimized re-key message numbers, minimized requirements on regular group user computation and key storage space make them suitable for a variety of secure large 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 one way function) 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.

References

Y. Amir, Y. Kim, C. C. Nita-Rotaru, and G. Tsudik, "On the Performance of Group Key Agreement
C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-Secure Key
C. Blundo and A. Cresti, "Space Requirements for Broadcast Encryption", Proceedings of Advances in
M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System", Proceedings
and Some Efficient Constructions", Proceedings of the 18th Annual Joint Conference of the IEEE
R. Canetti, T. Malkin, and K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast
1993.
P. Garrett, "The Mathematics of Coding Theory: Information, Compression, Error Correction, and Finite
W. Geiselmann and R. Steinwandt, "Attacks on a Secure Group Communication Scheme with Hierarchical
1997.
1997.
Y. Kim, A. Perrig, and G. Tsudik, "Tree-Based Group Key Agreement", ACM Transactions on Information
A. Perrig, D. Song, and J. D. Tygar, "ELK, a New Protocol for Efficient Large-Group Key Distribution",
S. Rafaei and D. Hutchison, "A Survey of Key Management for Secure Group Communication", ACM
O. Rodeh, K. P. Birman, and D. Dolev, "Using AVL Trees for Fault Tolerant Group Key Management",