Computers and Mathematics with Applications 65 (2013) 1360–1368

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Centralized key distribution protocol using the greatest common divisor method

P. Vijayakumar [a,*], S. Bose [a], A. Kannan [b]

[a] Department of Computer Science and Engineering, Anna University, Chennai-600 025, Tamilnadu, India

[b] Department of Information Science and Technology, Anna University, Chennai-600 025, Tamilnadu, India

ARTICLE INFO

Keywords:
Multicast communication
Key distribution
GCD
Computation complexity
Karatsuba multiplication
Extended Euclidian algorithm

ABSTRACT

Designing a key distribution protocol with minimal computation and storage complexity is a challenging issue in secure multimedia multicast. In most of the multimedia multicast applications, the group membership requires secured dynamic key generation and updation operations that usually consume much of the computation time. In this paper, we propose a new GCD (Greatest Common Divisor) based Key Distribution Protocol which focuses on two dimensions. The first dimension deals with the reduction of computation complexity which is achieved in our protocol by performing fewer multiplication operations during the key updation process. To optimize the number of multiplication operations, the existing Karatsuba divide and conquer approach for multiplication is used in this proposed work. The second dimension aims at reducing the amount of information stored in the Group Center and group members while performing the update operation in the key content. The proposed algorithm which focuses on these two dimensions has been implemented and tested using a Cluster tree based key management scheme and has been found to produce promising results. Comparative analysis to illustrate the performance of various key distribution protocols is shown in this paper and it has been observed that this proposed algorithm reduces the computation and storage complexity significantly.

1. Introduction

Multimedia services, such as pay-per-view, videoconferences, some sporting events, audio and video broadcasting are based upon multicast communication where multimedia messages are sent to a group of members. In such a scenario, groups can be either opened or closed with regard to senders. In a closed group, only registered members can send messages to this closed group. In contrast, data from any sender is forwarded to the group members in open groups. Groups can be classified into static and dynamic groups. In static groups, membership of the group is predetermined and does not change during the communication. In dynamic groups, membership can change during multicast communication. Therefore, in a dynamic group communication, members either join or depart from the service at any time. When a new member joins into the service, it is the responsibility of the Group Center (GC) to disallow new members from having access to previous data. This provides backward secrecy in a secure multimedia communication. Similarly, when an existing group member leaves from any group, he/she should not have further access to data. This achieves forward secrecy. In order to provide forward and backward secrecy the keys are frequently updated whenever a member joins/leaves the multicast service. Furthermore, if a device lacks storage capabilities, it may be impossible within the receiving device to implement a group key management protocol based on a key tree structure. Hence the amount of information to be stored to find the updated key by GC and group

* Correspondence to: Department of CSE, University College of Engineering Tindivanam, Melpakkam, Tamilnadu, 604001, India. Tel.: +91 9940896665.
E-mail addresses: vijibond2000@gmail.com (P. Vijayakumar), sbs@cs.annauniv.edu (S. Bose), kannan@annauniv.edu (A. Kannan).

0898-1221/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2012.01.038

members should also be minimized. GC also takes care of the job of distributing the Secret key and Group key to the group members. In this paper, we propose a Key Distribution algorithm that reduces the computational complexity and at the same time, it decreases the number of keys to be stored by GC and group members. The remainder of this paper is organized as follows: Section 2 provides the features of some of the related works. Section 3 discusses the proposed key distribution protocol and a detailed explanation of the proposed work. Section 4 explains the Cluster tree based key management where the proposed key distribution is employed. Section 5 provides the optimization method based on Karatsuba multiplication approach for key updation process. Section 6 analyzes the comparative performance of our proposed algorithm with the other existing key distribution methods. Section 7 gives concluding remarks and suggests some future directions.

2. Literature survey

There are many works that are present in the literature on key management and key distribution [1–13]. In most of the existing Key Management schemes, different types of group users obtain a new distributed multicast Group key which is used for encrypting and decrypting multimedia data for every session update. Among the various works on key distribution, Maximum Distance Separable (MDS) [14] method focuses on error control coding techniques for distributing re-keying information. In MDS, the key is obtained based on the use of Erasure decoding functions [15] to compute session keys by the GC/group members. Moreover, the Group center generates n message symbols by sending the code words into an Erasure decoding function. Out of the n message symbols, the first message symbol is considered as a session key and the group members are not provided with this particular key alone by the GC. Group members are given the $(n-1)$ message symbols and they compute a code word for each of them. Each of the group members uses this code word and the remaining $(n-1)$ message symbols to compute the session key. The main limitation of this scheme is that it increases both computation and storage complexity. The computational complexity is obtained by formulating $l_r + (n-1)m$ where l_r is the size of r bit random number used in the scheme and m is the number of message symbols to be sent from the group center to group members. If $l_r = m = l$, computation complexity is nl. The storage complexity is given by $\lceil \log_2 L \rceil + t$ bits for each member. L is number of levels of the Key tree. Hence Group Center has to store $n\,(\lceil \log_2 L \rceil + t)$ bits.

Secure communication using the extended Euclidean algorithm [16] was proposed for centralized secure multicast environments. The main advantage of this algorithm is that only one message is generated per rekeying operation and only one key is stored in each user's memory. In this algorithm, two values (δ, L) are computed in the intermediate steps of GC. The main limitation of the Euclidean algorithm is that the two computed values must be relatively prime. If this is not the case, then the algorithm fails in which the user cannot recover the secret information sent by GC. Also, the time taken for defining a new multiplicative group is high, whenever a new member joins or leave the multicast operation. This approach is only suitable for a star based key management scheme.

The Data Embedding Scheme proposed in [17] is used to transmit a rekeying message by embedding the rekeying information in multimedia data. In this scheme, the computation complexity is $O(\log n)$. The storage complexity also increases to the value of $O(n)$ for the server machine and $O(\log n)$ for group members. This technique is used to update and maintain keys in a secure multimedia multicast via a media dependent channel. One of the limitations of this scheme is that a new key called an embedding key has to be provided to the group members in addition to the original keys, which causes a lot of overheads. A level homogeneous key tree [18] based key management scheme was proposed in [19] to reduce computation and storage complexity. A Key management scheme using key graphs has been proposed by Wong Gouda [20] which consists of the creation of secure group and basic key management graphs scheme using a Star and Tree based method. The limitation of this approach is that scalability is not achieved. A new group keying method that uses one-way functions [21] to compute a tree of keys, called the One-way Function Tree (OFT) algorithm has been proposed by David and Alan. In this method, the keys are computed up the tree, from the leaves to the root. This approach reduces re-keying broadcasts to only about $\log n$ keys. The major limitation of this approach is that it consumes more space. However, the time complexity is more important than space complexity. The storage complexity of GC is $2nK$ and group member is LK, where K is the key size in bits. In our work, we focused on reduction of computation time complexity.

Wade Trappe and Jie Song proposed a Parametric One Way Function (POWF) [22] based binary tree Key Management. Each node in the tree is assigned a Key Encrypting Key (KEK) and each user is assigned to a leaf and given the IKs of the nodes from the leaf to the root node in addition to the session key. These keys must be updated and distributed using top down or bottom up approach. The storage complexity is given by $(\log_\tau n)+2$ keys for a group center. The amount of storage needed by the individual user is given as $\frac{(\tau^{L+1}-1)}{\tau-1}$ keys. Computation time is represented in terms of amount of multiplication required. The amount of multiplication needed to update the KEKs using the bottom up approach *is* $\tau \log_\tau n - 1$. Multiplication needed to update the KEKs using the top down approach *is* $\frac{(\tau-1)\log_\tau n(\log_\tau n+1)}{2}$. This complexity can be reduced substantially if the numbers of multiplications are reduced. Therefore, in this paper we propose a new cluster tree based Key Management Scheme that reduces computation time by reducing the number of multiplications required in the existing approaches. We also use the Karatsuba fast multiplication algorithm to optimize the multiplication operations used in the key distribution protocol in the GC. The proposed method also reduces the amount of information that needs to be stored for updating the keys when there is a change in the group membership. Our proposed algorithm is suitable for single join/leave operation (Single Rekeying operation). When the number of joining or leaving operations is more, batch join and leaving operations can be integrated for a group of users in our proposed key distribution protocol. To perform batch joining and leaving operation

marking, merging and Batch balanced algorithms have been proposed in the past [23–26] and the merging algorithm takes fewer re-keying operations than marking algorithms. The batch balanced algorithm requires less rekeying and updating cost than merging algorithms.

3. GCD based key distribution protocol

The proposed framework works in three phases. The first phase is the Group Center Initialization, where a multiplicative group is created at GC. In the second phase called the Member Initial Join phase, the members send join requests to the GC and obtain all the necessary keys for participation through secure channel. The final phase of this protocol is known as "Member Leave" that deals with all the operations to be performed after a member leaves the group (providing forward secrecy). The proposed work mainly concentrates on the third phase of "Member Leave" phase because the computation time is extremely large in most of the existing systems for providing forward secrecy. This is an extremely great challenge in most of the multimedia multicast applications.

3.1. GC initialization

Initially, the GC selects a large prime number p and q, where $p > q$ and $q \leq \lceil p/4 \rceil$. The value p helps in defining a multiplicative group z_p^* and q is used to fix a threshold value μ, where $\mu = a + q$. The value a is a random element from the group z_p^* and hence when the 'a' value increases, the value of μ also increases.

3.2. Member initial join

Whenever a new user 'i' is authorized to join the multicast group for the first time, the GC sends (using a secure unicast) a secret key K_i which is known only to the user U_i and GC. K_i is a random element in z_p^* and the necessary condition is that all K_i are greater than μ. If this condition is not satisfied the value of 'a' must be adjusted so that it is possible to select $K_i > \mu$. Using this K_i, the Sub Group Keys (SGK) or auxiliary Keys, γ and a Group key k_g are given for that user u_i which will be kept in the user u_i database. The following steps describe the key updation process used for a member join operation at the Group Center.

(1) Initially, GC selects a random element β from z_p^*.
(2) GC now computes the shared secret key $\gamma = \beta^a \mod p$.
(3) The GC calculates $\partial_g = \prod_{i=1}^{n} (k_i)$.
(4) The GC computes a GCD value of (μ, ∂_g) by using the extended Euclidian algorithm described in [27] from which it finds x, y, d such that

$$x \times \mu + y \times \partial_g = d.$$

(5) The GC multicasts β, x, p, q and d to the group members. Upon receiving all the above information (β, x, p, q, d) from the GC, an authorized user u_i of the current group executes the following steps to obtain the new group key γ.

(1) Computes $x1$ using the relation $x \mod k_i = x1$.
(2) Computes μ using $x1^{-1} \mod k_i = \mu$.
(3) Performs the following operation to find the shared secret key.

$$\frac{\beta^{d \times \mu}}{\beta^q} \mod p = \beta^{(d \times \mu) - q} \mod p = \gamma.$$

The γ obtained in this way must be equal to the γ computed in Step 2 of GC.

Security.

Computing the newly updated γ in the proposed scheme depends on the method used to calculate the member's secret key K_i in a particular amount of time. In this scheme, the group center distributes the elements β, x, p, q and d to the group members through multicast communication. Hence an attacker will try to capture all the distributed elements and by using these elements, the attacker can try to find the value of μ. This μ can be computed only by using the user's secret key K_i.

If the attacker is not an active adversary, (i.e., not a previous member of the multicast group) the attacker can use brute force attack to learn about any one member's secret key K_i. If the size of K_i is w bits, then the attacker has to use the total number of trials of 2^w. The time taken to derive K_i can be increased by choosing the large K_i for each user's secret key. In this work, the size of K_i must be 64 bits even though the experiments were conducted with 32 bits and 64 bits. If the time required to perform one attempt using brute force attack is 1μs, then the total time required will be 2^{63} μs = 292 471 years. Therefore when a large size K_i is used it is not possible to find the value of μ and hence γ can't be computed by an adversary.

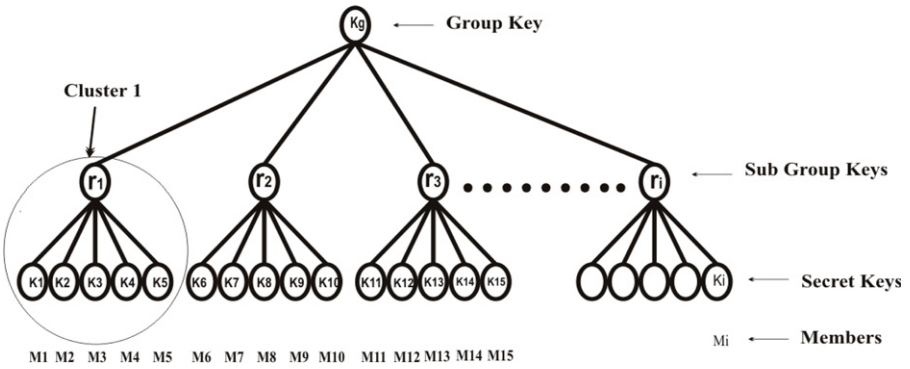

Fig. 1. Clustered tree based key management with cluster size $M = 5$ and number of users $= N$.

3.3. Member leave

Whenever some new members wish to join or some old members wish to leave the multicast group, the GC needs to distribute a new Group key to all the current members in a secure way with minimal computation time. When a new member joins the service, it is easy to communicate the new group key with the help of the old group key. Since the old group key is not known to the new user, the newly joining user cannot view the past communications. This provides backward secrecy. Member leave operation is completely different from member join operation. In member leave operation, when a member leaves the group, the GC must avoid the use of an old Group key/SGK to encrypt a new Group key/SGK. Since the old member knows the old GK/SGK, it is necessary to use each user's secret key to perform a re-keying operation when a member departs from the services. In the existing key management approaches [1,2,6,28] this process increases GC's computation time, because the number of multiplication operations to be done in the key updation is more. In our proposed key distribution scheme, the computation times are equalized for member join and leave operations. Therefore, our work aims at reducing the computation time by decreasing the number of multiplication operations to be carried out. Moreover, this work reduces the number of keys to be stored by the group members to recover the new GK.

4. Clustered tree based key management scheme

Scalability can be achieved in this proposed key distribution approach by applying this scheme in a clustered tree based key management scheme to update the GK and SGK. Fig. 1 shows a cluster tree in which the root is the group key, leaf nodes are individual keys, and the intermediate level is Sub Group Key (SGK). The tree shown in Fig. 1 consists of only three levels. The lowest level (0th level) is the group key. The next higher level (1st level) contains the shared secret keys, γ_i where $i = 1, 2, \ldots, n$. The last level (2nd level) is the user's level, where M number of users are grouped into k clusters, C_k. Each cluster is attached to the upper level (1st level) node and in turn with the group key node. When the number of joining users exceeds the cluster size, a new node is created from the root to form the second cluster. The number of clusters formed is based on the cluster size M which is fixed by GC and the number of joining users. If the cluster tree based key management consists of N number of users $M_1, M_2, \ldots, M_N$ and each cluster size is of size M then there will be $\lceil N/M \rceil$ clusters. In this cluster tree based key management scheme, updating is necessary for each rekeying operation used for member leave and member join operations. For example, if a member M_{10} in cluster 2 from the Fig. 1 leaves the group, then the keys on the path from his leaf node to the tree's root node must be changed.

Hence, only the keys γ_2 and k_g will become invalid. Therefore, these two keys must be updated.

In order to update these two keys, two approaches are used in the member's departure (Leave) operation. In the first approach, updation of the sub group key, γ_2 for the cluster 2 is performed as given in Algorithm 1. When a member M_{10} leaves from the service, GC computes $\partial_g\left(k_{6,9}\right)$ for the existing users using their own secret keys which are kept in GC. When computing $\partial_g\left(k_{6,9}\right)$ for the members M_6, M_7, M_8 and M_9 the GC uses K_6, K_7, K_8 and K_9 which are the secret keys for the remaining members of cluster 2. Since the secret key K_{10} is known to the member M_{10} who had left from the service, GC is not using the secret key K_{10} when it computes the function $\partial_g\left(k_{6,9}\right)$ for the members M_6, M_7, M_8 and M_9. However, the computation time of $\partial_g\left(k_{6,9}\right)$ can be reduced by dividing the γ_2 by k_{10} as shown in Step 1 rather than multiplying all users secret key once again. Next the GC computes μ, GCD value of $\left(\mu, \partial_g\left(k_{6,9}\right)\right)$ and generates a multicast message as indicated in Step 4 and sends the message to all the existing members of the cluster in order to update the new SGK γ_2^1.

Algorithm 1.

(1) $\partial_g\left(k_{6,9}\right) = \frac{\partial_g\left(k_{6,10}\right)}{k_{10}}$.

(2) GC generates the new β, q and computes μ and γ_2^1 values as explained in Sections 3.1 and 3.3.

(3) Now GC computes GCD value of $\left(\mu, \partial_g\left(k_{6,9}\right)\right)$ and finds out x, y, d values.
(4) Finally GC multicasts β, x, p, q, and d to the existing group members.
Group members M_6, M_7, M_8 and M_9 execute the following steps to obtain the new sub group key γ_2^1.
(5) Compute $x \mod k_i = x1$.
(6) Compute $x1^{-1} \mod k_i = \mu$.
(7) Perform the following operation to find the shared secret key.

$$\frac{\beta^{d\times\mu}}{\beta^q} \mod p = \beta^{(d\times\mu)-q} \mod p = \gamma_2^1.$$

After updating the above SGK successfully, GC has to use the second approach in order to update the group key k_g using a different procedure as explained below. The new group key k_g is used to encrypt the multimedia data. For updating the GK, GC generates a new group key from z_p^*, with a condition that the new group key $k_g^1 < \gamma_i$. If this condition is not satisfied then append a value 1 in front [1] of γ_i in order to make γ_i a greater value than k_g^1. Every time a new cluster is created its corresponding SGK is multiplied with all other SGKs and the result is stored in a temporary variable X. Therefore whenever a new cluster is created, only the new γ_i of the newly created cluster is multiplied with the value X which is stored in GC. Hence only one multiplication is needed for updating the GK. Similarly when an existing cluster is completely deleted X is divided by the corresponding γ_i value and hence only one division is necessary for updating the GK. In order to understand the key updation when a single member leaves a group, consider an example using Fig. 1 where only one member M_{10} is allowed to leave the cluster (cluster 2). In this case, γ_2 must be updated and let the updated γ_2 be represented as γ_2^1. In order to update γ_2, the GC must divide X by γ_2 first and then the result must be multiplied with the newly computed γ_2^1 and the final result is stored in the variable X. This X is added with the newly generated group key k_g^1 to obtain γ_g and the rekeying message is formed by using the equation $\gamma_g = k_g^1 + X$. In this way, member leave operations are handled effectively by reducing the number of multiplications/divisions.

The resultant value γ_g is broadcast to the remaining members of the group. The members of the group can recover the updated group key with the help of γ_i using the relation,

$$\gamma_g \mod (\gamma_i) = k_g^1.$$

The key strength of our algorithm is that the scalability increases sufficiently. The number of keys to be used by the GC and group members are reduced in comparison to the other existing approaches [1,2,15,4,22,6]. Each user has to store 3 keys, since the tree described in the proposed algorithm has 3 levels. If the numbers of clusters are k and each cluster consists of n users, then the storage complexity of GC is $(n \times k) + 2k + 1$, where $2k$ is used to denote the total number of $\partial_g(k_{i,j})$ and γ_i used for every cluster that are stored in GC and 1 represents group key storage area.

5. Optimization method

Assumption. Let $\partial_g = \prod_{i=1}^{n}(k_i)$ be a multiplication function which is used for member join operation, where $k_i =$ secret is the key of a user. Now, 'σ_i' is the size of the k_i, where $i = 1, 2, 3, \ldots, n$ ($n =$ size of the group).

Algorithm. Consider a scenario, where $\sigma_1 = 2$ and $\sigma_2 = 2$. A set of 'σ_i' are multiplied to form ∂_g as shown in Step 3 in Section 3.3 using the traditional multiplication operation present in most of the existing key distribution approaches [3,15]. In order to multiply σ_1 and σ_2 using traditional multiplication operation, the algorithm takes 4 multiplication operations. In general, when two 'σ' digit numbers are to be multiplied, it takes (σ^2) multiplication operations in order to obtain the solution.

For optimizing the number of multiplication operations used for computing ∂_g Karatsuba fast multiplication, the divide and conquer approach [29–31,5] is used in our proposed key distribution algorithm. The number of multiplication operations to be performed in total to obtain the solution for the 'σ' digit number will be $(\sigma^{1.585})$. If σ is a long digit number, it divides the number into two halves, and those products can be multiplied by recursive calls of the Karatsuba divide and conquer algorithm. The recursion can be applied until the numbers are so small that they can (or must) be computed directly. We can obtain the whole product used in the function ∂_g by using the "divide and conquer" method recursively as they compute the Batch verification process which is implemented in the Merkle tree [32]. Hence, the multiplication time complexity is given by $O\left(\sigma^{\log_2 3}\right)$. Therefore it is faster than the traditional multiplication, which requires σ^2 single-digit products and the complexity is $O\left(\sigma^{\log_2 4}\right)$. The Karatsuba fast multiplication approach works well when the value of $\sigma > 16$ digits. However, if the number of digits of $\sigma < 16$, this algorithm shall not show a significant difference. In order to optimize the use of the Karatsuba fast multiplication approach, the group size in our proposed key distribution algorithm can have 16-digits, 32-digits, 64-digits, 128-digits, etc. In the proposed algorithm given in Section 3.3, we have analyzed and tested the algorithm for a group size p as 16-digit, 32-digit and 64-digit prime numbers. The key values used in our algorithm are 16 and 32 digit numbers.

Table 1
Computation, storage complexity of various key management approaches.

Parameters	ETF	MDS	Binary	Proposed method
Computation cost (GC)	$\tau + 2(\log_\tau n - 2) + \log_\tau n$	$l_r + (n-1)m$	$\tau \times \log_\tau n - 1$	$3 + G$
Computation cost(user)	$(\log_\tau n - 1)(M + E)$	$(\log_\tau n - 1)(H + ED)$	$(\log_\tau n - 1)(M + Ap)$	$1 + I + 1M$
Storage complexity (user)	$\log_\tau n + 1$	$\lceil \log_2 L \rceil + t$	$(\log_\tau n) + 2$	3
Storage complexity (GC)	$n(\log_\tau n + 1)$	$n(\lceil \log_2 L \rceil + t)$	$\frac{(\tau^{L+1}-1)}{\tau-1}$	$(n \times k) + 2k + 1$
Communication complexity	$(\log_\tau n - 1)[(3 \times p) - 1] + 2p$	$\tau \log_\tau n$	$(\log_\tau n - 1)[(3 \times p) + 1] + (2p + 1)$	2multicast + 1 unicast

Theorem 1. *The number of multiplications in the computation of ∂_g is in the order of $O\left(\sigma^{1.585}\right)$ when Karatsuba divide and conquer multiplication is employed for the key computation process where the key size is a σ digit number.*

Proof. Initially, divide the input number σ into three $\frac{\sigma}{2}$ digit numbers, each can take three multiplications of $\frac{\sigma}{2}$ digits, which is represented as $\frac{\sigma}{2} \times 3$. Break each of the resulting numbers further into $\frac{\sigma}{4}$ digit parts and perform the multiplications with these parts. Do the same steps until we get single digits of the input numbers in the level $\log_2 \sigma$. Simplifying the above statements, the following formulations are done.

$$
\begin{aligned}
&= \sigma + \frac{\sigma}{2} \times 3 + \frac{\sigma}{4} \times 3^2 + \frac{\sigma}{8} \times 3^3 + \cdots + \frac{\sigma}{2^{\log_2 \sigma}} \times 3^{\log_2 \sigma} \\
&= \sigma + \frac{3}{2} \times \sigma + \frac{3^2}{2^2} \times \sigma + \frac{3^3}{2^3} \times \sigma + \cdots + \frac{3^{\log_2 \sigma}}{2^{\log_2 \sigma}} \times \sigma \\
&= \sigma + \frac{3}{2} \times \sigma + \left(\frac{3}{2}\right)^2 \times \sigma + \left(\frac{3}{2}\right)^3 \times \sigma + \cdots + \left(\frac{3}{2}\right)^{\log_2 \sigma} \times \sigma \\
&= \sigma \left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \left(\frac{3}{2}\right)^3 + \cdots + \left(\frac{3}{2}\right)^{\log_2 \sigma}\right).
\end{aligned}
$$

This series is in the form of finite geometric series $1 + \omega + \omega^2 + \cdots + \omega^r$ and this can be written as, $1 + \omega + \omega^2 + \cdots + \omega^r = \frac{\omega^{r+1}-1}{\omega-1}$.

Substitute $\omega = \frac{3}{2}$ to simplify this equation.

$$
\begin{aligned}
&= \sigma \left(\frac{\left(\frac{3}{2}^{\log_2 \sigma + 1} - 1\right)}{\left(\frac{3}{2} - 1\right)}\right) = \sigma \left(\frac{\left(\frac{3}{2}^{\log_2 \sigma + 1} - 1\right)}{\left(\frac{1}{2}\right)}\right) \\
&= 2\sigma \left(\frac{3}{2}^{\log_2 \sigma + 1} - 1\right) \\
&= 2\sigma \left(\left(\frac{3}{2}\right)^{\log_2 \sigma} \left(\frac{3}{2}\right)^1 - 1\right) = 2\sigma \left(\frac{3}{2}\right)^{\log_2 \sigma} \left(\frac{3}{2}\right)^1 - 2\sigma = 3\sigma \left(\frac{3}{2}\right)^{\log_2 \sigma} - 2\sigma = 3\sigma \left(\frac{3^{\log_2 \sigma}}{2^{\log_2 \sigma}}\right) - 2\sigma \\
&= 3\sigma \left(\frac{3^{\log_2 \sigma}}{\sigma}\right) - 2\sigma = 3\sigma^{\log_2 3} - 2\sigma = O(\sigma^{1.585}). \quad \square
\end{aligned}
$$

6. Performance analysis

The proposed method has been implemented in JAVA (Intel Core i3 processor, with 2 GB RAM) for more than 6000 users and we have compared the computation time with the existing approaches to perform the rekeying operation. For implementing this approach a cluster based key tree is created in which cluster size is fixed as 100 users, hence totally 65 clusters(in approx) are created. Table 1 shows the computation (in terms of the number of multiplications required) and storage complexities of various key management approaches [1,15,22]. The notations used for comparisons are defined as: n is the number of members, k is the number clusters available in the clustered tree, τ is the maximum number of children of each node of the tree, p is the size of the prime number used to define the multiplicative group, I is the time taken to find the inverse element of a multiplicative group, G is the time taken to perform a GCD operation, L is the number of levels and t is used to represent the size of the seed key in bits used in MDS.

E, ED, M, Ap and H denote the computation costs of finding Euler's value, erasure decoding, modulo division, appending and hash operation respectively. Erasure decoding operations for an MDS code need $O\left(n^2\right)$ arithmetic operations if standard

Table 2
Group key computation time for various key distribution approaches.

No. of users	Length of prime number (in digits)	Length of new GK (in digits)	ETF (ns)		MDS (ns)		Binary tree (ns)		GCD (ns)	
			Server	Client	Server	Client	Server	Client	Server	Client
2	8	5	211948432	87477763	753 408	520 372	599 231	494 678	347 744	419 111
2	8	6	74401457	62202985	846 185	588 792	760 156	637 209	347 898	587 531
2	16	5	108734791	87105870	876 905	640 457	790 874	569 879	485 644	559 196
2	16	6	87838705053	86853329641	976 493	433 325	878 956	397 543	485 232	532 064
3	8	5	66234234	58699556	894 126	526 888	832 145	578 213	356 298	466 762
3	8	6	66173260	61553685	1 027 245	450 018	965 623	481 432	356 398	449 892
3	16	5	134731825	154361288	1 079 355	897 951	945 647	844 304	696 690	758 903
3	16	6	1317281552	1244607297	1 316 729	987 455	1 162 346	913 456	866 194	890 654
10	8	5	1.12197632E9	2.883584E8	4 201 236	977 812	3 645 136	971 778	1 287 652	932 178
10	16	6	2.46677504E9	3.9834256E8	9 206 421	1 976 342	6 828 321	1 334 098	3 706 212	1 043 527

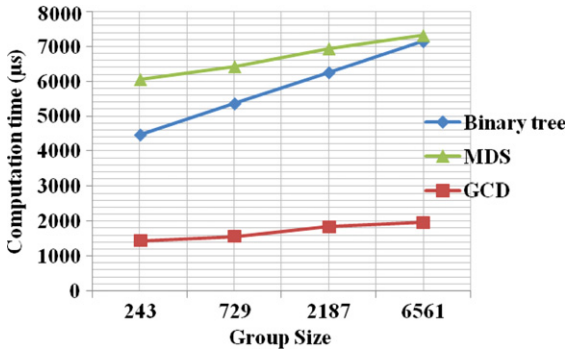

Fig. 2. GC key computation time of various key distribution schemes.

erasure decoding algorithms are used. Even if the fast decoding algorithms are used it needs $O(n \log n)$ operations in the receiver side. Similarly, the time taken to compute the totient value for a 32 digit key value used in ETF is extremely high compared with performing simple subtraction and modulo division used in our proposed approach.

Table 2 compares the computation time which is defined as the time taken to compute the group key at the Group Center area and the Group Members' area for various key management approaches.

For comparing the computation time of the proposed algorithm with three different algorithms present in the literature, the group size is taken to be 8-digit and 16-digit. Table 2 shows the measured computation time in nanoseconds for such comparisons. It is evident from the values that the computation time for our proposed algorithm is found to be better both in the GC area and the client area than the other algorithms. It is to be noted that in our algorithm the key recovery time taken by single user is large when compared to GC's key computation time for M users. The reasons behind this amount of time are (1) calculating mod value of x is high, when the x value is computed for an entire cluster, and (2) finding an inverse element also takes an enormous amount of time.

The graphical result shown in Fig. 2 is used to compare the GC key computation time of the proposed method with the existing methods. It compares the results obtained from our proposed Cluster tree key distribution protocol with the Binary tree based, and erasure encoding which is named MDS. The comparison results of key computation time taken for computing the group key using Euler's totient function [1] based method is not included in Fig. 2.

Because the group key computation time increases rapidly when Euler's totient function is used, since the function takes longer to find the prime factors of 16-digit key value. However, if security is a major concern in implementing the multicast security, the Euler's totient based approach can be used. Most of the applications like PAY-TV, sporting events are fully based on the idea of reducing the computation time where our proposed algorithm will be more suitable.

From Fig. 2 it is observed that when the group size is around 6000, the key computation time is found to be 1968 µs in our proposed approach for updating all the keys from the leaf node to the root node where the key size=16 digits, which is better in comparison with the other existing schemes. Moreover if the number of members who are joining and leaving with in a cluster increases, then the computation time gradually increases. However it is less in comparison with the existing approaches. The result shown in Fig. 3 is used to compare the user's key computation time of our proposed method with the existing methods. It compares the results obtained from the Cluster tree based key distribution scheme with existing approaches and it is observed that when the group size is 6561, the key recovery time of a user is found to be 1781 µs in our proposed approach, which is better in comparison with the other existing schemes.

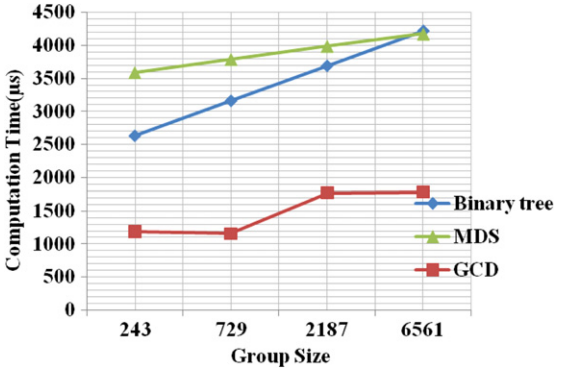

Fig. 3. User key computation for obtaining the new key value.

7. Concluding remarks

This paper proposes a new solution to reduce the computation and storage complexity while providing secure multimedia multicast through effective key management techniques. For this purpose a new Clustered tree-based key distribution protocol (key value = '*n*' bit numbers) has been proposed in this paper. The proposed algorithm has two dimensional focuses—minimal computation complexity and minimal storage complexity. When the key size is small (key size=8 digits), the computation time decreases by 0.2 ms and when the key size increases (16 or 32 digits) the computation time decreases by 0.15–0.18 ms for updating a single key from any level of the clustered tree by using Karatsuba fast multiplication. With regard to the storage complexity, the amounts of keys stored by GC and group members are reduced substantially by employing the cluster tree approach. Further extensions to this work are to devise techniques for reducing the communication complexity which is the number of keys to be sent from GC to the group members' area in order to recover the updated keying information and to reduce rekeying cost for batch join and batch leave operations.

References

[1] P. Vijayakumar, S. Bose, A. Kannan, S. Siva Subramanian, A secure key distribution protocol for multicast communication, in: P. Balasubramaniam (Ed.), Gandhigram-India, 2011, in: CCIS, vol. 140, Springer, Heidelberg, 2011, pp. 249–257.
[2] P. Vijayakumar, S. Bose, A. Kannan, S. Siva Subramanian, An effective key distribution protocol for secure multicast communication, in: IEEE International Conference on Advanced Computing, Chennai, December 14–16, 2010, pp.102–107.
[3] Mingyan Li, R. Poovendran, A. David McGrew, Minimizing center key storage in hybrid one-way function based group key management with communication constraints, Information Processing Letters (2004) 191–198. Elsevier.
[4] R. Poovendran, J.S. Baras, An information-theoretic approach for design and analysis of rooted-tree-based multicast key management schemes, IEEE Transactions on Information Theory 47 (2001) 2824–2834.
[5] T. Jebelean, Using the parallel Karatsuba algorithm for long integer multiplication and division, in: European Conference on Parallel Processing, in: Lecture Notes in Computer Science, 1997, pp. 1169–1172.
[6] Sandeep S. Kulkarni, Bezawada Bruhadeshwar, Key-update distribution in secure group communication, Computer Communications 33 (6) (2010) 689–705. Elsevier.
[7] Bezawada Bruhadeshwar, Kishore Kothapalli, A family of collusion resistant symmetric key protocols for authentication, ICDCN (2008) 387–392.
[8] Bezawada Bruhadeshwar, Kishore Kothapalli, Maddi Sree Deepya, Reducing the cost of session key establishment, ARES (2009) 369–373.
[9] Bezawada Bruhadeshwar, Kishore Kothapalli, M. Poornima, M. Divya, Routing protocol security using symmetric key based techniques, ARES (2009) 193–200.
[10] Shiuh-Jeng Wang, Yuh-Ren Tsai, Chien-Chih Shen, Pin-You Chen, Hierarchical key derivation scheme for group-oriented communication systems, International Journal of Information Technology, Communications and Convergence 1 (1) (2010) 66–76.
[11] Mohsen Imani, Mahdi Taheri, M. Naderi, Security enhanced routing protocol for ad hoc networks, Journal of Convergence 1 (1) (2010) 43–48.
[12] Pinaki Sarkar, Amrita Saha, Security enhanced communication in wireless sensor networks using Reed–Muller codes and partially balanced incomplete block designs, Journal of Convergence 2 (1) (2011) 23–30.
[13] Bin Xie, Anup Kumar, David Zhao, Ranga Reddy, Bing He, On secure communication in integrated heterogeneous wireless networks, International Journal of Information Technology, Communications and Convergence 1 (1) (2010) 4–23.
[14] Mario Blaum, Jehoshua Bruck, Alexander Vardy, MDS Array codes with independent parity symbols, IEEE Transactions on Information Theory 42 (2) (1996) 529–542.
[15] Lihao Xu, Cheng Huang, Computation-efficient multicast key distribution, IEEE Transactions on Parallel and Distributed Systems 19 (5) (2008) 1–10.
[16] J.A.M. Naranjo, J.A. Lopez-Ramos2, L.G. Casado, Applications of the extended Euclidean algorithm to privacy and secure communications, in: Proceedings of the 10th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE, 2010.
[17] Wade Trappe, Jie Song, Radha Poovendran, K.J. Ray Liu, Key distribution for secure multimedia multicasts via data embedding, in: IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 3, 2001, pp. 1449–1452.
[18] J.S. Lee, J.H. Son, Y.H. Park, S.W. Seo, Optimal level-homogeneous tree structure for logical key hierarchy, in: Proc. of IEEE Conference on Communication System Software and Middleware Workshop, COMSWARE, 2008.
[19] Dong-Hyun Je, Jun-Sik Lee, Yongsuk Park, Seung-Woo Seo, Computation-and-storage-efficient key tree management protocol for secure multicast communications, Computer Communications 33 (6) (2010) 136–148. Elsevier.
[20] C. Wong, M. Gouda, S. Lam, Secure group communications using key graphs, IEEE/ACM Transactions on Networking 8 (2002) 16–30.
[21] A. David McGrew, Alan T. Sherman, Key establishment in large dynamic groups using one-way function trees, IEEE Transactions on Software Engineering 29 (5) (2003) 444–458.
[22] Wade Trappe, Jie Song, Radha Poovendran, K.J. Ray Liu, Key management and distribution for secure multimedia multicast, IEEE Transactions on Multimedia 5 (4) (2003) 544–557.

[23] Ng W. Hock Desmond, M. Howarth, Z. Sun, H. Cruickshank, Dynamic balanced key tree management for secure multicast communications, IEEE Transactions on Computers 56 (5) (2007) 590–605.
[24] Ng W. Hock Desmond, Z. Sun, H. Cruickshank, Scalable balanced batch rekeying for secure group communication, Computers and Security (2006) 265–273. Elsevier.
[25] C. Jin-Hee, C. Ing-Ray, E. Mohamed, On optimal batch rekeying for secure group communications in wireless networks, Journal on Wireless Networks (2008) 915–927. Springer.
[26] Bezawada Bruhadeshwar, Sandeep S. Kulkarni, Balancing revocation and storage trade-offs in secure group communication, IEEE Transactions on Dependable and Secure Computing 8 (1) (2011) 58–73.
[27] Wade Trappe, Lawrence C. Washington, Introduction to Cryptography with Coding Theory, second ed., Pearson Education, 2007, pp. 66–70.
[28] P. Vijayakumar, S. Bose, A. Kannan, P.H. Himesh, Key Distribution protocol for secure multicast with reduced communication delay, in: Active Media Technology 2011, in: Lecture Notes in Computer Science, vol. 6890, Springer-Verlag, 2011, pp. 312–323.
[29] Xianjin Fang, Longshu Li, On karatsuba multiplication algorithm, in: Proc. 7th Int. Data, Privacy, and E-Commerce Symp., Washington, DC, USA, 2007, pp. 274–276.
[30] Tom St Denis, BigNum Math Implementing Cryptographic Multiple Precision Arithmetic, SYNGRESS Publishing, 2003.
[31] S. Meftah, R. Razali, A. Samsudin, R. Budiator, Enhancing public-key cryptosystem using parallel Karatsuba algorithm with socket programming, in: Proc. 9th Asia-Pacific Conference on Communication, Penang, Malaysia, September 21–24, 2003, pp. 706–710.
[32] Yun Zhou, Xiaoyan Zhu, Yuguang Fang, MABS: multicast authentication based on batch signature, IEEE Transactions on Mobile Computing 9 (2010) 982–993.