Computation Efficient Multicast Key Distribution

Lihao Xu
Department of Computer Science
Washington University
St. Louis, MO 63130
lihao@cs.wustl.edu

***Abstract*— Efficient key distribution is an important problem for secure group communications. The communication and storage complexity of the multicast key distribution problem has been studied extensively. In this paper, we propose a new multicast key distribution scheme whose *computation* complexity is significantly reduced. Instead of using conventional encryption algorithms, the scheme employs MDS codes, a class of error control codes, to distribute multicast key dynamically. The computation load of key distribution is more evenly distributed among the group key distribution controller (GC) and all the group members. Easily combined with any key-tree based schemes, this scheme provides much lower computation complexity, while maintaining low and balanced communication complexity and storage complexity for secure dynamic multicast key distribution.**

***Index Terms*— key distribution, multicast, MDS codes, erasure decoding**

I. Introduction

In many applications, multicast is an efficient means of distributing data in terms of resource (such as network bandwidth, server computation and I/O load) usage. The privacy of a multicast communication session is usually ensured using (symmetric) encryption. All the designated receivers or members in a multicast group share a session (encryption) key. In many applications, however, the multicast group membership changes dynamically, i.e., some new members are authorized to join a new multicast session while some old members should be excluded. Thus session keys shall change dynamically to ensure both *forward secrecy* and *backward secrecy* of multicast sessions. The forward secrecy is maintained if an old member who has been excluded from the current session can *not* access the communication of the current session, and the backward secrecy is guaranteed if a new member of the current session can *not* recover the communication of past sessions. This requires each session need a new key that is only known to the current session members, i.e., session keys need to be *dynamically* distributed to authorized session members.

In this paper, we adopt a common model where session keys are issued and distributed by a central *group controller* (GC). The resources needed for the GC to distribute session keys to group members include communication, storage and computation resources. The *communication complexity* is usually measured by the number of data bits that need to be transmitted from the GC to group members to convey information of session keys, while the *storage complexity* is measured by the number of data bits that the GC and group members need to store to obtain session keys, and the *computation complexity* is measured by the computation the GC and group member need to perform to derive session keys. Hereafter, the problem of how to effectively use resources to distribute session keys is referred to as the *group key distribution* problem.

The group key distribution problem has been studied extensively, mainly on balancing the storage complexity and the communication complexity. There are two trivial schemes to distribute a session key to a group of n members. The first one is that the GC shares an individual key with each group member, which can be used to encrypt a new group session key. In this scheme, the communication complexity is $O(n)$, while the GC needs to store $O(n)$ key information and each member stores $O(1)$ key information, and $O(n)$ encryption and decryption operations are needed. In the second scheme, the GC shares an individual key with each *subset* of the group, which can then be used to multicast a session key to a designated subset of group members. Now the communication complexity and computation complexity reduce to $O(1)$, at the cost of increasing the storage complexity to $O(2^n)$ for both the GC and each group member. It is easy to see neither of the two schemes works for practical applications with a reasonable group size n. Thus research efforts have been made to achieve low communication and storage complexity for group key distribution.

Static secret sharing via broadcast channel was studied in [19], [14]. However, this *threshold*-based scheme can distribute a session key to a designated group of members for only *one-time* use. Once a session key is distributed

to the group, any member can calculate the secret information other members in the same group hold. Thus the scheme does not provide forward or backward secrecy. A secure lock method based on the Chinese Remainder Theorem was proposed in [9]. But its prohibitively high communication complexity and computation complexity make it only practical for a very small group with limited number of members. Various theoretical measures and schemes for group key distribution were introduced in [11]. Along the same line, many research efforts have been made on balancing communication complexity and storage complexity of the group key distribution problems, e.g., [7], [12], [5], [6], [21], [22].

For a multicast group with a large number of members, *key-tree* based schemes were introduced to decompose a large group into multiple layers of subgroups with smaller sizes [23], [16], [24], [25]. Using these schemes, a group membership change can be effectively handled in the corresponding subgroups without affecting other ones. Thus communication complexity is reduced at the cost of increase in storage and computation complexity together with extra communication delays. For a group of n members, key-tree based schemes have a communication complexity of $O(logn)$, and storage complexity of $O(n)$ for the GC and $O(logn)$ for each group member. It has been shown that if a group member can store at most $O(logn)$ keys, then the lower bound of communication complexity is $O(logn)$, if a *structure-preserving* protocol is used for group key distribution[8]. Thus the key-tree based schemes are of practical interest for a variety applications because of its balance between communication complexity and storage complexity.

While most research on group key distribution has been on balancing communication complexity and storage complexity, little efforts have been made to reduce *computation complexity*. It has been long assumed that expensive encryption and decryption operations are necessary to distribute group keys. However, in the model where a GC is used to distribute group keys, the computation pattern is *asymmetric*, i.e., the GC needs to perform many more computation operations than each individual group member. Thus besides heavy communication load, the GC can become a bottleneck also because of expensive computation operations it has to perform. Hence it is important to explore approaches of reducing computation complexity by 1) lowering the overall computation cost for both the GC and each individual group member, and 2) distributing some computation load away from the GC to each group member.

In this paper, we propose a new *dynamic* group key distribution scheme that reduces computation complexity and yet maintain at least the same security degree of using symmetric encryption algorithms. In our scheme, information related to session keys is encoded using *error control codes* rather than encryptions. In general, encoding and decoding of a proper error control code have much (at least one order) lower computation complexity than existing encryption and decryption algorithms. Thus computation complexity of key distribution can be significantly reduced. The similar idea of using error control codes to achieve privacy was employed in [19], [14], [3]. The major difference between these schemes and ours is that our scheme allows dynamic group membership changes with very low storage complexity, while the other schemes only work for a predefined static group.

The security strength of this scheme will be evaluated, as well as its communication, storage and computation complexity. Besides its low computation complexity, this scheme also has low storage complexity: $O(1)$ for an individual group member, and $O(n)$ for the group controller, where n is the number of group members. Based on the basic scheme using error control codes, concrete design parameters are derived to apply this scheme to key trees. Experimental results are also given for the computation complexity of this scheme on a 3-ary balanced key tree.

This paper is organized as follows: a basic scheme using error control codes to distribute multicast keys is described in Section II. Section II also evaluates the security and resource consumption of the basic scheme. Section III applies the general scheme to a balanced 3-ary key tree to achieve low communication, computation and storage complexity. Implementation and experimental results are given as well. Section IV concludes the paper.

II. The Basic Scheme: Dynamic Key Distribution Using MDS Codes

A. MDS Codes

MDS (*Maximum Distance Separable*) codes are a class of error control codes that meet the Singleton bound [13]. An (n, k) (block) error control code is a mapping from $GF(q)^k$ to $GF(q)^n$: $E(m) = c$, where $m = m_1 m_2 \cdots m_k$ is the original message and $c = c_1 c_2 \cdots c_n$ is its codeword, and $E(\cdot)$ is an encoding function with $k \leq n$. If a decoding function $D(\cdot)$ exists such that $D(c_{i_1} c_{i_2} \cdots c_{i_k}, i_1, i_2, \cdots, i_k) = m$ for $1 \leq i_j \leq n$ and $1 \leq j \leq k$, then this code is called an (n, k) MDS code. For an (n, k) MDS code, the k symbols of the original message can be recovered from *any* k symbols of its codeword. The process of recovering the k symbols is called *erasure decoding*. All the symbols are defined over $GF(q)$, and usually $q = 2^m$. The well-known RS (Reed-Solomon) codes [17] are a class of widely used

MDS codes. Notably RS codes and other MDS codes can be used to construct secret-sharing and threshold schemes [19], [14].

B. Description of the Basic Scheme

For a dynamic multicast group, a session key is issued by a *group controller* (GC). Using this session key, the GC can establish a secure multicast channel with the authorized group members. Every time group memberships change because of join or leave of some group members, the GC reissues a new session key which is independent of all the old session keys. This *rekeying* procedure ensures the security of the current session, as well as that of the old sessions, namely, the newly-joined members cannot recover the communications of the old sessions and those old members who left the group cannot access the current session. Thus both backward secrecy and forward secrecy of group communication are maintained.

The complexity of the rekeying operation is asymmetric when a new member joins and an old member leaves. When a new member joins, the GC can easily multicast the new session key encrypted by the current session key to all the current members, followed by a unicast to the new member to send the new session key encrypted by a predetermined encryption key shared between the GC and the new member. Thus join is easy with low communication and computation cost. However, when an old member leaves, the current session key cannot be used to convey the new session key information securely, since it is also known to the old member. So hereafter we will focus on the rekeying operation for member leaves.

In any key distribution schemes, a basic operation is needed to distribute a piece of secret data to a small group of n members, where each member shares a different individual key with the GC. In all current existing schemes, this operation is fulfilled by the GC using n encryptions followed by n unicasts. Now we describe a new scheme that realizes this operation using *one* erasure decoding of certain MDS code, followed by one multicast to all the n members. We call this scheme the *basic* scheme of key distribution. We will then show this basic scheme can be easily integrated into any key distribution scheme, especially the schemes based on key trees, to reduce computation cost.

The basic scheme consists of three phases: 1) initialization of the GC; 2) join of a new member; and 3) rekeying procedure whenever a group member leaves. Here again, the size of the targeted multicast group is n.

1) GC Initialization: Initially, the GC constructs an (L, n) MDS code C over $GF(q)$ ($L \geq 2n$) and a secure *one-way* hash function $H(\cdot)$ whose codomain is $GF(q)$. The domain of $H(\cdot)$ can be an arbitrary space F large enough so that $H(\cdot)$ has a secure one-way property: given any arbitrary $y \in GF(q)$, it is impossible or computationally hard to get $x \in F$ such that $H(x) = y$. Since other strong properties, such as 2nd-preimage resistance[15, Ch.9.2], are *not* necessary, the hash function H can be implemented more efficiently. The GC then makes public both the MDS code C and the one-way hash function H.

2) Member Initial Join: Whenever a new member i is authorized to join the multicast group for the first time, the GC sends it (using a secure unicast) a pair (j_i, s_i), where s_i is a random element in $H(\cdot)$'s domain F, and j_i is a positive integer satisfying 1) $n + 1 \leq j_i \leq L$; and 2) $j_i \neq j_k$ for all k's, where k is a current member of the multicast group. The pair (j_i, s_i) will be used as member i's *seed key* (denoted as S_i) and kept in GC's local database as long as member i remains a potential member of the multicast group.

3) Rekeying: Whenever some new members join or some old members leave the multicast group, the GC needs to distribute a new session key to all the current members. As already discussed, we will focus on the rekeying operation when an old member leaves. After an old member leaves, the GC needs to distribute a new key to n remaining members to achieve both forward and backward secrecy of the session key.

The GC executes the rekeying process in the following steps:

1) The GC randomly chooses a *fresh* element r in F, which has not been used to generate previous keys;
2) In the remaining group of n members, for each member i of the current group with its seed key (j_i, s_i), the GC constructs an element c_{j_i} in $GF(q)$: $c_{j_i} = H(s_i + r)$, where + is a simple combining operation in F, e.g., string concatenation;
3) Using all the c_{j_i}'s in the above step, the GC constructs a codeword c of the (L, n) MDS code C: set the j_ith symbol of the codeword c to be c_{j_i}. Since C is an (L, n) MDS code, the codeword c is uniquely determined by its n symbols. Using an efficient erasure decoding algorithm for C, the GC can easily calculate the first n symbols $c_1 c_2 \cdots c_n$ of the codeword c;
4) The GC sets the new session key k to be the first symbol c_1 of the corresponding original message: $k = c_1$.
5) The GC multicasts r and $c_2 \cdots c_n$.

The above rekeying process is illustrated in Figure 1(a).

Upon receiving r and $c_1 c_2 \cdots c_{n-1}$ from the GC, an authorized member i of the current group executes the following steps to obtain the new session key:

1) Calculate $c_{j_i} = H(s_i + r)$ with its seed key (j_i, s_i);
2) Decode the first symbol of the codeword c from its n symbols $c_2 \cdots c_n c_{j_i}$ $(n+1 \leq j_i \leq L)$; (notice here only one symbol needs to be recovered, thus the computation cost of erasure decoding is further reduced)
3) Recover the new session key k: $k = c_1$.

This key recover process, as shown in Figure 1(b), finishes the whole rekeying procedure.

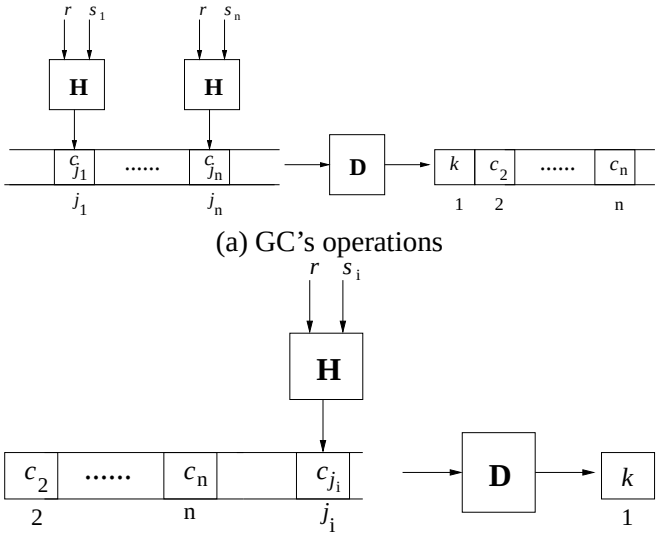

Fig. 1. Rekeying Process of the Basic Scheme

The MDS property of the code C ensures that each authorized member with its c_{i_j} and $c_2 \cdots c_n$ can decode the new session key $k = c_1$ as the GC. Any unauthorized receiver, however, cannot deduce c_1 just from $c_2 \cdots c_n$, since it needs one more symbol of the codeword c. Any stale seed key (j_i', s_j') cannot generate a valid symbol of the current codeword c, since the pair (j_i', s_j') is not used in the generation of c. Thus the forward and backward secrecy is achieved.

C. Evaluation of the Basic Scheme

As can be seen from the above basic scheme, for all the authorized group members, the GC generates a new session key by generating a *common* new message word, the first symbol of which is used as the new session key. The new session key is decided by a random element r in F, as well as all the seed keys of the current authorized group members. The random element r and the $(n-1)$ symbols of the codeword corresponding to the message word are multicast in *plaintext* to all the group members, thus computational cost is much lower than using secure point-to-point communications for the rekeying process. The computation operations needed for this new scheme are erasure decoding of a chosen MDS code, a one-way hash function and some simple combining functions, all of which can be implemented efficiently and are computationally much cheaper than encryptions.

As already noted, the purpose of this basic scheme is to replace separate encryptions followed by unicasts, and be used as a building block for any key distribution schemes whenever applicable. Thus we examine the communication, computation and storage costs of the basic scheme, and compare them with those of conventional rekeying schemes using point-to-point unicasts secured by separate encryptions.

For simplicity, hereafter we assume $q = 2^m$ and the size of a new session key is l bits.

1) Security: Since r and $c_2 \cdots c_{n-1}$ are multicast in plaintext and thus known to all interested parties, including unauthorized receivers who attempt to access the current session, the security of the new session key relies on the secrecy of a codeword symbol c_{j_i} that the authorized member i of the current multicast group has. The secrecy of c_{j_i}, in turn, depends on the secrecy of the seed key that member i has. Thus an attacker who attempts to deduce the new session key k has three ways:

1) *brute force attack*: guess the session key k itself;
2) guess a symbol c_k of the codeword;
3) guess a current member's seed key.

The effort an attacker makes to deduce a session key depends on the parameters of the scheme, namely, the size of finite field $GF(2^m)$, the size t of member i's seed key s_i, and the size of random number r. Intuitively, when proper hash function $H(\cdot)$ and random number generator are chosen, the larger these parameters are, the more effort an attacker needs to make, and thus the more secure this scheme is. The following theorem states the exact sizes of the parameters to ensure the security of this scheme:

Theorem 1: Over a finite field $GF(2^m)$, if $m = t = l_r = l$, then the effort an attacker needs to make to deduce a session key from the basic scheme is *no less* than that of a brute force attack, where t is the size of member i's seed key s_i, l_r is the size of random number r and l is the size of session key k.

Proof: The effort of an attacker needs to make can be measured by the *entropy* of the information an attacker attempts to recover [10]. We give an information theoretical proof to show that the entropy of either a symbol c_k of codeword or a current member's seed key is no less than the entropy of the session key itself. Hence an attacker makes no less effort than a brute force attack of directly guessing the session key itself.

First of all, the entropy of a random session key k is $\mathcal{H}(k) = l$, where $\mathcal{H}(X)$ is the entropy of a random vari-

able X.

If an attacker chooses to guess a symbol c_{j_i} in the codeword directly, then the entropy of c_{j_i} is $\mathcal{H}(c_{j_i}) = log_2\, q = m = l$. Even though the correct location of a symbol in the codeword is needed to generate the corresponding session key, an attacker can exploit the fact that for any location j_i, there always exists a correct symbol c_{j_i} which is a symbol of the codeword that generates a new session key. Thus the attacker can first pick up an arbitrary location j_i, where $n \leq j_i \leq L$, then guess the codeword symbol c_{j_i} at that location. So the entropy of getting a correct codeword symbol is $\mathcal{H}(c_{j_i}) = m = l$.

On the other hand, a seed key consists of two components: an element s_i in F and an integer j_i. It is easy to see that $\mathcal{H}(s_i) = t = l$. Also notice that the two components s_i and j_i of a seed key S_i are chosen *independently*, thus $\mathcal{H}(S_i) = \mathcal{H}(s_i) + \mathcal{H}(j_i) = l + log_2(L - n)$, when an (L, n) MDS code is used.

Finally, the inputs to the one-way hash function $H(\cdot)$ are s_i and r, each of which has size l, and the output is c_{j_i}, a symbol of a codeword, whose size is l, as shown in Figure 1(a). Considering the fact that r is also known the attacker, the effort an attacker needs to deduce s_i from r and c_{j_i} is the *conditional* entropy [10] $\mathcal{H}(s_i|r, c_{j_i}) = \mathcal{H}(s_i) = l$, if $H(\cdot)$ is a properly-chosen secure hash function.

Thus the amount of information an attacker needs to guess to obtain an l-bit session key k is at least l bits, i.e., the attacker needs to make as much effort as a brute force attack to deduce a session key from the basic scheme. □

Now we examine the security strength of this scheme against *conspiracy attack*, where some old members cooperate to deduce the current session key. Since its freshness is determined by a fresh random number r, a new session key is totally independent of all the old session keys. This guarantees the *forward secrecy* and *backward secrecy*, i.e., old members cannot access the new session and new members cannot recover communications of all the previous sessions. One possible way for an old member to get a new session key is to calculate the seed key of a current member from old keys. It is easy to compute a symbol c_k of a codeword c from a message word m, once k is known. However, because of the secure one-way property of the hash function $H(\cdot)$ used to the generate c, it is impossible (or at least computationally hard) to compute a seed key (s_i, j_i) from c_{j_i} even if c_{j_i} is known. Cooperation among d old members helps reduce neither the number of possible locations for j_i of a current member nor computational hardness to get s_i from c_{j_i}, since j_i can be recycled from old ones as long as the current member i has unique j_i which is different from that of all other current members. Thus this scheme is *resilient* to conspiracy attack.

Finally, it is conjectured that for all (L, n) MDS codes over $GF(q)$, where $q = 2^m$, the largest possible L is $q + 1 = 2^m + 1$ [13]. On the other hand, it is necessary $L \geq 2n$ to ensure that each group member receives enough information to obtain the session key. Thus for a session key of l-bit, this scheme can support up to 2^{l-1} members for one group.

2) Complexity: When a new member i is authorized to join the multicast group, the GC assigns a seed key pair (j_i, s_i) to it. This seed key pair remains valid until member i leaves the multicast group permanently. Thus the seed key pair is unicast only *once* to an authorized member. Compared to the rekeying procedure that is needed whenever there is a membership change for the multicast group, this communication cost (in terms of number of the bits transmitted) is negligibly small.

In the rekeying procedure, the GC needs to multicast a fresh random number r and $(n - 1)$ symbols of the new codeword $c_2 \cdots c_n$. Each of the codeword symbols has m bits. The random number r is used to guarantee that the new session key is different from all the old keys used. The length of r determines the total number of sessions the scheme supports. Let r be l_r-bit long, then this scheme can support up to 2^{l_r} sessions. Thus the number of bits that the GC needs to multicast for a rekeying procedure is $l_r + (n - 1)m$. As already discussed in the previous section, it is secure enough to set $l_r = m = l$, thus the communication complexity of this basic scheme is nl, the same as that of separate encryptions followed by unicasts. Hence the basic scheme does *not* incur additional communication complexity to the conventional schemes it replaces.

As of storage complexity, a current member i only needs to store its own seed key pair (j_i, s_i). Since $j_i \leq L$ and s_i is t-bit long, member i needs to store $log_2 L + t$ bits locally. On the other hand, the GC needs to store the seed keys of all the current members of the multicast group, i.e., the GC needs to store $n(log_2 L + t)$ bits. Since it is secure to set $m = t = l$, and $L = 2^m + 1$, for session keys of size l, an individual member's storage complexity is $(2l + 1)$ and GC's is $n(2l + 1)$, both approximately about *twice* of those for conventional schemes.

Finally we examine the computation cost of this scheme. Just as all other key distribution schemes, a seed key pair of a current member are distributed only once using a secure unicast when the member joins the multicast group for the first time. In this join procedure, the GC needs only one encryption operation and the newly-joined member needs only one decryption operation. Most com-

putations are thus carried in the rekeying procedure. During the rekeying procedure, the GC needs n hashing operations with n current members in the multicast group and one erasure decoding operation for n symbols. Symmetrically a current member needs one hashing operation and one erasure decoding operation for only *one* symbol, i.e., the session key. An erasure decoding operation is to solve certain number of unknowns from n linear equations. This usually can be done using *Gaussian Elimination*. The erasure decoding time needed for solving n unknowns (codeword symbols) is about *twice* for solving 1 unknown (codeword symbol). Thus the GC spends about twice as much time on decoding as an individual member. In comparison, in conventional schemes, the GC needs to perform n encryptions while an individual member only needs 1 decryption. Since hashing operations are usually much less expensive than encryption operations, (exact cost of which depends on the hash function chosen,) computation load is hence more evenly distributed from the GC to all the members. In most applications, the rekeying operation is *blocking*, i.e., the operation does not finish until all the members do. Thus computation load balancing is important to reduce the overall execution time of such operations.

The erasure decoding operations for an (L, n) MDS code only need $O(n^2)$ arithmetic operations if standard erasure decoding algorithms are used. Fast decoding algorithms only need $O(nlogn)$ operations [13]. Particularly, when a proper MDS code is chosen, both encoding and decoding only need $O(mn^2)$ binary exclusive OR operations [2], [4], [26].

III. Practical Key Distribution: Applying the Basic Scheme to Key Trees

The basic key distribution scheme reduces computation complexity by replacing computation expensive encryption and decryption operations with more efficient erasure decoding operations of MDS codes. This basic scheme has the same communication complexity as conventional key distribution schemes using secure unicasts. Thus the basic scheme can be readily used as a building block to replace encrypted unicasts in any key distribution schemes, particularly schemes with low communication complexity.

To reduce the communication complexity of rekeying operation, a key-tree based scheme and many of its variations have been proposed [23], [16], [24], [25], [20]. This scheme reduces the communication complexity of rekeying operation to $O(logn)$, while each member needs to store $O(logn)$ keys and the GC needs to store $O(nlogn)$ keys, where n is the multicast group size. This is the most practical key distribution scheme, which balances the communication and storage complexity for dynamic multicast key distribution.

Here we briefly describe a basic key-tree based scheme for the rekeying operation.

A. Key-Tree Based Rekeying Scheme

The main idea to reduce the rekeying communication complexity of this scheme is to have the GC distribute *subgroup* keys in addition to individual member keys and the group session key. These keys are arranged in a *logical* tree hierarchy, where the group session key serves as the root, the individual member keys are the leaves, and the subgroup keys correspond to intermediate nodes. Each member stores all the keys along the path from the corresponding leaf to the root in the tree. Then each subgroup key can be used to securely multicast to the members that are leaves of the corresponding subtree. During the rekeying process, the GC can thus securely multicast to a subgroup of members using their shared subgroup key instead of individual member keys.

Figure 2 shows a key-tree for a 9-member group. K_i ($1 \leq i \leq 9$) is the individual key of member i. K_{1_9} is the group session key that is shared by all the members. Finally, K_{1_3}, K_{4_6} and K_{7_9} are 3 subgroup keys for the 3 corresponding subgroups respectively, e.g., K_{1_3} is shared by members 1 through 3, who form the first subgroup.

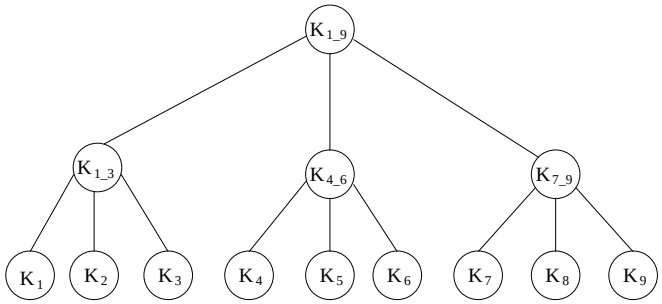

Fig. 2. A Key Tree for a 9-member Group

Now, suppose member 9 leaves the group. Then a rekeying operation is needed to change the current group session key K_{1_9} and the corresponding subgroup key K_{7_9}. This can be achieved by the GC multicasting the following 5 messages (of equal size) to the remaining 8 members: $E_{K_7}(K_{7\_8})$, $E_{K_8}(K_{7\_8})$, $E_{K_{7\_8}}(K_{1\_8})$, $E_{K_{1\_3}}(K_{1\_8})$, $E_{K_{4\_6}}(K_{1\_8})$, where K_{1_8} is the new group session key, K_{7_8} is the new subgroup key for the new subgroup formed by members 7 and 8, and $E_k(m)$ is the encrypted version of message m using key k.

While the key-tree can be arbitrary, it has been shown that a balanced tree is required to achieve low commu-

nication complexity of the rekeying operation. In general, it is easy to show that the communication complexity of a rekeying operation on a balanced d-ary key tree is $O(dlog_d n)$, and each member needs to store $O(log_d n)$ keys. It is not hard to show $dlog_d n$ is minimized when d is 3[20]. The only cost of this key-tree-based scheme is the storage increase at the GC: the GC now needs to store all the additional subgroup keys as well. The total number of the keys stored on the GC increases to $\frac{dn-1}{d-1}$ from $n+1$ for a d-ary balanced key tree.

B. MDS Code Based Rekeying on A Key-Tree

The GC initialization and each member's initial join can be performed exactly same on a key tree as in the basic scheme. Thus we focus on adaption of the basic scheme for rekeying on a key tree.

As in other key-tree based rekeying schemes, MDS codes are used to rekey from bottom (leaves) up, so that child subtrees have their keys for use when their parent node is rekeyed. In Figure 2, when member 9 leaves, the new subgroup key K_{7_8} is rekeyed before the server changes the new group session key to K_{1_8}. When MDS codes are used for the rekeying process, each node (leaf or intermediate) key becomes a pair of (j_i, s_i) as discussed in the previous section. The GC stores all the key pairs on the key-tree. Whenever encryptions are needed for rekeying a subgroup key, a new MDS codeword is constructed from all the key pairs ((j_i, s_i)'s) of the corresponding immediate child nodes and then multicast by the GC. Note that in the rekeying process, each level of the key tree may use the same or different MDS codes. But for the simplicity of implementation, the same MDS code can be used for all levels, since the security of the basic scheme does *not* depend on the MDS code.

In Figure 2, when member 9 leaves the group, the GC first uses the key pairs $K_7 = (j_7, s_7)$ and $K_8 = (j_8, s_8)$, together with a fresh random r, to construct a codeword of an $(L, 2)$ MDS code, and then follow the rekeying procedure of the basic scheme as described in the previous section. After proper decoding, member 7 and 8 share a new subgroup key K_{7_8} that is only known to them, besides the GC. This finishes the rekeying of K_{7_8}. Next, the GC constructs another codeword of an $(L, 3)$ MDS code from the subgroup keys K_{1_3}, K_{4_6} and K_{7_8}, and the decoding output from this codeword produces a new group session key K_{1_8}, which is shared by all the remaining group members.

Note that when the key tree is a d-ary balanced tree, only an $(L, d-1)$ MDS code is needed to rekey the immediate subgroup key shared by the leaf node corresponding to the just-left old member. Then another (L, d) MDS code is needed for rekeying all the other subgroup keys and the new group session key. Since the rekeying scheme based on MDS codes does *not* change the communication and storage complexity of the underlining key-tree-based rekeying scheme, the communication complexity still remains $O(dlog_d n)$.

C. Implementation and Experiments

As pointed out in the previous subsection, d needs to be 3 to minimize the rekeying communication complexity. Thus the MDS codes needed are $(L, 2)$ and $(L, 3)$ codes, and their encoding and decoding are significantly simpler than general (L, n) MDS codes, where $n \gg 3$. Thus the MDS-Code-based scheme is particular suitable and practical for rekeying operations on 3-ary balanced key trees.

In most current applications, a session key needs to be at least 128 bits to be considered reasonably secure. Thus a multicast key distribution scheme is implemented for distributing 128-bit session keys. The scheme uses a 3-ary balanced key tree. The well-known RS (Reed-Solomon) code [17] is employed as an MDS code. As discussed in the basic scheme, $m = l_r = t = 128$ bits. Thus the an RS code would be constructed in a finite field $GF(2^{128})$. However, $GF(2^{128})$ is too large to be computationally feasible to implement. Thus $GF(2^{16})$ is chosen instead. Since an RS code in $GF(2^{16})$ can only process 16-bit symbols, 8 erasure decoding operations in $GF(2^{128})$ are needed to produce a 128-bit session key. A (2^{16},2) RS code is used to distribute the immediate subgroup key to the remaining 2 members of the lowest level subgroup tree affected by the old member. A $(2^{16}, 3)$ RS code is used to distribute all other affected subgroup keys. In Figure 2, when member 9 leaves, the (2^{16},2) RS code is used for distributing a new key K_{7_8} to members 7 and 8, and the $(2^{16}, 3)$ RS code is then used to distribute the remaining keys, including the new session key. Finally, the MD5 algorithm is used as the one-way hash function to produce 128-bit hash outputs. (A more computation efficient hash function can certainly be used, as long as the one-way property is guaranteed. This implementation simply uses existing widely used components which are readily available and whose properties have been studied for a long time.)

Experiments have been conducted to compare computation time using our scheme (which are called RS(2) and RS(3) in the following table and figures, since $(2^{16}, 2)$ and $(2^{16}, 3)$ RS codes are used.) and conventional schemes using encryptions. The encryption algorithms compared are CAST, IDEA, AES/Rijndael and RC4. The encryption keys for these algorithms are all of 128 bits, and an optimal C implementation of AES is used [18]. (DES and

triple-DES are experimented as well, but their test results are not included here, since 1) their encryption keys are not 128-bit and 2) they are much slower to compute than the above algorithms.)

Table I lists the experimental computation time of *one* encryption/decryption of a 128-bit data symbol on a 450 MHZ Pentium 3 PC running Linux. In our scheme using RS codes, encryption and decryption of a 128-bit data symbol correspond to 8 erasure decoding of $(2^{16}, 2)$ RS code (RS(2)) or $(2^{16}, 3)$ RS code (RS(3)) in GF(2^{16}). The erasure decoding of the RS codes are implemented by solving linear equations in GF(2^{16}) using standard Gaussian Elimination. Additions and multiplications in GF(2^{16}) are based on table look-ups. In addition, it takes about 2 usecs to produce a 128-bit hash output using the MD5 on the same PC.

Algorithms	Encryption Time (usecs)	Decryption Time (usecs)
RC4	43	43
CAST-128	16	16
IDEA	12	34
FAST AES-128	8	12
RS(2)	6	6
RS(3)	12	8

TABLE I
COMPUTATION TIME OF A 128-BIT SYMBOL

Notice that in distributing a key, i.e., in the encryption process, RS(3) needs to decode 2 128-bit symbols, while in recovering a key, i.e., in the decryption process, RS(3) needs to decode only 1 128-bit symbol. On the other hand, RS(2) only needs to decode 1 128-bit symbol in both processes. The FAST AES-128 uses an optimal implementation which avoids expensive additions and multiplications in finite field GF(2^8) by using only binary exclusive ORs. The erasure decoding time of RS codes can certainly be further reduced by carefully employing binary exclusive ORs rather than additions and multiplications in finite field GF(2^{16}). Also observe for most encryption algorithms, before encrypting user data, encryption keys need to be preprocessed. This *key scheduling* process takes relatively little time when the user data to be encrypted is large. In key distribution schemes, however, the user data has only 128 bits while the encryption key has 128 bits too. Thus the key scheduling time is usually much longer than that on encrypting the 128-bit user data symbol. Unfortunately such key scheduling operations can neither be avoided nor be amortized to large user data, since data to different members have to be encrypted using different keys. The erasure decoding of RS codes (as well as all other practical MDS codes), on the other hand, does not require such a key scheduling process. Thus key distribution using MDS codes is more efficient than using conventional encryption algorithms.

Based on encryption/decryption time listed in Table I, it is easy to calculate total rekeying time of changing a new session key to a group of n members using a 3-ary balanced key tree. As already discussed, the rekeying process consists of two phases: *key dissemination* (during which the GC's computes and sends necessary data to group members) and *key recovery* (during which group members receive and compute necessary data to obtain a new session key).

The computation time for key dissemination T_{dis} using conventional encryptions is not hard to calculate: 2 encryptions at the lowest level (leaf level) on the key tree, and 3 encryptions on all other ($log_3 n - 1$) levels, thus

$$T_{dis}^{encry} = t_{enc}[2 + 3(log_3 n - 1)]$$

where t_{enc} is encryption time of a single 128-bit data symbol. On the other hand, using RS codes for key dissemination, in additional to erasure decoding, 2 and 3 hashing operations are respectively needed at the leaf and all the other levels on the key tree, so the key dissemination time becomes

$$T_{dis}^{MDS} = (2h + t_{rs2}) + (3h + t_{rs3})(log_3 n - 1)$$

where h is the hashing time to produce a 128-bit output, t_{rs2} is the erasure decoding time of one 128-bit symbol using $(2^{16}, 2)$ RS code, and t_{rs3} is the erasure decoding time of two 128-bit symbols using $(2^{16}, 3)$ RS code.

The key recovery phase is a blocking operation, i.e., the process does not finish until all members obtain the session key. In this sense, the computation time for key recovery should be measured by the computation time of a member who executes the most computations. It is easy to see that the remaining members sharing the similar leaf subgroup with the just-left old member execute the most computations, thus for schemes using a conventional encryption algorithm, the key recovery time is

$$T_{rec}^{encry} = t_{dec}[2 + 3(log_3 n - 1)]$$

where t_{dec} is decryption time of a single 128-bit data symbol. Similar the key recovery time using MDS codes is

$$T_{rec}^{MDS} = (h + t'_{rs2}) + (h + t'_{rs3})(log_3 n - 1)$$

where t'_{rs2} and t'_{rs3} are the erasure decoding time of one 128-bit symbol using $(2^{16}, 2)$ RS and $(2^{16}, 3)$ RS codes respectively.

Finally the total computation time for rekeying is measured as the sum of the key dissemination time and key recovery time. For different group sizes, computation times for rekeying process are plotted in Figure 3, Figure 4 and Figure 5. (Our scheme using RS code is referred to as RS in the figures.) These figures show key distribution using MDS code is more efficient in computation than using conventional encryption algorithms.

IV. Conclusion

We have presented a dynamic multicast key distribution scheme using MDS codes. The computation complexity of key distribution is greatly reduced by employing only erasure decoding of MDS codes instead of more expensive encryption and decryption computations. The overall execution time of group key distribution is further lessened by balancing the computation load among the GC and all the group members. Easily combined with key trees, this scheme provides much lower computation complexity, while maintaining low and balanced communication complexity and storage complexity for dynamic group key distribution. This scheme is thus practical for many applications, including Internet multicast.

References

[1] Michel Abdalla, Yuval Shavitt, and Avishai Wool, "Towards Making Broadcast Encryption Practical", IEEE/ACM Transactions on Networking, 8(4):443–454, August 2000.

[2] M. Blaum, J. Bruck, A. Vardy, "MDS Array Codes with Independent Parity Symbols", *IEEE Trans. on Information Theory*, 42(2), 529-542, March 1996.

[3] R. Blom, "An Optimal Class of Symmetric Key Generation Systems", *Advances in Cryptology*, EUROCRYPT'84, LNCS 209, 335-338, 1984.

[4] J. Bloemer, M. Kalfane, M. Karpinski, R. Karp, M. Luby and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme", ICSI Technical Report, TR-95-048, August 1995.

[5] C. Blundo, A. Cresti, "Space Requirement for Broadcast Encryption", *Advances in Cryptology*, EUROCRYPT'94, LNCS 950, 287-298, 1995.

[6] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung, "Perfectly Secure Key Distribution in Dynamic Conferences", *Advances in Cryptology*, EUROCRYPT'92, LNCS 740, 471-486, 1993.

[7] C. Blundo, L. A. Frota Mattos, D. R. Stinson, "Trade-offs between Communication and Storage in Unconditionally Secure Schemes for Broadcast Encryption and Interactive Key Distribution", *Advances in Cryptology*, CRYPTO'96, LNCS 1109, 387-400, 1996.

[8] R. Canetti, T. Malkin, K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption", *Advances in Cryptology*, EUROCRYPT 1999, May 1999.

[9] G. H. Chou and W. T. Chen, "Secure Broadcasting Using the Secure Lock", *IEEE Trans. on Software Engineering*, 15(8), 929-934, Aug. 1989.

[10] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Son, 1991.

[11] A. Fiat and M. Naor, "Broadcast Encryption", *Advances in Cryptology*, CRYPTO'93, LNCS 773, 480-491, 1994.

[12] M. Luby and J. Staddon, "Combinatorial Bounds for Broadcast Encryption", *Advances in Cryptology*, EUROCRYPT'98, LNCS 1403, 512-526, 1998.

[13] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, Amsterdam: North-Holland, 1977.

[14] R. J. McEliece and D. V. Sarwate, "On Sharing Secrets and Reed-Solomon Codes", *Communications of ACM*, 26(9), 583-584, Sep. 1981.

[15] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 4th Printing, 1999.

[16] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", *Proc. of ACM SIGCOMMi'97*, 277-288, Sep. 1997.

[17] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", *J. SIAM*, 8(10), 300-304, 1960.

[18] V. Rijmen, A. Bosselaers and P. Barreto, *Optimised C code V3.0 (of AES/Rijndael)* `http://www.esat.kuleuven.ac.be/~rijmen/rijndael-fst-3.0.zip`

[19] A. Shamir, "How to Share a Secret", *Communications of ACM*, 24(11), 612-613, Nov. 1979.

[20] J. Snoeyink, S. Suri and G. Varghese, "A Lower Bound for Multicast Key Distribution", *Proc. IEEE INFOCOM 2001*, Apr. 2001.

[21] D. R. Stinson, "On Some Methods for Unconditionally Secure Key Distribution and Broadcast Encryption", *Designs, Codes and Cryptography*, 12, 215-243, 1997.

[22] D. R. Stinson and T. van Trung, "Some New Results on Key Distribution Patterns and Broadcast Encryption", *Designs, Codes and Cryptography*, 14, 261-279, 1998.

[23] M. Waldvogel, G. Caronni, D. Sun, N. Weiler and B. Plattner, "The VersaKey Framework: Versatile Group Key Management", *IEEE Journal on Selected Areas in Communications*, 7(8), 1614-1631, August 1999.

[24] D. M. Wallner, E. J. Harder and R. C. Agee, "Key Management for Multicast: Issues and Architectures", *Internet Draft* (work in progress), draft-wallner-key-arch-01.txt, Sep. 15, 1998.

[25] C. K. Wong, M. Gouda and S. S. Lam, "Secure Group Communications Using Key Graphs", *Proc. ACM SIGCOMM'98*, Sep. 1998.

[26] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding," *IEEE Trans. on Information Theory*, 45(1), 272-276, Jan., 1999.

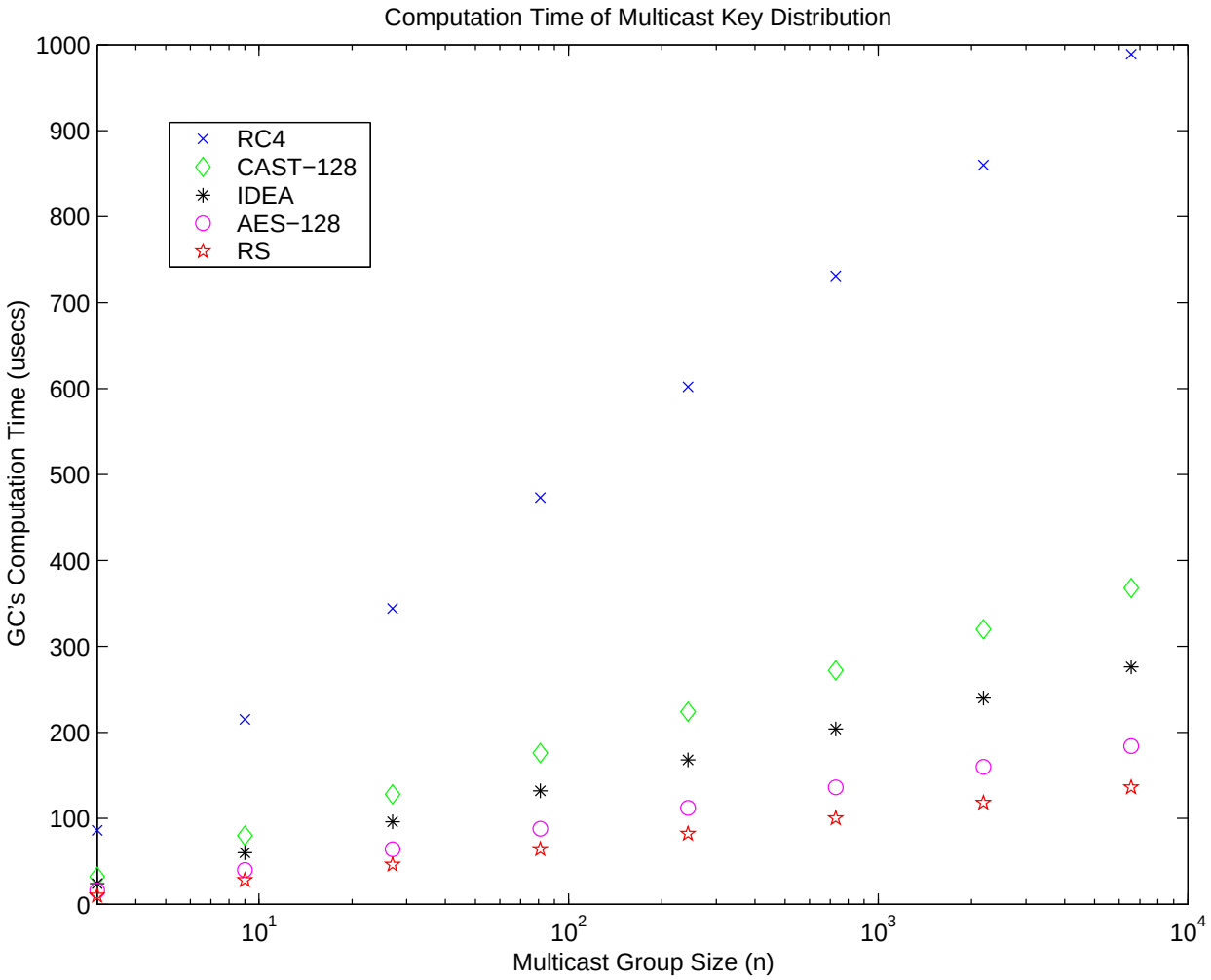

Fig. 3. Computation Time for Key Dissemination

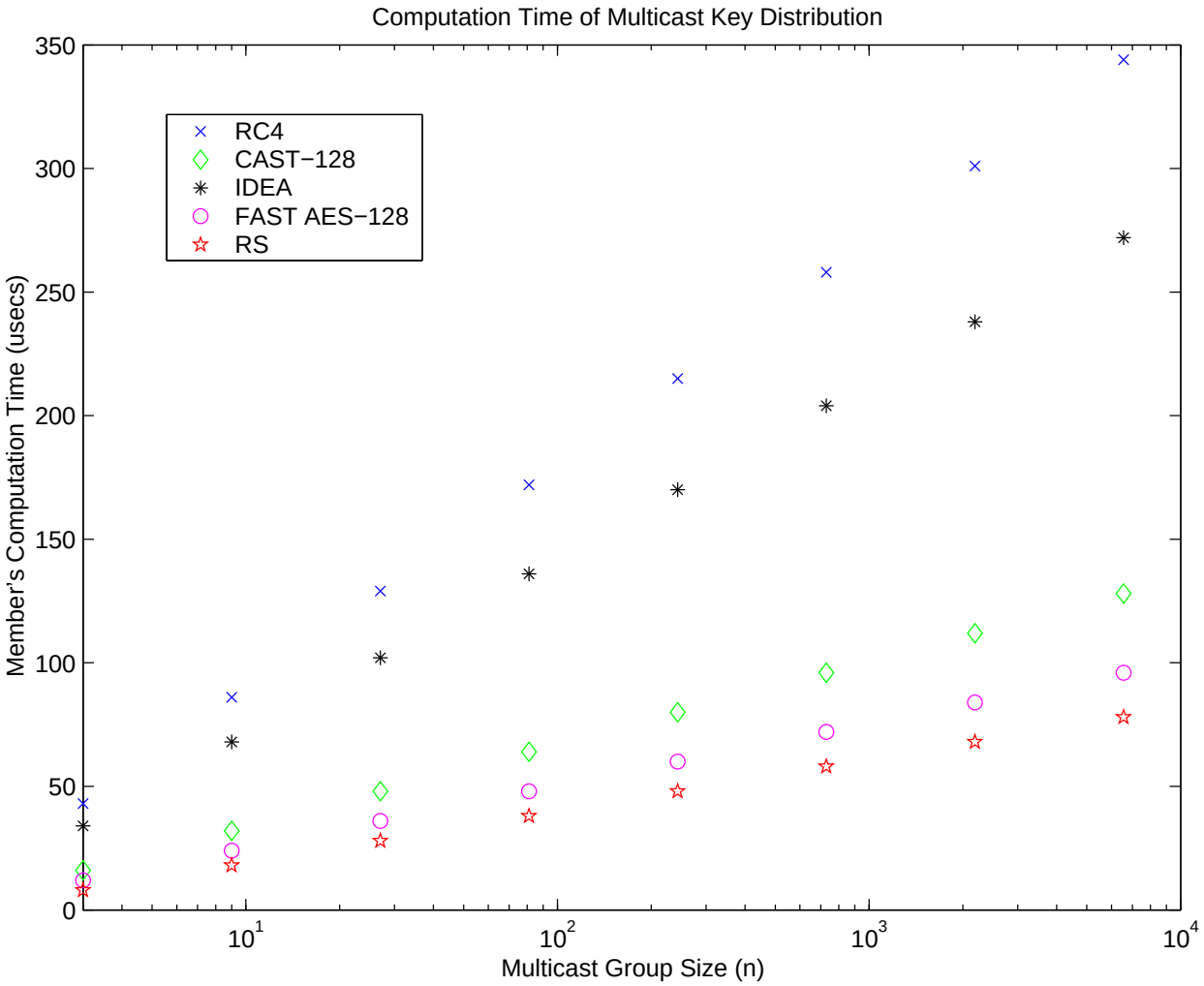

Fig. 4. Computation Time for Key Recovery

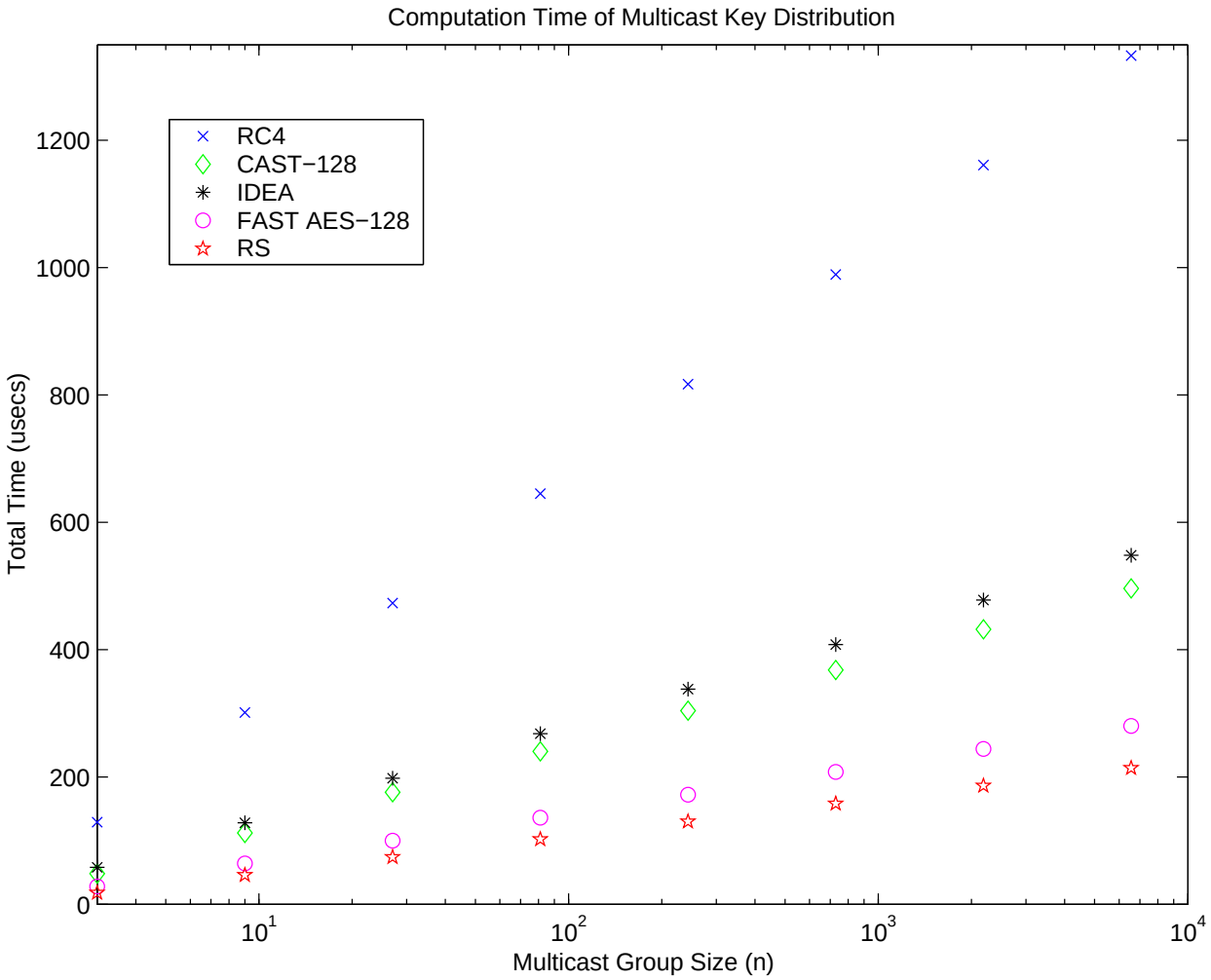

Fig. 5. Total Computation Time for ReKeying