Scalable Group Key Management with Partially Trusted Controllers

Himanshu Khurana[1], Rafael Bonilla[1], Adam Slagell[1], Raja Afandi[2],
Hyung-Seok Hahm[1], and Jim Basney[1]
[1]NCSA, University of Illinois
{hkhurana, bonillla,slagell, hahm, jbasney}@ncsa.uiuc.edu
[2]Computer Science Department, University of Illinois
afandi@cs.uiuc.edu

Abstract

Scalable group key management solutions are crucial for supporting Internet applications that are based on a group communication model. Many solutions have been proposed and of these the most efficient and scalable ones are based on logical key hierarchies (LKH) with symmetric keys organized in a tree. However, these solutions centralize trust in the group controller and make it an attractive attack target for access to communication keys for all groups supported by the controller. In this paper we propose a novel group key management approach, which uses a partially trusted controller that does not have access to communication keys and yet provides the same level of efficiency and scalability as LKH schemes. For this we develop a new public-key encryption scheme, which is based on El Gamal, and we show that the scheme is as secure as El Gamal.

1 Introduction

Many collaborative applications such as conferencing, command-and-control systems, white-boards, publish-subscribe systems, interactive distance-learning, and shared instruments need reliable and secure communication for large groups comprising as many as thousands of members that may leave or join groups at any time. Recently proposed solutions for reliable multicast communications for both IP multicast (e.g., Adamson *et al.* [1]) and applications-layer multicast (e.g., SpinGlass [2]) can support such large groups. Group key management is the cornerstone of providing secure communication, and the key management problem translates to ensuring that only the current group members have the session key. Therefore, the key management scheme must be efficient in changing the session key on member join and leave events and scalable to support large groups. Solutions proposed by Wong *et al.* [20], Wallner *et al.* [19], and Carroni *et al.* [5] that are based on logical key hierarchies (LKH) of symmetric keys organized in a tree provide both efficiency and scalability by reducing both group rekey operation complexity and key storage requirement to $\mathcal{O}(logN)$ for group size N.

Consider the tree-based, group-oriented LKH key management scheme of Wong *et al.* [20]. In this scheme a trusted group controller (GC) creates and distributes symmetric keys to group members, and maintains the user-key relations. For N users GC organizes keys as the nodes of a full and balanced d-ary tree where the root node key serves as the group's session key while the remaining keys serve as *key encrypting keys* that are used to efficiently update the session key on member join and leave events. When a user leaves the group the server only needs to perform approximately $d * log_d(n)$ symmetric encryptions and send a message of size $\mathcal{O}(d * log_d(n))$ to update the session key. This illustrates the efficiency and scalability provided by this approach. However, this scheme requires GC to be fully trusted because its compromise would result in disclosure of the session key and the key encrypting keys managed by GC. Extending that argument, when a group controller manages multiple groups, its compromise leads to compromise of keys for all those groups; e.g., a GC supporting

an interactive distance learning application with different groups representing different classrooms, or a GC supporting a command-and-control system where a large number of groups have permissions to access different components of the system. This has two consequences. First, the data protected by the session keys can be deciphered by the adversary. Second, the only way to recover from such a compromise is to re-key all groups managed by GC since the key encrypting keys are known to the adversary.

Both decentralized key distribution solutions (e.g., [14]) and contributory key agreement solutions (e.g., [4], [9], [17], [16]) solve this problem by eliminating the GC but, unfortunately, they do not scale to large groups. In this paper we address this problem by developing a group key management approach that offers the efficiency and scalability of LKH schemes while minimizing trust in GC at the same time. The new approach is based on a (discrete-log) public key encryption scheme that we have developed utilizing concepts of proxy re-encryption [3] [8] [10]. The encryption scheme enables GC to maintain user-key relations but does not allow GC access to session keys directly or indirectly via decryption keys. Consequently, GC's compromise does not provide the adversary with access to session keys or to (session) key decrypting keys. The latter implies that recovery from GC's compromise does not require re-keying of the entire group. We feel that this is a useful contribution that aims to mitigate the consequences of server compromise, which is an inescapable reality indicated by recent statistics on electronic crime [6].

The rest of this paper is organized as follows. In Section 2 we present our group key management scheme and analyze it in Section 3. In Section 4 we discuss related work and conclude in Section 5.

2 The Proposed Scheme: TASK

In this section we describe our group key management scheme, TASK $-$ a Tree-based scheme that uses Asymmetric Split Keys. TASK comprises a communication group with n members M_1, M_2, ..., M_n, and a partially trusted group controller GC. The group is created by the first member and GC. Members then join (leave) the group with the help of a *sponsor* (an existing group member) and GC. TASK is similar to LKH schemes in that GC manages a d-ary key tree except that TASK uses split asymmetric keys, and GC only manages shares of the split keys (members manage the other part of the shares). Session keys are computed with member shares and, consequently, GC does not have access to them. To ensure security, session keys are updated whenever members join and leave the group.

We now provide some defintions, present the TASK public-key encryption scheme, provide an instance of the key-tree, and show how TASK supports member join and leave events.

2.1 Definitions

1. *El Gamal*. Let $\mathcal{E}_{eg} = (Gen, Enc, Dec)$ be the notation for standard El Gamal encryption [7]. Gen is the key generating function. Hence $Gen(1^k)$ outputs parameters (g, p, q, a, g^a) where g, p and q are group parameters, (p being k bits), a is the private key, and $y = g^a \bmod p$ is the public key. The Enc algorithm is the standard El Gamal encryption algorithm and is defined as $e = (mg^{ar} \bmod p,\ g^r \bmod p)$, where r is chosen at random from Z_q and m is the message. To denote the action of encrypting message m with public key y, we write $Enc_{PK_y}(m)$. Dec is the standard El Gamal decryption algorithm and requires dividing mg^{ar} (from e) by $(g^r)^a \bmod p$.

2. *TASK Encryption Keys*. All members manage *private keys* that they use for encrypting and decrypting protocol messages. Each member has a unique private key and a set of $\mathcal{O}(log_d n)$ private keys in its path to the root node that will be common to other members. For example, K_1 denotes member M_1's private key, $PK_1 = g^{K_1} \bmod p$ is M_1's *public key*, and K_{1-9} and K_{1-8} are the root private keys as illustrated in Figure 1. GC manages *corresponding private keys* for members and for intermediate and root nodes that it uses for re-encrypting protocol messages; e.g., K'_1 is M_1's corresponding private key, and K'_{1-9} and K'_{1-8} are the corresponding root private keys. All private and corresponding private keys are essentially El Gamal

keys and every pair of private and corresponding private keys adds up to the same value $-GKEK$, the *Group Key Encrypting Key*. However, no entity knows the value of $GKEK$.

3. *Session Keys*. The session key, also known as the Data Encrypting Key (DEK), is computed by members to be the (one-way) hash of the root private key.

4. *Key Randomization*. TASK private keys are often updated by either adding or subtracting random numbers. To simplify representation of key updates we define two functions. $A_{(r_1,r_2,...,r_n)}(K)$ denotes the addition of n random numbers $r_1, r_2, ..., r_n$ to key K (mod q). Similarly, $S_{(r_1,r_2,...,r_n)}(K)$ denotes the subtraction of n random numbers $r_1, r_2, ..., r_n$ from key K (mod q).

5. *External PKI Keys*. Some protocol messages are encrypted with external PKI keys (using El Gamal) and all protocol messages are signed and verified with external PKI keys[1] (using the DSA signature algorithm). We differentiate between TASK keys and external PKI keys by placing a bar on top of external PKI keys (e.g., $Enc_{\overline{PK_{GC}}}(m)$ denotes encryption of message m with GC's external PKI public-key).

2.2 TASK public-key encryption scheme $\mathcal{E}$

We denote the TASK Asymmetric Encryption scheme by $\mathcal{E}$= $(IGen, UGen, KU, AEnc, ADec, \Gamma)$. Here $IGen$ is a distributed protocol executed by M_1 (the first member of the group) and GC to generate group parameters g, p and q, private and corresponding private keys K_1 and K_1' and public key $PK_1 = g^{K_1}$. (K_1 is simply a random number modulo q chosen by M_1, and K_1' is a random number modulo q chosen by *GC*). $UGen$ is a distributed protocol executed by joining member M_i, an existing member sponsor M_s, and GC to generate (1) a public-private key pair for M_i, (2) if required, a private key for a (new) intermediate node along the path from M_i's location in the key-tree to the root, and (3) the corresponding private keys for GC. The $UGen$ protocol requires M_i, M_s, and GC to generate random numbers and add/subtract them from K_s and K_s'. It is guaranteed that $K_i + K_i' = K_s + K_s' = GKEK$. KU is a key update protocol initiated by an existing member sponsor and executed by all group members and GC to update $GKEK$. It involves the sponsor choosing a random value r and distributing it to all other group members who add it (mod q) to their private keys, and GC choosing a random value r_{GC} and adding it (mod q) to all corresponding private keys; i.e., $GKEK$ is modified by adding r and r_{GC}. $AEnc$ and $ADec$ are identical to Enc and Dec defined above for El Gamal. Γ is a transformation function which transforms a message encrypted with one TASK public key into a message encrypted with another TASK public key. For example, once $UGen$ has been executed for members M_i and M_j, $\Gamma_{(K_i',K_j')}$ would take as input an encrypted message of the form $(g^{RK_i}S, g^R)$ and output $(g^{RK_j}S, g^R) = ((g^{RK_j'})^{-1}g^{RK_i'}g^{RK_i}S, g^R)$ where S is the message. The fact that $K_j + K_j' = K_i + K_i'$ is crucial in enabling this transformation.

The encryption scheme $\mathcal{E}$ is secure if (1) it retains the same level of security as the standard El Gamal scheme against all adversaries $\mathcal{A}$, (2) the GC cannot distinguish between encryptions of two messages even with access to multiple *corresponding private keys*, and (3) a group member cannot distinguish between encryptions of two messages for which he does not have a decryption key (even if he has other private keys). The proofs are given in the Appendix.

Theorem 1 Let $\mathcal{E} = (IGen, UGen, KU, AEnc, ADec, \Gamma)$ be the TASK encryption scheme. $\mathcal{E}$ is CPA (chosen plaintext attack) secure against the GC, group members, and any adversary $\mathcal{A}$.

2.3 Member Join Protocol

The first member of the group, M_1, creates the group along with GC. First, M_1 generates group parameters g, p and q, chooses a private key K_1 (mod q), and computes the public key $PK_1 = g^{K_1}$. M_1 then sends the group

[1]Most other group key management schemes (e.g., [9], [16], [20]) assume the presence of an external PKI for signature verification but not for encryption. We assume an external PKI for encryption as well but argue that this imposes little or no additional costs in terms of certificate distribution; e.g., PGP certificates typically contain both encryption and signature verification keys.

parameters to GC who chooses the corresponding private key K'_1 (also mod q). Both M_1 and GC implicitly agree that $K_1 + K'_1 = GKEK$ mod q, though neither knows its value.

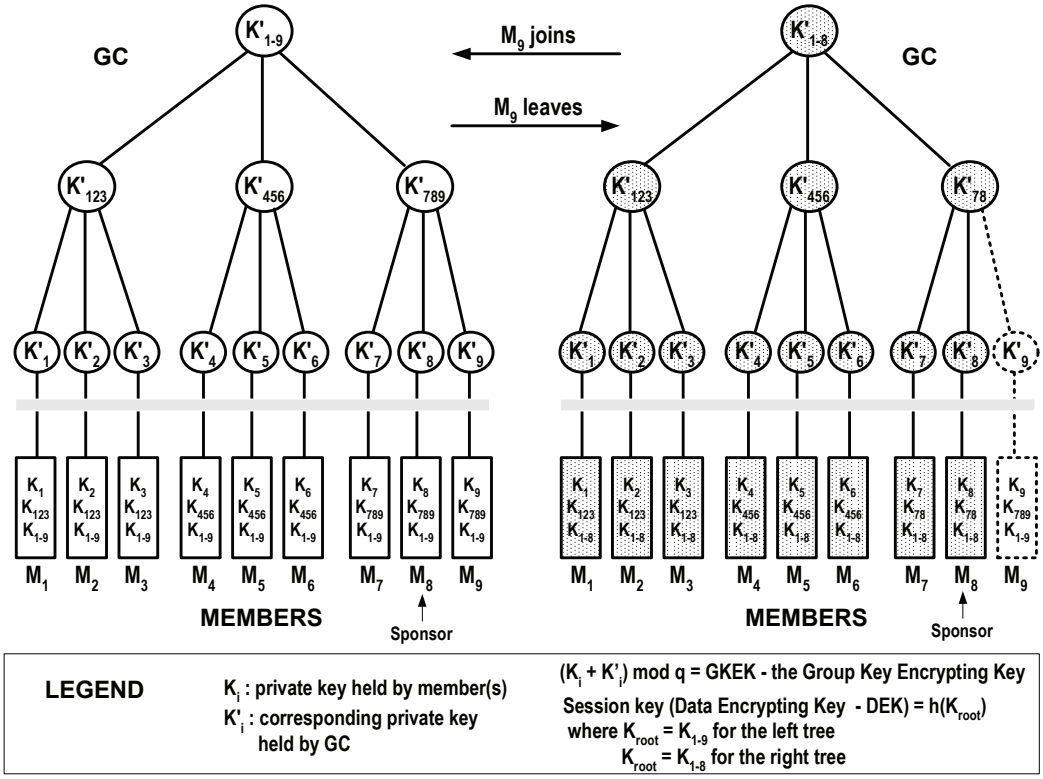

Figure 1: An Example of a TASK Key Tree with Member Join and Leave Events

All other members join the group with the help of a sponsor (an exisiting group member) and GC. We illustrate the join of member M_9 in an existing group of eight members in Figure 1. To join the group, M_9 sends a JOIN message to GC. GC then determines the insertion node in the tree, which will either be shallowest rightmost node if the join does not increase the height of the tree, or the root node otherwise. GC also determines (and then informs) the sponsor for the join event to be the rightmost leaf node in the subtree rooted at the insertion point; M_8 in this case. These join and sponsor determination messages are separate from any join messages generated by the underlying group communication system, though they may be combined for increased efficiency. The TASK join protocol is a three-step protocol illustrated below but consists of only two communication rounds: one in which the sponsor M_8 sends unicast messages to M_9 and GC, and a multicast message to members M_1 ... M_7, and the second in which M_9 sends a unicast message to GC. All TASK multicast messages are encrypted with TASK keys while unicast messages are encrypted with either the receiver's external PKI public-key or with TASK keys.

To add M_9 to the group, the sponsor M_8 generates private key K_{789}, M_9 generates private key K_9, and GC generates corresponding private keys K'_{789} and K'_9. All key generations are accomplished by adding and subtracting random values from previously held keys, and are therefore computationally inexpensive operations. Furthermore, in order to ensure that M_9 cannot access the previous session key, all private keys in the path from the joining node to the root node are updated by adding a random value r to the previous values. However, since TASK requires all private and corresponding private keys to add up to the same value ($GKEK$), r is added to all private keys held by the group members. A new session key is then computed from the updated root private key. In Figure 1, the change in key values is denoted by different subscripts and fill patterns for the tree nodes. The three-step join protocol is detailed below.

TASK PROTOCOL FOR MEMBER M_9's JOIN EVENT

Step 1:

- M_8 generates random value r and adds it all private keys; i.e., $K_8 \leftarrow A_r(K_8)$, $K_{789} \leftarrow A_r(K_{78})$, and $K_{1-9} \leftarrow A_r(K_{1-8})$. Compute new session key: $DEK \leftarrow h(K_{1-9})$.
- M_8 generates random value r_9 and computes a temporary key for M_9: $TK_9 \leftarrow A_{r_9}(K_8)$.
- $M_8 \longrightarrow M_9$: $Enc_{\overline{PK_9}}(g, p, q, TK_9, K_{789}, K_{1-9})$
- $M_8 \longrightarrow GC$: $Enc_{\overline{PK_{GC}}}(r_9)$.
- $M_8 \longrightarrow M_1 ... M_7$: $AEnc_{PK_{1-8}}(r)$.

Step 2:

- Members $M_1 ... M_7$ decrypt r from M_8's message using private key K_{78}, and add r (mod q) to all their private keys; e.g., member M_1 computes $K_1 \leftarrow A_r(K_1)$, $K_{123} \leftarrow A_r(K_{123})$, and $K_{1-9} \leftarrow A_r(K_{1-8})$. Members also compute the new session key: $DEK \leftarrow h(K_{1-9})$.
- GC decrypts random values from M_8 and stores them temporarily.
- M_9 decrypts and stores private keys from M_8, computes new session key from K_{1-9}, generates random value $r_{\bar{9}}$ and computes his private key: $K_9 \leftarrow A_{r_{\bar{9}}}(TK_9)$.
- $M_9 \longrightarrow GC$: $Enc_{\overline{PK_{GC}}}(r_{\bar{9}})$

Step 3:

- GC computes corresponding private key for M_9: $K'_9 \leftarrow S_{r_9, r_{\bar{9}}}(K'_8)$.
- GC chooses a random value r_{GC} and adds it (mod q) to all corresponding private keys; i.e., $\forall i \; K'_i = A_{r_{GC}}(K'_i)$.

In the first step, M_8 generates a random value r that it adds to all private keys and a temporary key for M_9. M_8 also computes the new session key from K_{1-9}. M_8 then sends (1) to M_9 the group parameters, the temporary key, and the private keys in the path from M_9's location in the tree to the root (encrypted with M_9's public key), (2) to GC the random values for generating corresponding private keys (encrypted with GC's public key), and (3) to members $M_1 ... M_7$ the random value r (encrypted with public key PK_{1-8}). In the second step, members $M_1 ... M_7$ decrypt r with private key K_{1-8}, update their private keys by adding r, and compute the new session key $DEK = h(K_{1-9})$. In addition, M_9 generates private key K_9 and sends to GC a random value for generating corresponding private key K'_9. Note that M_9 has a private key that is not known to the sponsor even though the sponsor's key was used to generate it. In the third step GC computes corresponding private keys K'_9 and chooses a random value r_{GC}, which it adds (mod q) to all corresponding private keys. At the end of the join protocol we have added r and r_{GC} to $GKEK$ and all the splits of this key add up to the new value.

The join protocol also presents an easy way for a sponsor to *refresh* the private keys and the session key at any point in time. The sponsor simply chooses a random value r, adds it to all her private keys, and broadcasts it to the entire group by encrypting it with the (old) root private key. On receiving this message, all other group members decrypt r and add it to their private keys. The new session key will be the hash of the new root private key.

2.4 Member Leave Protocol

We illustrate the leave of member M_9 in Figure 1 above. M_9 informs GC of his desire to leave the group and GC then determines (and informs) the sponsor to the right-most leaf node of the subtree rooted at the leaving member's sibling node; M_8 in this case. The leave of member M_9 is enforced by deleting corresponding private key K'_9, updating private keys in the path from M_9's node to the root node, and changing the session key. Private keys are changed by adding a random number to previous value and since TASK requires all private and corresponding values to add to $GKEK$, the random number is added to all private keys. The TASK leave protocol is a three-step protocol that requires two communication rounds; one in which the sponsor sends a unicast message to GC, and the second in which GC sends a multicast message to members $M_1 ... M_7$. The

three-step leave protocol is detailed below.

In the first step, M_8 generates a random value r to be added to all private keys and sends r to GC encrypted with his TASK public key PK_8. In the second step, GC deletes the key (and node) K'_9, transforms the encrypted random value using a minimal set of corresponding private keys (in this case $\mathcal{O}(log_d n)$ keys) such that all remaining members can decrypt r (but M_9 cannot), and sends the multicast message to the remaining group members. (See Section 2.2 for details on the transformation function.) The transformation is used to distribute r because we cannot use the old root key to encrypt r (since M_9 has that key). In the third step, group members decrypt r and compute the new session key, and GC adds a random value r_{GC} to all corresponding private keys. Note that at the end of the leave protocol we have added r and r_{GC} to $GKEK$ and that all the splits of this key add up to the new value.

TASK PROTOCOL FOR MEMBER M_9's LEAVE EVENT

Step 1:

- M_8 generates random value r and it adds it (mod q) to all its remaining private keys; i.e., $K_8 \leftarrow A_r(K_8)$, $K_{78} \leftarrow A_r(K_{789})$, and $K_{1-8} \leftarrow A_r(K_{1-9})$. Compute the new session key: $DEK \leftarrow h(K_{1-8})$.
- $M_8 \longrightarrow GC$: $X = AEnc_{PK_8}(r)$

Step 2:

- GC deletes member node K'_9.
- $GC \longrightarrow M_1 ... M_7$: $\Gamma_{(K'_8, K'_{123})}(X), \Gamma_{(K'_8, K'_{456})}(X), \Gamma_{(K'_8, K'_7)}(X)$

Step 3:

- Members $M_1 ... M_7$ decrypt r from the message sent by GC and add it (mod q) to all their private keys. They compute the new session key: $DEK \leftarrow h(K_{1-8})$.
- GC chooses a random value r_{GC} and adds it (mod q) to all corresponding private keys.

3 Analysis

In this section we analyze the communication and computation costs for TASK and discuss its advantages over LKH schemes. In Table 1 we provide costs focusing on the number of communication rounds, total number of messages, serial number of expensive modular exponentiations, and the serial number of signature generations and verifications (we use DSA signatures for message authentication). The serial cost is the greatest cost incurred by any member in any given round, and assumes parallelization within each round. The total number of messages is the sum of all messages sent by the members in any given round (unicast and multicast messages counted separately). In addition, we provide the maximum size of any unicast and multicast message sent by the members as well as the key storage costs for executing TASK protocols. We separate costs incurred by members and by GC. We compare the costs of our scheme with those for the LKH scheme of Wong *et al.* [20]. We ignore tree balancing costs, which are computed to be $\mathcal{O}(log_2 n)$ for binary trees by Moyer *et al.* [11] and we expect balancing costs of similar complexity for our scheme.

Comparison. From Table 1 we can see that TASK scales to large groups just like LKH. There are only minor differences in both communication and computation costs except for modular exponentiations, which are an artifact of using asymmetric keys. Even there the costs are constant except when GC needs to re-encrypt messages for a member leave event in which case the costs are $\mathcal{O}(log_d n)$ and scale to large groups. (We have not included the average $2(h + 1)$ cost for symmetric encrytions incurred by GC in LKH [20] because these costs are orders of magnitude smaller than exponentiation costs).

Advantages. While providing the efficiency and scalability of LKH, TASK also provides minimization of trust in GC. If GC is compromised, the adversary will get access to all corresponding private keys but will not be able to get any session keys since the corresponding private keys cannot be used to decrypt any session keys. (Note that GC is a *partially trusted* entity because it manages corresponding private key trees, and yet (1) it can get compromised and (2) it cannot discover any session keys). Furthermore, once GC has been re-instated

after recovery, the adversary's advantage will be nullified by updating the private and corresponding private keys without having to re-key the entire group. A member will initiate and execute a refresh operation to update private keys and establish a new session key, and GC will update corresponding private keys by adding a random number to them. However, if GC and one group member are simultaneously compromised, the adversary gets access to the current session key, the member's private keys, GC's corresponding private keys, and to the $GKEK$ for that member's group – by adding the compromised member's private and corresponding private keys. In that case the group will have to be re-keyed, but this will not affect any other groups supported by GC.

Table 1: TASK and LKH Costs

Schemes	Entity	Events	Communication					Computation			Storage
			Rounds	Messages				Exponentiations	Signatures	Verifications	
				Unicast		Multicast					
				# Msgs	Msg Size	# Msgs	Msg Size				
TASK	GC	Join	2	1	O(1)	0	N/A	0	1	3	O(n)
		Leave	2	0	N/A	1	O(dh)	$\lceil dh \rceil$	1	1	
	Member	Join	2	3	O(h)	1	O(1)	7	3	1	O(h)
		Leave	2	1	O(1)	0	N/A	3	1	1	
LKH**	GC	Join	1	2	O(h)	1	O(h)	0	3	1	O(n)
		Leave	1	0	N/A	1	O(dh)	0	1	0	
	Member	Join	1	1	O(1)	0	N/A	0	1	2	O(h)
		Leave	1	0	N/A	0	N/A	0	0	1	

** We provide the costs for a group-oriented scheme presented in Wong *et al.* [20]

LEGEND	n: number of current group members d: degree of tree h: height of tree

3.1 Security Properties

A session key should be known only by the current group members. The following four security properties of session key establishment are supported by TASK:

If the sequence of m changes of the session key is $\mathcal{K} = \{DEK_0, ..., DEK_m\}$, then

1. *Group Key Secrecy:* It is computationally infeasible for a passive adversary or a passive group controller to discover any session key $DEK_i \in \mathcal{K}$ for all i.

2. *Forward Secrecy:* It is computationally infeasible for a passive adversary who knows a contiguous subset of old session keys (say $\{DEK_0, DEK_1, ..., DEK_i\}$) to discover any new session key DEK_j for $j > i$. The contiguous subset of old session keys can include those obtained by the adversary as a former group member or those that were inadvertently leaked to him.

3. *Backward Secrecy:* It is computationally infeasible for a passive adversary who knows a contiguous subset of session keys (say $\{DEK_i, DEK_{i+1}, ...DEK_j\}$) to discover a previous session key DEK_k for $k < i < j$. The contiguous subset of session keys can include those obtained by the adversary when he joins the group or those that were inadvertently leaked to him.

4. *Key Independence:* It is computationally infeasible for a passive adversary who knows a proper subset of session keys $\hat{K} \subset \mathcal{K}$ to discover any other session key not in $\hat{K}$.

Note that Key Independence implies all the other properties, either of Backward or Forward Secrecy implies Group Key Secrecy, and the combination of Backward and Forward Secrecy implies Key Independence. Group Key Secrecy is provided by the TASK encryption scheme $\mathcal{E}$ (in particular, Theorem 1), which ensures than any passive adversary (or a passive group controller) must break El Gamal in order to discover a session key. Forward Secrecy is provided by the Leave Protocol, which uses $\mathcal{E}$ to encrypt the random number that establishes the new session key with a set of public keys. The leaving member does not have a private key for any public key in this set and, therefore, cannot decrypt the random number (stated formally in Theorem 1); i.e., the leaving member cannot discover subsequent session keys. Backward Secrecy is provided by the Join Protocol, which uses $\mathcal{E}$ to encrypt the random number that establishes the new session key with the old root public key. Since the joining member was not part of the group when that public key was established he must break El Gamal just like any other general passive adversary to decrypt the random number and get access to the previous session key.

Another security property of interest is Perfect Forward Secrecy (PFS), which guarantees that compromise of long-term keys in a given session does not cause the compromise of any earlier session keys. TASK provides PFS because even if the private and corresponding private keys are compromised, only the current session key is revealed. This is due to the fact that all private keys and corresponding private keys are changed whenever a new session key is established and, therefore, unless previous private keys are compromised the adversary will not get access to previous session keys (we assume that all random numbers used to modify private keys are deleted after they are used, and that members and GC do not store previous private and corresponding private keys).

4 Related Work

A large number of group key management solutions have been proposed and [12] provides a comprehensive survey of these solutions. Recently, several performance optimization and reliable rekeying techniques have been proposed for LKH schemes. Periodic rekeying techniques [15], [21] and tree management based on usage patterns [22] further improve the scalability and efficiency of LKH schemes while reliable rekeying [23], [21] enables secure group communication in environments where reliable multicast communication may not be available. However, all of these schemes fully trust the GC, and we provide an efficient and scalable group key management scheme that minimizes this trust.

5 Conclusion

In this paper we presented a novel group key management scheme, TASK, which minimizes trust in the group controller and yet retains the efficiency and scalability of LKH schemes. We have shown that TASK can be extended to support application driven group merges and partitions, and will present that work in the near future. In the future, we will also apply reliable re-keyeing and performance optimization techniques proposed for LKH to TASK and analyze the results.

6 Acknowledgements

This work was funded by the Office of Naval Research under contract numbers N00014-03-1-0765 and N00014-04-1-0562. The views and conclusions contained in this document are those of the authors and should not be

interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research or the United States Government.

References

[1] B. Adamson, C. Bormann, M. Handley, J. Macker, "NACK-Oriented Reliable Multicast Protocol (NORM)", RMT Working Group INTERNET-DRAFT, draft-ietf-rmt-pi-norm-09, January 2004.

[2] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal Multicast", *ACM Transactions on Computer Systems*, Vol. 17, No. 2, pp 41-88, 1999.

[3] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography", in Eurocrypt'98, LNCS 1403, Springer-Verlag, 1998.

[4] M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System", (Extended Abstract), EUROCRYPT 1994: 275-286.

[5] G. Caronni, M. Waldvogel, D. Sun, and B. Plattner, "Efficient security for large and dynamic groups," in Proceedings of the 7th Workshop Enabling Technologies, Cupertino, CA: IEEE Comp. Soc. Press, 1998.

[6] CERT E-crime Watch Survey. Carnegie Mellon Software Engineering Institute, May 2004. *http://www.cert.org/about/ecrime.html*

[7] T. E. Gamal, "A Public Key Cryptosystem and a Signature Scheme Based on the Discrete Logarithm", *IEEE Transactions of Information Theory*, 31(4): 469-472, 1985.

[8] A. Ivan and Y. Dodis, "Proxy Cryptography Revisited", in Proceedings of the Network and Distributed System Security Symposium (NDSS), February 2003.

[9] Y. Kim, A. Perrig and G. Tsudik, "Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups", in Proceedings of 7th ACM Conference on Computer and Communication Security (CCS), 2000.

[10] M. Mambo and E. Okamoto, "Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts", *IEICE Transactions on Fundamentals*, vol. E80-A, No. 1, 1997.

[11] M. Moyer, J. Rao, and P. Rohatgi, "Maintaining Balanced Key Trees for Secure Multicast", draft-irtf-smug-key-tree-balance-00.txt, IETF Secure Multicast Group, June 1999.

[12] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication", *ACM Computing Surveys*, Vol.35, No.3, September 2003, pp. 309-329.

[13] R. V. Renesse, K. P. Birman, M. Hayden, A. Vaysburd, and D, Karr, "Building adaptive systems using ensemble," *Software–Practice and Experience* Vol28, No. 9, pp, 963-979, August 1998.

[14] O. Rodeh, K. Birman, and D. Dolev, "The Architecture and Performance of the Security Protocols in the Ensemble Group Communication System", *Journal of ACM Transactions on Information Systems and Security (TISSEC)*, 2001.

[15] S. Setia, S. Koussih, S. Jajodia and E. Harder, "Kronos: A Scalable Group ReKeying Approach for Secure Multicast", in Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 215-228, 2000.

[16] M. Steiner, G. Tsudik and M. Waidner, "Key Agreement in Dynamic Peer Groups", *IEEE Transactions on Parallel and Distributed Systems*, August 2000.

[17] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system", in CRYPTO '88, 1988.

[18] 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*,17(9),pp: 1614–1631, September 1999.

[19] D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures", Internet-draft, September 1998.

[20] C. K. Wong, M. G. Gouda, S. S. Lam, "Secure group communications using key graphs", *IEEE/ACM Transactions on Networking* 8(1): 16-30, 2000.

[21] X. B. Zhang, S. S. Lam, D.-Y. Lee, Y. R. Yang, " Protocol design for scalable and reliable group rekeying", *IEEE/ACM Transactions on Networking*, 11(6): 908-922, 2003.

[22] S. Zhu, S. Setia, and S. Jajodia, "Performance Optimizations for Group Key Management Schemes for Secure Multicast", in Proc.of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS 2003), May 2003.

[23] S. Zhu, S. Setia, S. Jajodia, "Adding Reliable and Self-healing Key Distribution to the Subset Difference Group Rekeying Method for Secure Multicast", in Proc. of Networked Group Communication Conference, 2003.

A TASK Encryption Scheme $\mathcal{E}$

Theorem 1 - Let $\mathcal{E} = (IGen, UGen, KU, AEnc, ADec, \Gamma)$ be the TASK encryption scheme. $\mathcal{E}$ is CPA secure against any Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$, if El Gamal is CPA secure against such an adversary.

Define,

$$Succ_{GC,\mathcal{E}} \stackrel{def}{=} \Pr\left[b = \hat{b} \;\middle|\; \begin{array}{l} (g,p,q,K_s,g^{K_s},K'_s,g^{K'_s}) \leftarrow IGen(1^k), \\ (K_u,K'_u) \leftarrow UserGen(1^k,K'_s,K_s), b \leftarrow \{0,1\}, \\ (m_0,m_1) \leftarrow GC(g^{K_u},K'_u), \hat{b} \leftarrow GC(g^{K_u},K'_u,Enc_{g^{K_u}}(m_b)) \end{array} \right]$$

Then $\mathcal{E}$ is CPA (Chosen Plaintext Attack) secure against GC if $|Succ_{GC,\mathcal{E}} - \frac{1}{2}|$ is negligible for GC. A similar formulation can be made for other adversaries with slight notational changes.

Proof: We have three types of adversaries to consider: the *Group Controller* (*GC*), members that are not intended recipients of a message, and users outside the group. We first consider *GC* and assume that it can break the TASK encryption scheme $\mathcal{E}$. Then $|Succ_{GC,\mathcal{E}} - \frac{1}{2}|$ is non-negligible. Based on *GC's* algorithm to break $\mathcal{E}$, we create a probabilistic, polynomial time (PPT) algorithm $\mathcal{B}$ to mount a successful chosen plaintext attack against the standard El Gamal encryption scheme. However, our premise is that El Gamal is CPA secure. This fact will provide the contradiction to our assumption that *GC* can mount a successful CPA attack against $\mathcal{E}$.

We note that *GC* has knowledge of multiple corresponding private keys. Intuitively, we see that these corresponding private keys cannot be used to decrypt a message encrypted with any member's public key and that *GC* gains no knowledge of $GKEK$, (which can also be used to decrypt messages encrypted with members' public keys), because its view of the corresponding private keys can be made consistent with any value of $GKEK$. However, we formally prove that *GC* cannot break $\mathcal{E}$ without breaking El Gamal. To prove this we use the idea that an adversary can simulate the role of group members for GC.

An oracle $\mathcal{O}$ creates a group by executing $IGen$ with GC, and generates the sponsor's public-private key pair (K_s, g^{K_s}). The oracle then executes $UGen$ with GC to generate public key g^{K_u} as the El Gamal challenge

key. The oracle then gives $\mathcal{B}$ public key g^{K_u}, and the sponsor's public-private key pair. GC at this point has corresponding private keys K_s' and K_u'. $\mathcal{B}$ then simulates the join and leave of (polynomial in k) w members for GC via repeated execution of $UGen$ and KU. For every execution l = 0,1, .., w of KU, $\mathcal{B}$ stores the random value r_l used to generate the session key.

GC now chooses two messages (m_0, m_1) to challenge the security of our encryption scheme $\mathcal{E}$. $\mathcal{B}$ considers these messages as the challenge to standard El Gamal and receives the challenge $Enc_{g^{K_u}}(m_b)$ from a left-right oracle, $\mathcal{O}_{l/r}$. This challenge is modified to $AEnc_{g^{(K_u+r_1+r_2+\ldots+r_w)}}(m_b)$ and is forwarded to *GC* who has a distinguisher able to determine b with probability greater than .5 by the assumption that $|Succ_{GC,\mathcal{E}} - \frac{1}{2}|$ is non-negligible. However, this is a contradiction to the assumption that El Gamal is CPA secure. Therefore, if El Gamal is CPA secure, our encryption system $\mathcal{E}$ is CPA secure against *GC*.

$\mathcal{B}$'s Pseudocode

$(g, p, q, g^{K_u}, K_s, g^{K_s}) \leftarrow \mathcal{O}(1^k, IGen, UGen)$ //Oracle produced challenge
$w \xleftarrow{R} \mathcal{B}$
for $1 \leq l \leq w$ **do**
  Choice $\xleftarrow{R} \{0, 1\}$
  if Choice == 0 and NumMembers ≥ 1 **then**
    $r_l \leftarrow KU(K_s', K_s)$
    NumMembers := NumMembers -1
  else
    $(r_l, K_{u_l}, K_{u_l}') \leftarrow UserGen(1^k, K_s', K_s)$
    NumMembers := NumMembers $+1$
  end if
  $i := i + 1$
end for
$(m_0, m_1) \xleftarrow{R} GC(p, q)$ //Message for encryption challenge
$Enc_{g^{K_u}}(m_b) \leftarrow \mathcal{O}_{l/r}(m_0, m_1, g^{K_u}, g, p, q)$ //Encryption challenge from a left-right oracle to $\mathcal{B}$
$AEnc_{g^{(K_u+r_1+r_2+\ldots+r_w)}} \leftarrow \mathcal{B}(Enc_{g^{K_u}}, r_1, r_2, \ldots, r_w, g, p, q)$ //Transformed encryption challenge for *GC*
$\bar{b} \leftarrow GC(AEnc_{g^{(K_u+r_1+r_2+\ldots+r_w)}}, g^{(K_u+r_1+r_2+\ldots+r_w)}, \bar{K}_u')$ [2]

Group members are considered passive adversaries if they try to decrypt protocol messages encrypted with public keys for which they do not have a private key. The setup for the proof of security against this type of adversary is very similar to the simulation argument for the GC as an adversary. However, in this case $\mathcal{B}$ plays the role of GC as well. An oracle executes $IGen$ and $UGen$ with $\mathcal{B}$ (while $\mathcal{B}$ plays the role of GC), generates public key g^{K_u}, and gives this key to $\mathcal{B}$. $\mathcal{B}$ then executes $UGen$ for the adversary to join the group, and simulates interactions with GC and other members for the adversary. At some point in time the adversary will be able to distinguish a translated encryption challenge allowing $\mathcal{B}$ to break El Gamal for a message encrypted with g^{K_u}.

We now consider adversaries outside of the list. It is trivial to see that outsiders would have to break El Gamal to decrypt a message. This is because both messages from the sender to *GC* and messages from *GC* to the receivers are encrypted with valid El Gamal keys, which are unknown in part or whole to any outsider. The formal proof is similar to the first one. The main difference in this case is that the adversary does not participate in the protocol at all but can only view messages sent and received in the protocol, and have access to the public-keys in the system. That is, once $\mathcal{B}$ is given g^{K_u}, it simulates an entire TASK system for the adversary giving him access to all communications and public keys. At some point in time the adversary will be able to distinguish a translated encryption challenge allowing $\mathcal{B}$ to break El Gamal for a message encrypted with g^{K_u}.□

[2] Here $\bar{K}_u = K_u + r_1 + r_2 + \ldots + r_w$, and $\bar{K}_u'$ is the corresponding private key to $\bar{K}_u$.