# Efficient Group Key Agreement & Recovery in Ad hoc Networks

**Nikos Komninos*, Georgios Mantas***

*Algorithms and Security Group
Athens Information Technology
GR-190 02 Peania (Attiki), Greece
{nkom, gman}@ait.edu.gr

## Abstract

Ad hoc networks are dynamic peer-to-peer wireless networks composed of a collection of nodes which employ wireless transmission methods in a self-organized way without relying on fixed infrastructure or predetermined connectivity. Such networks pose great challenges in group communication. In this paper, we propose an efficient group key agreement and recovery mechanism based on key escrow systems for ad hoc networks. Nodes randomly change their operation and perform authentication services for specific groups.

**Keywords:** key escrow, Clipper, key agreement, key recovery.

## 1 Introduction

Ad hoc networks are characterized by the lack of any centralized entity, as any centralized entity is very easy to be attacked. Furthermore, an ad hoc network is extremely dynamic as its nodes are able to join or leave the network at any time [1, 5]. Moreover, the deployment of security mechanism in an ad hoc network is a challenging issue because of its above inherent characteristics. First of all, conventional authentication techniques can not be used in ad hoc networks since public key infrastructures with a centralized, trusted entity is not possible to be implemented. Thus, only distributed solutions are employed. In addition, group key agreement protocols are applied in ad hock networks instead of key agreement protocols due to lack of trust in the network. However, ad hoc networks are subject to a lot of passive and active attacks which can be derived from outside malicious nodes or from inside compromised hosts [2]. Frequently, as a result from the attacks in an ad hoc network is the loss or the destruction of the secret key. Thus, it is very useful for the ad hoc networks the existence of key recovery mechanisms so as the key, which encrypts the data transferred among nodes, to be obtained at any time it is necessary. A key recovery mechanism can be achieved with a key escrow system. A key escrow system requires a lot of complex computations and storage of information. These requirements of a key escrow system can be satisfied by smart cards.

In this paper, we proposed an efficient group key agreement and recovery mechanism for ad hoc networks. Each node of the ad hoc network is equipped with a smart card which performs the complex computations required according the group session key agreement protocol, as well as the key recovery mechanism. Following the introduction, in section 2, we present the related work of group session key agreement protocols, key recovery, key escrow systems as well as the Clipper key escrow system. In section 3, the design of the proposed group session key agreement protocol is discussed. Furthermore, in section 4, the modified Clipper key escrow system is described. In section 5, the experimental results are presented. Finally, section 6 concludes the paper.

## 2 Related Work

### Group Session Key Agreement Protocols

Group key agreement protocols are generalized key agreement protocols which establish a common group session key among a group of parties and not only between two parties. Also, group key agreement protocols take into consideration the cases that parties may join or leave a group at any time. In these dynamic cases, supplementary group key agreement protocols are used in order that a new group session key to be derived. Moreover, group key agreement protocols do not require central authority. Thus, group key agreement protocols are suitable for ad hoc networks characterized by their dynamic changing topology, the lack of centralized control and trusted third parties as well as the resetting of connections [4]. Several group key agreement protocols are presented in [5].

### Key Recovery

The objective of a key recovery system is to permit access to encrypted communication data, when the encryption key is lost or destroyed due to equipment failure or malicious activities. An establishment of a session key is always required in order to achieve a key recovery. A key recovery system derives the encryption key from information stored in a secure back up copy. Key escrow is a way to achieve key recovery. According to this mechanism, information associated with the decryption key is divided into several parts and these parts are distributed and stored to trusted third parties (escrow agents). Thus, the escrow agents are able to reconstruct the decryption key from their stored parts at any time [4].

**Key escrow systems – The Clipper key escrow system**

A key escrow system provides encryption of user data using a session key ($K_s$) which can be recovered by an authorized third party under special circumstances. Thus, a third party, which has monitored the encrypted user data with the session key, is able to decrypt them. A very famous implementation of a key escrow system was the Clipper key escrow system, which uses the Clipper chip. This chip was developed and started to be promoted by the U.S. government as an embedded encryption device for voice communication systems in 1993. Clipper key escrow system offers encryption of the user's data as well as capability of session key recovery ($K_s$). This system is based on the fact that two key components, which can create an encryption key, can be stored into two escrow agents (authorized third parties) which are going to be part of the user data recovery mechanism when a recovery request exists [4].

## 3 Proposed Group Key Agreement Protocol

First of all, we consider a group of N nodes. We suppose that this group is a cluster created by applying any clustering algorithm in an ad hoc network [3]. We consider that the cluster-head (one of the N nodes) which is elected according the applied clustering algorithm is our Checker. Furthermore, we consider that the Checker and each node of this cluster are connected with a smart card. Firstly, we employ our group session key agreement protocol on the created cluster. Then, our modified Clipper key escrow system can be employed for key recovering at any time in our group (cluster). The Checker is considered as the only key escrow agent in our system. Furthermore we consider that each node has a unique identity number, ID. Also, each node knows the secret master key ( $K_M$ ), which is the stored key in the smart card, and the ID of the Checker. In addition, each node has embedded a unique key ( $K_U$ ).

In the **first step**, each node sends to the Checker a message that includes its ID (id_node), the ID of the Checker (id_checker), and an encrypted message with the master key $K_M$, which is derived from the concatenation of the ID of the node (id_node), the ID of the Checker (id_checker) and a nonce (nonce_node) generated randomly by each node (i.e. $E_{K_M}(id\_node \,\|\, id\_checker \,\|\, nonce\_node)$ ).

In the **second step**, the Checker decrypts the received encrypted message from each node and obtains the ID of each node. Then, the Checker compares it with the ID of each node sent outside of the encrypted message in order to authenticate each node. After that, the Checker sends to each node a message that includes its ID (id_checker), the ID of the corresponding node (id_node), and an encrypted message with the master key $K_M$, which is derived from the concatenation of the ID of the Checker (id_checker), the ID of the corresponding node (id_node), the nonce generated by the corresponding node in the step one increased by one (nonce_node+1) and a nonce (nonce_checker) generated by Checker,
(i.e. $E_{K_M}(id\_checker \| id\_node \| nonce\_node+1 \| nonce\_checker)$).

In the **third step**, each node decrypts the received encrypted message from the Checker and obtains the ID of the Checker. Then, each node compares it with the ID of the Checker sent outside of the encrypted message in order to authenticate the Checker. After that, each node sends to the Checker a message that includes its ID (id_node), the ID of the Checker (id_checker), and an encrypted message with the master key $K_M$, which is derived from the concatenation of the ID of the node (id_node), the ID of the Checker (id_checker), the nonce (nonce_checker) generated by the Checker increased by one (nonce_checker+1) and the unique key of each node (i.e. $E_{K_M}(id\_node \| id\_checker \| nonce\_checker+1 \| K_{U\_node})$ ).

In the **forth step**, the Checker decrypts all the received encrypted messages and obtains the unique keys of all nodes. Thus, the Checker is able to create the family key ( $K_F$ ). The family key is calculated in the Checker by the following formula:

$$K_F = K_{U\_node_1} \oplus K_{U\_node_2} \oplus \ldots \oplus K_{U\_node_{N-1}}$$

We note that the family key ( $K_F$ ) is a key which contains key contributions of each node apart from the Checker. Then, the Checker broadcasts the family key ( $K_F$ ) to all nodes. In particular, the Checker broadcasts a message which includes its ID (id_checker), and an encrypted message with the master key ( $K_M$ ), which is derived from the concatenation of the ID of the Checker (id_checker), the family key ( $K_F$ ), a random quantity (S_checker) generated by the Checker and another random quantity (nonce1) generated by the Checker
(i.e. $E_{K_M}(id\_checker \| K_F \| S\_checker \| nonce1)$ ).

In the **fifth step**, each node decrypts the received encrypted message from the Checker, obtains the random quantity (S_checker) generated by the Checker, the random quantity (nonce1) generated by the Checker and the family key ( $K_F$ ) which was created by the Checker in the previous step. After that, each node constructs a session key ( $K_i$ ) with the following XOR function:

$$K_i = K_F \oplus S\_checker$$

Then, each node sends to the Checker a message that includes its ID (id_node), the ID of the Checker (id_checker), and the hash value produced by the hash function H of a message derived from the concatenation of the ID of the Checker (id_checker), the random quantity (nonce1) generated by the Checker in the forth step increased by one (nonce1+1) and the calculated session key ( $K_i$ )

(i.e. $H(id\_checker \| nonce1+1 \| K_i)$ ).

In the **sixth step**, the Checker compares the hash values that it received from each node. If the Checker finds that all the received hash values are the same ( $K_s = K_1 = K_2 = \ldots = K_{N-1}$ ), it means that each node has generated the same session key ( $K$ ). Then, the Checker notifies all nodes that the session key has been established successfully. Thus, the Checker sends to each node a message that includes its ID (id_checker), and an encrypted message with the master key ( $K_M$ ), which is derived from the concatenation of the ID of the Checker (id_checker) and an number (ack_code) which is known to the nodes a priori and means that the session key has been established successfully (i.e. $E_{KM}(id\_checker \| ack\_code)$ ).

Thus, the group session key agreement protocol flow is the following:

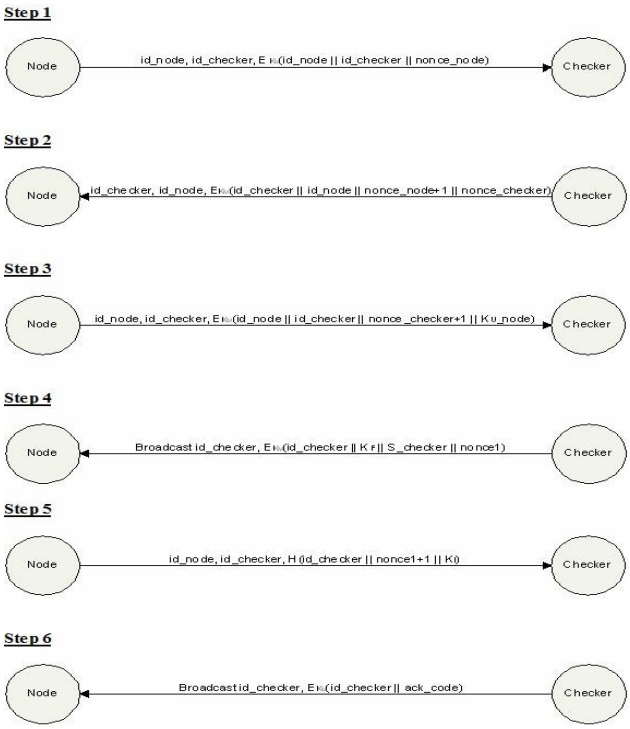


Fig. 1. The proposed group session key agreement protocol flow

## 4 Modified Clipper Key Escrow System

When the group session key agreement protocol is accomplished, each node has obtained the two critical keys which are going to be used for the creation of its modified LEAFs. These two critical keys are the family key ( $K_F$ ) and the session key ( $K_i$ ) of each node which is the group session key ( $K_s = K_i$ ) as all nodes have created the same session key according to the sixth step of group session key agreement protocol. Our modified key escrow system consists of the following processes:

**Process 1**

First of all, the modified LEAF is created. The modified LEAF is a data block which contains the ID of the node, the encrypted session key with the unique key of each node ( $E_{KU\_node}(K_s)$ ), a hash value (hash_value) and a timestamp.

The hash value (hash_value) is created by the hash function H of a message derived from the concatenation of the session key ( $K_S$ ) and a random value.

Then, the LEAF block is encrypted with the family key ( $K_F$ ) (i.e. $E_{KF}(id\_node \| E_{KU\_node}(K_S) \| hash\_value \| timestam)p$). After that, the node sends its ID and the LEAF block to the Checker. The Checker stores the received LEAF in a file according to the ID of the node that sent it. Thus, the Checker stores the LEAFs of each node.

**Process 2**

In case that a node wants to recover the session key, it needs to send a recovery request message to the Checker. Thus, the node sends to the Checker a recovery request message that includes its ID (id_node), and an encrypted message with the master key $K_M$, which is derived from the concatenation of the ID of the node (id_node) and the recovery code (rec_code) (i.e. $E_{KM}(id\_node \| rec\_code)$ ).

**Process 3**

When the Checker receives the recovery request message, it decrypts the received encrypted message and obtains the ID of the node and the recovery code. Then, the Checker compares the obtained ID with the ID of the node sent outside of the encrypted message in order to authenticate the node. After that, the Checker recognizes the recovery code and restores the LEAF that corresponds to the node that sent the recovery request message. Then, the Checker decrypts the restored LEAF with the family key ( $K_F$ ), which is common for the Checker and all nodes according to the group session key protocol, and obtains the encrypted session key with the unique key of the node ( $E_{KU\_node}(K_s)$ ). Then, the Checker sends to the node a message that includes its ID (id_checker) and an encrypted message with the master key $K_M$, which is derived from the concatenation of the ID of the Checker (id_checker) and the encrypted session key with the unique key of the node ( $E_{KU\_node}$ )( $K_s$ )

(i.e. $E_{KM}(id\_checker \| E_{KU\_node}(K_s))$ ).

**Process 4**

The node that sent the recovery request message, receives the response message of the Checker, decrypts the received encrypted message and obtains the ID of the Checker as well as the encrypted session key with its unique key ( $E_{KU\_node}(K_s)$ ).Then, the node compares the obtained ID with the ID of the Checker sent outside of the encrypted message in order to authenticate the Checker. After that, the node decrypts the

quantity ($E_{KU\_node}$)($K_s$) with its unique key in order to obtain the session key ( $K_S$ ).

## 5 Experimental Results

For the implementation, we considered that our group consists of three nodes and each node is connected with a smart card. One of them is the Checker. Thus, the simulation environment consists of the Checker, the Node1 and the Node2. For the communication between the Checker and Node1 and for the communication between the Checker and Node2 we used the client/server model. Furthermore, all required cryptographic functions (encryption, decryption, hashing), were executed by the Cryptoflex Smart Card. There are three types of encryption: encryption with the master key, encryption with the family key and encryption with the unique key of each node.

The master key is a DES key stored in the smart card. In case that one of the three applications needs to make encryption with the master key, then the application gets connection with the smart card, sends the data for encryption to it and the smart card encrypts these data with the stored DES key. Then, the smart card returns the encrypted data back to the application.

The family key is created during the group key agreement protocol. Now, the encryption with the family key includes two steps. In the first step, the application, which wants to make encryption with the family key, gets connection with the smart card, sends the data for encryption to it and the smart card encrypts these data with the stored DES key. Then, the smart card returns the encrypted data back to the application. In the second step, the application is XORing the returned encrypted data with the family key.

Each node has embedded a unique key. The encryption with the unique key of a node includes two steps too. In the first step, the application, which wants to make encryption with the unique key, gets connection with the smart card, sends the data for encryption to it and the smart card encrypts these data with the stored DES key. Then, the smart card returns the encrypted data back to the application. In the second step, the application is XORing the returned encrypted data with the unique key.

Furthermore, for each of the above encryption type there is the corresponding decryption type: decryption with the master key, decryption with the family key and decryption with the unique key of each node. Thus, in case that one of the three applications needs to decrypt encrypted data with the master key, then the application gets connection with the smart card, sends the encrypted data for decryption to it and the smart card decrypts these data with the stored DES key. Then, the smart card returns the decrypted data back to the application.

In case that one application needs to decrypt encrypted data with the family key, then two steps are required. In the first step, the application is XORing the encrypted data with the family key. In the second step, the application gets connection with the smart card, sends the data which is the result of XOR for decryption to the smart card and the smart card decrypts these data with the stored DES key. Then, the smart card returns the decrypted data back to the application.

In case that one application needs to decrypt encrypted data with the unique key of a node, then two steps are required. In the first step, the application is XORing the encrypted data with the unique key. In the second step, the application gets connection with the smart card, sends the data which is the result of XOR for decryption to the smart card and the smart card decrypts these data with the stored DES key. Then, the smart card returns the decrypted data back to the application.

Regarding hashing, in case that one application needs to calculate the hash value of an amount of data, then the application gets connection with the smart card, sends the data to it and the smart card calculates the corresponding hash value.

We measured that the proposed group session key agreement protocol requires 21 seconds until to be accomplished. In other words, Node1 should wait 21 sec until to receive the ack_code in step 6. Furthermore, we calculated that Node1 should wait 4 sec until to recover the session key. This is the required time from the moment that Node1 sends the rec_code in process 2 until to achieve the recovery.

## 6 Conclusion

Ad hoc networks suffer from lack of reliable security mechanisms due to their inherent characteristics. In this paper, we proposed an efficient group key agreement and recovery mechanism. Our mechanism performs better than other protocols [5] at the key agreement and recovery phase.

## References


[1] M. Bechler, H.-J. Hof, D. Kraft, F. Pählke, L. Wolf, “A Cluster-Based Security Architecture for Ad Hoc Networks”,
*IEEE INFOCOM* 2004

[2] Kai Inkinen, “New Secure Routing in Ad Hoc Networks: Study and Evaluation of Proposed Schemes*”, HUT T-110.551*
*Seminar on Internetworking*, Sjökulla, 2004-04-26/27

[3] Kadri, A. M’hamed, M. Feham, “Secured Clustering Algorithm for Mobile Ad Hoc Networks”, *IJCSNS International Journal of Computer Science and Network Security*, **volume 7 No.3**, March 2007

[4] Menezes A., Oorschot van P. and Vanstone S., 1996, *Handbook of Applied Cryptography*, CRC Press

[5] Bing Wu, Jie Wu and Mihaela Cardei, "A Survey of Key Management in Mobile Ad Hoc Networks", *HANDBOOK OF RESEARCH ON WIRELESS SECURITY*, Y. Zhang, J. Zheng, and M. Ma