Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 51 (2007) 2805–2817

Computer Networks

www.elsevier.com/locate/comnet

Secure multicast in dynamic environments

Chun-Ying Huang [*], Yun-Peng Chiu, Kuan-Ta Chen, Chin-Laung Lei

Department of Electrical Engineering, National Taiwan University, Room 604, EE-3 Building, No. 1, Section 4, Roosevelt Road, Taipei 106, Taiwan, ROC

Received 1 June 2006; received in revised form 29 November 2006; accepted 30 November 2006
Available online 19 December 2006

Responsible Editor: P. Dowd

Abstract

A secure multicast framework should only allow authorized members of a group to decrypt received messages; usually, one "group key" is shared by all approved members. However, this raises the problem of "one affects all", whereby the actions of one member affect the whole group. Many researchers have solved the problem by dividing a group into several subgroups, but most current solutions require key distribution centers to coordinate secure data communications between subgroups. We believe this is a constraint on network scalability. In this paper, we propose a novel framework to solve key management problems in multicast networks. Our contribution is threefold: (1) We exploit the ElGamal cryptosystem and propose a technique of key composition. (2) Using key composition with proxy cryptography, the key distribution centers used in secure multicast frameworks are eliminated. (3) For key composition, the framework is designed to resist node failures and support topology reconstruction, which makes it suitable for dynamic network environments. Without reducing the security or performance of proxy cryptography, we successfully eliminate the need for key distribution centers. Our analysis shows that the proposed framework is secure, and comparison with other similar frameworks demonstrates that it is efficient in terms of time and space complexity. In addition, the costs of most protocol operations are bounded by constants, regardless of a group's size and the number of branches of transit nodes.
© 2006 Elsevier B.V. All rights reserved.

Keywords: ElGamal cryptosystem; Key composition; Key management; Proxy cryptography; Secure multicast

1. Introduction

The original design of multicast [1] did not incorporate mechanisms that guaranteed the integrity and confidentiality of messages. Since multicast routers simply deliver messages via a tree-like structure according to group addresses, when an arbitrary user registers with a multicast group, he can receive all the messages from the distributor. This raises a serious security problem. Therefore, a great deal of research has focused on security mechanisms for multicast and group communications. The goal of these mechanisms is simple: *A secure multicast mechanism should be able to control member access rights and guarantee the confidentiality of data. Only*

[*] Corresponding author. Tel.: +886 2 33663544.
E-mail addresses: huangant@fractal.ee.ntu.edu.tw (C.-Y. Huang), frank@fractal.ee.ntu.edu.tw (Y.-P. Chiu), jethro@fractal.ee.ntu.edu.tw (K.-T. Chen), lei@cc.ee.ntu.edu.tw (C.-L. Lei).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.11.027

authorized members with a proper key should be able to decrypt ciphertexts received from the group. Thus, the secure multicast problem can be reduced to two questions: (1) How can cryptographic keys be generated and distributed? (2) Who should have access to these cryptographic keys to read confidential messages?

In general, secure multicast mechanisms, can be classified as centralized, decentralized, or distributed mechanisms [2]. Centralized mechanisms, such as [3,4], employ a single entity to control the whole group and seek to minimize storage costs, computational power, and bandwidth requirements. However, since there is only a single entity, the possibility of a single point of failure increases. In contrast, methods like [5–8] divide a group into several subgroups; hence, better scalability and reliability can be achieved because failures are confined to subgroups. Even so, these methods still require a centralized controller to coordinate secure data communications between different subgroups. Distributed approaches, like [9,10], eliminate centralized controllers so that all group members can contribute to the required cryptographic key, or it can be generated by one member. The result is a group key shared by all members; therefore, each user must be aware of the group membership list. In addition, computation and communication requirements may grow linearly as the number of group members increases.

Each approach has benefits and drawbacks. However, to construct a large-scale secure multicast network, especially one that requires the collaboration of nodes in different administrative domains, we believe a hybrid framework that uses a decentralized model working in a distributed manner is a good design choice.

In this paper, we propose a new secure multicast framework that builds on the advantages of decentralized and distributed approaches. The framework is constructed in two steps. First, we exploit the mathematics used in ElGamal cryptography [11] and proxy cryptography [12,13] and extend the calculations to develop a distributed protocol for key composition. Then, using the protocol, we eliminate the key distribution centers used in proxy cryptography-based secure multicast networks like [7].

The framework adopts a network model similar to those in [7,8], which are source-based multicast tree networks. We use proxy cryptography to deliver key encryption keys (KEKs). In proxy cryptography, given a sender A, receiver B, and proxy P, a message encrypted by A's encryption key, k_A, and transformed by P's encryption key, k_P, can be decrypted by B's decryption key, k_B. *Proxy cryptography has several benefits, including low key storage requirements, message invisibility on intermediate proxies, and low computational complexity regardless the number of proxy branches.* Since a proxy transforms an encrypted message without understanding the original message, it is suitable for large-scale networks, especially when most intermediate nodes can not be totally trusted.

The remainder of this paper is organized as follows. In Section 2, we review several related works that propose solutions for the secure multicast problem. In Section 3, we introduce the basic components used in our framework, and then construct the framework in Section 4. In Section 5, we evaluate and discuss the proposed approach, and compare its performance with other methods. In Section 6, we present our conclusions.

2. Related works

Several solutions to the problem of secure group communications have been proposed. In centralized approaches, such as [3,4], the authors propose a logical key hierarchy (LKH), whereby a key distribution center (KDC) maintains a tree of keys. Except for the root node (the KDC) and the leaf nodes (the group members), all intermediate nodes are logical nodes. Initially, each node is assigned a key encryption key (KEK). The KDC knows the KEKs of all nodes and a group member knows the KEKs of nodes on the path from the KDC to itself. Thus, in a balanced key tree with m members, each member has to keep $O(\log(m))$ keys. When a group secret key is being delivered to a member, the key is encrypted using the KDC's KEK and is then sent to all group members by a multicast. If a member joins or leaves a balanced key tree containing m members, $O(\log(m))$ keys and $O(\log(m))$ encryptions are required to update the corresponding KEKs in the key tree. Since there is a centralized key distribution center, the single point of failure and scalability problems cannot be avoided.

In distributed approaches, members in one group cooperate to generate a shared secret key. In group Diffie–Hellman (GDH) [9], the authors extend the Diffie–Hellman [14] key agreement protocol to n parties. Given a cyclic group G of order q with a generator g, each participant i chooses a random value $N_i \in G$. The GDH protocol starts with the first member passing a secret number g^{N_1} to the second member. On receipt of the set of secret numbers from the $(j-1)$th member, the jth member increases the numbers in the set by adding his secret number g^{N_j} and generates a new set for the next member. After $O(n)$ rounds of computation, the final group key k of $g^{N_1 \cdots N_n}$ can be generated and delivered to all participants. Since members in a distributed secure group communication environment are required to know the group membership list, such frameworks may be not suitable for large groups. In addition, they suffer from the "one affects all" problem.

To overcome the problems encountered in centralized and distributed models, several decentralized frameworks for secure multicast have been proposed. In IOLUS [5], members of a multicast group are divided into several subgroups. The connectivity between the subgroups is controlled by *group security agents* (GSAs) placed on the links between the subgroups. Each subgroup is relatively independent; therefore, membership changes can be confined to the subgroup in which they occur. The problem with IOLUS is that while an encrypted message is being transmitted from subgroup A to subgroup B, the GSA decrypts the message with A's secret key and then encrypts it with B's secret key. Although this achieves the goal of confidentiality, the GSAs must be totally trusted and well protected to prevent the leakage of confidential message.

To solve the problem of trusting third parties, another framework, the dual-encryption protocol (DEP) [6], has been proposed. In DEP, a multicast group is also partitioned into several subgroups, each of which has a corresponding subgroup manager (SGM). Confidentiality is guaranteed by a data encryption key (DEK) maintained by a group controller (GC). Three key encryption keys (KEKs) are used to deliver a DEK: KEK_1 is shared by the GC and the SGM; KEK_2 is shared by the SGM and members of its subgroup; and KEK_3 is shared by the GC and members of a subgroup, except the SGM. Let the encryption and decryption functions with secret key k be denoted by $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$, respectively. The GC delivers the DEK to a subgroup's members via the following steps. First, the GC sends $\mathsf{Enc}_{\text{KEK}_1}(\mathsf{Enc}_{\text{KEK}_3}(\text{DEK}))$ to the SGM. The received message is decrypted with KEK_1 and re-encrypted with KEK_2. The new encrypted key $\mathsf{Enc}_{\text{KEK}_2}(\mathsf{Enc}_{\text{KEK}_3}(\text{DEK}))$ is then sent to the subgroup's members. Since only those members know both KEK_2 and KEK_3, they can obtain the correct DEK. When a member joins or leaves a subgroup, KEK_2 is changed by the SGM and sent to all valid members. The problem with DEP is that the forward and backward secrecy depends on how often the DEK is updated. Therefore, if the DEK is changed infrequently, newcomers or ex-members can decrypt past/current messages easily.

Cipher-sequence (CS) [8] is another decentralized secure multicast framework that is built over a multicast tree. Every non-leaf node in the tree is assigned a cryptography function. When a message is sent out from a non-leaf node, it is encrypted by the assigned cryptographic function of that node. Thus, a message sent from the root of the tree to a member is encrypted by all nodes on the path. The encryptions performed by intermediate nodes form a cipher-sequence. Each leaf node contains members of a subgroup, all of whom share the same reverse function to the cipher-sequence. Since different paths generate different cipher-sequences, the reverse function for each subgroup is different. When a member joins or leaves a subgroup, the last inner node on the path closest to the subgroup's members changes its encryption function; thus, a new cipher-sequence and a corresponding reverse function are generated for that subgroup. Although this framework is secure and scalable for message delivery, it still requires a centralized key distribution center to understand the topology, assign encryption functions to non-leaf nodes, and compute reverse functions for subgroup members.

It is widely recognized that [7] was the first paper to discuss the possibility of applying proxy cryptography to secure multicast communications. The approach uses a similar network model to that of cipher-sequence; however, instead of assigning cryptographic functions to nodes, it assigns cryptographic keys. A key control component associated with the group controller generates encryption and decryption keys for the sender, receiver, and all intermediate transformation nodes (proxies). When a node joins the multicast tree, the group controller splits the decryption key of the new-comer's parent node into an encryption key and a decryption key. The new encryption key is kept by the parent node for further message transformation. Meanwhile, the new decryption key is sent to the new-comer to decrypt confidential messages. This approach, however, still requires a controller to handle key distribution, so it has similar drawbacks and limitations to the cipher-sequence approach.

Our work differs from previous studies in three ways: (1) We do not require a centralized key control component to distribute keys. Encryption keys are generated by the transformation nodes themselves. (2) The decryption keys for message receivers are obtained by running the proposed key composition protocol, which should be done in roughly a single round trip time. (3) Since the key distribution process is totally decentralized and distributed, the proposed protocol can be used to construct a generic security service for multicast networks that can be shared by different communication groups.

3. Proxy cryptography and key composition

Proxy cryptography [12] comprises encryption and digital signature schemes. In secure multicast, we only leverage the encryption part of the technique. Proxy encryption allows a semi-trusted proxy to transform a ciphertext from Alice into a ciphertext for Bob without seeing the underlying plaintext. The proxy encryption scheme can be implemented using the ElGamal cryptosystem in which two public parameters, p and g, are shared by all users, where p is a prime of the form $2q+1$ and g is a generator in $\mathbb{Z}_p^*$. The sender, A, randomly selects a secret key x in $\mathbb{Z}_q$ and releases its public key as $g^x \bmod p$. The message m is then encrypted with a randomly selected secret parameter $k \in \mathbb{Z}_q$ and sent in the form of two cipher values (c_1, c_2), where $c_1 = g^k \pmod p$ and $c_2 = m((g^x)^k) \pmod p$. Decryption just reverses the process. Therefore, it is easy for a node that knows $-x$ to recover the original plain-text by computing $m = c_2/c_1^x \pmod p$.

To implement proxy encryption using the above example, we assume that the message receiver B has a secret key y, and an intermediate proxy P between A and B has a transformation key $(y-x)$. Then, the intermediate proxy P transforms the cipher values (c_1, c_2) from A by computing $c_2' = c_2 \cdot c_1^{(y-x)}$ and sends (c_1, c_2') as the new ciphertext to B. Thus, B can extract the original message m from the transformed ciphertext with its own secret key y. To improve flexibility, the single proxy mechanism can be extended to a multiple proxies mechanism, as shown in Fig. 1. Assume there is a sender A, a receiver B, and n intermediate proxies P_i $(i = \{1, 2, \ldots, n\})$ on the path between A and B. To deliver a message from A to B, the message path will be A–P_1–P_2–$\cdots$–P_n—B. The key distribution center first assigns secret keys k_0 and k_n to A and B, respectively. Then, it randomly generates $k_1, k_2, \ldots, k_{n-1}$ and computes proxy keys k_{P_i} as $k_i - k_{i-1}$ for each proxy on the path. *Note that only the k_{P_i} $(i = \{1, 2, \ldots, n-1\})$ keys are sent to the proxies. The k_i keys are only used to compute the final proxy keys, so they are not sent to the proxies.*

The concept of chaining proxies can be used to construct a secure multicast framework easily. In a multicast network like that in Fig. 2, the key distribution center (KDC) first assigns secret keys k_s and k_{R_1}–k_{R_4} to node S and R_1–R_4, respectively. Then, it computes proxy keys according to the network topology. For example, the proxy nodes P_1, P_3, and P_4 are assigned proxy keys $k_1 - k_s$, $k_{R_1} - k_1$, and $k_{R_2} - k_1$, respectively. When a member joins or leaves, the proxy closest to the event has to notify the KDC to perform a re-keying operation by updating the secret keys of the proxy and all the members directly connected to it.

Since the above framework requires a centralized key distribution center, it has several drawbacks. First, its scalability is restricted by the single key server or cluster, which could be overloaded by a large number of member join/leave requests. Second, it is common for a centralized service to encounter the

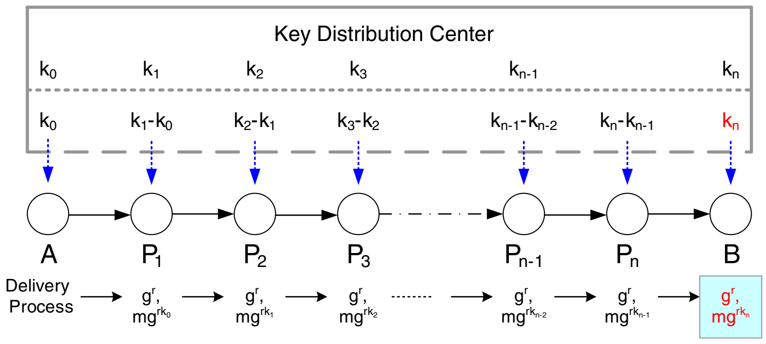

Fig. 1. An example of chaining proxies. The key distribution center generates cryptographic keys for the sender, the proxies, and the receiver. On receipt of the transformed message, the receiver can decrypt the ciphertext with its own secret key.

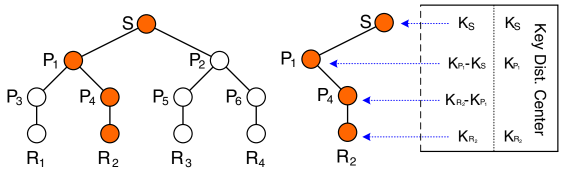

Fig. 2. An example of applying chaining proxies on a source-based multicast tree. Node S is the sender, nodes P_1–P_6 are multicast routers, and nodes R_1–R_4 are groups of receivers.

single point of failure problem. Third, since the secret keys for nodes are assigned by the key distribution center, the latter must be well protected and totally trusted. Furthermore, if there is a network dynamics issue, such as a proxy shutdown or a crash, it is hard for a single point to monitor and handle the whole network topology.

To eliminate the key distribution center, we propose a novel technique of key composition for proxy encryption. The basic idea is simple: *all stable nodes, including the sender and the proxies, decide their own secret keys. When a receiver joins the group, it obtains the decryption key by using the proposed key composition protocol.* Suppose there is a sender A, a receiver B, and the proxies P_1–P_n sit on the path between A and B. The sender A chooses a secret key k_0 and each proxy P_i ($i = \{1, 2, \ldots, n\}$) chooses a secret key k_i. When the ciphertext $\{c_1, c_2\} = \{g^r, m \cdot g^{rk_0}\}$ of a message m sent from A finally reaches B, the encrypted and transformed message will be in the form of

$$\{c_1, c_2\} = \{g^r, m \cdot g^{r(k_0+k_1+k_2+\cdots+k_n)}\},$$

where r is the random ephemeral key. The reason for this form is that when an arbitrary proxy P_i receives the ciphertext $\{c_1, c_2\}$ from the previous node, it always computes the new $\{c'_1, c'_2\}$ by $c'_1 = c_1$, $c'_2 = c_2 \cdot c_1^{k_i}$ and then forwards the result $\{c'_1, c'_2\}$ to the next node. Thus, to decrypt the message, the receiver must know the decryption key with a value k_c of $(k_0 + k_1 + k_2 + \cdots + k_n)$. We also call k_c the *composed key* for receiver B, since all nodes on the path from the sender to the receiver contribute to the secret key. In Fig. 3, four different message paths are extracted from the network shown in Fig. 2. As the value of a composed key is contributed by all nodes on the message path, the composed key for each message path is different. Members that share the same message path have the same composed key; hence, they form a subgroup. Since the composed key varies according to the message path, we also call it a *path key*. In the example

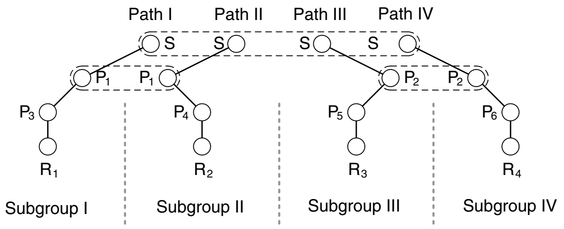

Fig. 3. The paths extracted from the network in Fig. 2. Each subgroup has a different composed key because the nodes on each message path are different.

in Fig. 3, suppose that node S has a secret key k_0 and node P_i has a secret key k_i; then, the composed keys for R_1, R_2, R_3, and R_4 are $(k_0 + k_1 + k_3)$, $(k_0 + k_1 + k_4)$, $(k_0 + k_2 + k_5)$, and $(k_0 + k_2 + k_6)$, respectively. With these observations, our problem is reduced to *"How can we securely generate and update the composed key for the group members in an efficient and distributed manner?"*

4. The framework

In this section, we explain the construction of our framework. First, we describe the roles and the assumptions used in this paper. Second, we define the operations and notations used in our framework. Then, the framework is constructed with essential group operations, namely, members joining and leaving, re-keying, and message delivery. Finally, we propose algorithms for handling network dynamics, such as topology changes and node failures.

There are three kinds of role in our framework: the sender, the proxy, and the receiver. The sender delivers messages to the receiver, while the proxies are responsible for transforming a received ciphertext into a new ciphertext that can be decrypted with a different secret key. We make the following assumptions.

- We assume that *both the sender and the receivers are trustworthy*. The definition of a receiver is a node that is authenticated by the sender and allowed to join the multicast network. Since a receiver can always access plain-text messages, we trust the receivers in our framework. The issue of leakage of confidential messages by receivers is not within the scope of this paper.
- We assume that *the proxies are only partially trusted*. That is, we trust a proxy to transform a received ciphertext and forward it to the next

node correctly. Since we use proxy encryption, the proxies cannot decipher the original plaintext message concealed in the received ciphertext. Thus, there is no concern about leakage of confidential messages by proxies. Note that a proxy can also be a receiver. In this special case like a receiver, the proxy is totally trusted.
- We assume that *a proxy knows about other proxies within a distance of at least two hops*. This is a requirement of dynamic networks. When the state of a proxy changes from on-line to off-line or vice versa, it should announce its new state to all its neighbors within at least two hops.
- We assume that *a secure channel can be established between any two nodes*. This can be done by using either a symmetric key cryptographic system or a public key cryptographic system. Although a secure channel between two nodes is sometimes required, the use of such channels should be kept to a minimum since they impact on the performance of key distribution, key updating, and message delivery.

Our framework is constructed over a multicast tree network. Thus, we also assume that there is already a source-based multicast tree network. As it is built on an overlay network, there should be no asymmetric logical link in the multicast tree network. However, messages exchanged between two adjacent overlay nodes may be transmitted using asymmetric physical links. Although asymmetric physical links do not affect the operation of the proposed protocol, it may affect the performance of communications between overlay nodes. Thus, the asymmetric physical links between two adjacent overlay nodes should be minimized. Readers may refer to [15] for optimizing the asymmetric link performance during multicast tree construction.

4.1. Operations and notations

Given two arbitrary keys, k_1 and k_2, the primitive operations used in our framework are the key compose operation $\mathsf{COMP}(k_1, k_2)$ and the key decompose operation $\mathsf{DECOMP}(k_1, k_2)$. The two operations are defined as

$$\mathsf{COMP}(k_1, k_2) = k_1 + k_2$$

and

$$\mathsf{DECOMP}(k_1, k_2) = k_1 - k_2,$$

respectively. In the key compose operation, a node adds its own secret key to a composed key, which is usually forwarded to the next node securely. The first key composition operation is always initiated by the sender. Before the composed key reaches the receiver, it is called a *partially composed key* (PCK). To generate the correct key for a receiver, each node on the path between the sender and receiver iteratively incorporates its own secret key into the PCK. This chain key composition operation is called the key composition process. The notations used in our framework are listed in Table 1.

4.2. The key composition protocol: joining a group

The key composition protocol (KCP) is initiated when a newcomer, R, requests to join a group. First, R has to enter the network by sending a network join request to the closest proxy, which then acts as the parent proxy of R. After R is admitted, the join request is forwarded along the upstream path to the root node. If the sender approves the request, it sends a session ID SID and a randomly generated session key, k_{SID}, to R securely via a direct connection or an indirect relayed channel. At the same time, the sender initiates a key composition process

Table 1
Notations used in the key composition process

Notation	Meaning
R	The message receiver
S	The message sender
SID	Session ID. The SID is used by a receiver to identify the key composition process
P_i	The intermediate proxy i on the message path that transforms ciphertexts during the message delivery process
k_i	The secret key of proxy P_i
k_0	The secret key of sender S
k_{SID}	A randomly generated session key for the key composition process, which is identified by an SID
PCK	Partially composed key, an incomplete key during the key composition process
CCK	Complete composed key. The final result of a key composition process

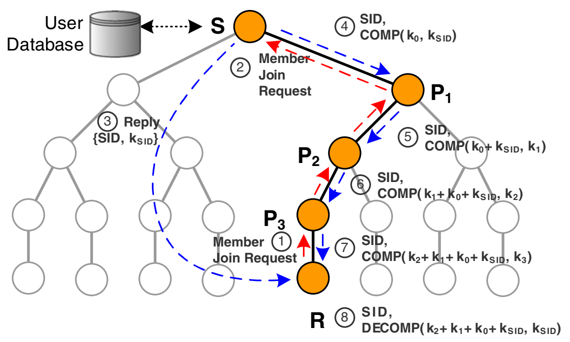
User Database
S
④ SID, COMP(k_0, k_{SID})
② Member Join Request
P_1
③ Reply {SID, k_{SID}}
P_2
⑤ SID, COMP(k_0+ k_{SID}, k_1)
⑥ SID, COMP(k_1+ k_0+ k_{SID}, k_2)
P_3
Member ① Join Request
⑦ SID, COMP(k_2+ k_1+ k_0+ k_{SID}, k_3)
R ⑧ SID, DECOMP(k_2+ k_1+ k_0+ k_{SID}, k_{SID})

Fig. 4. An example of joining a group. Node S is the sender, nodes P_1, P_2, P_3 are the proxies, and node R is the newcomer.

by sending a partially composed key (PCK) of COMP(k_0,k_{SID}) to the first proxy securely. The k_0 key is the secret key of the sender. When a proxy P_i receives a PCK, it performs the COMP(received-PCK,k_i) operation and forwards the new PCK to the next node securely. On receipt of the PCK, the receiver runs the DECOMP(received-PCK,k_{SID}) operation to obtain the final key, which is the complete composed key (CCK). Note that during the key composition process, the composed PCK is sent with an SID to guarantee that the newcomer can properly decompose the corresponding k_{SID} and obtain the final CCK. Also, when the parent proxy of the newcomer R receives a CCK for R, a re-keying operation must be performed before computing and returning the new CCK to R. This operation is discussed in the next subsection, and an example of a newcomer joining a group is illustrated in Fig. 4.

The success of the key composition process depends on whether the sender has delivered the k_{SID} to the receiver. The sender can reject sending a k_{SID} key to an unknown client. Thus, access control can be applied when the sender receives a request from a newcomer. In our framework, the generation of k_{SID} and access control can be offloaded to other trusted proxies. We discuss load-sharing with trusted proxies in Section 5.1.

4.3. Leaving a group and re-keying

Leaving a group is much easier than joining, as a member just needs to notify its parent proxy. The latter then performs a re-keying operation to change its own key, and sends the updated key to all members, except the departing member. Two problems in secure multicast are maintaining forward secrecy and backward secrecy. Maintaining the former means that a member that has left should not be able to read future confidential messages, while latter means that a newcomer should not be able to read previous confidential messages. Thus, when the membership of multicast group changes due to a member joining or leaving, a re-keying operation is always performed.

In our framework, a re-keying operation can be confined to a subgroup. Only the parent proxy and the subgroup members directly connected to the proxy need to perform the re-keying operation. Suppose a proxy has a key, k. To perform the re-keying operation, the proxy first chooses a new key, k', and computes $\Delta k = k' - k$. Then, it sends the result of Δk to all connected subgroup members securely. On receipt of a Δk from the parent proxy, a member updates its own key by COMP(original-CCK,Δk). Future ciphertexts can then be decrypted using the new CCK. A non-leaf node (e.g., the root node or any non-leaf proxies) may also perform a re-keying operation. In this case, the non-leaf node uses the same re-keying methodology as subgroups. Then, the Δk is multicast to all group members using the usual message delivery process.

The re-keying operation described above is quite straightforward. The cost of delivering key updates to each subgroup member is O(n), where n is the number of valid subgroup members. This, however, restricts the scalability of the subgroup. To solve the problem, we can adopt either distributed or centralized group key management mechanisms, such as TGDH [10] or LKH [3,4], to improve the re-keying performance of subgroups. Take the LKH approach as an example. A leaf proxy, which is the proxy closest to subgroup members, acts as a subgroup controller and maintains a logical key tree for all directly connected subgroup members. When a member joins or leaves the subgroup, the proxy updates the keys in the logical tree and then multicasts the encrypted Δk along with updated logical keys to all subgroup members. The LKH approach is efficient because it only requires O(1) multicasts and O(log(n)) computations to update the secret keys of all subgroup members.

4.4. The message delivery process

The message delivery process follows the nodes on a message path, as shown in Fig. 3. During the process, a message is encrypted by the sender and then transformed by each node on the path. A transformation is only performed as a message is leaving a node. Note that a message can only be encrypted/transformed once by the same node, which means there is no loop during the delivery process. Given the ElGamal encryption algorithm $\mathsf{Enc}_k(\cdot)$ and the decryption algorithm $\mathsf{Dec}_k(\cdot)$ with an arbitrary secret key k, the message delivery process on Path-II in Fig. 3 is as follows:

1. The sender S encrypts the plain-text message m with its own secret key k_0 and sends out $\mathsf{Enc}_{k_0}(m)$.
2. The transformation node P_1 encrypts the received message with its own secret key k_1 and sends out $\mathsf{Enc}_{k_1}(\mathsf{Enc}_{k_0}(m))$.
3. The transformation node P_4 encrypts the received message with its own secret key k_4 and sends out $\mathsf{Enc}_{k_4}(\mathsf{Enc}_{k_1+k_0}(m))$.

Finally, the receiver gets an encrypted message $\mathsf{Enc}_{k_4+k_1+k_0}(m)$, which can be decrypted using the previously obtained CCK of $k_0 + k_1 + k_4$.

When deploying proxy encryption or similar techniques in a secure multicast network, we use a "hybrid encryption" [16] technique to improve the overall performance. These techniques are mainly used to deliver key encryption keys (KEKs). In our scheme, the sender randomly generates a symmetric secret key k_r for a symmetric encryption algorithm $\mathsf{Enc}'_k(\cdot)$ and uses the key to encrypt the to-be-delivered message, m, as $\mathsf{Enc}'_{k_r}(m)$. During the message delivery process, k_r and $\mathsf{Enc}'_{k_r}(m)$ are sent together. However, the symmetric secret key k_r is encrypted and transformed by all the nodes that it passes. On receipt of the message, the receiver can decrypt k_r with its own CCK and then use k_r to decrypt the message m. In summary, the hybrid encryption technique uses the subgroup key, which acts as a key encryption key (KEK), to deliver a symmetric secret key so that a receiver can decrypt the encrypted message. The symmetric secret key should be changed periodically to guarantee both forward and backward secrecy.

4.5. Handling network dynamics

In a dynamic network, a proxy may join or leave the multicast tree network at any time. A proxy may also crash due to unexpected fatal errors. Such events make it necessary to modify the multicast tree's topology, which affects the path keys of related receivers. Since the receivers do not know the exact key of the affected proxy, the only way to update the path keys is to ask all receivers to run the key composition protocol again. This creates a substantial extra workload for both the sender and the network, especially when the number of affected receivers is large. To prevent this situation, we propose solutions that handle the network dynamics by only altering the keys of nodes close to the problematic proxy.

Assume there is already a working secure multicast network, and a new proxy can join the network as either a leaf proxy or a non-leaf proxy. Joining as a leaf proxy is fairly easy, since the new proxy only needs to randomly generate a secret key for itself and then find the closest available parent proxy to join the network. However, to join as a non-leaf proxy, the new proxy must first find the closest available parent proxy and a child proxy of the parent proxy. Then, the newcomer has to "insert" itself on the link between the child proxy and the parent proxy. The new proxy does not make the decision about its own secret key. Instead, the key is randomly generated by the child proxy and assigned to the new proxy. Assume that the child proxy randomly generates a secret key k_x to the new proxy. At the same time, the child must alter its own secret key, k_{child}, by $\mathsf{DECOMP}(k_{\mathsf{child}}, k_x)$. The two ways for a proxy to join a network are illustrated in Fig. 5. In the Part I of Fig. 5, a new proxy P_x may join as a leaf proxy under proxy P_3 or as a non-leaf proxy under the sender S. As shown in Part II-1, P_x can decide its own secret key as k_x, since joining as a leaf proxy does not affect any receivers. In the Part II-2, joining between the link of S and P_1 may affect the path keys of R_1 and R_2. Thus, the secret key of P_x is assigned by P_1 as k_x securely; then the secret key of P_1 is changed to $k_1 - k_x$.

Unlike membership operations, proxy leaving is more complex than proxy joining. When a proxy crashes or is off-line, the link between the sender and several receivers that passes the problematic proxy is broken. Thus, the proxies close to the problematic node should be responsible for repairing the topology. Here we make the following additional assumption: when a proxy joins a network, it sends information about its secret key to its parent and its children securely as $k_P + k_x$ and $k_P - k_x$, respectively. (k_P is the secret key of the proxy and k_x is a randomly generated value.) When the proxy crashes or is off-line, its parent node chooses one of its children to replace the problematic node. Then, the parent node sends $k_P + k_x$ to the chosen node securely. The secret key of the chosen node, k_{chosen}, is then set to $\mathsf{COMP}(k_{\mathsf{chosen}}, k_P)$. Fig. 6 shows the procedures for handling proxy leaving. Assume

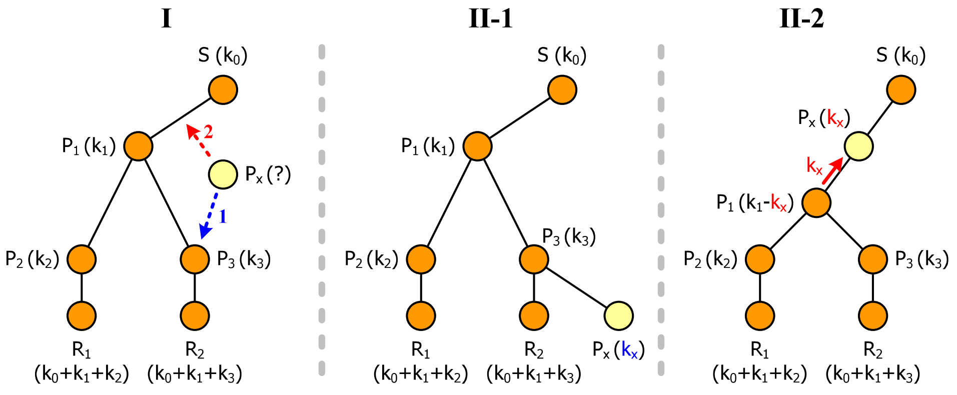

Fig. 5. Procedures to handle a newly joined proxy in a dynamic network.

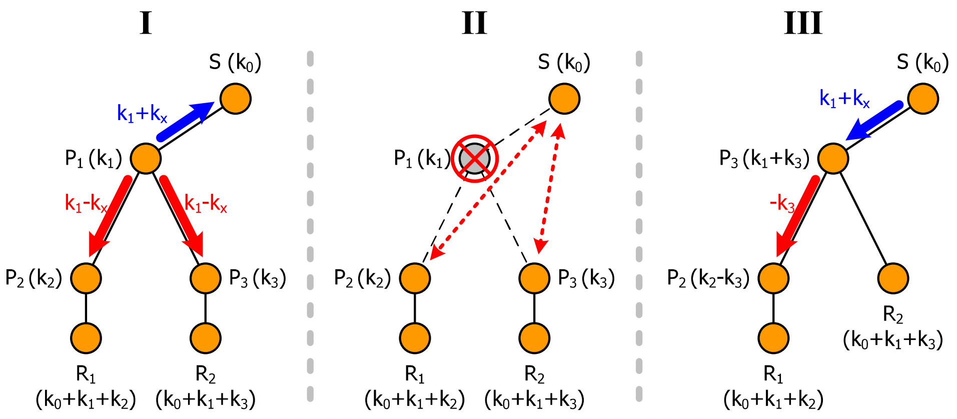

Fig. 6. Procedures for handling a crashed or off-line proxy in a dynamic network.

that the proxy P_1 will be off-line later. In Part I, P_1 sends secret key information $k_1 + k_x$ and $k_1 - k_x$ to its parent and children, respectively. When P_1 is off-line, the parent node S chooses node P_3 to replace P_1 and sends $k_1 + k_x$ to P_3; then, P_3 sets its own secret key to $k_1 + k_3$.

As the receivers' secret keys do not need to be updated with information about proxies joining or leaving, after a join or leave operation has been completed successfully, the receivers can decrypt encrypted messages as usual.

An important task for handling network dynamics is to detect proxy failures. An off-line node must broadcast the event to all its adjacent beforehand for them to prepare well for the possible topology changes. However, as a failed or a crashed proxy is mostly unable to complete the sign-out process, there should be some alternatives to detect failures of neighborhood proxies. While the failure of a node can be corrected locally, a simple way to detect unexpected node failures is to use *heartbeat messages*. Since a proxy knows all the adjacent children proxies, it can always periodically send "ping" messages to all these children and then wait for the corresponding "pong" messages. Thus, a node failure can be identified within at most the interval of two "ping" messages. Although heartbeat messages can detect unavailable children, it cannot tell whether the problem is on the child itself or is caused by link failures. No matter what the problem is, on detecting a child failure, the node will always try to contact the grandchildren under the problematic child proxy and try to pick one of the grandchildren to replace the possible crashed proxy. This also isolates only the problematic proxy. Then, an isolated proxy can detect link failures by both (1) it does not receive heartbeat messages from its parent and (2) it also does not receive notifications of parent crash from its grandparent. If a link failure is detected by an isolated node, the node can reenter the network by initiating a new network join request.

It is also worth noting that when a left proxy comes back, it cannot be placed at its original location before it leaves. Such a proxy is treated as a new comer and it must connect to the network as a leaf member or a leaf proxy. The reason for this design is to prevent the tree topology from oscillations caused by frequently joining or leaving of unstable nodes.

5. Evaluations and discussions

We now evaluate and discuss several aspects of the proposed framework. First, we discuss the possibility of sharing the load on the root node. Then, we discuss the security of our framework in detail. We also compare the performance of our framework with that of similar secure multicast frameworks.

5.1. Load-sharing with trusted proxy nodes

In our framework, a receiver initiates the network join process by forwarding a "member join request" to the sender. As a result, all the join requests from potential members are gathered at the root node. However, this may increase the load on the root node substantially and hence reduce the overall performance of the framework. To solve the problem, the loads induced by member join requests should be shared among several trusted proxies.

To do this, a root node must choose some trusted proxies. A selected proxy initiates the key composition protocol to obtain the CCK between the root and the parent proxy of the selected proxy. After successfully completing the key composition process, the selected proxy becomes a *delegated proxy* and handles future join requests generated by all downstream members. On intercepting a join request, just like a root node, the delegated proxy initiates the key composition process. The keys used for the key composition process are the previously obtained CCK plus the secret key of the delegated proxy and a randomly generated session key. An example of a running delegated proxy is illustrated in Fig. 7. In the figure, the proxy P_2 is chosen as a delegated proxy. Thus, it has to obtain the CCK between the root S and its parent proxy P_1, i.e., $k_0 + k_1$. A newcomer R joins the network as usual and initiates a member join request, which is sent to the root. However, the request does not reach the root, as it is intercepted by P_2. The remainder

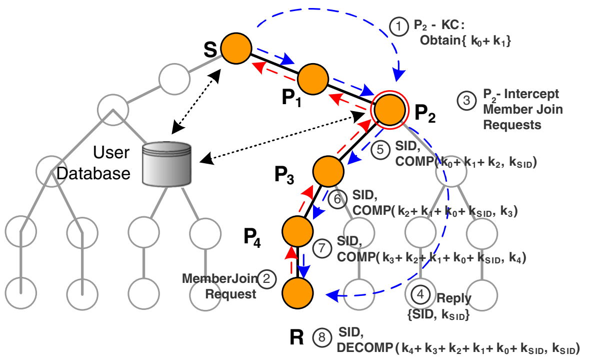
S
P1
P2
P3
P4
R
User Database
1 P2 - KC: Obtain{ k0+ k1}
2 MemberJoin Request
3 P2- Intercept Member Join Requests
4 Reply {SID, kSID}
5 SID, COMP(k0+ k1+ k2, kSID)
6 SID, COMP(k2+ k1+ k0+ kSID, k3)
7 SID, COMP(k3+ k2+ k1+ k0+ kSID, k4)
8 SID, DECOMP(k4+ k3+ k2+ k1+ k0+ kSID, kSID)

Fig. 7. An example of delegating access control to proxy nodes.

of the key composition process is the same as in the original (proposed) framework.

It should be noted that the delegated proxy must be trustworthy because it has the key to decrypt ciphertexts sent from the root. Besides, since a delegated proxy is responsible for access control, it should be able to access or at least query the membership database. The design of proxy delegation in our framework is inherently reliable. Once it is detected that a delegated proxy has crashed or is off-line, network join requests can simply be forwarded to the root node after the topology has been repaired. The root node can then choose a new delegated proxy.

5.2. Security analysis

In this section, we focus on several security issues. First, we discuss the common criteria that a secure multicast framework must possess. Then, we describe the use of ElGamal-based proxy cryptography for message delivery. We also examine the strength of the key composition process in terms of security. Finally, we discuss the problem of proxy compromise. The common criteria that a secure multicast framework must possess are: guaranteed of forward/backward secrecy, the ability to prevent collusion, and the "containment" property. In our framework, both forward and backward secrecy are achieved by re-keying at the parent proxy nodes. When a newcomer joins, or an existing member leaves, the parent proxy node must change its secret key and update the decryption keys of members under its control. With regard to collusion, as the decryption keys of members under different proxies are isolated because of different message paths, it is hard for those members to collude. Also, since subgroups are isolated, if one is compromised, it does not affect the whole multicast network. Hence, the framework can limit damage to one part of the network.

Our framework delivers messages using ElGamal-based proxy cryptography. Suppose an encrypted message $\{c_{(1,0)}, c_{(2,0)}\} = \{g^r, m \cdot g^{rk_0}\}$ sent from the root node is being delivered via a message path containing n proxies $P_1, P_2, \ldots, P_n$, and each proxy has a secret key k_i ($1 \leqslant i \leqslant n$). A message transformed by the kth proxy P_k is denoted as $\{c_{(1,k)}, c_{(2,k)}\}$. When a transformed message $\{c_{(1,j-1)}, c_{(2,j-1)}\}$ sent from the $(j-1)$th proxy passes the jth proxy P_j, the latter transforms the received message to another ciphertext $\{c_{(1,j)}, c_{(2,j)}\}$ by $\{c_{(1,j)}, c_{(2,j)}\} = \{c_{(1,j-1)}, c_{(2,j-1)} \cdot c_{(1,j-1)}^{k_j}\}$. In this scenario, an adversary could eavesdrop on network traffic of proxy P_j and know about $\{c_{(1,j)}, c_{(2,j)}\}$ and $\{c_{(1,j-1)}, c_{(2,j-1)}\}$. However, to break the secret key of P_j, the adversary has to solve $c_{(2,j)} = c_{(2,j-1)} \cdot c_{(1,j-1)}^{k_j}$, which is equivalent to solving the discrete logarithm problem (DLP); that is, given a prime number p, a group generator g, and $y = g^x \pmod{p}$, find x. The security of message delivery is thus guaranteed, since it is infeasible to compute the DLP, especially when p is large. It is recognized that the ElGamal cryptosystem is only secure against CPA (chosen plain-text attacks) [17], and cannot resist chosen ciphertext attacks (CCA). However, this is not a problem in our framework because proxies only transform passed ciphertexts by encrypting them again. Therefore, attackers cannot use these proxies as oracles to attack the ElGamal cryptosystem.

The security of the key composition process is guaranteed by the randomly generated session key k_{SID}, which is decided by the message sender. Since k_{SID} is added with real secret keys, the possibility of guessing k_{SID} and the real secret keys depends on the length of the secret keys, which vary from 256 bits to 2048 bits. Thus, it is impossible to determine either k_{SID} or the secret keys from the composed value of k_{SID} and the secret keys.

Two methods can be used to obtain the secret key of a proxy. One is to compromise the proxy, which allows an adversary to find out and access the secret key directly. The other method is to compromise both the parent proxy and a child proxy of the given proxy. In this case, if an adversary knows the secret keys of both the parent proxy and a child proxy, he can compute the difference between the PCKs of the parent and the child proxy during the key composition process. Extending such an attack, an adversary may compromise any two proxies on a message path and hence know the contributory secret keys of all proxies between the two compromised proxies. An example of compromising proxies on a message path is illustrated in Fig. 8. However, these kinds of compromise do not affect the security of the framework. Even if an adversary knows the secret keys of all the proxies on a message path, to decrypt the message sent out from a sender or received by the receivers, an adversary still has to compromise either the sender or one of the receivers to obtain the encryption key or the decryption CCK, respectively. The problem of compromising the sender or a receiver is not within the scope of this paper. If a sender or a receiver is compromised, the adversary does not need to know the key to decrypt the ciphertexts, since a decrypted message can be accessed directly via a compromised sender or receiver.

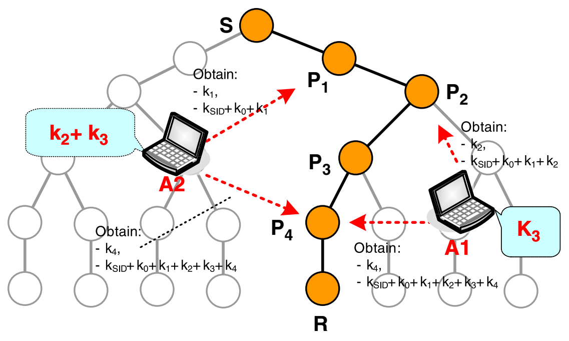

Fig. 8. An example of proxy compromises. There are two adversaries A_1 and A_2. The sender has a secret key k_0, and each proxy P_i has a secret key k_i. In this example, the adversary A_1 compromises the parent proxy P_2 and the child proxy P_4 of proxy P_3. Thus, it obtains the secret key of P_3. The adversary A_2 compromises the leading proxy P_1 and the trailing proxy P_4 of the message path and hence obtains $k_2 + k_3$.

5.3. Performance evaluation and comparison

The performance evaluation of our framework focuses on the storage, computation, and communications costs of the message delivery and key composition processes. The message delivery process is very efficient. When a message is being delivered to a receiver, each proxy passed only applies one encryption to transform the message. Receivers only need storage space for CCK, which is the decryption key. However, the root node and proxies require storage space for $2 + d$ keys, where d is the average number of branches for intermediate nodes, since

Table 2
Performance comparison with similar frameworks

	LKH	IOLUS	DEP	CS/PE	Ours
Number of keys per member	O(log(m))	1	2	1	*1*
Number of keys at key server	O(m)	2	O(s)	O(s)	–
Number of keys at intermediate nodes	–	d	2	1	$2+d$
Cost of re-keying	O(log(m))	O(d)	O(m)	O(n)	O(log(n))
Number of encryptions by a distributor	O(1)	O(d)	O(s)	O(1)	O(1)
Need to trust intermediate nodes	–	Yes	No	Partially	Partially

they need to store extra key information for the use if the network fails.

The cost of joining or leaving a group depends mainly on the re-keying cost. In our framework, the re-keying cost is O(log(n)), where n is the number of members in a subgroup. An additional cost of members joining is running the key composition process because a request must be sent from the receiver to the root node and then returned to the receiver. Thus, each proxy has to perform two forwarding operations. The process should take roughly a round trip time between the receiver and the root node.

We further evaluate the performance of our framework by comparing it with other similar frameworks. For the ease of comparison, we assume all these frameworks construct a balanced tree network. Table 2 shows the performance of all the compared frameworks. In the table, the number of group members is denoted by m, the number of subgroups for a scheme that uses a subgrouping technique is denoted by s, the number of members in a subgroup is denoted by n, and the average number of branches of intermediate nodes is denoted by d. The performance of our framework is shown in the column marked “Ours”.

6. Conclusions

In this paper, we have proposed a scalable and lightweight framework that provides secure multicast communications in a dynamic environment. The framework is based on a simple and novel technique called key composition, and can be applied to any source-based multicast tree network. With the properties of key composition and proxy encryption, our framework eliminates the need for key distribution centers used in proxy cryptography. In addition, our network's robustness against failure enables it to work in a distributed manner. The proposed framework is also efficient. Each node only needs to perform one encryption or transformation to guarantee the confidentiality of a message. The storage and computation costs of the key composition process are both constants. Therefore, our framework is scalable for large and dynamic multicast groups. For the above reasons, it can be easily applied in dynamic networks, such as peer-to-peer or overlay networks. The performance comparison also shows that our framework is more efficient than the compared frameworks.

References

[1] S.E. Deering, D.R. Cheriton, Multicast routing in datagram internetworks and extended lans, ACM Transactions on Computer Systems (TOCS) 8 (2) (1990) 85–110.

[2] S. Rafaeli, D. Hutchison, A survey of key management for secure group communication, ACM Computing Surveys (CSUR) 35 (3) (2003) 309–329.

[3] D. Wallner, E. Harder, R. Agee, Key management for multicast: issues and architectures, RFC 2627.

[4] C.K. Wong, M. Gouda, S.S. Lam, Secure group communications using key graphs, IEEE/ACM Transactions on Networking 8 (1) (2000) 16–30.

[5] S. Mittra, Iolus: a framework for scalable secure multicasting, in: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM SIGCOMM, 1997, pp. 277–288.

[6] L. Dondeti, S. Mukherjee, A. Samal, Scalable secure one-to-many group communication using dual encryption, Computer Communication 23 (17) (2000) 1681–1701.

[7] R. Mukherjee, J. Atwood, Proxy encryptions for secure multicast key management, in: Proceedings of the 28th Annual IEEE Conference on Local Computer Networks, IEEE LCN, 2003, pp. 377–384.

[8] R. Molva, A. Pannetrat, Scalable multicast security with dynamic recipient groups, ACM Transactions on Information and System Security 3 (3) (2000) 136–160.

[9] M. Steiner, G. Tsudik, M. Waidner, Diffie-Hellman key distribution extended to group communication, in: CCS'96: Proceedings of the 3rd ACM Conference on Computer and Communications Security, ACM Press, 1996, pp. 31–37.

[10] Y. Kim, A. Perrig, G. Tsudik, Simple and fault-tolerant key agreement for dynamic collaborative groups, in: CCS'00: Proceedings of the 7th ACM Conference on Computer and Communications Security, ACM Press, 2000, pp. 235–244.

[11] T. Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory 31 (1985) 469–472.
[12] M. Blaze, G. Bleumer, M. Strauss, Divertible protocols and atomic proxy cryptography, in: Proceedings of Advances in Cryptology: EUROCRYPT'98: International Conference on the Theory and Applications of Cryptology Techniques, LNCS, 1998, pp. 127–144.
[13] A. Ivan, Y. Dodis, Proxy cryptography revisited, in: Proceedings of Network and Distributed System Security Symposium, ISOC, 2003.
[14] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976) 654–655.
[15] S. Ramanathan, Multicast tree generation in networks with asymmetric links, IEEE/ACM Transactions on Networking 4 (4) (1996) 558–568.
[16] V. Shoup, A proposal for an ISO standard for public key encryption, Cryptology ePrint Archive, Report 2001/112, http://eprint.iacr.org/2001/112, September 2001.
[17] Y. Tsiounis, M. Yung, On the security of elgamal based encryption, in: PKC '98: Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptography, Springer-Verlag, London, 1998, pp. 117–134.

Chun-Ying Huang received his B.S. degree in Computer Science from National Taiwan Ocean University in 2000 and the M.S. degree in Computer Information Science from National Chiao-Tung University in 2002, respectively. He is now a Ph.D. candidate in the Department of Electrical Engineering, National Taiwan University. His current research interests focus on computer networks and network security, including key management, attack mitigation, intrusion detection, and traffic analysis.

Yun-Peng Chiu received his B.S. degree in Electrical Engineering from National Tsing-Hua University, Taiwan, in 1996, and the M.S. degree in Electrical Engineering from National Taiwan University, Taiwan, in 1998. He is now a Ph.D. candidate of Electrical Engineering at National Taiwan University, Taiwan. His current research interests include network security and cryptography. He is a student member of the Association for Computing Machinery, the Chinese Cryptology and Information Security Association, the Institute of Electrical and Electronic Engineers, and the Institute of Electronics, Information, and Communication Engineers.

Kuan-Ta Chen received his B.S. and M.S. in Computer Science from National Tsing-Hua University in 1998 and 2000, respectively. He received his Ph.D. in Electrical Engineering from National Taiwan University in 2006. Since then he joined the Institute of Information Science, Academia Sinica as an assistant research fellow. His research interests include multimedia networking, Internet measurement, network security, and entertainment networking. Much of his recent work focus on the analysis and design of networked entertainment systems, including traffic characterization, transport protocols, quality of service, human factors, and game cheat detections. He is a member of ACM and IEEE.

Chin-Laung Lei received his B.S. degree in Electrical Engineering from National Taiwan University in 1980, and his Ph.D. degree in Computer Science from the University of Texas at Austin in 1986. From 1986 to 1988, he was an assistant professor in the Computer and Information Science Department at the Ohio State University, Columbus, OH, USA. In 1988, he joined the faculty of the Department of Electrical Engineering, National Taiwan University, where he is now a professor. His current research interests include computer and network security, cryptography, parallel and distributed processing, design and analysis of algorithms, and operating system design. He is a member of the Institute of Electrical and Electronics Engineers and the Association for Computing Machinery.