Collusion-Resistant Multicast Key Distribution Based on Homomorphic One-Way Function Trees

Jing Liu and Bo Yang

***Abstract*— Providing security services for multicast, such as traffic integrity, authentication, and confidentiality requires securely distributing a group key to group receivers. In literature, this problem is called *multicast key distribution* (MKD). A famous MKD protocol — OFT (One-way Function Tree) has been found vulnerable to collusion attacks. Solutions to prevent these attacks have been proposed, but at the cost of a higher communication overhead than the original protocol. In this paper, we prove falsity of a recently-proposed necessary and sufficient condition for a collusion attack on the OFT protocol to exist by a counterexample and give a new necessary and sufficient condition for nonexistence of any type of collusion attack on it. We instantiate the general notion of OFT to obtain a particular type of cryptographic construction named *homomorphic one-way function tree* (HOFT). We propose two structure-preserving graph operations on HOFTs, tree product and tree blinding. One elegant quality possessed by HOFTs is that handling (adding, removing, or changing) leaf nodes in a HOFT can be achieved by using tree product without compromising its structure. We provide algorithms for handling leaf nodes in a HOFT. Employing HOFTs and related algorithms, we put forward a collusion-resistant MKD protocol without losing any communication efficiency compared to the original OFT protocol. We also prove the security of our MKD protocol in a symbolic security model.**

***Index Terms*— Multicast key distribution, One-way function tree, Homomorphism, Collusion**

I. INTRODUCTION

Many emerging group-oriented applications, for instance, IPTV, DVB (Digital Video Broadcast), videoconferences, interactive group games, collaborative applications, and so on all require a one-to-many or many-to-many group communication mechanism. One of the most efficient approaches to ensure confidentiality of group communications is employing a symmetric-key encryption scheme. But before the sender encrypts and transmits the data over a group communication channel to a group of privileged users, a shared key called *group key* must be

Manuscript received August 31, 2010; revised December 20, 2010; accepted April 08, 2011. This work was supported in part by the National Natural Science Foundation of China under Grant No. 60973134 and the Natural Science Foundation of Guangdong Province under Grants No. 10351806001000000.

Jing Liu is with the School of Information Science, and Technology and Guangdong Key Lab of Information Security and Technology, Sun Yat-Sen University, Guangzhou, 510006, China (e-mail: liujing3@mail.sysu.edu.cn).

Bo Yang is with College of Informatics, and College of Software, South China Agricultural University, Guangzhou, 510642, China (e-mail: byang@scau.edu.cn)

established among them. Group key establishment can be subdivided into *group key distribution* (GKD) and *group key exchange* (or *group key agreement*). Two parallel lines of research, commonly referred to as *broadcast encryption* (BE) [1] and *multicast key distribution* (MKD) (or multicast encryption), have been established to study the GKD problem. This paper only focuses on MKD protocols. In contrast with stateless receivers in BE protocols, each receiver in MKD protocols is *stateful*, which means that they are allowed to maintain a personal state and make use of previously learned keys for decrypting current transmissions. Generally speaking, for large dynamic groups, MKD protocols are often more efficient and scalable than BE protocols. Rather than tackling the general GKD problem as BE protocols, most MKD protocols aim to solve a more specific problem in the multicast encryption setting, called *immediate group rekeying*. To prevent a new member from decoding messages exchanged before it joins a group, a new group key must be distributed to the group when a new member joins. This security requirement is called *group backward secrecy* [2]. On the other hand, to prevent a departing member from continuing access to the group's communication (if it keeps receiving the messages), the key should be changed as soon as a member leaves. This security requirement is called *group forward secrecy* [2]. To provide both *group backward secrecy* and *group forward secrecy*, the group key must be updated upon every membership change and distributed to legitimate members. This process is referred to as *immediate group rekeying* in literature. Respectively, the rekeying process due to a joining membership change (resp. a departing membership change) is referred to as *join rekeying* (resp. *leave rekeying*). For large dynamic groups with frequent changes in membership, it is a big challenge to design a scalable MKD protocol.

Among all generic MKD protocols in which rekey messages are built using traditional cryptographic primitives (symmetric-key encryption and pseudorandom generators), a class of protocol called *tree-based* protocol [3],[4],[5] is the most efficient one to date in terms of communication overhead. They have a communication complexity of $O(\log_2 n)$ for a group size of n. A recent result by Micciancio et al. [6] has also confirmed that $\log_2 n$ is the optimal lower bound on the communication complexity of generic MKD protocols. The first tree-based MKD protocol is the *Logical Key Hierarchy* (LKH) [3],[4]. The OFT protocol [7],[5] halves the communication overhead of LKH in case of leaving rekeying by deriving its key tree in a bottom-up manner. However, Horng [8] showed that OFT is vulnerable to a particular kind of collusion attack. Soon after, Ku and Chen [9] found new types of collusion attacks, and they proposed an improved protocol to prevent any collusion attack. But leaving rekeying using their approach requires a communication complexity of $O((\log_2 n)^2+\log_2 n)$. Recently, Xu et al. [10] showed that all the known attacks on OFT can be subsumed by a generic collusion attack. They also derived a necessary and sufficient condition for such an attack to exist and further proposed a protocol to prevent collusion attacks while minimizing the average broadcast size of rekeying message. However their protocol requires a storage linear to the size of the key tree ($O(2n\text{-}1)$) and still has a much bigger broadcast size than the original OFT protocol.

In this paper, we prove falsity of Xu et al.'s necessary and sufficient condition for existence of a collusion attack on the OFT protocol by a counterexample and give a new necessary and sufficient condition. We introduce a new cryptographic construction named *homomorphic one-way function trees* (HOFT) by respectively substituting a homomorphic trapdoor function and a modular multiplication for the one-way function and the exclusive-or mixing function in the original one-way function trees. We propose two tree operations — tree product and tree

blinding — for HOFTs and prove that both are structure-preserving. Then, we provide algorithms for adding/removing/changing leaf nodes in a HOFT without compromising its structure (i.e., functional dependency between node secrets) by performing a tree product of the HOFT and a corresponding incremental tree. Utilizing HOFTs and related algorithms, we put forward a collusion-resistant MKD protocol without losing any communication efficiency compared to the original OFT protocol. In contrast, two existing solutions [9], [10] to improve the OFT protocol have to trade off communication efficiency for collusion resistance. We also prove the security of our protocol in a symbolic security model.

The remainder of this paper is organized as follows. Section II reviews related research results. In section III, we prove the falsity of Xu et al.'s necessary and sufficient condition for a collusion attack on OFT to exist by a counterexample. We give a new necessary and sufficient condition for nonexistence of an arbitrary type of collusion attack. In sections IV, we introduce a new cryptographic construction – Homomorphic OFT and related algorithms. Section V presents a collusion-resistant MKD protocol based on HOFTs and related algorithms. Section VI proves the security of our MKD protocol in a symbolic security model. Section VII gives a comparison between our protocol and other related protocols. Section VIII concludes this paper and gives some topics for future research.

II. RELATED RESEARCH

A. Tree-based MKD protocols

The *Logical Key Hierarchy* (LKH) protocol was independently proposed by Wong et al. [3] and Wallner et al. [4]. In the LKH protocol, each internal node in the key tree represents a key encryption key (KEK), each leaf node of the key tree is associated with a group member and the

root node represents the group key. A key associated with the internal node is shared by all members associated with its descendant leaf nodes. Every member is assigned to the keys along the path from its leaf to the root. When a member leaves the group, all the keys that the member knows should be changed. The key server generates new keys to replace those keys and sends the newly-generated keys encrypted with keys that the departing member does not have access to. If n represents the current number of members in a group and we consider a full and balanced binary tree, leave rekeying using LKH requires at least $2\log_2 n$ key encryptions and transmission by the key server. When a member joins, the key server creates a leaf node for it, and changes all the keys from this leaf node to the root. In addition to sending the newly-generated keys encrypted with the new member's leaf node key, the key server sends each new internal node key encrypted under the key it is replacing, to the other members. Join rekeying using LKH requires encryptions and transmission of $2\log_2 n$ keys by key server.

The OFT protocol was proposed by Sherman, Balenson and McGrew [5],[7]. Each internal node i of a key tree is associated with a node secret x_i, a blinded node secret y_i and a node key K_i. Let x_{2i} and x_{2i+1} denote the left child and right child of x_i, respectively. The node secret of the root node is the group key. There exist two different pseudorandom functions f and g. The function f is used to compute a corresponding blinded node secret from a node secret, i.e., $y_i = f(x_i)$; The function g is used to compute a corresponding node key from a node secret, i.e., $K_i = g(x_i)$. A node key rather than a node secret is used as a key encryption key (KEK) in a rekeying message. Unlike in the LKH protocol, the key server does not send each member those node secrets along the path from its associated leaf node to the root. Instead, it supplies each member with the blinded node secrets associated with the siblings of the nodes in its path to the root. Each member uses these blinded node secrets and its leaf node secret to compute the other node

secrets in its path to the root according to a functional relationship as below. A one-way function key tree is computed in a bottom-up manner using the pseudorandom functions f and a bitwise exclusive-or operation denoted by '⊕'. Specifically, each internal node secret x_i is computed by exclusive-oring y_{2i} with y_{2i+1}, i.e., $x_i = y_{2i} \oplus y_{2i+1}$. When a new member joins/leaves the group, all the node secrets in its path to the root must change (and therefore the corresponding blinded node secrets along this path also change). The key server sends the joining member the blinded node secrets it is entitled to, after encrypting them with this member's leaf node key. In addition, the key server encrypts each of those changed blinded node secrets under the corresponding sibling's node key, and sends them to old group members in the case of join rekeying or remaining members in the case of leave rekeying. Consider a full and balanced OFT with n members (after the join/leave). The key server needs to encrypt and send $2\log_2 n$ blinded node secrets when a member joins. It needs to encrypt and send $\log_2 n$ blinded node secrets when a member leaves.

B. Collusion attacks on the OFT protocol

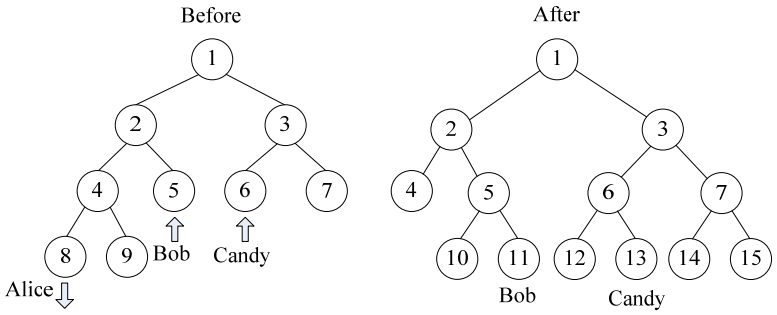

Fig. 1 Scenarios of collusion attacks on OFT

The functional dependency relationship among the node secrets in a one-way function tree allows leave rekeying using OFT to save half of communication cost as compared to that using LKH. However, the same relationship also renders it vulnerable to collusion attacks.

The first collusion attack on the OFT protocol attributes to Horng [8]. Referring to Figure 1, suppose that Alice, associated with node 8, leaves at time t_A, and later Candy joins the group at time t_C and is associated with node 13 (ignore Bob's joining for the time being). We use $x_{i[t_A,t_C]}$ to denote the node secret associated with node i in the time interval between t_A and t_C. Suppose that there are no changes in group membership between time t_A and t_C. Since after the eviction of Alice, x_3 is not changed until Candy joins, Alice holds its blinded version $y_{3[t_A,t_C]}$. Since x_2 is changed when Alice leaves, and then remains unchanged at least until Candy joins, Candy receives its blinded version $y_{2[t_A,t_C]}$ at the time of joining. Collectively knowing $y_{2[t_A,t_C]}$ and $y_{3[t_A,t_C]}$, Alice and Candy can collude to obtain the group key in the time interval [t_A, t_C] by computing $x_{1\,[t_A,t_C]} = y_{2[t_A,t_C]} \oplus y_{3[t_A,t_C]}$. Therefore, the OFT protocol fails to provide not only group forward secrecy against Alice but also group backward secrecy against Candy. Horng thus proposed two necessary conditions for a collusion attack to exist: (1) the two colluding nodes must be evicted and join at different subtree of the root; (2) no key update happens between time t_A and t_C. Later, Ku and Chen [9] showed that neither of these two conditions is necessary by proposing two new kinds of collusion attacks. Referring to Figure 1 again, the first kind of collusion attack can be described as follows. Suppose that Alice leaves at time t_A, and later Bob joins the group at time t_B and is associated with node 11. Also suppose that there are no changes in group membership between time t_A and t_B. For the same reason as above, Alice and Bob can collude to compute $x_{2[t_A,t_B]}$. Since x_3 remains unchanged during the time interval [t_A, t_B], Alice and Bob both hold its blinded version $y_{3[t_A,t_B]}$. Therefore, both can further compute the group key $x_{1[t_A,t_B]}$ by $x_{1\,[t_A,t_B]} = f(x_{2[t_A,t_B]}) \oplus y_{3[t_A,t_B]}$. This attack refutes Horng's necessary condition (1). The second kind of collusion attack given by them is described as follows. Suppose that Alice leaves

at time t_A, later Bob joins the group at time t_B, and Candy joins the group last at time t_C. We also assume that there are no changes in group membership not only between time t_A and t_B, but also between time t_B and t_C. After the eviction of Alice, x_3 is not changed until Candy joins the group. Therefore, Alice holds its blinded version $y_{3[t_A,t_C]}$ even after her eviction. Since x_2 is changed when Bob joins the group, and then remains unchanged at least until Candy joins the group, Candy receives its blinded version $y_{2[t_B,t_C]}$ at the time of joining. Collectively knowing $y_{3[t_A,t_C]}$ and $y_{2[t_B,t_C]}$, Alice and Candy can collude to compute the group key $x_{1[t_B,t_C]}$ (note that $[t_B,t_C] \subset [t_A,t_C]$). This attack refutes Horng's necessary condition (2).

C. Improvements on the OFT protocol

When a new member joins, it will be supplied with the blinded node secrets that were once used to compute the past group key. On the other hand, when a member leaves, it brings out the blinded node secrets that may be used to compute the future group key. It is thus possible for a pair of evicted member and joining member to combine their knowledge together to compute a valid group key between the time of eviction and that of joining. Therefore, it becomes reasonable to devise a solution to prevent collusion attacks either by preventing a departing member from bringing out any blinded node secrets that contain any information about the future group key or by supplying joining member with blinded node secrets that contain no information about the past group key. Each of the following two improvements on the OFT protocol is just aiming at one aspect to achieve collusion-resistance.

Ku and Chen improve the OFT protocol by changing all the keys known by a departing member. That is to say, when a member leaves, not only all the node secrets in its path to the root, but also all the blinded node secrets associated with the siblings of those nodes in that path

must be changed. The additional updates of node secrets increase the broadcast size by $(\log_2 n)^2$ keys. The key server needs to encrypt and send $(\log_2 n)^2+\log_2 n+1$ keys in total.

Xu et al. [10] observed that collusion between an evicted member and a joining member is not always possible and its success depends on a temporal relationship between them. It is not necessary to always change additional blinded node secrets as above unless a collusion attack is indeed possible. They proposed a stateful approach in which the key server tracks all evicted members and records all the knowledge held by them. Every time a new member joins, the key server checks against that knowledge to decide whether this joining member could have a successful collusion with any previous evicted member. For that purpose, their protocol has a storage requirement linear to the size of the key tree. The key server will not change additional blinded node secrets as Ku and Chen's protocol until a successful collusion is detected. Therefore it has a communication overhead lower than Ku and Chen's protocol, but still bigger than the original OFT protocol.

In their paper [10], Xu et al. put forward three propositions to support the correctness and security of their protocol. They first consider a generic collusion attack on the OFT protocol (depicted in Figure 2, and notice that this figure actually combines two different key trees respectively at t_A and t_C). Before introducing their propositions, let us get familiar with some notations to be used. Suppose that *A* leaves at time t_A and *C* joins at a later time t_C. Let *B*, *D*, *E*, and *F* respectively denote the subtrees rooted at nodes *L*, *R*, *R'*, and *R"*. Let t_{DMIN}, t_{EMIN}, and t_{FMIN} denote the time of the first group key update after t_A that happens in *D*, *E*, and *F*, respectively. Let t_{BMAX}, t_{EMAX}, and t_{FMAX} denote the time of the last group key update before t_C that happens in *B*, *E*, and *F*, respectively.

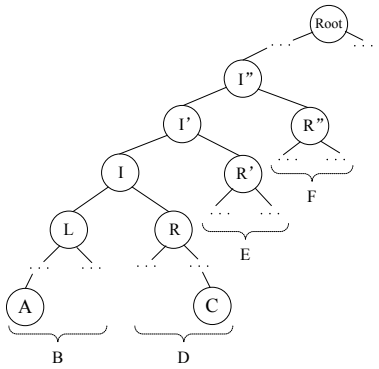

Fig. 2 A generic collusion attack on OFT

Xu's proposition 1**:** *For the OFT protocol, referring to Figure 2, the only node secrets that can be computed by A and C when colluding are:*

- x_I *in the time interval* $[t_{BMAX}, t_{DMIN}]$,
- $x_{I'}$ *in* $[t_{BMAX}, t_{DMIN}] \cap ([t_A, t_{EMIN}] \cup [t_{EMAX}, t_C])$,
- $x_{I''}$ *in* $[t_{BMAX}, t_{DMIN}] \cap ([t_A, t_{EMIN}] \cup [t_{EMAX}, t_C]) \cap ([t_A, t_{FMIN}] \cup [t_{FMAX}, t_C])$,

and so on, up to the root.

In fact, it is easy to verify that all kinds of collusion attacks presented in Section II-B are subsumed by this generic attack.

Xu's proposition 2**:** *A pair of colluding members A and C cannot compute any node secret which they are not supposed to know by the OFT protocol, if one of the following conditions holds*

- *A is removed after C joins.*
- *A and C both join.*
- *A and C are both removed.*

This proposition confirms that the above generic collusion attack is the only pattern of two-party collusion attack. The same authors also give the following sufficient and necessary condition for a collusion attack to exist.

Xu's proposition 3**:** *For the OFT protocol, an arbitrary collection of removed members and joining members can collude to compute some node secret not already known, if and only if the same node secret can be computed by a pair of members in the collection.*

Unfortunately, in their proof of this proposition, the authors claim that to compute a node secret not already known, the set of colluding members must already know both child blinded node secrets of it. This claim is wrong, since the colluding members may not know those child blinded node secrets at first, but can collude to compute them.

III. A NEW NECESSARY AND SUFFICIENT CONDITION

In this section, we first present an interesting counterexample that refutes the necessity of Xu's proposition 3, and then propose a new necessary and sufficient condition for the nonexistence of an arbitrary type of collusion attack on the OFT protocol. At last, we reveal that for OFT-based protocols, security against collusion attacks follows from security against collusion between an arbitrary pair of evicted member and later-joining member.

A. A counterexample

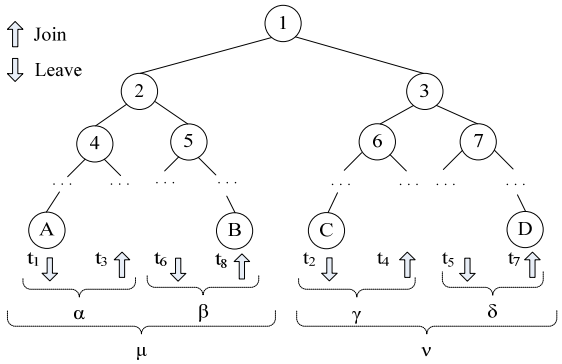

Fig. 3 A counterexample against Xu's proposition 3

We consider a collusion scenario depicted in Figure 3 (notice that this figure actually combines multiple key trees respectively at different time t_i (i =1, …, 7)). Suppose that Alice (A) and Colin (C) leave the group at time t_1 and t_2, respectively, and Dean (D) and Bob (B) join the group at time t_7 and t_8, respectively. It is assumed that the chronological order of t_1, t_2, … , and t_8 corresponds with the numerical order of their subscripts. Let α, β, γ, δ, μ, and ν denote the subtrees rooted at node 4, 5, 6, 7, 2, and 3, respectively. In addition to the above changes in group membership, there are changes at time t_3, t_4, t_5, and t_6, which happened in α, γ, δ, and β, respectively. Let $t^X_{\alpha MAX}$ denote the time of the last group key update before X joins the group that happens in α. Let $t^Y_{\beta MIN}$ denote the time of the first group key update after Y leaves the group that happens in β. Recall that x_v denotes the node secret associated with node v and y_v denotes the blinded version of it. Moreover, $x_{v\,[t1,\,t2]}$ denotes the value of node secret x_v in the time interval [t_1, t_2].

According to Xu's proposition 1, Alice and Bob can collude to compute x_2 in the time interval $[t^B_{\alpha MAX}, t^A_{\beta MIN}]$, i.e., $x_{2[t_3,t_6]}$; Colin and Dean can collude to compute x_3 in the time interval $[t^D_{\gamma MAX}, t^C_{\delta MIN}]$, i.e., $x_{3[t_4,t_5]}$. Thus, collectively knowing $x_{2[t_3,t_6]}$ and $x_{3[t_4,t_5]}$, Alice, Bob, Colin and Dean can collude

to compute $x_{1[t_4,t_5]}$. However, we shall show that each pair of evicted member and joining member cannot collude to compute $x_{1[t_4,t_5]}$.

According to Xu's proposition 1, all the node secrets that can be computed by Alice and Bob when colluding are:

- x_2 in the time interval $[t^B_{\alpha MAX}, t^A_{\beta MIN}]$, i.e., $x_{2[t_3,t_6]}$,

- x_1 in the time interval $[t^B_{\alpha MAX}, t^A_{\beta MIN}] \cap ([t_1, t^A_{\nu MIN}] \cup [t^B_{\nu MAX}, t_8])$, but evaluation of this formula results in $[t_3, t_6] \cap ([t_1,t_2] \cup [t_7,t_8]) = \phi$.

Thus, Alice and Bob cannot collude to compute $x_{1[t_4,t_5]}$. By the same argument, we can prove that for the rest of eviction-joining scenarios, i.e., the collusion between Colin and Dean, that between Alice and Dean, or that between Colin and Bob, $x_{1[t_4,t_5]}$ cannot be computed either. This counterexample thus falsifies the necessity of Xu's proposition 3.

B. A new necessary and sufficient condition

For OFT-based protocols, the functional dependency between node secrets follows an important result concerning collusion attacks as follows.

Theorem 1: *For a One-way Function Tree X with n legitimate members*, *an arbitrary collection of k (2 ⩽k⩽n) parties cannot collude to compute any node secret not already known* (*including the group key*)*, if and only if an arbitrary pair of parties cannot collude to compute any node secret not already known.*

Proof: For an arbitrary node secret x_i in a key tree X, we use x_{2i} and x_{2i+1} to denote its left child and right child respectively. The blinded version of x_i is denoted by y_i. The necessity is trivial. We prove sufficiency using induction over k. It is trivially true for k=2. Now suppose that it is

true for any arbitrary $k<m$ ($m<n$), we prove it is true for $k=m$ by contradiction. Suppose that there exists a key tree X_0 and a group of m parties (denoted by S) who can collude to compute a node secret $x_{i[a,b]}$ not already known, we divide S into an arbitrary sub-group S' of size m-1 and a remaining party m_i. According to the inductive hypothesis, an arbitrary collection of parties in S' cannot collude to compute any node secret not already known (including $x_{i[a,b]}$). On the other hand, $x_{i[a,b]}$ is unknown to m_i itself since $x_{i[a,b]}$ is unknown to the group S (including m_i). In this case, the only possible way for parties in S to produce a new node secret $x_{i[a,b]}$ on the one-way function tree X_0 is that some party m_i' in S' already knows $x_{2i[c,d]}$ ($[a,b]\subseteq[c,d]$) (or $x_{2i+1[e,f]}$ ($[a,b]\subseteq[e,f]$)) and m_i already knows $x_{2i+1[e,f]}$ ($[a,b]\subseteq[e,f]$) (or $x_{2i[c,d]}$ ($[a,b]\subseteq[c,d]$)) such that they can collude to compute $x_{i[a,b]}= f(x_{2i[c,d]})\oplus f(x_{2i+1[e,f]})$. But that is a contradiction to the initial assumption that an arbitrary pair of parties cannot collude to compute any node secret not already known. □

According to Theorem 1, preventing an arbitrary collusion attack on an OFT-based protocol (Notice that here we mean preventing any collection of parties from colluding to compute any not-already-known node secret on an OFT tree) is reduced to preventing collusion between an arbitrary pair of parties. Furthermore, Xu's proposition 2 confirms that the latter can be reduced to preventing collusion between an arbitrary pair of evicted member and later-joining member. Thus, if one is interested in analyzing the security of an OFT-based protocol against collusion attacks, it suffices to prove it secure against collusion between an arbitrary pair of evicted member and later-joining member only; immunity to an arbitrary collusion attack follows from this automatically! Thus, we have the following theorem.

Theorem 2: *An OFT-based protocol is secure against collusion attacks if it is secure against collusion between an arbitrary pair of evicted member and later-joining member.*

As we discussed in Section II-C, both Ku and Chen's improvement on OFT protocol and Xu's are meant to prevent collusion between an arbitrary pair of evicted member and later-joining member. It is easy to prove that both are secure against this type of collusion attack using Xu's proposition 1.

IV. HomoMorphic One-way functioN trees

A. Definition

Before we give the definition of a homomorphic one-way function tree, let's review some basic mathematical concepts. A group G with its operation "$*$" is denoted by $(G, *)$. Given two groups $(G, *)$ and $(H, \cdot)$, a *group homomorphism* from $(G, *)$ to $(H, \cdot)$ is a function $f: G \rightarrow H$ such that for all u and v in G, it holds that $f(u*v) = f(u) \cdot f(v)$. One can easily deduce that a group homomorphism f maps the identity element e_G of G to the identity element e_H of H, and maps inverses to inverses in the sense of $f(u^{-1}) = f(u)^{-1}$. According to this definition, the *Rabin function* [11] and the *RSA function* [12] are both homomorphic.

Since all nodes in an OFT are homogeneous (i.e., cryptographic keys), we choose to use self-homomorphism that maps an Abelian group G to G. If every node secret in an OFT X is an element of an Abelian group G, we say *X is defined over G*.

Definition 1 Homomorphic OFT — A *homomorphic OFT* (HOFT) over an Abelian group $(G, *)$ is a binary key tree that is computed using a self-homomorphic OWF f and the multiplicative operation "$*$" in a bottom-up manner as follows. For an arbitrary node secret x_i in a HOFT X, suppose that its left child and right child are denoted by x_{2i} and x_{2i+1} respectively, and we have $x_i = f(x_{2i}) * f(x_{2i+1})$.

B. Two structure-preserving operations on HOFTs

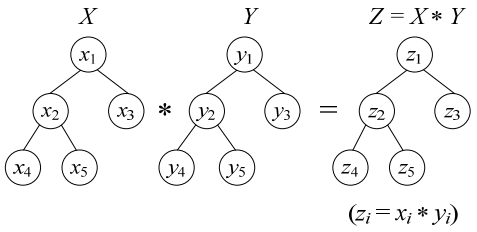

Fig. 4 Tree product

A binary operation (resp. unary operation) is said to be structure-preserving if the operation takes two HOFTs (resp. one HOFT) as inputs (resp. input) and outputs a HOFT. For convenience, we shall interchangeably use the same notation "x_i" or "y_i" to denote either a node itself or its associated node secret in Section IV.

Definition 2 Tree product — Given two arbitrary HOFTs X and Y, both defined over an Abelian group $(G, *)$, if X and Y are isomorphic, a tree product of X and Y, denoted by $X * Y$, is computed by multiplying their corresponding node secrets (see Figure 4).

Note that although we use the same notation "$*$" for both group operation and tree product, its meaning is context-evident.

Theorem 3: *Given two arbitrary isomorphic HOFTs X and Y, both defined over an Abelian group (G, *), the result of a tree product X * Y is also a HOFT.*

Proof: Let X and Y are two arbitrary HOFTs defined over an Abelian $(G, *)$, and $Z = X * Y$. We prove Z is also a HOFT. For an arbitrary node secret $z_i \in Z$, we have

$z_i = x_i * y_i$ (Definition 2)

$= (f(x_{2i}) * f(x_{2i+1})) * (f(y_{2i}) * f(y_{2i+1}))$ (Definition 1)

$= (f(x_{2i}) * f(y_{2i})) * (f(x_{2i+1}) * f(y_{2i+1}))$ ("$*$" is commutative and associative)

$= f(x_{2i} * y_{2i}) * f(x_{2i+1} * y_{2i+1})$ (f is homomorphic)

$= f(z_{2i}) * f(z_{2i+1})$ (Definition 2).

Thus, Z is a HOFT according to Definition 1. □

It follows that tree product is *structure-preserving*.

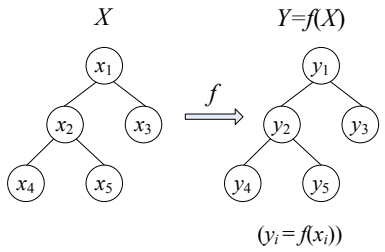

Fig. 5 Tree blinding

Definition 3 Tree blinding — For an arbitrary HOFT X defined over an Abelian group $(G, *)$ in conjunction with a homomorphic one-way function (HOWF) f, a *tree blinding* operation based on f maps X to another key tree Y, denoted by $Y = f(X)$. Y is computed by applying f to every node of X (see Figure 5). We call Y a *blinded tree* of X.

Theorem 4: *For an arbitrary HOFT X over (G, *), the blinded tree of X,* i.e., $f(X)$ *is also a HOFT.*

Proof: Let X is an arbitrary HOFT and $Y = f(X)$. We prove Y is also a HOFT. For an arbitrary node secret $y_i \in Y$, we have

$y_i = f(x_i)$

$= f(f(x_{2i}) * f(x_{2i+1}))$ (X is a HOFT)

$= f(y_{2i} * y_{2i+1})$ ($Y = f(X)$)

$= f(y_{2i}) * f(y_{2i+1})$ (f is homomorphic)

Thus, Y is a HOFT according to Definition 1. □

It follows that tree blinding is also a structure-preserving operation. Due to one-wayness of *f*, tree blinding operation helps conceal information about each node secrets of a key tree without compromising its structure.

C. Algorithms for handling leaf nodes in HOFTs

In tree-based MKD protocols, adding or removing members involves adding or removing corresponding leaf nodes in a key tree. In this section, we provide algorithms for handling (adding, removing, or changing) leaf nodes in a HOFT X without compromising its structure by performing a tree product of X and an *incremental tree* (or *incremental chain*). First of all, we introduce a concept called *Combined Ancestor Tree* that was proposed by Sherman and McGrew [5]. For a set of leaf nodes, the subtree consisting of all ancestors of those leaf nodes is called a *Combined Ancestor Tree* (CAT). Especially, an *ancestor chain* is an instance of a CAT that has one single leaf node. In below, we only discuss the general (complicate) case — adding/removing/changing multiple leaf nodes in a HOFT. Algorithms for adding/removing/changing a single leaf node can be easily derived from those given here.

1) Algorithm 1 — adding multiple leaf nodes

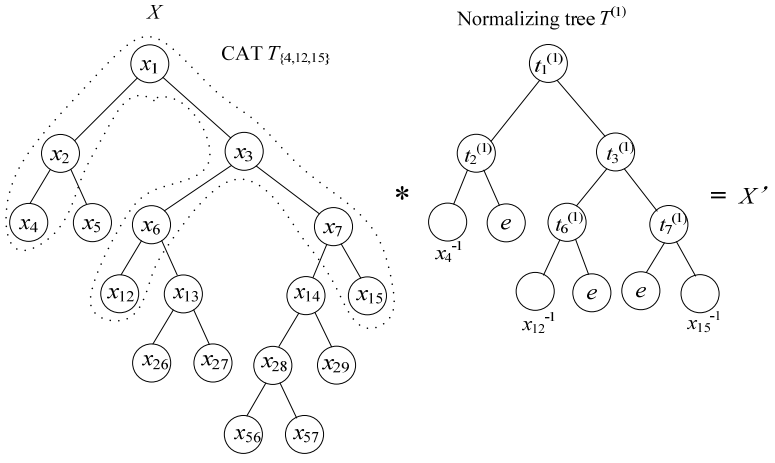

Fig. 6 Normalization for multiple additions

Referring to Figure 6, we want to add leaf nodes x_9, x_{25}, and x_{31} to X. To maintain the balance of a key tree, we prefer to choose those shallowest nodes as the *growing points* (i.e., the place where a new node will be added), for example, x_4, x_{12}, and x_{15} in this case. We use $T_{\{4,12,15\}}$to denote the CAT induced by x_4, x_{12}, and x_{15}. Adding multiple leaf nodes to a HOFT without compromising its structure takes two steps. The first step called *normalization* (depicted in Figure 6) is in fact to perform a tree product of X and a normalizing tree $T^{(1)}$. The purpose is to turn the value of each leaf node secret of the CAT $T_{\{4,12,15\}}$ into the identity element of G. The normalizing tree $T^{(1)}$ is constructed from $T_{\{4,12,15\}}$ by first changing the value of each leaf node secret of $T_{\{4,12,15\}}$ into its corresponding inverse in G, and then computing all the other internal node secrets in a bottom-up manner as described in Definition 1.

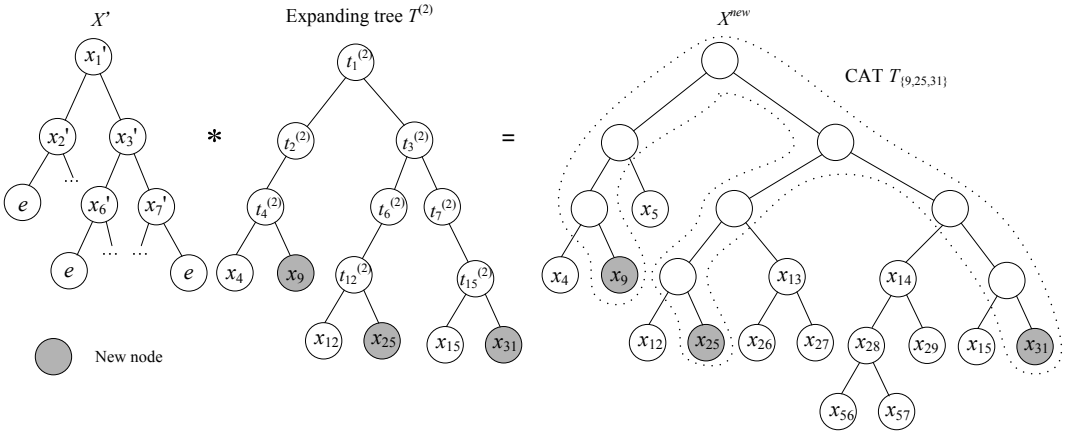

Fig. 7 Expansion for multiple additions

The second step called *expansion* (illustrated in Figure 7) is to perform a tree product of the output of the first step, i.e., X' and an expanding tree $T^{(2)}$. New leaf nodes are actually added onto X in this step. The expanding tree $T^{(2)}$ is also constructed from $T_{\{4,12,15\}}$ by first creating two new child nodes for each leaf node x_i of $T_{\{4,12,15\}}$ such that the node secret formerly associated with x_i is now associated with the left child of x_i, and a new node secret is associated with the right child

of x_i, and then computing all the other internal node secrets in a bottom-up manner. The output of expansion is the final result — an updated key tree X^{new}.

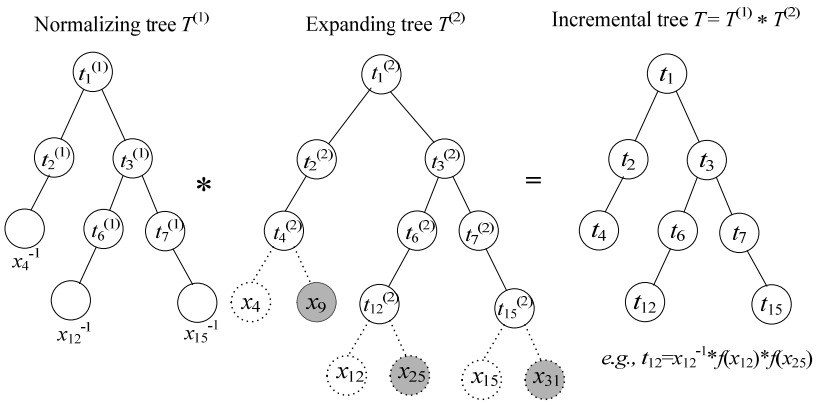

Fig. 8 An incremental tree for multiple additions

To simplify the two-step process, we introduce a concept called *incremental tree*. From X to X^{new}, the corresponding incremental tree T depicted in Figure 8 is obtained by performing a tree product of the normalizing tree $T^{(1)}$ and its counterpart in the expanding tree $T^{(2)}$. Notice that in Figure 8, we use a dotted circle (shaded or not shaded) to denote node that does not directly involve in a tree product computation, but whose position should be remembered. Now, X^{new} can be obtained from X by first directly performing a tree product of X and T, then creating two new child nodes for each node whose place is formerly a growing point, such that the node secret formerly associated with a growing point is now associated with one child, and a new node secret is associated with the other child.

2) Algorithm 2 — removing multiple leaf nodes

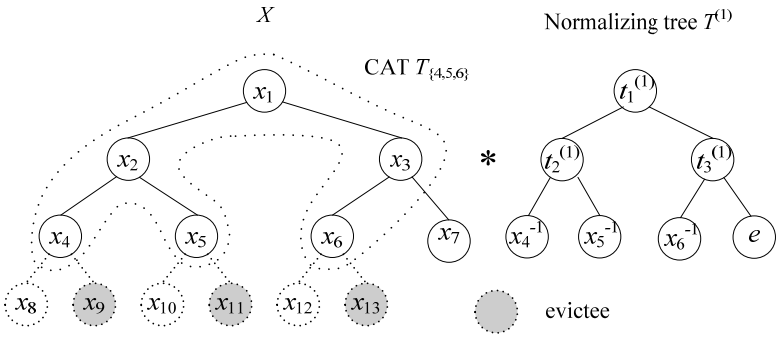

Fig. 9 Normalization for multiple removals

As illustrated in Figure 9, to remove x_9, x_{11} and x_{13} from a key tree X without compromising its structure, the first step is similar to the normalization step for adding multiple leaf nodes.

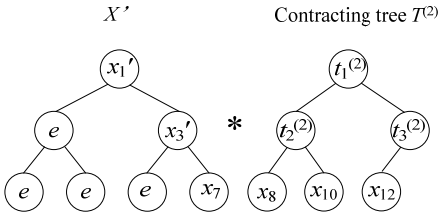

Fig. 10 Contraction for multiple removals

The second step called *contraction* as illustrated in Figure 10, is to perform a tree product of the output of the first step, i.e., X' and a contracting tree $T^{(2)}$. The contracting tree $T^{(2)}$ is constructed from CAT $T_{\{4,5,6\}}$ by first replacing each leaf node of $T_{\{4,5,6\}}$ with its child in X not to be removed, and then computing all the other internal node secrets in a bottom-up manner.

Similar to the incremental tree for multiple additions (Figure 8), the incremental tree T for these multiple removals can also be obtained by performing a tree product of the normalizing tree $T^{(1)}$ and the contracting tree $T^{(2)}$. Now, the updated new key tree X^{new} can be obtained from X by first deleting each leaf node in need of removing and its corresponding sibling (even if it is a subtree), then performing a tree product of the resulting tree and T.

3) *Algorithm 3 — changing the value of multiple leaf nodes*

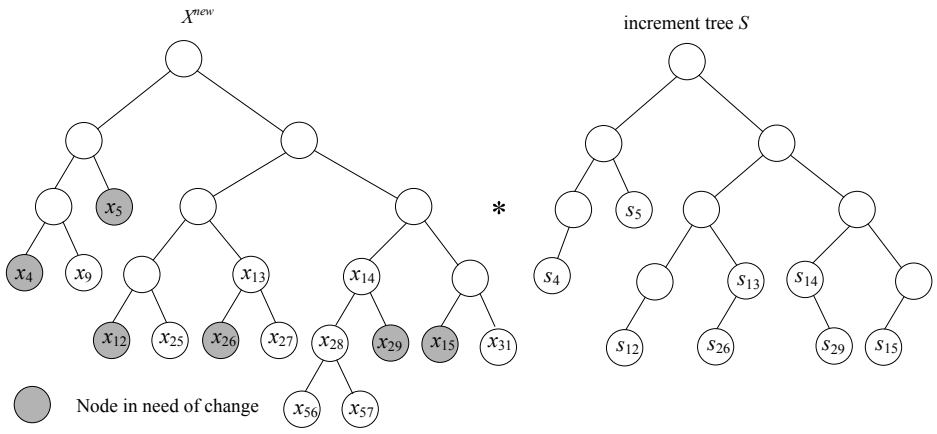

Fig. 11 Changing the value of multiple leaf node

As illustrated in Figure 11, to change the value of leaf nodes x_4, x_5, x_{12}, x_{26}, x_{29}, and x_{15} in X^{new} into a new value without affecting other leaf nodes and compromising the structure of X^{new}, we randomly select seeds s_4, s_5, s_{12}, s_{26}, s_{29}, and s_{15}. Then we compute an increment tree S in the bottom-up manner and perform a tree product of X^{new} and S. Of course, the value of all the internal nodes in the path from the changed leaf node to the root node is also changed. Obviously, to trigger changing the value of the root node of a HOFT T without compromising T's structure, we can choose to change a shallowest leaf node in T using the above algorithm. An efficient way to change all leaf nodes of a HOFT without compromising it structure is using the tree blinding operation.

V. A COLLUSION-RESISTANT MKD PROTOCOL BASED ON HOFTS

Employing the three algorithms provided in Section IV, we are able to put forward a collusion-resistant MKD protocol (we call it the HOFT protocol). When members join or leave, all the node secrets on the corresponding CAT should be changed. The key server uses algorithms provided in Section IV to construct an incremental tree (or chain), and update the key

tree by performing a tree product of it and the incremental tree. After that, the key server needs to communicate all the changes in the key tree to group members by broadcasting the incremental tree (or chain) such that legitimate members can update their rekeyed node secrets and rekeyed blinded node secrets by multiplying those secrets by their corresponding incremental secrets. The essential task of a MKD protocol based on HOFTs is to strictly control access to the incremental tree (or chain) to ensure group forward secrecy and group backward secrecy.

Bursty behaviour (a number of membership changes happen simultaneously), periodic group rekeying or batch group rekeying all require a *bulk operation* that can process multiple membership changes simultaneously. The broadcast size and computational effort of multiple additions and removals can be substantially reduced by using a bulk operation that removes and adds multiple members simultaneously rather than repeatedly applying individual adding or removing operations. This reduction results from the fact that a set of individual operations may repeatedly change node secrets along common segments of the key tree. Although we only present the protocol for adding/removing multiple members, group rekeying protocol for adding/removing a single member can be easily derived from those given here.

In the following passages, for any leaf node of a key tree, we shall interchangeably refer to it and its associated member for simplicity. Similar to OFT, we also use a pseudorandom OWF g to compute each node key K_v from its corresponding node secret x_v, i.e, $K_v = g(x_v)$.

A. Removing multiple members in a bulk operation

Taking the scenario depicted in Figure 9 as an example, to remove members x_9, x_{11} and x_{13} from the current key tree X, the key server uses algorithm 2 provided in Section IV to compute a incremental tree T except that during contraction operation, it needs to replace each leaf node of

CAT $T_{\{4,5,6\}}$ (resp. x_4, x_5, x_6) with a rekeyed sibling of an evictee (resp. x_8', x_{10}', x_{12}'). Then the key server sends the blinded version of each incremental secret t_i encrypted under the node key associate with the sibling of x_i^{new} (if exists) in the updated key tree X^{new}, for example, $\{f(t_2)\}\_K_3^{new}$, $\{f(t_3)\}\_K_2^{new}$, $\{f(t_4)\}\_K_5^{new}$ (recall that $K_i^{new} = g(x_i^{new})$). In addition, the key server sends the sibling of each evictee a new node secret encrypted under its old value, i.e., $\{x_8'\}\_K_8$, $\{x_{10}'\}\_K_{10}$, $\{x_{12}'\}\_K_{12}$ (recall that $K_i = g(x_i)$).

After every legitimate member receives the rekeying message, it extracts all blinded incremental secrets it is entitled to, and computes all incremental secrets it is entitled to in the incremental tree T in a bottom-up manner, and then updates its own rekeyed node secrets/ blinded node secrets by multiplying their old values by their corresponding incremental secrets.

Consider a full and balanced HOFT with n members. When l members leave, we can compute the broadcast overhead by the size of the induced incremental tree. We denote the size of the incremental tree induced by l leaving (joining or changing) members by $S(l)$. According to [5], $S(l)$ satisfies $2l-1+\lfloor \log_2(n/l) \rfloor < S(l) < 2l-1+l*\lfloor \log_2(n/l) \rfloor$. To remove l members, the key server needs to encrypt and send $S(l)+l$-1 secrets. In addition, the key server needs to perform $2S(l)$ modular multiplication computations and $S(l)$-1 OWF computations.

B. Adding multiple members in a bulk operation

At first, we give the following concept.

Definition 4 Dangling subtree — For a tree X and a subtree T of X, let v be a vertex in T. If v has a child not in T then every subtree rooted at a child of v not in T is called a *dangling subtree* of T in X.

For example, referring to the right part of Figure 7, all the dangling subtrees of $T_{\{9, 25, 31\}}$ in X^{new} are subtrees respectively rooted at x_4, x_5, x_{12}, x_{13}, x_{14}, and x_{15} (Notice that some subtree may contain only one node). For a full binary tree with n leaves, the number of dangling subtrees of an arbitrary subtree with l leaves are at most $l*\log_2(n/l)$ (this claim immediately follows from Claim 1 of [13]).

Taking the scenario depicted in Figure 6 as an example, to add members x_9, x_{25}, and x_{31} to a HOFT X, the key server needs to perform the following two steps:

(1) Add x_9, x_{25}, and x_{31} onto X

The key server uses the same algorithm 1 provided in Section IV on X to compute the corresponding incremental tree T (refer to Fig. 8, and here we call it *the first incremental tree*) and update the key tree X. Let X^{new} denote the updated key tree obtained after the first step (Fig. 7).

(2) Change all the blinded node secrets before supplying them to each joining member

As mentioned in Section II-C, to prevent the possible collusion between a pair of evicted member and joining member, one of solutions is changing all the blinded node secrets before supplying them to each joining member. Referring to the HOFT X^{new} in Figure 7, all the corresponding node secrets in need of change are accordingly the root nodes of all the dangling subtrees of CAT $T_{\{9,25,31\}}$, i.e., x_4, x_5, x_{12}, x_{13}, x_{14}, and x_{15}. As stated in algorithm 3 in Section VI, to trigger changing the root node of each dangling subtree, we can choose to change a shallowest leaf node of it. Therefore, to change the root nodes of all the dangling subtrees of CAT $T_{\{9,25,31\}}$, the leaf nodes in need of change accordingly are x_4, x_5, x_{12}, x_{26}, x_{29}, and x_{15}. Now we use algorithm 3 provided in Section IV to compute an increment tree S (see Fig. 11, here we call it

the second incremental tree) and update the tree X^{new}. Let X^{Fin} denote the final updated key tree obtained after the second step.

To distribute the first incremental tree T to old members, the key server sends every leaf node secret of T encrypted under the old group key K_1 [recall that $K_1 = g(x_1)$], i.e., $\{t_4, t_{12}, t_{15}\}\_K_1$. To distribute the second incremental tree S to old members, the key server sends every leaf node secret of S encrypted under the old group key K_1, , i.e., $\{s_4, s_5, s_{12}, s_{26}, s_{29}, s_{15}\}\_K_1$. After decrypting these two messages, every old member can reconstruct these two incremental trees T and S. Therefore, they can accordingly update their own rekeyed node secrets and rekeyed blinded node secrets. In addition, the key server also needs to supply every joining member with blinded node secrets it is entitled to, i.e., $\{f(x_4^{Fin}), f(x_5^{Fin}), f(x_3^{Fin})\}\_K_9$, $\{f(x_{12}^{Fin}), f(x_{13}^{Fin}), f(x_2^{Fin})\}\_K_{25}$, $\{f(x_{15}^{Fin}), f(x_{14}^{Fin}), f(x_6^{Fin}), f(x_2^{Fin})\}\_K_{31}$ (recall that the final updated key tree is denoted by X^{Fin}).

Consider a full and balanced OFT with n members (after l members join). When l members join the group, the key server needs to encrypt and send $(l+l*\log_2(n/l)+l*\log_2 n)$ (Keys). Recall that $S(l)$ denotes the size of the incremental tree induced by l leaving (joining, or changing) members. Also recall that the number of leaf nodes of the second incremental tree equals to the number of dangling trees of the CAT, i.e., $l*\log_2(n/l)$ at most. Therefore, the size of the first increment tree and the second incremental tree are respectively $S(l)$ and $S(l*\log_2(n/l))$. In addition, the key server needs to perform $l+S(l)+S(l*\log_2(n/l))$ OWF computations and $2(S(l)+S(l*\log_2(n/l))+l)$ multiplication computations.

VI. SECURITY PROOF

In [14], Panjwani develops a symbolic security model for studying GKD protocols. In this model, all keys and messages generated by a GKD protocol are treated as abstract data types and cryptographic primitives as abstract functions over such data types. Security notions for GKD protocols are defined in such a model. Panjwani also proves the security of the LKH protocol [3] and subset cover protocols [13] using a straightforward inductive argument. We prove the security of the HOFT protocol under this model as well. Since a rigorous treatment will occupy a full-length paper, we prove it in a less rigorous (i.e., heuristic) manner. In the following discussion, we will use some notations given by [14]. Consider a multicast group of n users, labelled 1, 2, … , n. Each user i shares a long-lived key K_i with the key server. At any time t, users in a specific set $S^{(t)} \subseteq \{1,2, \ldots , n \}$, referred to as legitimate members at that time, are authorized to receive information sent over the multicast channel. The key used to encrypt all the information sent to $S^{(t)}$ is called the group key, denoted by $K^{(t)}$. The HOFT corresponding to $S^{(t)}$ is denote by $Tr^{(t)}$. Let $[n]$ denote the set $\{1, \ldots, n\}$ and let $2^{[n]}$ denote the power set of $[n]$. Obviously, the group dynamics up to time t can be represented by a sequence of sets $\vec{S}^{(t)} = (S^{(0)}, S^{(1)}, \cdots, S^{(t)}) \in (2^{[n]})^t$. A sequence $\vec{S}^{(t)} \in (2^{[n]})^t$ is called *simple*, if for all $t \geq 1$, $S^{(t-1)}$ changes into $S^{(t)}$ through a single change in membership. According to [14],we only need to consider simple sequence as arbitrary group membership updates can be simulated using simple sequences only.

We present the following security definitions adapted from the related security definitions given by [14].

Definition 5: A n-user OFT-based protocol Π (including the HOFT protocol) is called *correct*, if for all $t\geq 0$, for all simple sequence $\vec{S}^{(t)} \in (2^{[n]})^t$, $\forall i \in S^{(t)}$, i always knows the node secrets on the path from its associated leaf node to the root (and therefore the corresponding blinded keys along this path) in $Tr^{(t)}$, and the blinded node secrets that are siblings to this path, and no other node secrets nor blinded secrets in $Tr^{(t)}$.

Definition 6: An n-user GKD protocol Π is called *secure against single user attack*, if for all $t\geq 0$, for all simple sequence $\vec{S}^{(t)} \in (2^{[n]})^t$, $\forall i \notin S^{(t)}$, i can never recover (compute) any node secret in $Tr^{(t)}$ (including $K^{(t)}$) from K_i and the entire rekeying messages throughout the lifetime of the group.

Definition 7: An n-user GKD protocol Π is called *secure against collusion attacks*, if for all $t\geq 0$, for all simple sequence $\vec{S}^{(t)} \in (2^{[n]})^t$, an arbitrary set of users $Col=\{i | i \notin S^{(t)}\}$, Col can never recover any node secret in $Tr^{(t)}$ (including $K^{(t)}$) from $\{K_i | \ i \in Col\}$ and the entire rekeying messages throughout the lifetime of the group.

It is easy to derive that security against collusion attacks implies group forward secrecy and group backward secrecy for a GKD protocol.

Theorem 5: *The HOFT protocol is correct and secure against single user attack.*

Proof: We prove this claim using induction over t. For $t=0$, since $S^{(0)}=\phi$, the claim is trivially true. Now we argue that if the claim is true for some $t\text{-}1\geq 0$, then it is true for t as well. For any simple sequence $\vec{S}^{(t)} = (S^{(0)}, S^{(1)}, \cdots, S^{(t-1)}, S^{(t)})$, we only need to consider the following cases:

Case 1 ($i \in S^{(t-1)} \wedge i \in S^{(t)}$, and $S^{(t-1)}$ changes into $S^{(t)}$ due to a member's leaving): According to the HOFT protocol, i can only recover all those incremental secrets corresponding to its rekeyed node secrets in $Tr^{(t-1)}$ and all the blinded incremental secrets corresponding to its rekeyed blinded

node secrets in $Tr^{(t-1)}$ from the rekeying messages. From inductive hypothesis, i holds all the node secrets and blinded node secrets in $Tr^{(t-1)}$ as required by Definition 5. Therefore, it still holds and only holds all the node secrets and blinded node secrets in $Tr^{(t)}$ as required by Definition 5.

Case 2 ($i \in S^{(t-1)} \wedge i \in S^{(t)}$, and $S^{(t-1)}$ changes into $S^{(t)}$ due to a member's joining): According to the HOFT protocol, even though i can recover all the incremental secrets in the first incremental chain (notice that $\vec{S}^{(t)}$ is simple) and the second incremental tree at time t from the rekeying messages, it can only update those rekeyed node secrets and blinded node secrets it holds in $Tr^{(t-1)}$. From inductive hypothesis, i holds all the secret materials in $Tr^{(t-1)}$ as required by Definition 5. Therefore, it still holds and only holds all the secret materials in $Tr^{(t)}$ as required by Definition 5.

Case 3 ($i \notin S^{(t-1)} \wedge i \in S^{(t)}$): That is to say, i joins the group at time t. According to the HOFT protocol, every newly joining member i can recover just the same secret materials as required by Definition 5 from the rekeying messages.

Case 4 ($i \in S^{(t-1)} \wedge i \notin S^{(t)}$): That is to say, i is evicted at time t. From the inductive hypothesis, all secrets that i knows are the node secrets on its path to the root and the blinded node secrets that are siblings to this path in $Tr^{(t-1)}$. However, according to the HOFT protocol, all the node secrets on i's path to the root in $Tr^{(t-1)}$ are changed at time t. Even though some blinded node secrets in $Tr^{(t-1)}$ that i brings out may not change at time t, each incremental secret in the incremental chain at time t is encrypted not by any blinded node secret, but by a node secret that is one of the siblings to i's path to the root in $Tr^{(t-1)}$, and thus unknown to i. Therefore the whole incremental chain at time t is inaccessible to i, i can never compute any node secret on $Tr^{(t)}$.

Case 5 ($i \notin S^{(t-1)} \wedge i \notin S^{(t)}$): That is to say, i is evicted before time t-1. From the inductive hypothesis, i can never recover (compute) any node secret in $Tr^{(t-1)}$. While in the rekeying message at time t,

every incremental secret is encrypted by a node secret in $Tr^{(t-1)}$ (in the case of both leave rekeying and join rekeying). Therefore, the whole incremental chain in the case of leave rekeying at time t (resp. the first incremental chain and the second incremental tree in the case of join rekeying at time t) is inaccessible to i, and thus i can never compute any node secret on $Tr^{(t)}$. □

In fact, the OFT protocol can also be proved to be correct and secure against single user attack in the same way. In their proof of Xu's proposition 1 and Xu's proposition 2, Xu et al. implicitly assume that the OFT protocol is correct and secure against single user attack.

Theorem 6: *An arbitrary pair of colluding members A and C cannot compute any node secret which they are not supposed to know by a HOFT protocol that is correct and secure against single user attack.*

Proof: As in the proof of Xu's proposition 2, we use Figure 2 to analyse all possible node keys that can be computed by a pair of colluding members. In the following three cases: (1) A is evicted after C joins; (2) C joins after A joins; (3) C is evicted after A is evicted, our proof is just the same as Xu's proposition 2 and thus omitted. We only consider the remaining cases as below.

(4) A and C are both evicted at the same time. Referring to Figure 2, according to the HOFT protocol, all the node secrets on their paths to the root are changed after A and C are evicted. Furthermore, their knowledge about the siblings (such as node R' and R'') to the intersection of their paths is same. Therefore, they cannot compute any node secret besides what they already know.

(5) A and C join at the same time. This case is similar to case (4). Thus the proof is omitted.

(6) C joins after A is evicted. Referring to Figure 2, suppose that A is evicted at time t_A and later C joins the group at time t_C. According to the HOFT protocol, all the blinded node secrets of siblings (such as L, R', and R'') to C's path to the root are changed at time t_C. Therefore,

according to Xu's proposition 1, we have $[t_{BMAX}, t_{DMIN}]= \phi$. Thus, A and C cannot compute any node secret besides what they already know. □

Theorem 7: *The HOFT protocol is secure against collusion attacks.*

Proof: We prove this claim by contradiction. According to Definition 7, suppose that there exist a $t_0 \geq 0$ and a simple sequence $\vec{S}^{(t_0)} \in (2^{[n]})^{t_0}$ such that a set of users $Col_0 = \{i \mid i \notin S^{(t_0)}\}$ can recover a node secret x_i in $Tr^{(t_0)}$ using their long-lived keys and the entire rekeying messages. Because the HOFT protocol is secure against single user attack (according to Theorem 5), for each $i \in Col_0 \wedge i \notin S^{(t_0)}$, i can never recover any node secret in $Tr^{(t_0)}$. Therefore, users in Col_0 must collude to compute the node secret x_i. According to Theorem 1, there exists a pair of users (who may not belong to Col_0) who can collude to compute a node secret not already known. This contradicts Theorem 6. □

VII. COMPARISON WITH OTHER PROTOCOLS

TABLE 1 COMPARISON WITH RELATED PROTOCOLS

(1) ADDING A MEMBER

	LKH	OFT	Ku&Chen	Xu	HOFT
Collusion attack	no	**yes**	no	no	no
Broad. size(bits)	$2\log_2 n * K$	$(2\log_2 n + 1) * K$	$(2\log_2 n + 1) * K$	$(2\log_2 n + 1) * K$ or $((\log_2 n)^2 + 2\log_2 n + 1) * K$ (when detecting collusion)	$(2\log_2 n + 1) * K$
Server comp.	$2\log_2 n * C_E$	$(2\log_2 n + 1) * C_E + (\log_2 n + 1) * C_h$	$(2\log_2 n + 1) * C_E + (\log_2 n + 1) * C_h$	$(2\log_2 n + 1) * C_E + ((\log_2 n)^2 + 2\log_2 n + 1) * C_E + ((\log_2 n)^2 + \log_2 n + 1) * C_h$ (when detecting collusion)	$(2\log_2 n + 1) * C_E + (1 + \log_2 n + S(\log_2 n)) * C_f + (2S(\log_2 n) + 1) * C_M$ (at most)
Max. old mem. comp.	$\log_2 n * C_D$	$2C_D + \log_2 n * C_h$	$2C_D + \log_2 n * C_h$	$2C_D + \log_2 n * C_h$ or $(\log_2 n + 1) * C_D + 0.5 * (\log_2 n)^2 * C_h$ (when detecting collusion)	$(1 + \log_2 n) * C_D + (\log_2 n + S(\log_2 n)) * C_f + 2\log_2 n * C_M$ (at most)

(2) REMOVING A MEMBER

	LKH	OFT	Ku&Chen	Xu	HOFT
Collusion attack	no	yes	no	no	no
Broad. size(bits)	$2\log_2 n * K$	$(\log_2 n + 1) * K$	$((\log_2 n)^2 + \log_2 n + 1) * K$	$(\log_2 n + 1) * K$	$(\log_2 n + 1) * K$
Server comp.	$2\log_2 n * C_E$	$(\log_2 n + 1) * C_E + \log_2 n * C_h$	$((\log_2 n)^2 + \log_2 n + 1) * C_E + ((\log_2 n)^2 + \log_2 n) * C_h$	$(\log_2 n + 1) * C_E + \log_2 n * C_h$	$(\log_2 n + 1) * C_E + (\log_2 n - 1) * C_f + (\log_2 n + 2) * C_M$
Max. mem. comp.	$\log_2 n * C_D$	$C_D + \log_2 n * C_h$	$\log_2 n * C_D + 0.5 * (\log_2 n)^2 * C_h$	$C_D + \log_2 n * C_h$	$C_D + \log_2 n * C_f + (\log_2 n + 1) * C_M$

(3) ADDING *l* MEMBERS

	LKH	OFT	HOFT
Collusion attack	no	yes	no
Broad. size(bits)	$(2S(l) - l) * K$	$(S(l) + l * \log_2 n) * K$	$(l + l * \log_2(n/l) + l * \log_2 n) * K$ (at most)
Server comp.	$(2S(l) - l) * C_E$	$S(l) * C_E + (2S(l) - l) * C_h$	$(l + l * \log_2(n/l) + l * \log_2 n) * C_E + (l + S(l) + S(l * \log_2(n/l))) * C_f + 2(S(l) + S(l * \log_2(n/l)) + l) * C_M$ (at most)
Max. old mem. comp.	$\log_2 n * C_D$	$(l + 1) * C_D + \log_2 n * C_h$	$(l + l * \log_2(n/l)) * C_D + (S(l) + S(l * \log_2(n/l))) * C_f + 2\log_2 n * C_M$ (at most)

(4) REMOVING *l* MEMBERS

	LKH	OFT	HOFT
Collusion attack	no	yes	no
Broad. size(bits)	$(2S(l) - l) * K$	$(S(l) + l - 1) * K$	$(S(l) + l) * K$
Server comp.	$(2S(l) - l) * C_E$	$(S(l) + l - 1) * C_E + 2S(l) * C_h$	$(S(l) + l) * C_E + (S(l) - 1) * C_f + 2S(l) * C_M$ (at most)
Max. mem. comp.	$\log_2 n * C_D$	$l * C_D + \log_2 n * C_h$	$l * C_D + \log_2 n * C_f + (2\log_2 n + 1) * C_M$ (at most)

We summarize related discussions in Section II and Section V to present a comparison between our protocol and related protocols, covering the following measures: collusion attack,

broadcast size (in bits), key server's computational overhead and maximum (old) member computational cost. The two solutions to improve the OFT protocol respectively proposed by Ku et al. and Xu et al. are referred to as Ku&Chen protocol and Xu protocol. Since both protocols did not give the specific algorithms for processing multiple membership changes, we omit them from relevant comparison. Cost analysis for batch group rekeying using LKH is based on protocol proposed by Li et al. [15]. In Table 1, n is the number of members in the group and $S(l)$ is the size of the CAT induced by l leaving (joining or changing) members (Recall that $2l-1+\lfloor \log_2(n/l) \rfloor < S(l) < 2l-1+l*\lfloor \log_2(n/l) \rfloor$). K is the size of a cryptographic key or (blinded) node secret in bits. C_E, C_D, C_h, C_f, and C_M respectively denote the computational cost of one evaluation of the encryption function E, one evaluation of the decryption function D, one evaluation of hash function, one evaluation of trapdoor OWF f, and one modular multiplication respectively.

In case of leave rekeying, the OFT protocol and the HOFT protocol nearly half the communicational overhead of the LKH protocol. Another advantage of the OFT protocol over LKH lies in member's computational overhead. Surprisingly, this merit possessed by OFT has been noticed neither by its inventors nor by existing literatures. Unfortunately the OFT protocol is subject to collusion attack. Ku & Chen protocol improves the OFT protocol to prevent collusion attack by changing all the node secrets and blinded node secrets known by an evictee on every member eviction, which require a broadcast of quadratic size. Xu protocol only performs additional blinded node secret update when detecting a possible collusion between an evictee and a joining member, thus it has lower communication overhead than Ku & Chen protocol. Among all collusion-resistant protocols based on OFT, HOFT is the only one that achieves collusion resistance without losing communicational efficiency compared to the

original OFT protocol. Moreover, when adding l multiple members in a bulk operation, the worst-case communication cost of HOFT protocol is less than that of the original OFT protocol by $(l\text{-}1)*K$ bits. However, due to using a trapdoor OWF (e.g., Rabin function) and modular multiplication instead of a much faster hash function (e.g., SHA1) and exclusive-or operation as in the original OFT protocol, HOFT has higher computational overhead than the original one, especially in conducting join rekeying. However, according to Canetti et al. [16], among all measures used to evaluate a MKD protocol, communication complexity is probably the most important one, as it is the biggest bottleneck in current applications. Therefore, it is worth trading off moderate computational cost for achieving collusion resistance and at the same time, not losing the communication efficiency compared to the original OFT protocol.

VIII. CONCLUSION AND FUTURE RESEARCH

In this paper, we instantiate the general notion of One-way Function Tree to obtain a new cryptographic construction named HOFT. Employing HOFTs and related algorithms, we propose a collusion-resistant MKD protocol without losing the communication efficiency compared to the OFT protocol. Finding other meaningful application of HOFT is a worthwhile topic. In our protocol, a HOFT is constructed in a bottom-up manner. We can construct a top-down homomorphic one-way function tree. But its meaningful application should receive further study either. To instantiate the general concept of the famous *Merkle authentication tree* [17] to obtain a *homomorphic authentication tree*, we must find a self-homomorphic OWF with respect to a non-commutative operation at first. We leave as an open problem the existence of homomorphic authentication tree.

Several GKD protocols — OFT, MARKS [18], the protocol proposed by Chang et al. [19], and LORE [20] — have been shown to be vulnerable to collusion attacks. Developing rigorous analysis methodology and formal verification method for these protocols are necessary. We don't see any research result related to formal verification of GKD protocols. To the best of our knowledge, the only result related to provable security of GKD protocols is [21]. In this work, Panjwani proves that a modified version of the LKH protocol is provably-secure against adaptive adversaries in computational security model. Our future work will focus on the provable security of OFT-based protocols (especially the HOFT protocol). Developing a formal method for automatic verification of GKD protocols is also an ongoing work.

ACKNOWLEDGMENT

The authors would like to thank R. L. Rivest for valuable discussion. The authors also would like to thank anonymous referees whose comments helped to improve this paper greatly.

REFERENCES

[1] A. Fiat, and M. Naor, "Broadcast encryption," *Advances in Cryptology - Crypto'93*, Lecture Notes in Computer Science D. R. Stinson, ed., USA-Santa Barbara, California: Springer-Verlag, August 1993.
[2] L. Cheung, J. A. Cooley, R. Khazan, and C. Newport, "Collusion-Resistant Group Key Management Using Attribute-Based Encryption," in First International Workshop on Group-Oriented Cryptographic Protocols (GOCP), 2007.
[3] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE-ACM Transactions on Networking,* vol. 8, no. 1, pp. 16-30, Feb, 2000.
[4] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key Management for Multicast: Issues and rchitectures," *Internet Draft*, Internet Eng. Task Force, 1998.
[5] A. T. Sherman, and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE Transactions on Software Engineering,* vol. 29, no. 5, pp. 444-458, May, 2003.
[6] D. Micciancio, and S. Panjwani, "Optimal communication complexity of generic multicast key distribution," *IEEE-ACM Transactions on Networking,* vol. 16, no. 4, pp. 803-813, Aug, 2008.
[7] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic groups: One-way function trees and amortized initialization," *draft-irtf-smug-groupkeymgmt-oft-00.txt, Internet Research Task Force*, August 2000.
[8] G. Horng, "Cryptanalysis of a Key Management Scheme for Secure Multicast Communications," *IEICE Transactions on Communications,* vol. E85-B, no. 5, pp. 1050-1051, 2002.
[9] W. C. Ku, and S. M. Chen, "An improved key management scheme for large dynamic groups using one-way function trees," in Proceedings of International Conference on Parallel Processing Workshops 2003, pp. 391-396.
[10] X. Xu, L. Wang, A. Youssef, and B. Zhu, "Preventing Collusion Attacks on the One-Way Function Tree (OFT) Scheme," in Proceedings of the 5th international conference on Applied Cryptography and Network Security, Zhuhai, China, 2007, pp. 177-193.
[11] M. O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Cambridge: Massachusetts Institute of Technology, Laboratory for Computer Science, 1979.
[12] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM,* vol. 21, no. 2, pp. 120-126, 1978.

[13] D. Naor, M. Naor, and J. B. Lotspiech, "Revocation and tracing schemes for stateless receivers," *Advances in Cryptology—CRYPTO '2001*, Lecture Notes in Computer Science, pp. 41–62, New York, 2001.
[14] S. Panjwani, "Private Group Communication: Two Perspectives and a Unifying Solution," Computer Science and Engineering Department, University of California, San Diego, San Diego, 2007.
[15] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam, "Batch rekeying for secure group communications," in Proceedings of the 10th international conference on World Wide Web, Hong Kong, Hong Kong, 2001, pp. 525-534.
[16] R. Canetti, T. Malkin, and K. Nissim, "Efficient communication-storage tradeoffs for multicast encryption," *Advances in Cryptology - Eurocrypt'99*, Lecture Notes in Computer Science J. Stern, ed., pp. 459-474, 1999.
[17] R. C. Merkle, *Secrecy, Authentication, and Public-Key Cryptosystems, Technical Report No. 1979-1*, Information Systems Laboratory, Stanford University Palo Alto, Calif, 1979.
[18] B. Briscoe, "MARKS: Zero side effect multicast key management using arbitrarily revealed key sequences," in Proceedings of Networked Group Communication 1999, pp. 301-320.
[19] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key management for secure lnternet multicast using Boolean function minimization techniques," in INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, 1999, pp. 689-698 vol.2.
[20] J. Fan, P. Judge, and M. H. Ammar, "HySOR: group key management with collusion-scalability tradeoffs using a hybrid structuring of receivers," in Proceedings of Eleventh International Conference on Computer Communications and Networks 2002, pp. 196 - 201.
[21] S. Panjwani, "Tackling adaptive corruptions in multicast encryption protocols," in Proceedings of the 4th conference on Theory of cryptography, Amsterdam, The Netherlands, 2007, pp. 21-40.

Jing Liu received the Ph.D. degree in computer application technology from University of Electronic Science and Technology of China, Chengdu, China, in 2003. From September 2003 to July 2005, he was with No.30 Institute of China Electronics Technology Group Corporation, Chengdu, China, as a postdoctoral fellow. Since 2005, he has been a lecturer at School of Information Science and Technology, Sun Yat-Sen University, China. He has also been affiliated with Guangdong Key Laboratory of Information Security and Technology, Guangzhou, China, since 2005. His current research interests mainly focus on applied cryptography and network security.

Bo Yang received the B. S. degree from Peking University in 1986, and the M. S. and Ph. D. degrees from Xidian University in 1993 and 1999, respectively. From July1986 to July 2005， he had been at Xidian University, from 2002, he had been a professor of National Key Lab. of ISN in Xidian University, supervisor of Ph.D. In May 2005, he has served as a Program Chair for the fourth China Conference on Information and Communications Security (CCICS'2005). He is currently dean, professor and supervisor of Ph.D. at College of Informatics and College of Software, South China Agricultural University. He is a senior member of Chinese Institute of Electronics (CIE), a member of specialist group on information security in Ministry of Information Industry of P.R.China and a member of specialist group on computer network and information security in Shanxi Province. His research interests include information theory and cryptography.